保密级别： 公开

报告版本：v2.0

数科转换迁移系统

接口指南

编 制 人：李 欣

审 核 人：刘 丹

批 准 人：

日 期：

修改历史记录

序号	版本号	内容	编制\日期	审核\日期	批准\日期
1	V1.0	初稿			
2	2021.10	新增 docx 预盖章			
3		新增转换带 css 的 html			
4	2021.12	新增附件 2，异常错误码对应说明			
5	2022.02	新增附件 2，状态码说明			

目　录

1. 转换服务接口

1.1. 调用接口

1.1.1. 调用方式

应用系统访问和调用转换服务的步骤如下：

1）引用数科提供的系统列基础 jar 包(具体参见 1.1.2 引用 jar 包)

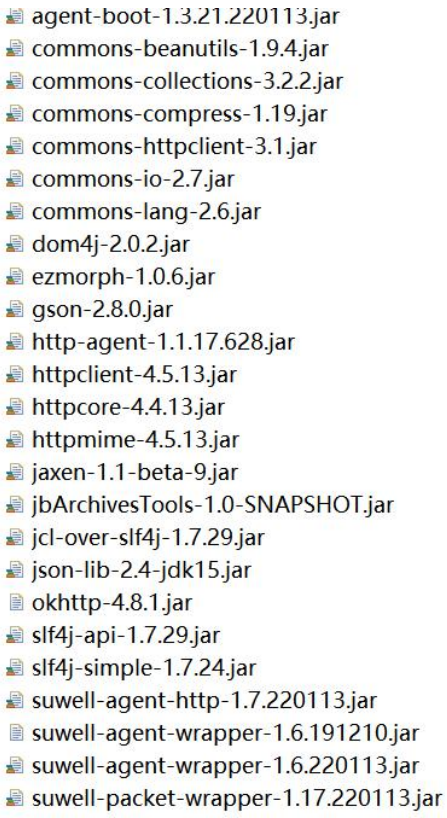
2）war 包版转换服务构造转换代理类 HTTPAgent 接口实现转换及加工，或通过 Packet 类自行拼包实现转换及加工。

3）Springboot 版转换服务构造转换代理类 HTTPAgent/AtomAgent 接口实现转换及加工，或通过 Packet 类自行拼包实现转换及加工。

4）调用相关接口（参见 1.2-1.4 章节）完成转换或加工，功能接口通用。

1.1.2. 引用 jar 包

以下 jar 包为调用转换服务依赖的基础 jar 包，不同时期发放的 jar 包小版本号可能不同，请以实际收到的资料为准。



Springboot 引用 jar 包示例图

1.1.3. 调用示例

```
    //1、定义代理对象，请求访问 war 基础版转换服务
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    private void off2OFD() {
         File in = null;
         OutputStream out = null;
         try {
              in = new File("D:\\convert\\ori\\doc_1.doc");
              out=new FileOutputStream(new File("D:\\convert\\tar\\doc_1.ofd"));
              ha.officeToOFD(in, out);
         } catch (Exception e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
         } finally {
              try {
                   ha.close();
              } catch (IOException e) {
                   e.printStackTrace();
              }
         }
     }
```

1.2. Agent 实现文件转换

com.suwell.ofd.custom.agent.HTTPAgent—使用 HTTP 方式传输文件。示例代码如下：

```
try{
 catch (Exception e) {
  while (true) {
              try {
                   Thread.sleep(50);
                  //重新提交。Packet 提交的请把 packet 也重新生成。
                  Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
                  packet.file(new Common(null, "pdf", new
FileInputStream("D:\\a.pdf")));
                  HTTPAgent  ha = new
HTTPAgent("http://localhost:8080/convert-issuer/");
                  ha.submit(packet);
                    break;
                   } catch (Exception e2) {
```

```
                    System.err.println("提交失败，重新提交！");
                }
            }
        }
```

1.2.1. 单文件转换生成 OFD

1.2.1.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 officeToOFD 方法将单个办公文件转换为 OFD 版式文件。

1.2.1.2. 接口定义

如传入源文件为文件对象，服务会根据传入的文件自动判断文件格式，并将其分配给转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。

```
void officeToOFD(File srcFile,
                 OutputStream out);
```

参数说明详见下表：

单文件转换生成 OFD 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	本地文件路径	new File("D:\\convert\\ori\\doc_1.doc")
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:\\convert\\tar\\doc_1.ofd"))

返回值说明详见下表：

单文件转换生成 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

如传入源文件为文件流(须更新 jar 包为 202201 以后版本)，需要对文件格式手动传参，服务会将其分配给指定转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。

```
void officeToOFD(InputStream in,
                 String ext,
                 OutputStream out);
```

参数说明详见下表：

单文件转换生成 OFD 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	in	InputStream 输入流	文件输入流	new FileInputStream("D:\\test1.docx")
2	ext	String 字符串	文件类型	"docx"
3	out	OutputStream 输出流	文件输出流	new FileOutputStream(new File("D:\\doc_1.ofd"))

返回值说明详见下表：

单文件转换生成 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.1.3. 接口示例

单文件转换 OFD 接口示例代码如下：

```
//java 代码调用
//参考示例：com.springboot.single.OfficeToOFD_1
//功能说明：将 doc_1.doc、test1.docx 等源文件转换成 OFD 版式文件，将转换后的文件保存至设置的目录下
public class OfficeToOFD_1 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义方法实现 office 转 ofd
    private void off2OFD() {
        try {

            //2-1、传参文件对象转换输出 OFD 文件
            //1）定义待转换的文件对象
            File srcFile = new File("D:\\doc_1.doc");
            //2)、定义转换后 OFD 文件输出流
        OutputStream outFile = new FileOutputStream(new File("D:\\off2OFD_doc1.ofd"));
            //3)、调用接口实现 office 文件转为 ofd 文件
```

```
            ha.officeToOFD(srcFile, outFile);

            //2-2、传参文件流转换输出 OFD 文件
            //1)定义待转换的文件流
            InputStream in= new FileInputStream("D:\\test1.docx");
            //2)定义文件类型,必须与文件流类型一致，否则转换失败
            String  ext = "docx";
            //3)定义转换后 OFD 文件输出流
            OutputStream out= new FileOutputStream(new
File("D:\\off2OFD_test1.ofd"));
            //4)调用接口请求服务，实现 office 文件转为 ofd 文件
            ha.officeToOFD(in,ext,out);

       } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                //2-4、关闭
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }


    public static void main(String[] args) {
            OfficeToOFD_1 office = new OfficeToOFD_1();
            office.off2OFD();
            System.out.println("转换结束......");
    }
}
```

1.2.1.4. 接口约束

支持转换流式源文件：doc、docx、wps、rtf、txt、uof、ppt、pptx、xls、xlsx、et

支持转换版式源文件：pdf 、其它私有格式（请咨询技术人员）

支持转换图片源文件：jpg、jpeg、tif、tiff、bmp、png

1.2.2. 单文件附加元数据生成 OFD

1.2.2.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 officeToOFD 接口将单个办公文件转换为 OFD 版式文件并附加元数据。

1.2.2.2. 接口定义

如传入源文件为文件对象，服务会根据传入的文件自动判断文件格式，并将其分配给指定转换节点进行转换同时加入元数据，转换完成的文件以流的形式保存至设置的路径下。

```
void officeToOFD(File srcFile,
                 OutputStream out,
                 Map<String, String> metas);
```

参数说明详见下表：

单文件加元数据转换 OFD 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	本地文件路径	new File("D:\\convert\\doc_1.doc");
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:\\convert\doc_1.ofd"));
3	metas	Map 集合	附加的元数据	Map<String, String> metas=new HashMap<String, String>() metas.put(Const.Meta.DOC_ID.value(), "0002"); // DOCID

返回值说明详见下表：

单文件加元数据转换 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

如传入源文件为文件流(须更新 jar 包为 202201 以后版本)，需要对文件格式手动传参，服务会将其分配给指定转换节点进行转换同时加入元数据，转换完成的文件以流的形式保存至设置的路径下。

```
void officeToOFD(InputStream in,
                 String ext,
                 OutputStream out,
                 Map<String, String> metas);
```

参数说明详见下表：

单文件加元数据转换 OFD 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	in	InputStream 输入流	文件输入流	new FileInputStream("D:\\test2.docx")
2	ext	String 字符串	文件类型	"docx"
3	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:\\docx2ofd_meta.ofd"));
4	metas	Map 集合	附加的元数据	Map<String, String> metas=new HashMap<String, String>() metas.put(Const.Meta.DOC_ID.value(), "0002"); // DOCID

返回值说明详见下表：

单文件加元数据转换 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.2.3. 接口示例

单文附加元数据生成 OFD 接口示例代码如下：

```
//java 代码调用
//参考示例：com.springboot.single.OfficeToOFD_2
//功能说明：将 doc_1.doc 源文件转换成 OFD 版式文件并附加元数据，将转换后文件保存至设置的目录下
public class OfficeToOFD_2 {

    //1、定义 http 代理对象，请求访问转换服务 (war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义方法实现  office 转为 ofd 并添加元数据
    private void off2OFD_meta() {
        try {
            //1)、定义 map 集合存储元数据
```

```
            Map<String, String> metas =new HashMap<String, String>();
            //2)、OFD 标准元数据的添加
            metas.put(Const.Meta.DOC_ID.value(), "111");// DOCID
            metas.put(Const.Meta.AUTHOR.value(), "我是元数据文档作者");// 作者
            metas.put(Const.Meta.TITLE.value(), "我是元数据文档标题"); // 标题
            metas.put(Const.Meta.ABSTRACT.value(), "我是元数据文档摘要");// 摘要
            metas.put(Const.Meta.SUBJECT.value(), "我是元数据文档主题");// 主题
            metas.put(Const.Meta.DOC_USAGE.value(), "我是元数据文档类型");// 文档类型
            metas.put(Const.Meta.MOD_DATE.value(), "2021-01-10");// 修改日期
            metas.put(Const.Meta.CREATOR.value(), "我是元数据创建者");
            //3)用户自定义元数据添加
            //单个自定义元数据的添加，自己定义 key 值---注意只能用一次
            metas.put(Const.Meta.CUSTOM_DATA.value(), "name=我是元数据自定义数据");
            //多个自定义元数据添加
            metas.put(Const.Meta.CUSTOM_DATAS.value(), "a=自定义数据 1,b=自定义 2");
            // 关键字添加
            metas.put(Const.Meta.KEYWORD.value(), "我是元数据关键字");
            // 关键字集，“，”分隔
            metas.put(Const.Meta.KEYWORDS.value(), "元数据关键字 1,关键字 2");



            //2-1、通过文件对象转换输出 OFD 文件并添加元数据
            //1)定义待转换的 office 文件
            File file = new File("D:/公文正文.wps");
            //2)、定义转换后的 OFD 文件输出流
            OutputStream out = new FileOutputStream(new File("D:/off2OFD_meta.ofd"));
            //3)、调用方法将 office 文件转为 OFD 文件并添加元数据
            ha.officeToOFD(file, out, metas);



            //2-2、通过文件流转换输出 OFD 文件并添加元数据
            //1)定义待转换的 office 文件
            InputStream in= new FileInputStream(new File("D:\\test1.docx"));
            //2)定义源 office 文件类型
            String  ext = "docx";
            //3)定义转换后的 OFD 文件输出流
        OutputStream out2= new FileOutputStream(new File("D:\\off2OFD_meta2.ofd"));
            //4)调用接口请求服务，将 office 文件转为 OFD 文件并添加元数据
            ha.officeToOFD(in, ext , out2, metas);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
```

```
                //关闭 ha 资源
                ha.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        OfficeToOFD_2 office2ofd = new OfficeToOFD_2();
        office2ofd.off2OFD_meta();
        System.out.println("结束......");
    }

}
```

1.2.2.4. 接口约束

源文件约束，请参考 1.2.1.4 章节。
元数据约束，请参考附件 1。

1.2.3. 多文件合并生成 OFD

1.2.3.1. 接口描述

实现该功能需引用 Agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 officesToOFD 接口将多个办公文件转换为OFD版式文件。用于将多个源文件合并成一个OFD版式文件，文档合并的前后顺序取决于在 List 集合里添加文件的顺序。

1.2.3.2. 接口定义

```
void officesToOFD(List<File> srcFiles,
                  OutputStream out);
```

参数说明详见下表：

多文件合并生成 OFD 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

序号	参数名称	参数类型	参数含义	内容举例

1	srcFile	List 集合	本地文件路径集合	List list=new ArrayList<File>();
				list.add(new File("D:\\convert\\ori\\doc_1.doc"));
				list.add(new File("D:\\convert\\ori\\docx_1.docx"));

2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:\\convert\\tar\\offices_1.ofd"));

返回值说明详见下表：

多文件合并生成 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.3.3. 接口示例

多文件合并生成 OFD 接口示例代码如下：

```
//java 代码调用
//参考示例：com.springboot.single.OfficesToOFD_3
//功能说明：将 xxx.doc，xxx.docx 等文件合并生成 OFD 并保存至设置的目录下

public class OfficesToOFD_3 {
      //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
     //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


     //2、定义方法实现 offices 转 ofd 方法
    private  void offs2OFD() {
        //2-1、定义集合，存储待转换的文件
         List<File> fileList =new ArrayList<File>();
         //2-2、定义转换后文件输出流
         OutputStream out = null;
         try {
             //2-3、将待转换的文件添加到集合中
             fileList.add(new File("D:/测试文件.docx"));
             fileList.add(new File("D:/公文附件 002.doc"));
             fileList.add(new File("D:/a.pdf"));
             fileList.add(new File("D:/测试文件.xls"));
             //2-4、给转换后 OFD 文件流赋值
             out=new FileOutputStream(new File("D:/convert_ofd/offices2ofd.ofd"));
```

```
            //2-5、调用接口请求服务，将多个文件转为 ofd
            ha.officesToOFD(fileList, out);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }  finally {
            try {
                //2-6、关闭 ha
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、调用方法
    public static void main(String[] args) {
        OfficesToOFD_3 office2ofd = new OfficesToOFD_3();
        office2ofd.offs2OFD();
        System.out.println("转换结束......");
    }
}
```

1.2.3.4. 接口约束

源文件约束，请参考 1.2.1.4 章节。

1.2.4. 多文件合并添加元数据生成 OFD

1.2.4.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 officesToOFD 接口将多个办公文件转换为 OFD 版式文件并附加元数据。用于将多个源文件合并成一个 OFD 版式文件，文档合并的前后顺序取决于在 List 集合里添加文件的顺序。

1.2.4.2. 接口定义

```
void  officesToOFD(List<File> srcFiles,
                   OutputStream out,
                   Map<String, String> metas);
```

参数说明详见下表：

多文件合并添加元数据生成 OFD 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	srcFile	List<File>类型	多个文件列表集合	List list=new ArrayList<File>();
				list.add(new File("D:\\convert\\doc_1.doc"));
				list.add(new File("D:\\convert\\doc_2.doc"));

2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:\\convert\\tar\\doc_1.ofd"));
3	metas	Map 集合	附加的元数据	Map<String, String> metas=new HashMap<String, String>(); metas.put(Const.Meta.DOC_ID.value(), "0002");

返回值说明详见下表：

多文件合并添加元数据生成 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.4.3. 接口示例

多文件合并附加元数据生成 OFD 接口示例代码如下：

```
//java 代码调用
//参考示例: com.springboot.single.OfficesToOFD_4
//功能说明: 将 doc,docx 等类型文件合并生成 OFD 并添加附加元数据信息，将转换后的 OFD 文件保存至设置的目录下

public class OfficesToOFD_4 {
    //1、定义 http 代理对象，请求访问转换服务(war 包版)
     HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义方法实现 offices 转 ofd 添加元数据
```

```
    private static void offs2OFD() {
        //2-1、定义集合，用于存储待转换的文件
        List<File> fileList =new ArrayList<File>();
        //2-2、定义转换后 ofd 文件输出流
        OutputStream out = null;
        //2-3、定义 map 集合，用于添加元数据
        Map<String, String> metas =new HashMap<String, String>();
        try {

            //2-4、添加待转换文件
            fileList.add(new File("D:/测试文件 1.docx"));
            fileList.add(new File("D:/测试文件 4.doc"));
            fileList.add(new File("D:/测试文件 3.xls"));
            fileList.add(new File("D:/a2.pdf"));

             //2-5、添加元数据
            metas.put(Const.Meta.DOC_ID.value(), "11111111111111");// DOCID
            metas.put(Const.Meta.TITLE.value(), "我是元数据标题");//标题
            metas.put(Const.Meta.AUTHOR.value(), "我是元数据作者");//作者
            metas.put(Const.Meta.ABSTRACT.value(), "我是元数据摘要");//摘要
            metas.put(Const.Meta.SUBJECT.value(), "我是元数据 SUBJECT"); //主题
            metas.put(Const.Meta.CREATOR.value(), "我是元数据创建者");//创建者
            metas.put(Const.Meta.DOC_USAGE.value(), "我是元数据 DOCUSAGE");//文档类型
            //最后修改日期，注意该参数必须为日期
            metas.put(Const.Meta.MOD_DATE.value(), "2021-01-10");
            //单个自定义元数据的添加，自已定义 key 值---注意只能用一次
            metas.put(Const.Meta.CUSTOM_DATA.value(), "name=我是元数据自定义数据");
            //多个自定义元数据的添加
            metas.put(Const.Meta.CUSTOM_DATAS.value(), "a=自定义数据 1,b=自定义 2");
            // 关键字
            metas.put(Const.Meta.KEYWORD.value(), "我是元数据关键字");
            // 关键字集，“，”分隔
            metas.put(Const.Meta.KEYWORDS.value(), "元数据关键字 1,关键字 2,关键字 3");



             //2-6、定义转换后 ofd 文件输出流
             out=new FileOutputStream(new File("D:/convert_ofd/offices_metas.ofd"));
             //2-7、调用接口请求服务，将 offices 文件集合转为 ofd 文件并添加元数据
             ha.officesToOFD(fileList, out,metas);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
```

```
                //2-8、关闭 ha
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        OfficesToOFD_4 offices2ofd = new OfficesToOFD_4();
        offices2ofd.offs2OFD();
        System.out.println("-----转换结束------");
    }
}
```

1.2.4.4. 接口约束

源文件约束，请参考 1.2.1.4 章节

元据据约束，请参考附件 1。

1.2.5. 单图片文件生成 OFD，自定义宽度

1.2.5.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 imagesToOFD 接口将单个图片文件转换为 OFD 版式文件。

1.2.5.2. 接口定义

```
void imagesToOFD(InputStream in,
                 String ext,
                 OutputStream out,
                 float pgWidth);
```

参数说明详见下表：

单图片文件生成 OFD，自定义宽度接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	in	InputStream 输入流	源文件输入流	new FileInputStream("D:\\test.jpg");
2	ext	String 字符串类型	源文件的文件类型	"jpg"
3	out	OutputStream 输出流	转换后 OFD 文件输出流	new FileOutputStream(new File("D:\\images_3.ofd"));
4	pgWidth	float 类型 单位：像素	输出图片宽度	float pgWidth=200;

返回值说明详见下表：

单图片文件生成 OFD，自定义宽度接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.5.3. 接口示例

单个图片文件生成 OFD，自定义宽度生成 OFD 接口示例代码如下：

```
//java 代码调用
//参考示例: com.springboot.single.ImagesToOFD_5
//功能说明: D:/1.png 文件生成 OFD 并将宽度设置为 200,保存至设置的目录下

public class ImageToOFD_5 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


    //2、定义方法实现 图片转为 ofd 文件并设置宽度
    private void imageToOFD() {


            try {
                //1)定义待转换的文件流
                InputStream in= new FileInputStream("D:\\1.png");
                //2)定义文件类型,必须与文件流图片类型一致，否则转换失败
                String  ext = "png";
              //3)定义转换后 OFD 文件输出流
        OutputStream out=new FileOutputStream(new File("D:/img2ofd_width.ofd"));
                //4)、定义图片宽度
```

```
                float imgWidth = 50;
                //5)、调用接口请求转换服务，将图片转为 ofd 并设置图片宽度
                ha.imagesToOFD(in, ext, out,imgWidth);

            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } finally {
                try {
                    //2-7、关闭 ha
                    ha.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        //3、方法调用
        public static void main(String[] args) {
            ImageToOFD_5 iamge2ofd = new ImageToOFD_5();
            iamge2ofd.imageToOFD();
            System.out.println("转换结束......");
        }
}
```

1.2.5.4. 接口约束

支持转换图片源文件：jpg、jpeg、tif、tiff、bmp、png

1.2.6. 单图片文件生成 OFD 自定义宽度，添加元数据

1.2.6.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 imagesToOFD 接口将单个图片文件转换为 OFD 版式文件添加元数据。

1.2.6.2. 接口定义

```
void imagesToOFD(InputStream in,
```

```
                String ext,
                OutputStream out,
                Map<String, String> metas,
                float pgWidth);
```

参数说明详见下表：

单图片文件生成 OFD 自定义宽度，添加元数据接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	in	InputStream 输入流	源文件输入流	new FileInputStream("D:\\test.jpg");
2	ext	String 字符串类型	源文件的文件类型	"jpg"
3	out	OutputStream 输出流	转换后 OFD 文件输出流	new FileOutputStream(new File("D:\\images_3.ofd"));
4	metas	Map 集合	附加的元数据	Map<String, String> metas=new HashMap<String, String>(); metas.put(Const.Meta.DOC_ID.value(), "0002");
5	pgWidth	float 类型 单位：像素	输出图片宽度	float pgWidth=200;

返回值说明详见下表：

单图片文件生成 OFD 自定义宽度，添加元数据接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.6.3. 接口示例

单个图片文件合并生成 OFD，自定义宽度生成 OFD 接口示例代码如下：

```
//java 代码调用
//参考示例: com.springboot.single.ImageToOFD_6
//功能说明: D:/1.png 文件生成 OFD 并将宽度设置为 200,添加元数据，保存至设置的目录下

public class ImageToOFD_6 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");



    //2、定义方法实现 图片转为 ofd 文件自定义宽度，添加元数据
```

```
    private void imageToOFD() {
        try {
                //1)定义待转换的文件流
                InputStream in= new FileInputStream("D:\\1.png");
                //2)定义文件类型,必须与文件流类型一致，否则转换失败
                String  ext = "png";
               //3)定义转换后 OFD 文件输出流
        OutputStream out=new FileOutputStream(new File("D:/img2ofd_width4.ofd"));
                //4)、定义 map 集合，存储元数据
                Map<String, String> metas =new HashMap<String, String>();
                //4-1)、OFD 标准元数据的添加
                metas.put(Const.Meta.DOC_ID.value(), "111");// DOCID
                metas.put(Const.Meta.AUTHOR.value(), "我是元数据文档作者");// 作者
                metas.put(Const.Meta.TITLE.value(), "我是元数据文档标题"); // 标题
                metas.put(Const.Meta.ABSTRACT.value(), "我是元数据文档摘要");// 摘要
                metas.put(Const.Meta.SUBJECT.value(), "我是元数据文档主题");// 主题
                 metas.put(Const.Meta.DOC_USAGE.value(), "我是元数据文档类型");// 类型
                metas.put(Const.Meta.MOD_DATE.value(), "2021-01-10");// 修改日期
                metas.put(Const.Meta.CREATOR.value(), "我是元数据创建者");
                //4-2)用户自定义元数据添加
                //单个自定义元数据的添加，自已定义 key 值---注意只能用一次
                metas.put(Const.Meta.CUSTOM_DATA.value(), "name=我是自定义元数据");
                //多个自定义元数据添加
                metas.put(Const.Meta.CUSTOM_DATAS.value(), "a=自定义 1,b=自定义 2");
                // 关键字添加
                metas.put(Const.Meta.KEYWORD.value(), "我是元数据关键字");
                // 关键字集，“，”分隔
                metas.put(Const.Meta.KEYWORDS.value(), "元数据关键字 1,关键字 2");

                //5)、定义图片宽度
                float imgWidth = 50;
                //6)、调用接口请求转换服务，将图片转为 ofd 并设置图片宽度
                ha.imagesToOFD(in, ext, out,metas,imgWidth);

            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } finally {
                try {
                    //2-7、关闭 ha
                    ha.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
```

```
            }
        }

        //3、方法调用
        public static void main(String[] args) {
            ImageToOFD_6 iamge2ofd = new ImageToOFD_6();
            iamge2ofd.imageToOFD();
            System.out.println("转换结束......");
        }
}
```

1.2.6.4. 接口约束

支持转换图片源文件：jpg、jpeg、tif、tiff、bmp、png

1.2.7. 多个图片文件合并生成 OFD，自定义宽度

1.2.7.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 imagesToOFD 接口将多个图片文件转换为OFD版式文件。用于将多个图片文件合并成一个OFD版式文件，文档合并的前后顺序取决于在 List 集合里添加文件的顺序。

1.2.7.2. 接口定义

```
void  imagesToOFD(List<File> srcFiles,
                  OutputStream out,
                  float pgWidth);
```

参数说明详见下表：

多图片文件合并生成 OFD，自定义宽度接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	srcFile	List<File>类型	文件集合	List<File> inList =new ArrayList<File>();
				inList.add(new File("D:\\convert\\ori\\png_1.png"));
				inList.add(new File("D:\\convert\\ori\\jpg_1.jpg"));

2	out	OutputStream 输	文件输出流	new FileOutputStream(new

序号	参数名称	参数类型	参数含义	内容举例
		出流		File("D:\\convert\\tar\\images_3.ofd"));
3	pgWidth	float 类型 单位：像素	图片宽度	float pgWidth=200;

返回值说明详见下表：

多图片文件合并生成 OFD，自定义宽度接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.7.3. 接口示例

多个图片文件合并生成 OFD，自定义宽度生成 OFD 接口示例代码如下：

```
//java 代码调用
//参考示例: com.springboot.single.ImagesToOFD_7
//功能说明: D:/2.png 和 D:/3.tiff 文件合并生成 OFD 并将设置宽度,将转换后文件保存至设置的目录下

public class ImagesToOFD_7 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
     //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
     //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义方法实现 图片转为 ofd 文件并设置宽度
     private void imagesToOFD() {
         //2-1、定义集合，用于存储待转换的 image 文件
         List<File> inList =new ArrayList<File>();
         //2-2、定义输出流，用户输出转换后的 ofd 文件
         OutputStream out = null;

         try {
              //2-3、将图片存储到集合中
              inList.add(new File("D:/1.jpg"));
              inList.add(new File("D:/2.png"));
              inList.add(new File("D:/3.bmp"));

              //2-4、赋值 输出 ofd 文件的输出流
              out=new FileOutputStream(new File("D:/convert_ofd/img2ofd_width.ofd"));
              //2-5、定义图片宽度
              float imgWidth = 200;
```

```
            //2-6、调用接口请求转换服务，将图片转为 ofd 并设置图片宽度
            ha.imagesToOFD(inList, out,imgWidth);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-7、关闭 ha
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        ImagesToOFD_7 iamges2ofd = new ImagesToOFD_7();
        iamges2ofd.imagesToOFD();
        System.out.println("转换结束......");
    }

}
```

1.2.7.4. 接口约束

支持转换图片源文件：jpg、jpeg、tif、tiff、bmp、png

1.2.8. 多个图片文件合并生成 OFD，自定义 dpi

1.2.8.1. 接口描述

将多个图片合并转换为 OFD 文件。 实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 imagesToOFD 接口将多个图片转换为 OFD 版式文件。用于将多个图片文件合并成一个 OFD 版式文件，文档合并的前后顺序取决于在 List 集合里添加文件的顺序，转换完成的文件保存至设置的路径下。

1.2.8.2. 接口定义

```
void  imagesToOFD(List<File> srcFiles,
                  OutputStream out,
                  int dpi);
```

参数说明详见下表：

多个图片文件合并生成 OFD，自定义 dpi 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	srcFile	List<File>类型	多个文件列表集合	List list=new ArrayList<File>();
				list.add(new File("D:\\convert\\ori\\png_1.png"));
				list.add(new File("D:\\convert\\ori\\jpg_1.jpg"));

2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/convert/tar/images_1.ofd"));
3	dpi	int 整型	图片的 dpi	Int dpi=300;

返回值说明详见下表：

多个图片文件合并生成 OFD，自定义 dpi 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.8.3. 接口示例

多个图片文件合并生成 OFD，自定义 dpi 接口示例代码如下：

```
//java 代码调用
//参考示例：com.springboot.single.ImagesToOFD_8
//功能说明：xxx.png,xxx.jpg,xxx.bmp 文件合并生成 OFD 并将 dpi 设置成 200，保存至设置的目录下
public class ImagesToOFD_8 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    static HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //static AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //static HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 图片转为 ofd 并自定义 dpi
    private void imagesToOFD() {
        //2-1、定义文件列表集合，用于存储待转换的图片文件
        List<File> inList =new ArrayList<File>();
        //2-2、定义文件输出流，用于输出转换后的 ofd 文件
```

```
        OutputStream out = null;

        try {
             //2-3、集合添加图片文件
             inList.add(new File("D:/1.jpg"));
             inList.add(new File("D:/2.png"));
             inList.add(new File("D:/3.bmp"));

             //2-4、文件输出流赋值
             out=new FileOutputStream(new File("D:/convert_ofd/images_dpi.ofd"));
             //2-5、定义图片 dpi
             int dpi =200;

             //2-6、调用方法，请求转换服务，将图片集合转为 ofd 文件并设置 dpi
             ha.imagesToOFD(inList, out,dpi);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        ImagesToOFD_8 images2ofd = new ImagesToOFD_8();
        images2ofd.imagesToOFD();
        System.out.println("转换结束......");
    }
}
```

1.2.8.4. 接口约束

支持转换图片源文件：jpg、jpeg、tif、tiff、bmp、png

1.2.9. 多图片文件合并附加元数据生成 OFD，自定义 width

1.2.9.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 imagesToOFD 接口将多个个图片文件转换为 OFD 版式文件。用于将多个图片文件合并成一个 OFD 版式文件，进行转换同时加入元数据。文档合并的前后顺序取决于在 List 集合里添加文件的顺序，转换完成的文件保存至设置的路径下。

1.2.9.2. 接口定义

```
void imagesToOFD(List<File> srcFile,
                 OutputStream out,
                 Map<String, String> metas,
                 float pgWidth);
```

参数说明详见下表：

多图片文件合并附加元数据生成 OFD，自定义宽度接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	List<File>类型	本地文件路径	List<File> inList =new ArrayList<File>(); inList.add(new File("D:/convert/ori/png_1.png")); inList.add(new File("D:/convert/ori/jpg_1.jpg"));
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/convert/tar/imagesToOFD_2/images_4.ofd"));
3	metas	Map<String,String>	元数据	Map<String, String> metas =new HashMap<String, String>(); metas.put(Const.Meta.DOC_ID.value(), "0002");
4	pgWidth	float 类型，单位：像素	输出图片宽度	float pgWidth=600;

返回值说明详见下表：

多图片文件合并附加元数据生成 OFD，自定义宽度接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.9.3. 接口示例

多图片文件合并附加元数据生成 OFD，自定义宽度接口示例代码如下：

```
//java 代码调用
//参考示例：com.springboot.single.ImagesToOFD_9
//功能说明：xxx.png,xxx.jpg,xxx.bmp 文件合并生成 OFD 并设置宽度，保存至设置的目录下
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class ImagesToOFD_9 {

    //1、定义 http 代理对象，请求访问转换服务（war 包版）
    static HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //static AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //static HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 图片转为 ofd 自定义宽度，添加元数据
    private void imageToOFD() {
        //2-1、定义集合，用于存储待转换的 image 文件
        List<File> inList =new ArrayList<File>();
        ///2-2、定义输出流，用于输出转换后的 ofd 文件
        OutputStream out = null;

        try {
            //2-3、将图片存储到集合中
            inList.add(new File("D:/b.jpg"));
            inList.add(new File("D:/a.png"));
            inList.add(new File("D:/c.bmp"));
            //2-4、赋值 OFD 文件输出流
            out=new FileOutputStream(new File("D:/convert_ofd/images_meta.ofd"));

            //2-5、定义 map 集合，用于存储元数据
            Map<String, String> metas =new HashMap<String, String>();
            //2-6、添加 OFD 标准元数据
            metas.put(Const.Meta.DOC_ID.value(), "111111111");
            metas.put(Const.Meta.AUTHOR.value(), "我是元数据作者");
            metas.put(Const.Meta.MOD_DATE.value(), "2021-01-10");
            metas.put(Const.Meta.TITLE.value(), "我是元数据标题");
            metas.put(Const.Meta.EXTEND_FILE.value(), "我是元数据 Extend_File");
            metas.put(Const.Meta.ABSTRACT.value(), "我是元数据摘要");
            metas.put(Const.Meta.SUBJECT.value(), "我是元数据主题");
            metas.put(Const.Meta.DOC_USAGE.value(), "我是元数据 DOCUSAGE");
            metas.put(Const.Meta.CREATOR.value(), "我是元数据创建者");
```

```
            //2-7、添加自定义元数据
            //单个自定义元数据的添加，自已定义 key 值---注意只能用一次
             metas.put(Const.Meta.CUSTOM_DATA.value(), "name=我是元数据自定义数据");
            //多个自定义元数据的添加，用","分开
            metas.put(Const.Meta.CUSTOM_DATAS.value(), "a=自定义数据 1,b=自定义 2");
            //2-8、添加关键字元数据
            metas.put(Const.Meta.KEYWORD.value(), "我是元数据关键字");
            metas.put(Const.Meta.KEYWORDS.value(), "我是元数据关键字 1,关键字 2");
            //2-9、定义宽度
            float width = 200;

            //2-10、调用方法，请求转换服务，将多个图片转为 ofd 文件并设置宽度，添加元数据
            ha.imagesToOFD(inList, out,metas,width);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-11、关闭 ha
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        ImagesToOFD_9  images2ofd = new ImagesToOFD_9();
        images2ofd.imageToOFD();
        System.out.println("转换结束......");
    }
}
```

1.2.9.4. 接口约束

支持转换图片源文件：jpg、jpeg、tif、tiff、bmp、png

元据据约束，请参考附件 1。

1.2.10. 多图片文件合并附加元数据生成 OFD，自定义 dpi

1.2.10.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 imagesToOFD 接口将多个图片转换为 OFD 版式文件并附加元数据。用于将多个图片文件合并成一个 OFD 版式文件，进行转换同时加入元数据。文档合并的前后顺序取决于在 List 集合里添加文件的顺序，转换完成的文件保存至设置的路径下。

1.2.10.2. 接口定义

```
void imagesToOFD(List<File> srcFiles,
                 OutputStream out,
                 Map<String, String> metas,
                 int  dpi);
```

参数说明详见下表：

多图片文件合并添加元数据生成 OFD，自定义 dpi 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	srcFile	List<File>类型	多个文件列表集合	List list=new ArrayList<File>();
				list.add(new File("D:/convert/ori/png_1.png"));
				list.add(new File("D:/convert/ori/jpg_1.jpg"));

2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/convert/tar/images_2.ofd"));
3	metas	Map 集合	附加的元数据	Map<String, String> metas=new HashMap<String, String>(); metas.put(Const.Meta.DOC_ID.value(), "0002");
4	dpi	int 整型	图片的 dpi	int dpi=200;

返回值说明详见下表：

多图片文件合并添加元数据生成 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.10.3. 接口示例

多图片文件合并附加元数据生成 OFD，自定义 dpi 接口示例代码如下：

```
//java 代码调用
//参考示例：com.springboot.single.ImagesToOFD_10
//功能说明：xxx.png，xxx.jpg，xxx.bmp 文件合并生成 OFD 并将 dpi 设置成 200，保存至设置的目录下
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class ImagesToOFD_10{

    //1、定义 http 代理对象，请求访问转换服务（war 包版）
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
     //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

     //2、定义方法实现  图片转为 ofd 并设置 dpi，添加元数据
    private  void imageToOFD() {
        //2-1、定义集合，用于存储待转换的 image 文件
        List<File> inList =new ArrayList<File>();
        //2-2、定义输出流，用于输出转换后的 ofd 文件
        OutputStream out = null;

        try {
             //2-3、将图片存储到集合中
             inList.add(new File("D:/1.jpg"));
             inList.add(new File("D:/2.png"));
             inList.add(new File("D:/3.bmp"));
             //2-4、赋值 OFD 文件输出流
             out=new FileOutputStream(new File("D:/convert_ofd/dimages_meta.ofd"));

             //2-5、定义 map 集合，用于存储元数据
             Map<String, String> metas =new HashMap<String, String>();
             //2-6、OFD 标准元数据的添加
             metas.put(Const.Meta.DOC_ID.value(), "111111111");
             metas.put(Const.Meta.AUTHOR.value(), "我是元数据作者");
             metas.put(Const.Meta.MOD_DATE.value(), "2021-01-10");
             metas.put(Const.Meta.TITLE.value(), "我是元数据标题");
             metas.put(Const.Meta.ABSTRACT.value(), "我是元数据摘要");
             metas.put(Const.Meta.SUBJECT.value(), "我是元数据主题 SUBJECT");
             metas.put(Const.Meta.DOC_USAGE.value(), "我是元数据 DOCUSAGE");
             metas.put(Const.Meta.CREATOR.value(), "我是元数据创建者");
```

```
            //2-7、添加自定义元数据
            //单个自定义元数据的添加，自已定义 key 值---注意只能用一次
            metas.put(Const.Meta.CUSTOM_DATA.value(), "name=我是元数据自定义数据");
            //多个自定义元数据的添加，用","分开
            metas.put(Const.Meta.CUSTOM_DATAS.value(), "a=自定义数据 1,b=自定义 2");
            //2-8、添加关键字元数据
            //单个关键字添加
            metas.put(Const.Meta.KEYWORD.value(), "我是元数据关键字");
            //多个关键字添加，用","分开
            metas.put(Const.Meta.KEYWORDS.value(), "我是元数据关键字,关键字 2");
            //2-9、定义 dpi
            int dpi = 200;

            //2-10、调用方法，请求转换服务，将多个图片转为 ofd 文件并设置 dpi，添加元数据
            ha.imagesToOFD(inList, out,metas,dpi);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-11、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        ImagesToOFD_10 images2ofd  = new ImagesToOFD_10();
        images2ofd.imageToOFD();
        System.out.println("转换结束......");
    }
}
```

1.2.10.4. 接口约束

支持转换图片源文件：jpg、jpeg、tif、tiff、bmp、png

元据据约束，请参考附件 1。

1.2.11. 单文件转换生成 PDF

1.2.11.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 OFDToPDF 接口将单个 OFD 版式文件转换为 PDF 文件。

1.2.11.2. 接口定义

如传入源文件为文件对象，服务会将其分配给转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。

```
void  OFDToPDF(File srcFile,
               OutputStream out);
```

参数说明详见下表：

单文件转换生成 PDF 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型。	本地文件路径	new File("D:\\convert\\ori\\cs.ofd")
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:\\convert\\tar\\doc_1.pdf"))

返回值说明详见下表：

单文件转换生成 PDF 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

如传入源文件为文件流(须更新 jar 包为 202201 以后版本)，服务会将其分配给指定转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。注：传入文件流的方式仅支持将 OFD 转为 PDF。

```
void OFDToPDF(InputStream in,
              OutputStream out);
```

参数说明详见下表：

单文件转换生成 PDF 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	in	InputStream 输入流	OFD 文件输入流	new FileInputStream(new File("D:\\test1.ofd"))

2	out	OutputStream 输出流	文件输出流	new FileOutputStream(new File("D:\\doc2pdf.pdf"))

返回值说明详见下表：

单文件转换生成 PDF 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.11.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.single.OfficeToPDF_9
//功能说明：将 xxx.doc/xxx.docx/xxx.xls...源文件转换成 pdf 版式文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class OfficeToPDF_9 {

    //1、定义 http 代理对象，请求访问转换服务（war 包版）
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


   //2、定义方法实现 单文件转换为 pdf 文件
   private void off2PDF() {

           try {
                 //2-1、传参文件对象转换输出 OFD 文件
                //1)定义待转换的原文件
                 File srcFile = new File("D:\\体检通知.docx");
                 //2)、定义转换后的 pdf 文件输出流
                 OutputStream outFile = new FileOutputStream(new
File("D:\\off2PDF_docx7.pdf"));
                 //3)、调用方法，执行将单个文件转为 pdf 文件
                 ha.OFDToPDF(srcFile, outFile);


                 //2-2、传参文件流转换输出 OFD 文件
                 //1)、定义待转换的原文件,必须是 OFD 文件流
                 InputStream in = new FileInputStream(new File("D:\\cs.ofd"));
                 //2)、定义转换后的 pdf 文件输出流
                 OutputStream outFile2 = new FileOutputStream(new File("D:\\cs.pdf"));
                 //3)、调用方法，执行将单个文件转为 pdf 文件
```

```
                ha.OFDToPDF(in, outFile2);

            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }finally {
                try {
                    //2-4、关闭资源
                    ha.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
    }

    //3、方法调用
    public static void main(String[] args) {
            OfficeToPDF_9  office2pdf = new OfficeToPDF_9();
            office2pdf.off2PDF();
             System.out.println("转换结束......");
    }
}
```

1.2.11.4. 接口约束

本接口支持的非 OFD 源文件请参考 1.2.1.4 章节，同时也支持将 OFD 文档转换成 PDF。

1.2.12. 单文件附加元数据生成 PDF

1.2.12.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 OFDToPDF 接口将单个办公文件转换为 OFD 版式文件并附加元数据。服务会根据传入的文件流自动判断文件格式，并将其分配给转换节点进行转换同时加入元数据，转换完成的文件保存至设置的路径下。

1.2.12.2. 接口定义

如传入源文件为文件对象，服务会将其分配给转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。

```
void  OFDToPDF(File srcFile,
               OutputStream out,
               Map<String, String> metas);
```

参数说明详见下表：

单文件附加元数据生成 PDF 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	本地文件路径	new File("D:/convert/ori/doc_1.doc");
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/convert/tar/offices_2.pdf"));
3	metas	Map 集合	附加的元数据	Map<String, String> metas=new HashMap<String, String>(); metas.put(Const.Meta.DOC_ID.value(), "0002");

返回值说明详见下表：

单文件附加元数据生成 PDF 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

如传入源文件为文件流(须更新 jar 包为 202201 以后版本)，服务会将其分配给指定转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。注：传入文件流的方式仅支持将 OFD 转为 PDF。

```
void OFDToPDF(InputStream in,
               OutputStream out,
               Map<String, String> metas);
```

参数说明详见下表：

单文件转换生成 PDF 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	in	InputStream 输入流	OFD 文件输入流	new FileInputStream(new File("D:\\test1.ofd"))
2	out	OutputStream 输出流	文件输出流	new FileOutputStream(new File("D:\\ofd2pdf.pdf"))
3	metas	Map 集合	附加的元数据	Map<String, String> metas=new HashMap<String, String>(); metas.put(Const.Meta.DOC_ID.value(), "0002");

返回值说明详见下表：

单文件转换生成 PDF 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.12.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.single.OfficeToPDF_10
//功能说明：将 doc_1.doc 文件转换成 pdf 版式文件，并添加文档 ID、作者等元数据
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class OfficeToPDF_10 {

    //1、定义 http 代理对象，请求访问转换服务（war 包版）
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
     //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
     //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


    //2、定义方法实现 单文件转换为 pdf 文件并添加元数据
    private void off2PDF_meta() {

        try {
             //1、定义 map 集合，用于存储元数据
             Map<String, String> metas =new HashMap<String, String>();
             //1)、OFD 标准元数据的添加
             metas.put(Const.Meta.DOC_ID.value(), "111111111");
             metas.put(Const.Meta.AUTHOR.value(), "我是元数据作者");
             metas.put(Const.Meta.MOD_DATE.value(), "2021-01-10");
             metas.put(Const.Meta.TITLE.value(), "我是元数据标题");
             metas.put(Const.Meta.ABSTRACT.value(), "我是元数据摘要");
             metas.put(Const.Meta.SUBJECT.value(), "我是元数据主题");
             metas.put(Const.Meta.DOC_USAGE.value(), "我是元数据 DOCUSAGE");
             metas.put(Const.Meta.CREATOR.value(), "我是元数据创建者");
             // 单个自定义元数据的添加，自已定义 key 值---注意只能用一次
             metas.put(Const.Meta.CUSTOM_DATA.value(), "name=自定义数据");
             // 多个自定义元数据的添加
             metas.put(Const.Meta.CUSTOM_DATAS.value(), "a=自定义数据 1,b=自定义 2");
             // 关键字添加
             //单个关键字添加
             metas.put(Const.Meta.KEYWORD.value(), "我是元数据关键字");
             //多个关键字添加
```

```
            metas.put(Const.Meta.KEYWORDS.value(), "我是元数据关键字,关键字 2");


            //2-1、传参文件对象转换输出 OFD 文件
            //1)定义待转换的原文件
            File srcFile = new File("D:/cs.doc");
            //2)、定义转换后的 pdf 文件输出流
            OutputStream out = new FileOutputStream(new File("D:/off2PDF_doc.pdf"));
          //3)、调用方法，执行将单个文件转为 pdf 文件,并添加元数据
            ha.OFDToPDF(srcFile, out, metas);


            //2-2、传参文件流转换输出 OFD 文件
            //1)、定义待转换的原文件,必须是 OFD 文件流
            InputStream in = new FileInputStream(new File("D:\\cs.ofd"));
            //2)、定义转换后的 pdf 文件输出流
            OutputStream outFile2 = new FileOutputStream(new File("D:\\cs.pdf"));
            //3)、调用方法，执行将单个文件转为 pdf 文件,并添加元数据
            ha.OFDToPDF(in,outFile2,metas);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
           try {
            //2-6、关闭资源
                ha.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        OfficeToPDF_10 office2pdf = new OfficeToPDF_10();
        office2pdf.off2PDF_meta();
        System.out.println("......");
    }

}
```

1.2.12.4. 接口约束

本接口支持的非 OFD 源文件请参考 1.2.1.4 章节，同时也支持将 OFD 文档转换成 PDF。

元据据约束，请参考附件 1。

1.2.13. 多文件合并生成 PDF

1.2.13.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 OFDToPDF 接口将多个文件（offices 文件或 OFD 或 PDF）合并转换为一个 PDF 版式文件。文档合并的前后顺序取决于在 List 集合里添加文件的顺序。

1.2.13.2. 接口定义

```
void  OFDToPDF(List<File> srcFiles,
               OutputStream out);
```

参数说明详见下表：

多文件合并生成 PDF 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	srcFile	List 集合	文件集合	List list=new ArrayList<File>();
				list.add(new File("D:/convert/ori/doc_1.ofd"));
				list.add(new File("D:\\2.pdf"));

2	out	OutputStream 输出流	文件输出流	new FileOutputStream("D:/convert/tar/offices_1.pdf"));

返回值说明详见下表：

多文件合并生成 PDF 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.13.3. 接口示例

```
//java 代码调用
```

```
//参考示例：com.springboot.single.OfficesToPDF_11
//功能说明：xxx.doc、xxx.ofd、xxx.docx 等文件合并生成 PDF
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class OfficesToPDF_11 {
    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    static HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //static AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //static HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 单文件转换为 pdf 文件并添加元数据
    private void offs2Pdf() {
        //2-1、定义集合，存储待转换的原文件
        List<File> fileList =new ArrayList<File>();
        //2-2、定义转换后的 pdf 文件输出流
        OutputStream out = null;
        try {
            //2-3、添加原文件到集合
            fileList.add(new File("D:/测试文件.docx"));
            fileList.add(new File("D:/公文附件 002.doc"));
            fileList.add(new File("D:/测试.ofd"));
            //2-4、赋值转换后的 pdf 文件输出流
            out=new FileOutputStream(new File("D:/convert_ofd/offices_pdf.pdf"));
            //2-5、调用方法，执行将多个文件合并转为 pdf 文件
            ha.OFDToPDF(fileList, out);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }  finally {
            try {
                //2-6、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
   //3、方法调用
    public static void main(String[] args) {
        OfficesToPDF_11  offices2pdf = new OfficesToPDF_11();
        offices2pdf.offs2Pdf();
        System.out.println("转换结束......");
   }
```

```
}
```

1.2.13.4. 接口约束

本接口支持的非 OFD 源文件请参考 1.2.1.4 章节，同时也支持将 OFD 文档转换成 PDF。

1.2.14. 多文件合并添加元数据生成 PDF

1.2.14.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 officesToOFD 接口将多个办公文件转换为 PDF 版式文件并附加元数据。用于将多个源文件合并成一个 PDF 版式文件，文档合并的前后顺序取决于在 List 集合里添加文件的顺序。进行转换同时加入元数据。

1.2.14.2. 接口定义

```
void OFDToPDF(InputStream srcFiles,
              OutputStream out,
               Map<String, String> metas);
```

参数说明详见下表：

多文件合并添加元数据生成 PDF 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	srcFile	List<File>类型	多个文件列表集合	List list=new ArrayList<File>();
				list.add(new File("D:\\convert\\ori\\doc_1.doc"));
				list.add(new File("D:\\convert\\ori\\ofd_1.ofd"));

2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:\\convert\\tar\\offices_1.pdf"));
3	metas	Map 集合	附加的元数据	Map<String, String> metas=new HashMap<String, String>() metas.put(Const.Meta.DOC_ID.value(), "0002");

返回值说明详见下表：

多文件合并添加元数据生成 PDF 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.14.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.single.OfficesToPDF_12
//功能说明: xxx.docx,xxx.doc,xxx.xls 等文件合并生成 PDF，并为其添加 DOCID、作者等元数据
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class OfficesToPDF_12 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    static HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //static AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //static HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 多文件合并转换为 pdf 文件并添加元数据
    private static void offsToPDF() {
        //2-1、定义集合，存储待转换的原文件
        List<File> fileList =new ArrayList<File>();
        //2-2、定义转换后的 pdf 文件输出流
        OutputStream out = null;
        try {
            //2-3、添加文件集合
            fileList.add(new File("D:/测试文件.docx"));
            fileList.add(new File("D:/公文附件 002.doc"));
            fileList.add(new File("D:/a.pdf"));
            fileList.add(new File("D:/测试文件.xls"));
            //2-4、定义 map 集合，用于存储元数据
            Map<String, String> metas =new HashMap<String, String>();
            //2-5、OFD 标准元数据的添加
           metas.put(Const.Meta.DOC_ID.value(), "111111111");
            metas.put(Const.Meta.AUTHOR.value(), "我是元数据作者");
            metas.put(Const.Meta.MOD_DATE.value(), "2021-01-10");//注意参数格式
            metas.put(Const.Meta.TITLE.value(), "我是元数据标题");
            metas.put(Const.Meta.ABSTRACT.value(), "我是元数据摘要");
            metas.put(Const.Meta.SUBJECT.value(), "我是元数据主题");
            metas.put(Const.Meta.DOC_USAGE.value(), "我是元数据 DOCUSAGE");
            metas.put(Const.Meta.CREATOR.value(), "我是元数据创建者");
            //2-6、添加单个自定义元数据---只能添加一次
            metas.put(Const.Meta.CUSTOM_DATA.value(), "name=元数据自定义数据");
            //2-7、添加多个自定义元数据
```

```
            metas.put(Const.Meta.CUSTOM_DATAS.value(), "a=自定义元数据 1,b=自定义 2");
            //2-8、添加关键字
            metas.put(Const.Meta.KEYWORD.value(), "我是元数据关键字");
            //2-9、添加多个关键字
            metas.put(Const.Meta.KEYWORDS.value(), "我是元数据关键字 S");
           //2-10、赋值 pdf 输出文件流
            out=new FileOutputStream(new File("D:/convert_ofd/offices_pdf.pdf"));
            //2-11、调用方法，执行将多个文件转为 pdf 文件并添加元数据
            ha.OFDToPDF(fileList, out,metas);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }  finally {
            try {
                //2-12、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        OfficesToPDF_12 offices2pdf = new OfficesToPDF_12();
        offices2pdf.offsToPDF();
        System.out.println("转换结束......");

    }
}
```

1.2.14.4. 接口约束

本接口支持的非 OFD 源文件请参考 1.2.1.4 章节，同时也支持将 OFD 文档转换成 PDF。

元据据约束，请参考附件 1。

1.2.15. 网页转换生成 OFD

1.2.15.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 htmlToOFD 接口将 html 文件转换为 OFD 文件。完成的文件保存至设置的路径下。

1.2.15.2. 接口定义

```
void htmlToOFD(String uri,
               OutputStream out);
```

参数说明详见下表：

网页转换成 OFD 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	uri	字符串类型（1-255）	待转换资源标识符	“http://www.baidu.com”，“D://a.html”，URI.create(“file:///D://a.html”).toString()
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/convert/tar/html_1.ofd"));

返回值说明详见下表：

网页转换成 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.15.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.single.HtmlToOFD_13
//功能说明: 将 wwww.baidu.com 的网页转换成 OFD 文件。
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class HtmlToOFD_13 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");
```

```
    //2、定义方法实现 html 转换生成 OFD
    private void html2Ofd() {

        //2-1、定义待转换的 html 资源
        String uri="http://www.baidu.com";
        //String uri2 = "D://a.html";
        //String uri3 = URI.create("file:///D:/a.html").toString();

        //2-2、定义转换后 ofd 文件输出流
        OutputStream out = null;

        try {
                //2-3、赋值转换后 ofd 文件输出流
                out=new FileOutputStream(new File("D:/convert_ofd/html_uri3.ofd"));
                //2-4、调用接口请求服务，将 html 文件转为 ofd 文件
                ha.htmlToOFD(uri, out);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-5、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        HtmlToOFD_13 htmlToOfd = new HtmlToOFD_13();
        htmlToOfd.html2Ofd();
        System.out.println("转换结束......");

    }
}
```

1.2.15.4. 接口约束

无。

1.2.16. 多网页合并转换生成 OFD

1.2.16.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 htmlToOFD 接口将多个 html 文件转换为 OFD 文件。用于将多个 html 文件转换为 OFD 版式文件。文档合并的前后顺序取决于在 List 集合里添加文件的顺序。转换完成的文件保存至设置的路径下。

1.2.16.2. 接口定义

```
void htmlToOFD(List<String> uris,
               OutputStream out);
```

参数说明详见下表：

多网页转换成 OFD 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	uris	List<String>	待转换资源集合	List uris=new ArrayList<File>(); uris.add(“http://www.baidu.com”); uris.add(“http://www.taobao.com”);
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/convert/tar/htmls_1.ofd"));

返回值说明详见下表：

多网页转换成 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.16.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.single.HtmlsToOFD_14
//功能说明: 将 wwww.baidu.comt 和 www.taobao.com 及本地网页合并转换成 OFD 文件。
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class HtmlsToOFD_14 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
```

```
    //1、定义代理对象，请求访问转换服务(Springboot 版)
     //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
     //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 html 多文件合并转换生成 OFD
    private void htmls2Ofd() {

        //2-1、定义集合，存储待转换的 html 资源
        List<String> uris=new  ArrayList<String>();
        uris.add("http://www.taobao.com");
        uris.add("http://www.baidu.com");
        uris.add(URI.create("file:///D://测试文件.html").toString());
        uris.add("D:/测试文件.html");
        //2-2、定义转换后 ofd 文件输出流
        OutputStream out = null;

        try {
            //2-3、赋值转换后 ofd 文件输出流
            out=new FileOutputStream(new File("D:/convert_ofd/htmls_236.ofd"));
            //2-4、调用接口请求服务，将多个 html 文件合并转为 ofd 文件
            ha.htmlToOFD(uris, out);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                //2-5、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        HtmlsToOFD_14 htmlsToOfd = new HtmlsToOFD_14();
        htmlsToOfd.htmls2Ofd();
        System.out.println("结束......");
    }
}
```

1.2.16.4. 接口约束

无。

1.2.17. 单 OFD 文件生成图片，自定义 width

1.2.17.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 OFDToImge 接口将单个 OFD 版式文件转换为图片文件(bmp、png、jpg、tiff、tif 格式)，转换完成的文件保存至设置的路径下。

1.2.17.2. 接口定义

如传入源文件为文件对象，服务会将其分配给转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。

```
void OFDToImge(File srcFile,
               OutputStream out,
               float width,
               boolean m);
```

参数说明详见下表：

单个 OFD 文件生成图片接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	本地文件路径	new File("D:/ofd-muti/1.ofd")
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/ofd_img.tiff"))
				new FileOutputStream(new File("D:/ofd_img.zip"))
3	width	浮点型 单位：像素	图片的宽度（像素）如果想用默认大小。填-1 即可	-1（原宽度）
				1000f (自定义宽度)
4	m	布尔型	输出格式。true 输出格式为 zip ； false 输出格式为 tiff	false :转换为 tiff 图片格式，输出格式为 tiff
				true :转换为非 tiff 图片格式。输出格式为 zip

返回值说明详见下表：

单文件转换生成图片接口返回值说明表

序号	返回类型	返回值含义
1	无	无

如传入源文件为文件流(须更新 jar 包为 202201 以后版本)，服务会将其分配给指定转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。

```
void OFDToImge(InputStream srcFile,
               OutputStream out,
               float width,
               boolean m);
```

参数说明详见下表：

单个 OFD 文件生成图片接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	InputStream 输入流	文件输入流	new FileInputStream("D:\\test1.ofd")
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/ofd_img.tiff"))
				new FileOutputStream(new File("D:/ofd_img.zip"))
3	width	浮点型 单位：像素	图片的宽度（像素）如果想用默认大小。填-1 即可	-1f（原宽度）
				1000f (自定义宽度)
4	m	布尔型	输出格式。true 输出格式为 zip ； false 输出格式为 tiff	false :转换为 tiff 图片格式，输出格式为 tiff
				true :转换为非 tiff 图片格式。输出格式为 zip

返回值说明详见下表：

单文件转换生成图片接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.17.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.single.OFDToImage_15
//功能说明: 将 xxx.ofd 文件转换成多页 tiff 文件或(bmp)zip 文件，并自定义图片宽度
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class OFDToImage_15 {
```

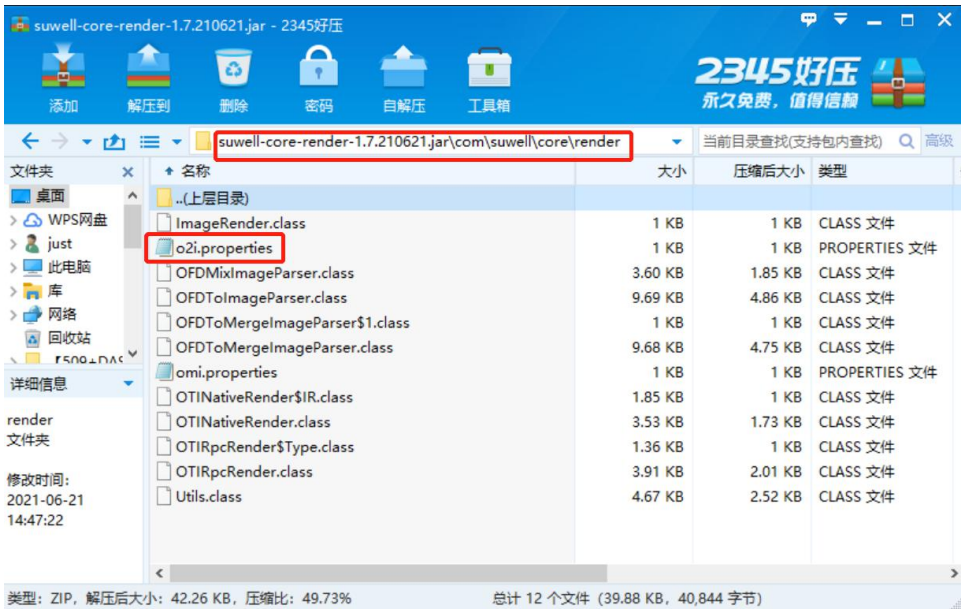
```
    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 单个 OFD 文件转换为图片并设置图片宽度
    private void ofd2Image() {

        try {

            //2-1、传参文件对象将 OFD 文件转换为图片
            //1)定义待转换的 OFD 文件对象
            File srcFile = new File("D:/response.ofd");
            //2)、定义转换后的图片输出流：非 tiff 格式图片文件
            OutputStream out=new FileOutputStream(new File("D:/ofd_img.zip"));
            //3)、定义生成后图片的宽度
            float imgWidth = 200;
            //4)、生成宽度为 200 的多页 jpg 压缩文件
            //ha.OFDToImge(in1, out, imgWidth,true);
            //5)、调用接口实现 OFD 文件转换生成原宽度的多页 jpg 文件,以压缩包格式文件输出
            ha.OFDToImge(srcFile, out, -1f, true);

            //2-2、传参文件流转换输出 OFD 文件
            //1)定义待转换的 OFD 文件流
            InputStream in = new FileInputStream(new File("D:/download.ofd"));
            //2)、定义转换后的图片输出流：非 tiff 格式图片文件压缩包
            OutputStream out2=new FileOutputStream(new File("D:/ofd_img2.zip"));
            //3)、调用接口实现 OFD 文件转为图片文件（非 tiff 格式图片以压缩包格式输出）
            ha.OFDToImge(in,out2, -1f,true);



            //定义转换后的图片输出流:tiff 格式图片文件
            //OutputStream out2=new FileOutputStream(new
File("D:/convert/tar/ofd_tiff.tiff"));
            //生成宽度为 1000 的 tiff 多页文件,注意 out 输出文件类型为.tiff
            //ha.OFDToImge(in, out2, imgWidth, false);
            //生成原宽度的多页 tiff 文件,注意 out 输出文件类型为.tiff
            //ha.OFDToImge(in, out2, -1f, false);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
```

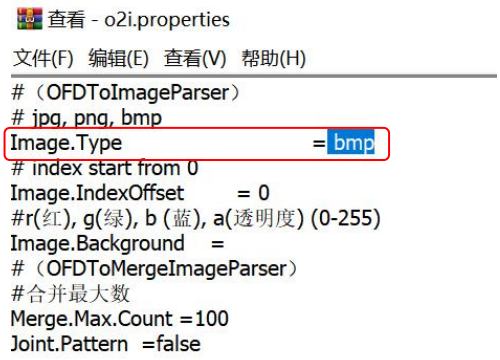
```
            try {
                //2-5、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        OFDToImage_15 ofdToImg = new OFDToImage_15();
       ofdToImg.ofd2Image();
       System.out.println("转换结束......");

    }
}
```

1.2.17.4. 接口约束

默认生成的格式是 bmp 格式图片，如需要其他格式图片，在节点路径下：build75\lib\suwell\suwell-core-render-1.7.xxxxxx.jar 修改 jar 包中的 com\suwell\core\render\o2i.properties 文件的配置项 Image.Type = bmp（可改为：jpg, png, bmp）

具体见下图：





1.2.18. 单 OFD 文件生成图片，自定义 dpi

1.2.18.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 OFDToImge 接口将单个 OFD 版式文件转换为图片文件，转换完成的文件保存至设置的路径下。

1.2.18.2. 接口定义

如传入源文件为文件对象，服务会将其分配给转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。

```
void OFDToImge(File srcFile,
               OutputStream out,
               int dpi,
               boolean m);
```

参数说明详见下表：

单文 OFD 文件生成图片接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	本地文件路径	new File("D:/ofd-muti/1.ofd")
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/ofd_img.tiff"))
				new FileOutputStream(new File("D:/ofd_img.zip"))
3	dpi	int 整型	图片的 dpi	-1（默认 96dpi）
				int dpi=600; （自定义 dpi）

4	m	布尔型	输出格式。true 输出格式为 zip ； false 输出格式为 tiff	false :转换为 tiff 图片格式，输出格式为 tiff
				true :转换为非 tiff 图片格式。输出格式为 zip

返回值说明详见下表：

单文件转换生成图片接口返回值说明表

序号	返回类型	返回值含义
1	无	无

如传入源文件为文件流(须更新 jar 包为 202201 以后版本)，服务会将其分配给指定转换节点进行转换，转换完成的文件以流的形式保存至设置的路径下。

```
void OFDToImge(InputStream srcFile,
               OutputStream out,
               int dpi,
               boolean m);
```

参数说明详见下表：

单个 OFD 文件生成图片接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	InputStream 输入流	文件输入流	new FileInputStream("D:\\test1.ofd")
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/ofd_img.tiff"))
				new FileOutputStream(new File("D:/ofd_img.zip"))
3	dpi	int 整型	图片的 dpi	-1（默认 96dpi）
				int dpi=600; （自定义 dpi）
4	m	布尔型	输出格式。true 输出格式为 zip ； false 输出格式为 tiff	false :转换为 tiff 图片格式，输出格式为 tiff
				true :转换为非 tiff 图片格式。输出格式为 zip

返回值说明详见下表：

单文件转换生成图片接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.18.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.single.OFDToImage_16
//功能说明: 将 xxx.ofd 文件转换成多页 tiff 文件或(bmp)zip 文件, 并自定义 dpi
//HTTPAgent/AtomAgent 为服务代理类, 通过此类连接及调用转换服务

public class OFDToImage_16 {

    //1、定义 http 代理对象, 请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象, 请求访问转换服务(Springboot 版)
   // AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    // HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 单个 OFD 文件转换为图片并设置图片 dpi
    private void ofd2Image_dpi() {

        try {
            //2-1、传参文件对象将 OFD 文件转换为图片
            //1)定义待转换的 OFD 文件对象
            File srcFile = new File("D:/response.ofd");
            //2)、定义转换后的图片输出流: 非 tiff 格式图片文件
            OutputStream out=new FileOutputStream(new File("D:/ofd_img.zip"));
            //3)、定义生成后图片的 dpi
            int imgDpi = 200;
            //4)、生成 dpi 为 200 的多页 bmp 压缩文件
            //ha.OFDToImge(in,out, imgDpi,true);
            //5)、调用接口实现 OFD 文件转换生成默认 dpi=96 的多页 bmp 压缩文件
            ha.OFDToImge(srcFile, out, -1, true);


            //2-2、传参文件流转换输出 OFD 文件
            //1)定义待转换的 OFD 文件流
            InputStream in = new FileInputStream(new File("D:/文档 4.ofd"));
            //2)、定义转换后的图片输出流: 非 tiff 格式图片文件压缩包
            OutputStream out2=new FileOutputStream(new File("D:/ofd_img2.zip"));
            //3)、调用接口实现 OFD 文件转换生成 dpi 为 200 的多页 bmp 压缩文件
          ha.OFDToImge(in,out2, imgDpi,true);


            //定义转换后的图片输出流:tiff 格式图片文件
            //OutputStream out2=new FileOutputStream(new
File("D:/convert/ofd_tiff.tiff"));
```

```
            //生成 dpi 为 200 的 tiff 多页文件,注意 out 输出文件类型为.tiff
            //ha.OFDToImge(in, out2, imgDpi, false);
            //生成默认 dpi=96 的多页 tiff 文件,注意 out 输出文件类型为.tiff
            //ha.OFDToImge(in, out2, -1, false);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        OFDToImage_16 ofdToImg = new OFDToImage_16();
        ofdToImg .ofd2Image_dpi();
        System.out.println("转换结束......");

    }
}
```

1.2.18.4. 接口约束

同 **1.2.15.4** 接口约束

1.2.19. 多 OFD 文件生成图片，自定义 width

1.2.19.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 OFDToImge 接口将多个 OFD 版式文件转换为图片文件，转换完成的文件保存至设置的路径下。

1.2.19.2. 接口定义

```
void OFDToImge(List<File> srcFiles,
               OutputStream out,
               float w,
               boolean m);
```

参数说明详见下表：

多个 OFD 文件生成图片接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	srcFiles	List<File> 类型	待转换源文件集合	List<File> files=new ArrayList<File>();
				File in1 = new File("D:/convert/ori/1.ofd");
				files.add(in1);
				File in2 = new File("D:/convert/ori/2.ofd");
				files.add(in2);

2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/ofds_img.tiff"))
				new FileOutputStream(new File("D:/ofds_img.zip"))
3	w	浮点型 单位：像素	图片的宽度（像素）如果想用默认大小。填-1 即可	-1（原宽度）
				1000f (自定义宽度)
4	m	布尔型	输出格式。true 输出格式为 zip ；false 输出格式为 tiff	false :转换为 tiff 图片格式，输出格式为 tiff
				true :转换为非 tiff 图片格式。输出格式为 zip

返回值说明详见下表：

多个 OFD 文件生成图片接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.19.3. 接口示例

```
//java 代码调用
//参考示例：com.http.single.OFDsToImage_17
//功能说明：将多个 OFD 文件（xx1.ofd、xx2.ofd 等文件）合并转换成一个多页 tiff 文件或 zip
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class OFDsToImage_17 {
```

```
    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


   //2、定义方法实现 多个 OFD 文件转换为图片并设置图片宽度
    private void ofdsToImage() {

        try {
            //2-1、定义集合，存储多个待转换的 OFD 文件
            List<File> files=new ArrayList<File>();

            //2-2、集合添加 OFD 文件
            files.add(new File("D:/33.ofd"));
            files.add(new File("D:/12311.ofd"));

            //2-3、定义文件转换后输出的文件流
            OutputStream out = new FileOutputStream(new
File("D:/convert_ofd/ofds_img.zip"));
            //2-4、生成宽度为 1000 的,bmp 多页压缩文件，注意 out 输出文件类型为.zip
            //ha.OFDToImge(files, out, 1000f,true);
            //2-4、生成原宽度的多页 bmp 压缩文件，注意 out 输出文件类型为.zip
            ha.OFDToImge(files, out, -1, true);



            //OutputStream out2 = new FileOutputStream(new
File("D:/convert_ofd/ofds_tiff.tiff"));
            //生成宽度为 1000 的多页 tiff 文件，注意 out 输出文件类型为.tiff
            //ha.OFDToImge(files, out2, 1000f, false);
            //生成原宽度的多页 tiff 文件，注意 out 输出文件类型为.tiff
            //ha.OFDToImge(files, out2, -1, false);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                //2-5、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
```

```
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        OFDsToImage_17 ofdsToImage = new OFDsToImage_17();
       ofdsToImage.ofdsToImage();
       System.out.println("转换结束......");
   }
}
```

1.2.19.4. 接口约束

同 1.2.15.4 接口约束。

1.2.20. 多 OFD 文件生成图片，自定义 dpi

1.2.20.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 OFDToImge 接口将多个 OFD 版式文件转换为图片文件，转换完成的文件保存至设置的路径下。

1.2.20.2. 接口定义

```
void OFDToImge(List<File> srcFiles,
               OutputStream out,
               int dpi,
               boolean m);
```

参数说明详见下表：

多个 OFD 文件生成图片自定义 dpi 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	srcFiles	List<File>类型	待转换源文件集合	List<File> files=new ArrayList<File>();
				File in1 = new File("D:/convert/ori/1.ofd");
				files.add(in1);
				File in2 = new File("D:/convert/ori/2.ofd");
				files.add(in2);

2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D:/ofds_img.tiff"))
				new FileOutputStream(new File("D:/ofds_img.zip"))
3	dpi	int 整型	图片的 dpi	-1（默认 96dpi）
				int dpi=600; （自定义 dpi）
4	m	布尔型	输出格式。true 输出格式为 zip ；false 输出格式为 tiff	false :转换为 tiff 图片格式，输出格式为 tiff
				true :转换为非 tiff 图片格式。输出格式为 zip

返回值说明详见下表：

多个 OFD 文件生成图片自定义 dpi 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.2.20.3. 接口示例

```
//java 代码调用
//参考示例：com.http.single.OFDsToImage_18
//功能说明：将多个 OFD 文件（xx1.ofd、xx2.ofd 等）转换成多页 tiff 文件或 zip
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class OFDsToImage_18 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 将多个 OFD 文件合并转换为图片并自定义图片 dpi
    private void ofds2Image() {

        try {
```

```
            //2-1、定义集合，存储待转换的 OFD 文件
            List<File> files=new ArrayList<File>();
            //2-2、集合添加待转换的 OFD 文件
            files.add(new File("D:/convert/ori/1.ofd"));
            files.add(new File("D:/convert/ori/2.ofd"));

            //2-3、定义转换后的图片输出流：非 tiff 格式图片文件（bmp,jpg,png）
            OutputStream out =new FileOutputStream(new
File("D:/convert/tar/ofds_img.zip"));
            //2-4_1、生成 dpi 为 200 的,bmp 多页压缩文件
            //ha.OFDToImge(files, out, 200,true);
            //2-4_2、生成默认 dpi=96 的 bmp 多页压缩文件
            ha.OFDToImge(files, out, -1, true);



            //2-3、定义转换后的图片输出流：tiff 格式图片文件
            //OutputStream out =new FileOutputStream(new
File("D:/convert/tar/ofds_tiff.tiff"));
            //2-4_1、生成 dpi 为多页 tiff 文件,注意 out 输出文件类型为.tiff
            //ha.OFDToImge(files, out, 200, false);
            //2-4_2、生成默认 dpi=96 的多页 tiff 文件,注意 out 输出文件类型为.tiff
            //ha.OFDToImge(files, out, -1, false);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-5、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    //3、方法调用
    public static void main(String[] args) {
        OFDsToImage_18 ofdsToImage = new OFDsToImage_18();
       ofdsToImage.ofds2Image();
        System.out.println("转换结束......");

    }

}
```

1.2.20.4. 接口约束

同 **1.2.15.4** 接口约束。

1.3. Agent 实现文件加工转换

1.3.1. 添加附件

1.3.1.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 addAttachment 接口将源文件转换为 OFD 版式文件并加入附件。转换完成的文件保存至设置的路径下。

1.3.1.2. 接口定义

```
void addAttachment(File srcFile,
                   OutputStream out,
                   Group<String,String,InputStream>... attachs)
```

参数说明详见下表：

添加附件接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	待转换源文件	new File("D:\\convert\\tar\\doc_1.ofd");
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream("D:\\convert\\tar\\attach.ofd");
3	attachs	Group 集合	附件集合	new Group<String, String, InputStream>("jpg_1", "jpg", new FileInputStream(new File("D:\\convert\\ori\\jpg_1.jpg")));

返回值说明详见下表：

添加附件接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.1.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.handle.AddAttachment_19
//功能说明：在文件中添加附件转为 xxx.ofd
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class AddAttachment_19 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 OFD 文件中添加附件
    private void addAttachment() {
        try {
            //2-1、定义待添加附件转换的原文件 xxx.doc,xxx.pdf,xxx.jpg,xxx.xls,xxx.ofd
            File srcFile = new File("D:\\cs.wps");
            //2-2、定义添加附件后的 OFD 文件输出流
            OutputStream out =new FileOutputStream(new
File("D:/convert/addAttr_19.ofd"));
            //2-3、定义 Group，添加附件
            Group<String, String, InputStream> attachment1 =new Group<String, String,
InputStream>("xml_1", "xml", new FileInputStream(new File("D://cs.xml")));
           Group<String, String, InputStream> attachment2 =new Group<String, String,
InputStream>("docx_1", "docx", new FileInputStream(new File("D://cs.docx")));
           Group<String, String, InputStream> attachment3 =new Group<String, String,
InputStream>("wps_1", "wps", new FileInputStream(new File("D://cs.wps")));
           Group<String, String, InputStream> attachment4 =new Group<String, String,
InputStream>("txt_1", "txt", new FileInputStream(new File("D://cs.txt")));
           Group<String, String, InputStream> attachment5 =new Group<String, String,
InputStream>("doc_1", "doc", new FileInputStream(new File("D://cs.doc")));
           Group<String, String, InputStream> attachment6 =new Group<String, String,
InputStream>("xls_1", "xls", new FileInputStream(new File("D://cs.xls")));
           Group<String, String, InputStream> attachment7 =new Group<String, String,
InputStream>("html_1", "html", new FileInputStream(new File("D://cs.html")));
           Group<String, String, InputStream> attachment8 =new Group<String, String,
InputStream>("pdf_1", "pdf", new FileInputStream(new File("D://cs.pdf")));
           Group<String, String, InputStream> attachment9 =new Group<String, String,
InputStream>("ofd_1", "ofd", new FileInputStream(new File("D://cs.ofd")));
           Group<String, String, InputStream> attachment10 =new Group<String, String,
InputStream>("png_1", "png", new FileInputStream(new File("D://cs.png")));
           Group<String, String, InputStream> attachment11 =new Group<String, String,
```

```
InputStream>("jpg_1", "jpg", new FileInputStream(new File("D://cs.jpg")));
            Group<String, String, InputStream> attachment12 =new Group<String, String,
InputStream>("tiff_1", "tiff", new FileInputStream(new File("D://cs.tiff")));

            //2-4、调用方法，给文件添加附件并转为 OFD 文件
            ha.addAttachment(srcFile,out,
attachment1,attachment2,attachment3,attachment4,attachment5,attachment6,attachment7,
attachment8,attachment9,attachment10,attachment11,attachment12);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-5、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        AddAttachment_19 addAttr = new AddAttachment_19();
        addAttr.addAttachment();
        System.out.println("转换结束----------------");

    }
}
```

1.3.1.4. 接口约束

无。

1.3.2. 添加图片水印

1.3.2.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 addImageMark 接口在

OFD 版式文件加入图片水印。此接口将源文件转换成 OFD 版式文件并将指定图片水印添加至指定位置。一般用于 OFD 版式文件的离线溯源。转换完成的文件保存至设置的路径下。

1.3.2.2. 接口定义

```
void addImageMark(File srcFile,
                  OutputStream out,
                  InputStream imge,
                  String imageType,
                  MarkPosition mk,
                  boolean printable,
                  boolean visible);
```

参数说明详见下表：

添加图片水印接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	待转换源文件	new File("D:\\cs.wps");
2	out	OutputStream 输出流	转换后 OFD 文件本地输出路径	new FileOutputStream("D:\\convert\\tar\\imageMark.ofd")
3	Imge	InputStream 输入流	要添加的图片水印	new FileInputStream(new File("D:\\convert\\ori\\jpg_1.jpg"));
4	imageType	String 字符串类型	图片格式	"jpg"
				"png"
5	mk	MarkPosition 类型，参见 1.6.2	水印位置，见 1.6.2 章节	MarkPosition mp=new MarkPosition(10,20,200,300,MarkPosition.INDEX_ALL);
6	Printable	boolean 布尔型	是否打印	true:打印
				false：不打印
7	visible	Boolean 布尔型	是否显示	true:显示
				false:不显示

返回值说明详见下表：

添加图片水印接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.2.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.handle.AddImageMark_20
//功能说明: 在原文件中添加图片水印(xxx.png,xxx.jpg)生成目标 OFD 文件
//HTTPAgent/AtomAgent 为服务代理类, 通过此类连接及调用转换服务

public class AddImageMark_20 {

    //1、定义 http 代理对象, 请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象, 请求访问转换服务(Springboot 版)
   // AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    // HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");



   //2、定义方法实现 OFD 文件中添加附件
   private void addImageMark() {
        try {
            //2-1、定义原文件, 待转换的文件
            File srcFile = new File("D:\\公文正文.wps");
            //2-2、定义转换后 OFD 文件输出流
            OutputStream out = new
FileOutputStream("D:\\convert\\addimageMark_jpg1.ofd");
            //2-3、定义图片水印文件输入流
            InputStream img=new FileInputStream(new File("D://1.jpg"));
        //2-4_1、定义水印位置: x,y,w,h,index 所有页添加,页数从 0 开始
           MarkPosition mp1=new
MarkPosition(10f,20f,50f,50f,MarkPosition.INDEX_ALL);

            //2-4_2、定义文件某连续页添加水印, 3-4 页添加水印
            Pair<Integer,Integer> pair =new Pair<Integer,Integer>(2,3);
            //2-4_2、、定义水印位置: x,y,w,h,index.
            MarkPosition mp2=new MarkPosition(10f,20f,50f,50f,pair);

            //2-4_3、定义某不连续页添加水印: 水印位置: x,y,w,h,index. 1,6 页添加水印
            MarkPosition mp3=new MarkPosition(10f,20f,50f,50f,new int[] {0,5});

            //2-4_4、定义水印位置: x,y,w,h,index.1、3-4、6 页添加水印
            MarkPosition mp4=new MarkPosition(10f,20f,50f,50f,new int[] {0,5},pair);

            //2-4_5: 定义水印位置: w,h,index. x 坐标,y 坐标都默认为 0
            MarkPosition mp5=new MarkPosition(50f, 50f,MarkPosition.INDEX_ALL) ;
```

```
            //2-4_6: 定义水印位置: index. x 坐标,y 坐标都默认为 0 w 为图片宽度,h 为图片高度
            MarkPosition mp6=new MarkPosition(MarkPosition.INDEX_ALL);



            //平铺(已过时)
            //MarkPosition mp7=new MarkPosition(new int[] {1},
Const.PatternType.Tile);

            //拉伸(已过时)
            //MarkPosition mp8=new MarkPosition(new int[] {1},
Const.PatternType.Stretch);

            //2-5、添加图片水印: 显示和打印都添加
            ha.addImageMark(srcFile,out,img, "jpg", mp3, true, true);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-6、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        AddImageMark_20 addImageMark = new AddImageMark_20();
        addImageMark.addImageMark();
        System.out.println("转换结束--------------------");
     }
}
```

1.3.2.4. 接口约束

无

1.3.3. 添加文字水印

1.3.3.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 addTextMark 接口对 ofd 文档添加文字水印。转换完成的文件保存至设置的路径下。此接口将源文件转换成 OFD 版式文件并将指定文字水印添加至指定位置。一般用于 OFD 版式文件的离线溯源。

1.3.3.2. 接口定义

```
void addTextMark (File srcFile,
                  OutputStream out,
                  TextInfo text,
                  MarkPosition mk,
                  boolean printable,
                  boolean visible);
```

参数说明详见下表：

添加文字水印接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	文件路径	new File("D:\\convert\\ori\\docx_1.docx")
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream("D:\\convert\\tar\\addTextInfo.ofd");
3	text	TextInfo 类型	文本的内容	new TextInfo("我是水印","宋体",18,"#000000")
3	mk	MarkPosition	水印位置信息，见 1.6.2 章节	new MarkPosition(100f, 200f, 100f, 50f, MarkPosition.INDEX_ALL)
4	printable	boolean 类型	是否打印，true 为打印，false 为不打印	true ：打印显示水印
				false ： 打印不显示水印
5	visible	boolean 类型	是否显示,true 为显示，false 为不显示	true ： OFD 文档显示水印
				false ： OFD 文档不显示水印

返回值说明详见下表：

添加文字水印接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.3.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.handle.AddTextMark_21
//功能说明：将“文字水印”字样设置成宋体 38 号蓝色，添加到源文件中并转换成 OFD 文件，保存到设置的目录下中
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class AddTextMark_21 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义方法实现 OFD 文件中添加文字水印
    private void addTextMark() {
        try {
            //2-1、定义原文件，待转换的文件
            File srcFile = new File("D:\\cs.docx");
            //2-2、定义转换后 OFD 文件输出流
            OutputStream out = new FileOutputStream("D:\\convert\\addTextMark1.ofd");
            //2-3、定义要插入的文字水印对象
            TextInfo textinfo=new TextInfo("文字水印","宋体", 38,"#0000ff");

            //2-4_1、定义水印位置：所有页加水印（x,y,w,h,index）
           MarkPosition mp=new
MarkPosition(10f,20f,200f,300f,MarkPosition.INDEX_ALL);

            //2-4_2、定义某不连续页添加水印:2,4,6 页加水印
            MarkPosition mp2=new MarkPosition(10f,20f,200f,300f, new int[] {1,3,5});

            //2-4_3、定义文件某连续页添加水印：3-4 页加水印
            Pair<Integer,Integer> pages =new Pair<Integer,Integer>(2,3);
            MarkPosition mp3=new MarkPosition(10f,20f,200f,300f,pages);

            //2-4_4、定义水印位置：x,y,w,h,index.1、6-7、9 页添加水印
            Pair<Integer,Integer> page2 =new Pair<Integer,Integer>(5,6);
            MarkPosition mp4=new MarkPosition(10f,20f,200f,300f, new int[]
{0,8},page2);

            //2-4_5：定义水印位置：w,h,index. x 坐标,y 坐标都默认为 0
            MarkPosition mp5=new MarkPosition(200f,100f,MarkPosition.INDEX_ALL);
```

```
            //2-5、添加文字水印：原文件，转换后 OFD 文件,水印信息，水印位置，打印，显示
            ha.addTextMark(srcFile,out, textinfo,mp4,true,true);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-6、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {

        AddTextMark_21 addTextMark = new AddTextMark_21();
        addTextMark.addTextMark();
        System.out.println("转换结束------------------");

    }

}
```

1.3.3.4. 接口约束

无

1.3.4. 添加权限

1.3.4.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 addPermission 接口对 ofd 文档进行权限控制。转换完成的文件保存至设置的路径下。此接口将源文件转换成 OFD 版式文件并对文档的某些操作进行控制。

1.3.4.2. 接口定义

```
void addPermission(File srcFile,
                   OutputStream out,
                   Map<Perm, Object> map);
```

参数说明详见下表：

添加权限接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File	文件路径	new File("D:\\convert\\ori\\docx_1.docx");
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream("D:\\convert\\tar\\addPermission.ofd");
3	map	Map<Perm, Object>	权限使用集合	Map map=new HashMap(); map.put(Const.Perm.PRINT,true);

返回值说明详见下表：

添加权限接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.4.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.handle.AddPermission_22
//功能说明: 将注释、水印、签章等权限进行限制
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class AddPermission_22 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义方法实现 OFD 文件中添加权限
    private void addPermission() {
        try {
            //2-1、定义原文件，待转换的文件
            File srcFile = new File("D:\\cs.docx");
            //2-2、定义转换后 OFD 文件输出流
            OutputStream out = new
FileOutputStream("D:\\convert\\addPermission_2.ofd");
```

```
            //2-3、定义 map 集合，用于存储用户权限
            Map<Perm, Object> map=new HashMap();

            //2-4、添加用户权限到集合中
            map.put(Const.Perm.ANNOT,false);//注释
            map.put(Const.Perm.WATERMARK, false);//水印
            map.put(Const.Perm.PRINT, true);//打印功能
            map.put(Const.Perm.PRINT_COPIES, 2);//允许打印的份数
            map.put(Const.Perm.SAVE_AS, false);//另存为功能
            map.put(Const.Perm.EXPORT, false);//导出
            map.put(Const.Perm.SIGNATURE, false);//签章
            //map.put(Const.Perm.COPY,false);//文本选择复制
            map.put(Const.Perm.VALID_END,java.sql.Date.valueOf("2021-08-06"));//有
效期结束日期
            map.put(Const.Perm.VALID_START,java.sql.Date.valueOf("2019-07-01"));//
有效期开始日期

            //2-5、将原文件转为 OFD 文件并添加权限
            ha.addPermission(srcFile, out,map);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-6、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用用
    public static void main(String[] args) {

        AddPermission_22 ofdAddPermission = new AddPermission_22();
        ofdAddPermission.addPermission();
        System.out.println("转换结束--------------------");
    }

}
```

1.3.4.4. 接口约束

添加权限接口中的第三个参数 Map<Perm, Object> map，map 中的 Perm 值请参考下表，

序号	Key 值	含义
1	Const.Perm.ANNOT	注释功能
2	Const.Perm.COPY	文本复制
3	Const.Perm.ASSEM	文件拆分组合
4	Const.Perm.EXPORT	文件导出，包含保存与另存为功能
5	Const.Perm.PRINT	打印功能
6	Const.Perm.PRINT_COPIES	允许打印的最大份数
7	Const.Perm.SAVE_AS	另存为功能
8	Const.Perm.SIGNATURE	签章功能
9	Const.Perm.VALID_END	有效期结束日期
10	Const.Perm.VALID_START	有效期开始日期
11	Const.Perm.WATERMARK	水印功能

1.3.5. 盖章

1.3.5.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 addSeal 接口对 ofd 文档进行盖章操作。转换完成的文件保存至设置的路径下。此接口将源文件转换成 OFD 版式文件并对文档进行盖章。

1.3.5.2. 接口定义

```
void addSeal(File srcFile,
             OutputStream out,
             String provider,
             String oesClass,
             SealInfo info);
```

参数说明详见下表：

添加盖章接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 文件类型	文件路径	new File("D:\\convert\\ori\\docx_1.docx");
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream("D:\\convert\\tar\\addSeal_1.ofd");
3	provider	String 字符串类型（1-255）	印章提供者名称	"Suwell_SDK"
4	oesClass	String 字符串类型（1-255）	关联的类名即第三方签章库实现的 JAVA 接口类名。转换服务可通过此类名找到实现类并调用相关方法	null
5	info	SealInfo 签章信息类型	印章的图像信息,见 1.6.4.SealInfo	SealInfo info=new SealInfo(SealInfo.NativeType.Normal, "2d7c5856-c55f-47c1-8b3c-466bddc60f46", 10, 10, 200, 200,"123456",1);

返回值说明详见下表：

添加盖章接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.5.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.handle.AddSeal_23
//功能说明: 调用 java 实现的接口完成服务端盖章
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class AddSeal_23 {

    //1、定义 http 代理对象，连接请求转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，连接请求转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


   //2、定义方法实现 OFD 文件中添加附件
    private void addSeal() {
        try {
            //2-1、定义原文件，待转换的文件
```

```
            File srcFile = new File("D:\\ofd\\产品使用说明书 v1.4.docx");
            //2-2、定义转换后 OFD 文件输出流
            OutputStream out = new FileOutputStream("D:\\convert\\addSeal_7.ofd");
            //2-3、签章提供商
            String provider = "Suwell_SDK";
            //2-4、签章类名
            String oesClass = null;
            //2-5、印章基本信息
            SealInfo info=new SealInfo(SealInfo.NativeType.All,
"2d7c5856-c55f-47c1-8b3c-466bddc60f46", 10, 10, 200, 200,"123456",0);
            //2-6、调用方法，实现源文件转为 ofd 文件并添加签章
            ha.addSeal(srcFile, out,provider,oesClass,info);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                //2-7、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        AddSeal_23 ofdAddSeal = new AddSeal_23();
        ofdAddSeal.addSeal();
       System.out.println("转换结束-----------------");
    }

}
```

1.3.5.4. 接口约束

无。

1.3.6. 添加签名

1.3.6.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 addSignature 接口对 ofd 文档进行签名操作。转换完成的文件保存至设置的路径下。此接口可实现转换并且进行签名。

1.3.6.2. 接口定义

```
void  addSignature(File srcFile,
                   OutputStream out,
                   String provider,
                   boolean lockSign,
                   com.suwell.ofd.custom.wrapper.model.SignInfo signInfo);
```

```
//该接口已过时
viod addSignature(File srcFile,
                   OutputStream out,
                   String provider,
                   String oesClass);
```

参数说明详见下表：

添加签名接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 文件类型	需要添加签名的文件路径	new File("D:\\convert\\ori\\ofd_1.ofd");
2	out	OutputStream 输出流	签名后文件输出路径	new FileOutputStream(new File("D:\\convert\\tar\\addSeal_2.ofd"));
3	provider	String 类型	数字签名提供者	"Suwell_SDK"
4	lockSign	Boolean 类型	是否锁定签名	true false
5	signInfo	SignInfo 类型	签名信息	new SignInfo("9a6fe709143103ff","123456" ,new TextInfo("我是签名","黑体",20,"#000000"), new MarkPosition(new int [] {1,3})); new

序号	参数名称	参数类型	参数含义	内容举例
				SignInfo("9a6fe709143103ff","123456" ,PackEntry.wrap(new File("D:/2.png")), new MarkPosition(new int [] {1,3}));

返回值说明详见下表：

添加签名接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.6.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.handle.AddSignature_24
//功能说明: 给 OFD,PDF 版式文件加签名转为 OFD 文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class AddSignature_24 {

       //1、定义 http 代理对象，请求访问转换服务(war 包版)
       static HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
       //1、定义代理对象，请求访问转换服务(Springboot 版)
       //static AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
       //static HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


       //2、定义添加签名方法
       private  void  addSignature() {

              try {

                    //2-1、定义原文件，待转换的文件
                     File srcFile = new File("D:\\1.ofd");
                     //2-2、添加签名后输出文件
                     OutputStream out = new FileOutputStream(new
File("D:\\convert\\addSignature_text.ofd"));
                     //2-3、数字签名提供者
                     String provider = "Suwell_SDK";
                     //2-4 是否锁定签名
                     boolean lockSign = true;

                     //2-5、定义签名需要用到的参数
                     //1)cerID,数字签名 ID
                     String certID="9a6fe709143103ff";
                     //2)密码和签章密码一致，123456
                     String password="123456";
                     //3)签名文本内容
                     TextInfo textInfo= new TextInfo("我是签名","黑体", 20,"#000000");
                     //4)签名位置（页码）信息对象
                     MarkPosition position= new MarkPosition(new int [] {1,3});
```

```
                    //2-6_1、签名信息对象:文字签名
                    SignInfo  signInfo = new SignInfo(certID,password, textInfo,
position,true,true);
                    //2-6_2、签名信息对象:图像签名
                    SignInfo  signInfo2 = new SignInfo(certID,password,
PackEntry.wrap(new File("D:/1.jpg")), position,"jpg",true,true);

                    //2-7_1、将原文件转换为 OFD 文件并添加文字签名
                    ha.addSignature(srcFile,out,provider,lockSign,signInfo);
                    //2-7_2、将原文件转换为 OFD 文件并添加图片签名
                    //ha.addSignature(srcFile,out,provider,lockSign,signInfo2);

            } catch (Exception e) {
                e.printStackTrace();
                // TODO: handle exception
            }finally {
                try {
                    //2-8、关闭 ha
                    ha.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

        }

        //3、方法调用
        public static void main(String[] args) {
            AddSignature_24  ofdAddSign = new AddSignature_24();
            ofdAddSign.addSignature();
            System.out.println("转换结束......");
        }

}
```

1.3.6.4. 接口约束

无。

1.3.7. 套模板转换生成 ofd

1.3.7.1. 接口描述

功能说明：

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 templateToOFD 方法来实现 OFD 文件套模板转换，套模板完成的文件保存至设置的路径下。此方式适用于简单单据。

1.3.7.2. 接口定义

```
void templateToOFD(String title,
                   InputStream template,
                   InputStream data,
                   OutputStream out);
```

参数说明详见下表：

套模板转换接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	title	String 字符串类型	模板的头名称	"ofd"
2	template	InputStream 输入流	模板 ofd 文件	new FileInputStream("D:\\1.ofd")
3	data	InputStream 输入流	对应模板的数据文件,格式为 xml	new FileInputStream("D:\\1.xml")
4	out	OutputStream 输出流	本地文件输出路径	new FileOutputStrem("D:\\2.ofd")

返回值说明详见下表：

套模板转换接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.7.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.handle.templateToOFD_25
//功能说明: 套模板转换生成 ofd
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class templateToOFD_25 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");
```

```
    //2、定义套模板转换生成 ofd
    public void template2ofd(){
         try {
                //2-1、定义原 OFD 文件
                InputStream ofdSrcFile = new FileInputStream("D:\\formbuild.ofd");
                //2-2、定义模板文件
                InputStream templeFile = new FileInputStream("D:\\formbuild.xml");
                //2-3、定义模板合成转换后的 OFD 文件
                OutputStream  ofdOutFile = new
FileOutputStream("D:\\convert\\template2ofd.ofd");
                //2-4、执行模板转换
               ha.templateToOFD("ofd", ofdSrcFile,templeFile,ofdOutFile);

         }catch(Exception e) {
              e.printStackTrace();
          }finally {
              try {
                  //2-5、关闭资源
                  ha.close();
              }catch(IOException e) {
                  e.printStackTrace();
              }
          }
    }

    //3、方法调用
    public static void main(String[] args){

         templateToOFD_25 temp=new templateToOFD_25();
         temp.template2ofd();
         System.out.println("转换结束........");

    }

}
```

1.3.7.4. 接口约束

无

1.3.8. 文件加密（已过时）

1.3.8.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 encrypt 接口对 ofd 文档进行加密操作。加密完成的文件保存至设置的路径下。此接口可实现加密。

1.3.8.2. 接口定义

```
Void encrypt(File srcFile,
             OutputStream out,
             com.suwell.ofd.custom.wrapper.model.CipherProvider provider);
```

参数说明详见下表：

文件加密接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 文件类型	需要加密的文件路径	new File("D://2.ofd");
2	out	OutputStream 输出流	文件输出路径	new FileOutputStream(new File("D://demo.zip"));
3	provider	com.suwell.ofd.custom.wrapper.model.CipherProvider	解密信息的提供者	New CipherProvider("xx","xx")

返回值说明详见下表：

文件加密接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.8.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.handle.OfdEncryption_26
//功能说明：给文件加密
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class OfdEncryption_26 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    // AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //2、定义文件加密方法
    private void encrypt(){

        try {
            //2-1、定义源文件，待加密的文件
           File srcFile = new File("D:/123.ofd");
```

```
            //2-2、定义加密后输出的文件流
            OutputStream outputStream = new FileOutputStream(new
File("D:/convert/encrypt.zip"));
            //2-3、定义安全服务代理对象
            CipherProvider cipherProvider = new CipherProvider("Suwell_SDK");
            //2-4、执行文件加密操作
            ha.encrypt(srcFile,outputStream,cipherProvider);

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                //2-5、关闭 ha
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        OfdEncryption_26 encryOfd = new OfdEncryption_26();
        encryOfd.encrypt();
        System.out.println("结束----------------");
    }
}
```

1.3.8.4. 接口约束

无。

1.3.9. 文件解密（已过时）

1.3.9.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 decrypt 接口对加密文档进行解密操作，解密完成的文件保存至设置的路径下。

1.3.9.2. 接口定义

```
void decrypt(File srcFile,
             OutputStream out,
             com.suwell.ofd.custom.wrapper.model.CipherProvider provider);
```

参数说明详见下表：

文件解密接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 文件类型	需要进行解密的文件路径	new File("D://demo.zip");
2	out	OutputStream 输出流	解密后的文件输出路径	new FileOutputStream(new File("D://2.ofd"));
3	provider	com.suwell.ofd.custom.wrapper.model.CipherProvider	解密信息的提供者	New CipherProvider("xx","xx")

返回值说明详见下表：

文件解密接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.9.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.handle.OfdDecryption_27
//功能说明：给加密的文件进行解密
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class OfdDecryption_27 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");

    //2、定义文件解密方法
    public void decryption()  {
        try {
            //2-1、定义源文件，待解密的文件
            File oriFile=new File("D://convert_ofd//test.sfd");
            //2-2、定义解密后生成的目标文件输出流
```

```
            OutputStream out=new FileOutputStream(new File("D://convert//解密后
的.ofd"));
            //2-3、解密信息的提供者
            CipherProvider provider = new CipherProvider("123456");
            //2-4、调用 decrypt 完成解密
            ha.decrypt(oriFile,out,provider);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-5、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        OfdDecryption_27 dp = new OfdDecryption_27();
        dp.decryption();
        System.out.println("结束----------------");
    }


}
```

1.3.9.4. 接口约束

无。

1.3.10. 删除文件页

1.3.10.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 deletePage 接口删除文件（PDF,OFD,Office 类型文件）指定页数并转换为 OFD 文件。

1.3.10.2. 接口定义

```
viod deletePage(File srcFile,
                OutputStream out,
```

```
                int... pages);
```

参数说明详见下表：

删除文件页接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 文件类型	文件路径	new File("D://ofd//2.ofd");
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D://ofd//2.ofd");)
3	pages	Int []	页数	Int[] pages= {2，4};或 int page=3;

返回值说明详见下表：

删除文件页接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.10.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.handle.DeletePage_28
//功能说明: 将流式文件、pdf 等常用办公文件转换成 OFD 并删除指定页面，或将 OFD 文件删除指定页面。
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class DeletePage_28 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    // AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    // HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");
    //2、定义方法删除文件指定页码并生成 OFD 文件
    private void delectpagedemo() {
     try {

         //2-1、定义源文件，待删除页码的文件(.pdf,.ofd,.docx,.wps...)
         File srcFile = new File("D:/测试.pdf");
         //2-2、定义删除页码后生成的目标文件输出流
         FileOutputStream outputStream = new FileOutputStream("D:/convert/删除页
    码.ofd");
          //2-3_1、指定要删除的页数(单页)
         //int pages= 3;
          //2-3_2、指定要删除的页数(多页)
```

```
        int[] pages= {1,3};
        //2-4、调用方法实现删除文件指定页码
        ha.deletePage(srcFile,outputStream,pages);
        //ha.deletePage(srcFile,outputStream,1,5,6);

    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } finally {
        try {
            //2-5、关闭资源
            ha.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
   }

   //3、方法调用
   public static void main(String[] args) {
        DeletePage_28 deletepage = new DeletePage_28();
        deletepage.delectpagedemo();
        System.out.println("结束----------------------");

   }
}
```

1.3.10.4. 接口约束

无。

1.3.11. 计算哈希值

1.3.11.1. 接口描述

实现该功能需调用 hashCode 接口，输出文件对象的哈希码值。

1.3.11.2. 接口定义

```
Int  hashCode();
```

参数说明：

无

返回值说明：

序号	返回类型	返回值含义
1	Int	对象的哈希码值

1.3.11.3. 接口示例

```
//java 代码调用
//功能说明：输入一个对象，并返回这个对象的哈希码值

package com.http.handle;

import java.io.File;

public class OfdHashCode_29 {

    public static void main(String[] args) {
         //源文件
        File f = new File("D:\\1.doc");
        //获取文件 hashCode
        System.out.println("文件的哈希值为:" + f.hashCode());
    }

}
```

1.3.11.4. 接口约束

无。

1.3.12. 插入页面

1.3.12.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 insertPage 接口，在定义插入的位置插入指定范围的 OFD 文档内容。

1.3.12.2. 接口定义

接口 1：

```
void insertPage(File srcFile,
                OutputStream out,
                InputStream source,
                int srcStart,
                int srcEnd,
                Int index);
```

参数说明详见下表：

插入页面接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 文件类型	文件路径	new File("D://ofd//1.ofd");
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream(new File("D://ofd//2.ofd");)
3	source	InputStream 输出流	待插入的文件，必须是 OFD 格式	new FileInputStream(new File("D://ofd//c.ofd"));
4	srcStart	Int 类型	待插入的范围 起始位置 1 表示第 1 页	int srcStart=1;
5	srcEnd	Int 类型	待插入的位置 结束位置 1 表示第一页	int srcEnd=10;
6	index	Int 类型	插入的位置 0 表示第一页前，1 表示第 1 页后	int index=0;

返回值说明详见下表：

插入页面的接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.12.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.handle.InsertPage_30
//功能说明：在指定插入位置，插入指定范围的 OFD 文件内容
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
```

```
public class InsertPage_30 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义方法插入页面到文件
    private void insertPageDemo() {
     try {

          //2-1、定义源文件，待插入内容的文件
          File srcFile = new File("D://11.ofd");
          //2-2、定义完成插入后生成的目标文件输出流
          FileOutputStream outputStream = new FileOutputStream("D://ofd//insert.ofd");
          //2-3、待插入的文件。可选其中的部分页面插入到源文件中
          InputStream source = new FileInputStream(new File("D://cs.ofd"));
          //2-4、待插入文件的起始页
          int srcStart=1;
          //2-5、待插入文件的尾页
          int srcEnd=3;
          //2-6、将待插入文件插入到源文件的位置。
          int index=1;
          //2-7、调用方法实现将指定页面插入到源文件指定页码
          ha.insertPage(srcFile,outputStream,source,srcStart,srcEnd,index);

     } catch (Exception e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
     } finally {
          try {
               //2-8、关闭资源
               ha.close();
          } catch (IOException e) {
               e.printStackTrace();
          }
     }
    }

    //3、方法调用
    public static void main(String[] args) {
       InsertPage_30 insertPage = new InsertPage_30();
       insertPage.insertPageDemo();
```

```
        System.out.println("------------------------");
    }
}
```

1.3.12.4. 接口约束

仅限 OFD 文件插入 OFD 页面。

1.3.13. 删除印章

1.3.13.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 removeSeal 接口将 OFD 版式文件中签章删除。转换完成的文件保存至设置的路径下。

1.3.13.2. 接口定义

```
void removeSeal(File srcFile,
                FileOutputStream  out,
                Pair<SealMode,Boolean>... pair);
```

参数说明详见下表：

单个 OFD 文件删除签章接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	待转换源文件	new File("D:\\ofd\\0216.ofd")
2	out	FileOutputStream 输出流	本地文件输出路径	new FileOutputStream("D:\\ofd\\0216-1.ofd")
3	pair	Pair<SealMode, Boolean>集合	签章集合(删除掉的信息枚举；是否删除)	Pair<SealMode, Boolean> pair = new Pair<SealMode, Boolean>(SealMode.Eseal, true); Pair<SealMode, Boolean> pair = new Pair<SealMode, Boolean>(SealMode.Signature, true);

返回值说明详见下表：

单个 OFD 文件删除签章接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.13.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.handle.RemoveSeal_31
//功能说明: 在 55.ofd 文件中删除签章并生成新的 OFD 文件
//HTTPAgent/AtomAgent 为服务代理类, 通过此类连接及调用转换服务
public class RemoveSeal_31 {

    //1、定义 http 代理对象, 请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象, 请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   //2、定义删除签章方法
    private void removeSeal() {
        try {
            //2-1、定义源文件,待删除签章的文件
            File srcFile = new File("D:/55.ofd");
            //2-2、定义删除签章后目标文件输出流
          FileOutputStream outputStream = new
FileOutputStream("D:/convert/removeSeal.ofd");
            //2-3、 SealMode.Signature 签名 SealMode.Eseal 签章
            Pair<SealMode, Boolean> pair = new Pair<SealMode, Boolean>(SealMode.Eseal,
true);
            //2-4、请求服务, 删除签章
          ha.removeSeal(srcFile, outputStream, pair);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-5、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    //3、方法调用
    public static void main(String[] args) {
        RemoveSeal_31 removeSeal = new RemoveSeal_31();
        removeSeal.removeSeal();
```

```
        System.out.println("结束.........");
    }
}
```

1.3.13.4. 接口约束

无。

1.3.14. 旋转页面

1.3.14.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 rotate 接口将 OFD 版式文件中的内容进行旋转，可以自定义旋转的角度、指定的页码等。转换完成的文件保存至设置的路径下。

1.3.14.2. 接口定义

```
void rotate(File srcFile,
            OutputStream out,
            float rotate,
            int... pages);
```

参数说明详见下表：

单文 OFD 文件旋转页面接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	待转换源文件	new File("D:\\ofd\\4.ofd")
2	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream("D:\\ofd\\4-1.ofd")
3	rotate	float	旋转的角度	如填写（180）就是将该页的内容旋转 180°。
4	pages	int 数组	需旋转的页码	如需旋转单页 int i=2; 如需旋转多页则 int[] i={1,2,3};

返回值说明详见下表：

单文 OFD 文件旋转页面接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.14.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.handle.RotatePage_32
//功能说明：在 ofd 文件中旋转页面
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class RotatePage_32 {

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");



   //2、定义旋转页面方法
    private void rotate() {
        try {

           //2-1、定义源文件，待旋转的文件
           File srcFile = new File("D:/cs.ofd");
           //2-2、定义完成旋转后生成的目标文件输出流
           FileOutputStream outputStream = new
FileOutputStream("D:/convert/rotatePage2.ofd");
           //2-3、定义旋转角度
           int rotate = 180;
           //2-4_1、定义需要旋转的页面（多页）
           int [] pages = new int[] {1,2,3};
           //2-4_2、定义需要旋转的页面（单页）
           //int page = 2;
           //2-5、将源文件的第 1，2，3 页旋转 180 度
          ha.rotate(srcFile, outputStream, rotate,pages);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-6、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

```
    //3、方法调用
    public static void main(String[] args) {
        RotatePage_32 rotate = new RotatePage_32();
        rotate.rotate();
        System.out.println("结束.........");

    }
}
```

1.3.14.4. 接口约束

无。

1.3.15. 交换页面接口

1.3.15.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 swapPage 接口将 OFD 版式文件中的某两个页面交换位置，可以指定交换的页码。转换完成的文件保存至设置的路径下。

1.3.15.2. 接口定义

```
void swapPage(File srcFile,
              OutputStream out,
              Pair<Integer, Integer>... swap);
```

参数说明详见下表：

单个 OFD 文件交换页面接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	待转换源文件	new File("D:\\ofd\\0216.ofd")
1	out	OutputStream 输出流	本地文件输出路径	new FileOutputStream("D:\\ofd\\0216-1.ofd")
2	swap	Pair<Integer, Integer>集合	Integer 类型集合(交换页面 1；交换页面 2)	Pair<Integer, Integer> pair = new Pair<Integer, Integer>(1, 2);代表页面 1 和页面 2 交换位置

返回值说明详见下表：

单个 OFD 文件交换页面接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.15.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.handle.SwapPage_33
//功能说明: 交换页面位置并生成 OFD 文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class SwapPage_33 {
    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


   //2、定义交换页面方法
    private void swapPage() {
        try {
            //2-1、定义源文件，待交换页面的文件
            File srcFile = new File("D:/55.ofd");
            //2-2、定义完成交换后生成的目标文件输出流
          FileOutputStream outputStream = new
FileOutputStream("D:/convert/swapPage.ofd");
          //2-3、定义需要交换的页面
            Pair<Integer, Integer> swap = new Pair<Integer, Integer>(1,2);
            //2-4、请求服务,实现交换页面 ha.swapPage(输入文件,输出文件,需要交换页面位置);
            ha.swapPage(srcFile, outputStream, swap);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-5、关闭资源
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

```
    }

    //3、方法调用
    public static void main(String[] args) {
        SwapPage_33 swapPage = new SwapPage_33();
        swapPage.swapPage();
       System.out.println("结束......");
    }
}
```

1.3.15.4. 接口约束

无。

1.3.16. 拷贝语义

1.3.16.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 copyCustomTag 接口将 OFD 版式文件中的语义拷贝到指定 OFD 文件中。转换完成的文件保存至设置的路径下。

1.3.16.2. 接口定义

```
void copyCustomTag(File srcFile,
                   OutputStream out,
                   InputStream ofd);
```

参数说明详见下表：

拷贝语义接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	srcFile	File 类型	带语义的原文件	new File("D:\\语义 OFD 文件.ofd");
2	out	OutputStream 输出流	拷贝语义后生成的 ofd 文件输出流	new FileOutputStream(new File("D:\\convert_ofd\\copy_ofd.ofd"))
3	ofd	InputStream 输入流	文件不带语义，拷贝语义到该文件	new FileInputStream(new File("D:\\没有语义 OFD 文件.ofd"));

返回值说明详见下表：

拷贝语义接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.16.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.handle.CopyCustomTag_34
//功能说明: 将 OFD 版式文件中的语义拷贝到指定 OFD 文件中
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class CopyCustomTag_34 {

      //1、定义 http 代理对象，请求访问转换服务(war 包版)
       static HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8080/convert-issuer/");
      //1、定义代理对象，请求访问转换服务(Springboot 版)
     //static AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");



     //2、定义拷贝语义方法
       private  void copyCustomTag() {

              try {
                   //2-1、带语义的原文件
                   File srcFile=new File("D:\\文档 3.ofd");
                   //2-2、拷贝语义后生成的 ofd 文件输出流
                   OutputStream out=new FileOutputStream(new
File("D:\\convert_ofd\\copy_ofd.ofd"));
                    //2-3、不带语义的文件，语义拷贝后与该文件生成新的文件流
                    InputStream ofd = new FileInputStream(new File("D:\\33.ofd"));
                    //2-4、执行拷贝语义操作
                    ha.copyCustomTag(srcFile, out,  ofd);

             } catch (Exception e) {
                 // TODO: handle exception
                 e.printStackTrace();
             }finally {
                    try {
                       //2-5、关闭资源
                       ha.close();
                 } catch (Exception e2) {
                     // TODO: handle exception
                     e2.printStackTrace();
```

```
                }
            }
        }

        //3、方法调用
        public static void main(String[] args) {
            CopyCustomTag_34  copyCustom = new CopyCustomTag_34();
            copyCustom.copyCustomTag();
            System.out.println("完成......");
        }

}
```

1.3.16.4. 接口约束

注意：文档语义结构相同时，拷贝语义后能够正常显示和使用。如文档语义结构不同，拷贝语义后会出现内容错乱。在执行该方法时需查看语义结构是否相同。

1.3.17. 带回调的异步转换

1.3.17.1. 接口描述

一般 Agent 代理里的接口可视为同步转换，即一个请求一个返回。如想通过 agent 仍实现异步转换，可引用 Agent*.jar，初始化 Agent 接口（该方法仅限 war 包版转换服务，仅 HTTPAgent 可调用），调用 convertNoWait 接口将文件上传并异步转换，通过回调可返回转换结果。

1.3.17.2. 接口定义

```
viod convertNoWait(InputStream packet,
                    ConvertCallback callback);
```

```
viod convertNoWait(InputStream packet,
                   ConvertCallback callback,
                    String token,
                    boolean raise);
```

```
viod convertNoWait(com.suwell.ofd.custom.wrapper.Packet packet,
```

```
                    ConvertCallback callback);
```

```
viod convertNoWait(com.suwell.ofd.custom.wrapper.Packet packet,
                    ConvertCallback callback,
                    String token,
                    boolean raise);
```

参数说明详见下表：

异步转换接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	packet	InputStream 类型	打包文件流（接口 1，2）	ConvertAgent.*store*(pa)
2	token	String 类型	系统注册的标识（接口 2）。注册方式参见 3.多节点部署及调用。	“8D43C8”
3	raise	boolean 类型	是否将优先级+1（接口 2）	如转换节点 8D43C8，级别：5，如设置升级为 true,则优先级可看成 6
4	packet	Packet 类型	打包文件类（接口 3）	Packet pa =new Packet(PackType.COMMON,Target.OFD);
5	callback	ConvertCallback 类型	转换的回调函数	内部类的方式和外部类的方式。参考 demo

返回值说明详见下表：

异步转换接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.3.17.3. 接口示例

```
//java 代码调用
//参考示例：com.http.nowait.OfficeToOFD_NoWait1
//功能说明：批量上传文件，异步转换，通过回调处理转换完成的文件，回调为内部类方式
//HTTPAgent 为服务代理类，通过此类连接及调用转换服务
public class OfficeToOFD_NoWait1 {

    //1、定义 http 代理对象，连接请求转换服务
    static HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
```

```
    //2、定义方法实现异步转换
    public void HATest() {

            try {
               //2-1、定义文件夹
                File f=new File("D:\\ofd\\test2");
                //2-2、获取文件夹中的文件
                File[] fs=f.listFiles();
                //2-3、循环文件夹中的文件列表
                for (File file : fs) {
                       //2-3-1、获取文件格式
                        String
format=file.getName().substring(file.getName().lastIndexOf(".")+1);
                        //2-3-2、定义起始时间
                        long begintime=System.currentTimeMillis();
                        //2-3-3、定义 packet 包
                        Packet pa =new Packet(PackType.COMMON,Target.OFD);
                        //2-3-4、packet 包存放源文件
                        pa.file(new Common("test", format, new
FileInputStream(file)));
                        //2-3-5、调用方法，执行异步请求转换

                        ha.convertNoWait(pa, new ConvertCallback() {

                          @Override
                          public OutputStream openOutput() throws IOException {
                            String name =file.getName();
                            File file=new File("D:/convert/2/"+name+".ofd");
                            FileOutputStream out=new FileOutputStream(file);
                            System.out.print("--------"+name+"--------");
                            return out;
                          }

                               @Override
                               public void onSuccess() {
                                   // TODO Auto-generated method stub
                                   System.out.println("onSuccess");
                               }

                               @Override
                               public void onStart() {
                                   // TODO Auto-generated method stub
                                   System.out.println("onStart");
```

```
                                }

                                @Override
                                public void onFinally() {
                                    // TODO Auto-generated method stub
                                    System.out.println("onFinally");

                                }

                                @Override
                                public void onFailed(String arg0, String arg1) {
                                    // TODO Auto-generated method stub
                                    System.out.println("onFailed");
                                }

                                @Override
                                public void onException(Exception arg0) {
                                    // TODO Auto-generated method stub
                                    System.err.println(arg0);
                                }
                            }, null, false);
                            //ticket  对应服务系统注册里的 Token,null 走 DEFAULT
                            //raise  是否优先级加 1，false 为不加。true 为加 1.

                            long endtime=System.currentTimeMillis();
                            long totaltime=endtime-begintime;
                            System.out.println("文件名称："+file.getName()+"的大小为
"+file.length()+",转换耗时为"+totaltime);

                        }


            } catch (Exception e) {
                e.printStackTrace();
            }

    }

    //3、方法调用
    public static void main(String[] args){
        OfficeToOFD_NoWait1  demo=new OfficeToOFD_NoWait1();
        demo.HATest();
    }
```

```
}
```

1.3.17.4. 接口约束

无

1.3.18. 文件转换

1.3.18.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 convert 接口将源文件的打包文件 Packet 包转换并输出转换后的目标文件流。转换完成的文件保存至设置的路径下。此接口需要配合 1.4 章节部分 Packet 拼包实现复合转换。

1.3.18.2. 接口定义

```
void convert(com.suwell.ofd.custom.wrapper.Packet packet,
             OutputStream result);
```

```
void convert(InputStream packet,
             OutputStream result);
```

参数说明详见下表：

文件转换接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	packet	Packet 类型	打包文件类（接口 1）	Packet pa =new Packet(PackType.COMMON,Target.OFD);
2	packet	InputStream 类型	打包文件流（接口 2）	ConvertAgent.*store*(pa)
3	result	OutputStream 输出流	文件输出流	new FileOutputStream(new File("D:\\result.ofd"))

返回值说明详见下表：

文件转换接口返回值说明表

序号	返回类型	返回值含义

1	无	无

1.3.18.3. 接口示例

参见 1.4 章节示例。

1.3.18.4. 接口约束

无

1.4. Packet 拼包实现复合转换及加工

复合转换是指用上面的单步接口(1.2-1.3)调用无法达到目地的转换或加工。用户可以通过自行实现源包拼装再调用 Agent 的转换接口实现。具体方法请参考 API。

常见的调用方式为：

```
Packet packet =new Packet(“”, “”);
packet .metadata();.....*
packet .file();......*
packet .envelope();
packet.digestResult();------md5 校验  essageDigest.getInstance("MD5")
.
.
Ha.convert();\\ha.submit();
```

1.4.1. 添加附件

1.4.1.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 attach 接口将原文件添加附件，并转换为 OFD 文件。可设置是否显示附件。转换完成的文件保存至设置的路径下。

1.4.1.2. 接口定义

```
Packet attach(String title,
              String type,
              InputStream in);
```

```
Packet attach(String title,
              String type,
              InputStream in,
              boolean visible);
```

```
Packet attach(String title,
              String type,
              PackEntry in);
```

参数说明详见下表：

添加附件接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	title	String 类型	作为附件时的显示名称,不能为空	"附件 1"
2	type	String 类型	附件的逻辑类型,可以为空	"pdf"
3	in	InputStream 类型	附件文件,不能为空	new FileInputStream("D:\\附件.pdf")
4	visible	Boolean 类型	附件是否可见	true:可见，false:不可见
5	in	PackEntry 类型	附件文件,不能为空	PackEntry.wrap(new File("D:\\1.jpg"))

返回值说明详见下表：

添加附件接口返回值说明表

序号	返回类型	返回值含义
1	Packet	数据包

1.4.1.3. 接口示例

```
//java 代码调用
//参考示例： com.springboot.packet.AddAttach
//功能说明：在原文件中添加附件并转换为 OFD 文件
```

```
//1、定义代理对象，请求访问转换服务(war 包版)
HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
//1、定义代理对象，请求访问转换服务(Springboot 版)
//AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
//HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

public void addAttach() {
   try {
      //1、定义数据包
      Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
      //2、添加公文文件
      packet.file(new Common(null, "ofd", new FileInputStream("D:\\测试.ofd")));
      //3-1、添加附件并设置隐藏。
      packet.attach("附件 1", "pdf", new FileInputStream("D:/测试.pdf"), false);
      packet.attach("附件 3", "png", new FileInputStream("D:/2.png"), false);

      //3-2、添加附件并设置显示
      packet.attach("附件 2", "jpg", new FileInputStream("D:/1.jpg"), true);
      packet.attach("附件 4", "html", new FileInputStream("D:/3.html"), true);

     //4、请求转换服务
     ha.convert(packet, new FileOutputStream("D:\\convert_ofd\\attach3.ofd"));
     packet.close();
  } catch (Exception e) {
        // TODO: handle exception
   }finally {
        try {
            //关闭 ha
            ha.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

     }

}

public static void main(String[] args){
     //3-1:定义添加附件对象
     AddAttach ad = new AddAttach();
     //3-2:执行添加附件方法
     ad.addAttach();
}
```

1.4.1.4. 接口约束

无。

1.4.2. 添加掩膜

1.4.2.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 addMask 方法来进行页面添加显示掩膜，添加完成的文件保存至设置的路径下，添加掩膜后的文件掩膜位置即传参设置的 x,y 位置。

1.4.2.2. 接口定义

```
Packet addMask(float x,
               float y,
               float w,
               float h,
               int ... pages);
```

```
Packet addMask(float x,
               float y,
               float w,
               float h,
               boolean readOnly,
               int... pages);
```

参数说明详见下表：

页面添加掩膜接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	x	float 类型	左定点的 x 坐标	24f
2	y	float 类型	左定点的 y 坐标	24f
3	w	float 类型	掩膜区域的宽	20f

序号	参数名称	参数类型	参数含义	内容举例
4	h	float 类型	掩膜区域的高	20f
5	pages	int 类型	需加掩膜的页码，可以为多页用“,”分隔	int[] i= {1,2,3}或 int i=1
6	readOnly	boolean 类型	掩膜是否只读。无参数默认是只读的。非只读可对掩膜进行删除修改等操作。	false
				true

返回值说明详见下表：

页面添加掩膜接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.4.2.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.packet.AddMask
//功能说明：页面添加掩膜（举例：第一页的 X 坐标 24，Y 坐标 24，添加宽 20，高 20 的掩膜）

public class AddMask{
    //1、定义代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8080/convert-issuer");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

      Packet pkg = new Packet(PackType.COMMON,Target.OFD);
       //源文件
       FileInputStream file=new FileInputStream("D:\\1.ofd");
             Pkg.file(new Common("1","ofd",file));
              //调用 addMask 方法实现给首页添加掩膜
             //pkg.addMask(24f, 24f, 200f, 200f, 1);

             //int[] i= {1,2,3};
              //调用 addMask 方法实现给 1,2,3 页添加掩膜
```

```
            //pkg.addMask(24f, 24f, 200f, 200f, i);



            int[] i= {1,2,3};
            //调用 addMask 方法实现给 1,2,3 页添加掩膜,并且此掩膜可编辑。
             pkg.addMask(24f, 24f, 200f, 200f,false, i);
            //掩膜加完输出文件
            ha.convert(pkg,new FileOutStream("D:\\2.ofd"));
            pkg.close();

        }
    }
}
```

1.4.2.4. 接口约束

无

1.4.3. 添加页码

1.4.3.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）,packet 接口，调用 addPageNumber 接口实现 OFD 文件添加页码。

1.4.3.2. 接口定义

```
Packet addPageNumber(boolean deleteOld,
                     PageNumber pageNumber);
```

参数说明详见下表：

添加页码接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	deleteOld	boolean	是否删除旧的页码	true false
2	pageNumber	PageNumber	页码数据对象	PageNumber pageNum = new PageNumber();

返回值说明详见下表：

添加页码接口返回值说明表

序号	返回类型	返回值含义
1	Packet 类型	packet 对象，添加页码后的文件数据包

1.4.3.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.packet.AddPageNumber
//功能说明：OFD 文件添加页码
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class AddPageNumber {


       //1、定义代理对象，请求访问转换服务（war 包版）
       HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
       //1、定义代理对象，请求访问转换服务(Springboot 版)
      //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
      //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


        public void addPageNumber() {

            try {
                //1、定义数据包
                Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
                //2、添加公文文件
                packet.file(new Common(null, "ofd", new FileInputStream("D:\\文档
4.ofd")));
                //3、添加页码
                //1)是否删除旧页码
                boolean deleteOld = true;

                //2_1)定义偶数页页码对象
                PageNumber pageNum_even = new PageNumber();
                //2_2)定义奇数页页码对象
                PageNumber pageNum_odd = new PageNumber();
                //2_3)定义所有页页码对象

               PageNumber pageNum_all = new PageNumber();

                //2-1）设置页码范围：EVEN--偶数，添加偶数页。
                pageNum_even.setPageRange("EVEN");
                //2-2)设置页码范围：ODD--奇数，添加奇数页。
                pageNum_odd.setPageRange("ODD");
                pageNum_all.setPageRange("ALL");
                //2-1)设置偶数页页码的参数
                //设置偶数页的页码 x 方法位置：右侧
                pageNum_even.setX("Right");
                //设置偶数页的页码 y 方向位置：下方
                pageNum_even.setY("Bottom");
```

```
                //设置偶数页的边距：左 0  上 0 右 30 下 30
                pageNum_even.setPagePadding("0 0 10 30");
                //设置偶数页页码的字体：宋体
                pageNum_even.setFontName("宋体");
                //设置偶数页页码的字体大小：12 float 类型
                pageNum_even.setFontSize(12f);
                //定义偶数页页码的字体颜色
                pageNum_even.setForeColor("#000000");
                //定义偶数页页码的显示格式：- 1 -
                pageNum_even.setPageNumberFormat("-- ${PageNumber} --");
                //定义偶数页页码的起始页码：2
                pageNum_even.setStartNumber(2);
                //定义偶数页页码的间隔：2
                pageNum_even.setStep(2);
                //3)、添加页码（pageNum_all 默认在左上角）
                packet.addPageNumber(deleteOld, pageNum_all);
                //4、请求转换服务
                ha.convert(packet, new
FileOutputStream("D:\\convert\\1\\addPage.ofd"));
                packet.close();
            } catch (Exception e) {
                // TODO: handle exception
                e.printStackTrace();
            }finally {
                try {
                    //关闭 ha
                    ha.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

            }

        }

        public static void main(String[] args){
            //3-1:定义添加附件对象
             AddPageNumber ad = new AddPageNumber();
            //3-2:执行添加附件方法
             ad.addPageNumber();
        }
}
```

1.4.3.4. 接口约束

无

1.4.4. 添加元数据

1.4.4.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet、agnet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 metadata 方法添加自定义元数据。转换完成的文件保存至设置的路径下。

1.4.4.2. 接口定义

```
Packet metadata(Const.Meta key,Object value);
```

参数说明详见下表：

添加自定义元数据接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	key	Const.Meta 枚举类型	枚举类型名称	Const.Meta.CUSTOM_DATAS 表示主定义源数据。
2	value	Object	自定义元数据的标识及值	"test1=45,test2=王洋"；多个用“，”隔开。

返回值说明详见下表：

添加自定义元数据接口返回值说明表

序号	返回类型	返回值含义
1	Packet 类型	packet 对象

1.4.4.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.packet.AddMetadata
//功能说明: 将其他文件转成一个 OFD 文件的同时并添加自定义元数据
//添加自定义元数据
public class AddMetadata {

    //1、定义代理对象，请求访问转换服务（war 包版）
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问 springboot 转换服务
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");
```

```
    public void testAddMetaData(){
            try {

                Packet packet =new Packet(PackType.COMMON, Target.OFD);
                packet.file(new Common("1", "png", 600,new FileInputStream(new
File("D:\\a.png"))));
                packet.file(new Common("2", "docx", 100f,new FileInputStream(new
File("D:\\原文件.docx"))));
                //添加自定义源数据。
                packet.metadata(Const.Meta.CUSTOM_DATAS, "test1=45,test2=王洋");

                //添加电子公文元数据。
                packet.metadata(Const.Meta.DOC_ID, "111");// DOCID
                packet.metadata(Const.Meta.AUTHOR, "我是元数据文档作者");// 作者
                packet.metadata(Const.Meta.TITLE, "我是元数据文档标题"); // 标题
                packet.metadata(Const.Meta.ABSTRACT, "我是元数据文档摘要");// 摘要
                packet.metadata(Const.Meta.SUBJECT, "我是元数据文档主题");// 主题
                packet.metadata(Const.Meta.DOC_USAGE, "我是元数据文档类型");// 类型
                packet.metadata(Const.Meta.MOD_DATE, "2021-01-10");// 修改日期
                packet.metadata(Const.Meta.CREATOR, "我是元数据创建者");

                ha.convert(packet, new FileOutputStream("D:\\customdata2.ofd"));
                packet.close();
            }catch(Exception ex) {
                 ex.printStackTrace();
            }finally {
                try {
                    ha.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

    }

    public static void main(String[] args) throws PackException, IOException,
ConvertException {
        AddMetadata ad = new AddMetadata();
        ad.testAddMetaData();
        System.out.println("--------结束-----------");
    }

}
```

1.4.4.4. 接口约束

元数据约束，请参考附件 1。

1.4.5. 添加文字水印接口

1.4.5.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）。通过调用 textMark 接口将原文件转换为 ofd 文件后并添加水印。

1.4.5.2. 接口定义

```
Packet textMark(TextInfo text, MarkPosition pos);
```

```
Packet textMark(TextInfo text,
                MarkPosition pos,
                boolean printable,
                boolean visible);
```

```
Packet textMark(TextInfo text,
                MarkPosition pos,
                boolean printable,
                boolean visible,
                String name);
```

参数说明详见下表：

添加文字水印接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	text	TextInfo	文本水印的内容	new TextInfo("我是水印","宋体", 18,"#000000")
2	pos	MarkPosition	水印位置信息，见 1.6.2 章节	new MarkPosition(100f, 200f, 100f, 50f, MarkPosition.INDEX_ALL)
3	printable	boolean	是否打印 ,默认是 true	true //打印
				false //不打印
4	visible	boolean	是否显示 ,默认是 true	true //显示

				false //不显示
5	name	String	水印的标识，允许为 null	null

返回值说明详见下表：

添加文字水印接口返回值说明表

序号	返回类型	返回值含义
1	Packet	添加文字水印后文件数据包

1.4.5.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.AddTextMark_3
//功能说明：将文件转换为 ofd 文件，同时添加文字水印
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
//添加文字水印
public class AddTextMark_3 {

    //1、定义代理对象，请求访问转换服务（war 包版）
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问 springboot 转换服务
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义添加文字水印的方法
    private void addTextMark() {
        try {

            //2-1、定义数据包对象
            Packet packet = Packet.common();
            //数据包添加源文件
            packet.file(new Common("", "docx", new FileInputStream("D:\\测试文
件.docx")));

            //2-2、定义水印内容对象。参数：水印内容，水印字体，水印字体大小，水印颜色
            TextInfo textInfo = new TextInfo("测试文本水印","黑体", 18, "#80000");

            //2-3、设置水印透明度(0-100)
            textInfo.setOpacity(20);

            //2-3_1、所有页添加水印.
            //定义水印位置对象：参数传入所有页
            MarkPosition  textMarkPosition = new
```

```
MarkPosition(MarkPosition.INDEX_ALL);
                //设置水印是否平铺：true:平铺，false:不平铺
                textMarkPosition.setTile(true);
                //设置水印平铺 x 轴步长:水印平铺时设置生效
                textMarkPosition.setxStep(100);
                //设置水印平铺 y 轴步长:水印平铺时设置生效
                textMarkPosition.setyStep(100);
                //设置水印旋转角度(0-360)
                textMarkPosition.setRotate(315);

                //2-3_2、2,4,6 页添加水印。
                //定义水印位置对象：参数：水印 x 坐标，水印 y 坐标，宽度，高度，页码下标（页码
下标从 0 开始，需页数-1）
                MarkPosition  textMarkPosition2 = new MarkPosition(0f, 0f, 200f,
200f,new int[] {1,3,5});
                //设置水印水平方向位置：右侧
                textMarkPosition2.setxAlign(XAlign.Center);
                //设置水印垂直方向位置：下方
                textMarkPosition2.setyAlign(YAlign.Bottom);
                //设置水印旋转角度
                textMarkPosition2.setRotate(315);

                //2-3_3、3-7 页添加水印。
                //定义 自定义参数 对象,定义水印添加的页码下标（页码下标从 0 开始，需页数-1）
连续页数：3-7 页
                Pair<Integer, Integer> page = new Pair<Integer, Integer>(2, 6);
                //定义水印位置对象：参数：水印 x 坐标，水印 y 坐标，宽度，高度，页码
                MarkPosition textMarkPosition3 = new MarkPosition(0, 0, 60, 20, page);



                //2-3_4、2,4,7-9 页加水印
                //定义 自定义参数 对象,定义水印添加的页码下标（页码下标从 0 开始，需页数-1）
连续页数：7-9 页
                Pair<Integer, Integer> pages = new Pair<Integer, Integer>(6, 8);
                //定义水印位置对象：参数：水印 x 坐标，水印 y 坐标，宽度，高度，页码
                MarkPosition textMarkPosition4 = new MarkPosition(0, 0, 60, 20, new int[]
{1,3}, pages);

                //2-4、调用 packet 添加文本水印方法给源文件数据包添加文本水印
                //packet.textMark(textInfo ,textMarkPosition);
                packet.textMark(textInfo, textMarkPosition, true, true);

              //2-4、请求转换服务，执行文件添加水印转换。
              ha.convert(packet, new FileOutputStream("D:\\addTextMark_4.ofd"));
```

```
            packet.close();

        } catch (Exception e) {
            // TODO: handle exception
        }finally {
            try {
                //2-5、关闭 ha
                ha.close();
            } catch (Exception e) {
                e.printStackTrace();
            }

        }
     }
    //3、调用方法
     public static void main(String[] args) {
           //3-1:定义添加文字水印对象
            AddTextMark_3 addTextMark = new AddTextMark_3();
        //3-2:执行添加文字水印方法
            addTextMark.addTextMark();

    }
}
```

1.4.5.4. 接口约束

无。

1.4.6. 添加对角线文字水印

1.4.6.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）,packet 接口，调用 addDiagonalTexMark 接口实现 OFD 文件添加对角线文字水印。

1.4.6.2. 接口定义

```
Packet addDiagonalTexMark(TextInfo text,
                          Const.LocationStyle style,
```

```
                        float x,
                        int w,
                        int... pages);
```

参数说明详见下表：

添加对角线文字水印接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	text	TextInfo	文字水印对象	new TextInfo("我是水印", "宋体", 100f);
2	style	Const.LocationStyle	斜线类型	NegativeDiagonal ：反对角线 PositiveDiagonal ：正对角线
3	x	float	水印 x 轴(水平)位置	10f
4	w	int	水印宽度	200
5	pages	int	添加水印的页码，可传多个	0,1,2

返回值说明详见下表：

添加对角线文字水印接口返回值说明表

序号	返回类型	返回值含义
1	Packet 类型	packet 对象，添加斜角线文字水印后的文件数据包

1.4.6.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.packet.AddDiagonalTexMark
//功能说明: 添加对角线文字水印
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class AddDiagonalTexMark {
       //1、定义代理对象，请求访问转换服务（war 包版）
       HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
       //1、定义代理对象，请求访问转换服务(Springboot 版)
      //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
      //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

       //2、定义添加对角线文字水印的方法
        public void addDiagonalTexMark() {

            try {
                //1、定义数据包
                Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
                //2、添加公文文件
                packet.file(new Common(null, "ofd", new
FileInputStream("D:\\csannot2.ofd")));
```

```
                //3、添加对角线水印。
                //1）定义文字水印对象
                TextInfo  txtInfo = new TextInfo("我是水印", "宋体", 100f);
                //2）调用方法，添加斜角线文字水印
                packet.addDiagonalTexMark(txtInfo,Const.LocationStyle.PositiveDiago
                nal,10f,200,0,1,2);
                //4、请求转换服务
                ha.convert(packet, new
FileOutputStream("D:\\addDiagonalTexMark5.ofd"));
                packet.close();
            } catch (Exception e) {
                e.printStackTrace();
            }finally {
                try {
                    //关闭 ha
                    ha.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

            }

        }

         //3、调用方法
        public static void main(String[] args){

             //3-1:定义添加附件对象
             AddDiagonalTexMark ad = new AddDiagonalTexMark();
            //3-2:执行添加附件方法
             ad.addDiagonalTexMark();
        }
}
```

1.4.6.4. 接口约束

无

1.4.7. 添加图片水印

1.4.7.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）。通过调用 addImageMark 接口将原文件转换为 ofd 文件并添加图片水印。

1.4.7.2. 接口定义

```
E addImageMark(T image,
               MarkPosition pos,
               boolean printable,
               boolean visible,
               int opacity);
```

参数说明详见下表：

添加图片水印接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	image	PackEntry 类型	图片水印文件	PackEntry.wrap(new File("D:\\1.jpg"))
2	pos	MarkPosition 类型	水印位置，见 1.6.2 章节	MarkPosition mp=new MarkPosition(10f,20f,200f,300f,MarkPosition.INDEX_ALL);
3	printable	boolean 布尔型	是否打印	true //打印
				false //不打印
4	visible	Boolean 布尔型	是否显示	true //显示
				false //不显示
5	opacity	Int 整数型	水印透明度，0--100 取值	100

返回值说明详见下表：

添加图片水印接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

注：上方为父类 MainXML 方法，下方为 Packet 方法。

```
Packet imageMark(InputStream image,
                 String imageType,
                 MarkPosition pos);
```

```
Packet imageMark(InputStream image,
                 String imageType,
                 MarkPosition pos,
                 int a,
                 boolean tile);
```

```
Packet imageMark(InputStream image,
                 String imageType,
```

```
                XAlign xa,
                YAlign ya,
                MarkPosition pos) ;
```

```
Packet imageMark(InputStream image,
                String imageType,
                MarkPosition pos,
                boolean printable,
                boolean visible);
```

```
Packet imageMark(InputStream image,
                String imageType,
                XAlign xa,
                YAlign ya,
                MarkPosition pos,
                boolean printable,
                boolean visible);
```

参数说明详见下表：

添加图片水印接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	image	InputStream 类型	图片水印文件流	new FileInputStream("D:/1.png");
2	imageType	String 字符串类型	图片水印类型	"png"
3	pos	MarkPosition 类型	水印位置,见 1.6.2 章节	MarkPosition mp=new MarkPosition(MarkPosition.INDEX_ALL);
4	alpha	Int 整数型	图片的透明和半透明度	0~255 之间
5	tile	boolean 布尔型	水印是否平铺	true //平铺 false //不平铺
6	xa	XAlign	水印水平方向位置	XAlign.Left //设置水平居左 XAlign.Center //设置水平居中 XAlign.Right //设置水平居右
7	ya	YAlign	水印垂直方向位置	YAlign.Bottom //设置垂直居下 YAlign.Middle //设置垂直居中 YAlign.Top //设置垂直居上
8	printable	boolean 布尔型	是否打印	true //打印 false //不打印
9	visible	Boolean 布尔型	是否显示	true //显示 false //不显示

返回值说明详见下表：

添加图片水印接口返回值说明表

序号	返回类型	返回值含义
1	Packet	添加图片水印后文件数据包

1.4.7.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.AddImageMark_4_1/4_2
//功能说明：将文件转换为 ofd 文件，同时添加图片水印
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

   //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义添加图片水印方法
     private void addImageMark() {
           try {

              //2-1、定义数据包对象
              Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
              //2-2、定义通用文件对象，源文件类型数据，即待转换的文件
              Common common = new Common("", "docx", new FileInputStream("D:\\测试文
件.docx"));
              //2-3、数据包添加源文件
              packet.file(common);

              //2-4、定义水印显示位置对象。添加水印需要用到该参数
              //2-4_1)所有页添加水印。
              //定义添加的水印位置信息对象。参数：水印图片宽度，水印图片高度，水印显示的页码
             MarkPosition imgMarkPosition = new MarkPosition(20,
                                                          30,
                                                        MarkPosition.INDEX_ALL);
              //设置水印旋转角度：平铺后未生效
              imgMarkPosition.setRotate(315);
              //设置水印平铺
              imgMarkPosition.setTile(false);
              //设置水印水平方向步长 :平铺后生效
              imgMarkPosition.setxStep(10);
```

```
            //设置水印垂直方向步长 :平铺后生效
            imgMarkPosition.setyStep(20);
            //设置水印水平方向位置
            imgMarkPosition.setxAlign(Const.XAlign.Right);
            //设置水印垂直方向位置
            imgMarkPosition.setyAlign(Const.YAlign.Top);

            //2-4_2)第 1-3 页添加水印。
            //2-4_2:1)定义自定义参数对象，传入 1-3 页页码
            Pair<Integer,Integer> pair =new Pair<Integer,Integer>(0,2) ;
            //2-4_2:2)定义添加的水印位置信息对象。参数：水印显示 x 轴起点坐标，水印显示 y
轴起点坐标，水印图片宽度，水印图片高度，水印显示的页码
            MarkPosition imgMarkPosition2 = new MarkPosition(0,0,20,30,pair);

            //2-4_3)第 1、3、6 页添加图片水印:参数：水印宽度，水印高度，加入水印页码
        MarkPosition imgMarkPosition3 = new MarkPosition(20,30,(new int [] {0,2,5}));

            //2-5、处理数据包添加图片水印。
            //1):所有页添加水印。参数：图片水印文件，水印位置信息，是否可打印，是否可显示，
透明度参数（0--100）
            //packet.addImageMark(PackEntry.wrap(new File("D:\\1.jpg")),
                                              imgMarkPosition,
                                              true,
                                              true,
                                              100);
             //2):1-3 页添加图片水印
             //packet.addImageMark(PackEntry.wrap(new File("D:\\1.jpg")),
                                                imgMarkPosition2,
                                                true,
                                                true,
                                                100);
            //3):第 1、3、6 页添加图片水印
            packet.addImageMark(PackEntry.wrap(new File("D:\\1.jpg")),
                                             imgMarkPosition3,
                                             true,
                                             true,
                                             100);

      //2-6、请求转换服务，执行文件添加图片水印转换。参数：原转换前文件数据包，转换后文件输
出流
         ha.convert(packet,
                    new FileOutputStream("D:\\convert_ofd\\addImageMark_4_5.ofd"));

            packet.close();
```

```
        } catch (Exception e) {
                // TODO: handle exception
         }finally {
                try {
                    //2-7: 关闭 ha
                    ha.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
        }
    }
    //3、调用方法
    public static void main(String[] args) {
            //3-1:定义添加文字水印对象
            AddImageMark_4_1 addImageMark = new AddImageMark_4_1();
            //3-2:执行添加文字水印方法
            addImageMark.addImageMark();
     }
```

1.4.7.4. 接口约束

无。

1.4.8. 添加签名

1.4.8.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 signature 接口实现将原文件转换为 ofd 文件后并添加签名。

1.4.8.2. 接口定义

```
Packet signature(String provider,
                boolean lockSign,
                SignInfo<PackEntry> signInfo);
```

参数说明详见下表：

添加签名接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	provider	String 类型	数字签名提供者	"Suwell_SDK"
2	lockSign	boolean 类型	是否锁定签名	true false
3	signInfo	SignInfo<PackEntry> 类型	签名信息	new SignInfo<PackEntry>("9a6fe709143103ff","123456" ,new TextInfo("我是签名","黑体", 20,"#000000"), new MarkPosition(new int [] {1,3})); new SignInfo("9a6fe709143103ff","123456" ,PackEntry.wrap(new File("D:/2.png")), new MarkPosition(new int [] {0}));

返回值说明详见下表：

添加签名接口返回值说明表

序号	返回类型	返回值含义
1	Packet	添加签名后文件数据包

1.4.8.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.AddSignature_8
//功能说明：将原文件转为 OFD 文件添加签名
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class AddSignature_8 {

     //1、定义 http 代理对象，请求访问转换服务(war 包版)
     HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问 springboot 转换服务
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   private  void  addSignature() {
        try {

               //1、定义 packet 对象
               Packet packet = new Packet(PackType.COMMON,Target.OFD);
               Common common = new Common(null, "pdf",new FileInputStream(new
File("D:/cs.pdf")));
               packet.file(common);
               //2、定义签名对象
```

```
                //1)cerID,数字签名 ID
                String certID="9a6fe709143103ff";
                //2)数字签名密码
                String password = "123456";
                //3)数字签名文本
                TextInfo textInfo = new TextInfo("我是数字签名", "simsun", 20f);
                //4)数字签名位置
                MarkPosition position = new MarkPosition(MarkPosition.INDEX_ALL);
                //5)签名对象: 文字
                SignInfo<PackEntry> sign = new SignInfo<PackEntry>(certID, password,
textInfo, position);
               //5)签名对象: 图片
                SignInfo<PackEntry> sign2 = new SignInfo<PackEntry>(certID,password,
PackEntry.wrap(new File("D:/a.png")), position,"png",true,true);

                //3、调用方法执行添加签名
                packet.signature("Suwell_SDK", false,sign2);
                //4、调用转换服务, 执行转换
                ha.convert(packet, new FileOutputStream("D:/addSign2.ofd"));
                packet.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    }

    public static void main(String[] args) {

        AddSignature_8  obj = new AddSignature_8();
        obj.addSignature();
        System.out.println("结束...");

    }
}
```

1.4.8.4. 接口约束

无。

1.4.9. 添加归档章

1.4.9.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 archivesSeal 接口实现将原文件转换为 ofd 文件并添加归档章。

1.4.9.2. 接口定义

```
Packet archivesSeal(Pair<String, String>[] ps,
                    float x,
                    float y,
                    int opacity,
                    boolean moveable,
                    String tid,
                    int... pages);
```

```
Packet archivesSeal(Pair<String, String>[] ps,
                    float x,
                    float y,
                    int opacity,
                    int... pages);
```

参数说明详见下表：

添加回档章接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	ps	Pair 类型	归档章需要的信息	Pair<String,String>[] pair = new Pair[6];
2	x	float 类型	x 轴坐标	10f
3	y	float 类型	y 轴坐标	10f
4	opacity	int 类型	不透明度	0 //透明
				100 //不透明 0~100 之间
5	moveable	boolean 类型	是否可以移动	true
				false

6	tid	String 类型	文号章文件名称	"归档章 1"
7	pages	int 类型	添加页码	1,2,3

返回值说明详见下表：

添加回档章接口返回值说明表

序号	返回类型	返回值含义
1	Packet	添加回档章后文件数据包

1.4.9.3. 接口示例

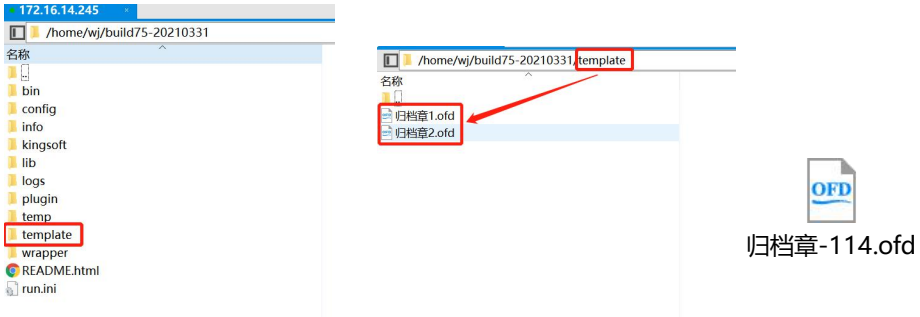
```
//java 代码调用
//参考示例：com.springboot.packet.AddArchiveSeal
//功能说明：将原文转换为 ofd 文件并添加归档章
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class AddArchiveSeal {

    public void addArchiveSeal() {

        //1、定义转换服务 http 代理对象
        HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");

        try {

            //定义归档章参数信息
            Pair<String,String>[] pair = new Pair[6];
            pair[0] =new Pair<String,String>("数科 111 号","#0000FF");
            pair[1] =new Pair<String,String>("数科网维","#0000FF");
            pair[2] =new Pair<String,String>("111 号","#0000FF");
            pair[3] =new Pair<String,String>("122 号","#0000FF");
            pair[4] =new Pair<String,String>("133 号","#0000FF");
            pair[5] =new Pair<String,String>("144 号","#0000FF");
            //定义页码
            int [] page = {1,2,3};
            //2、定义数据包
            Packet packet = Packet.archivesSeal(pair, 150f, 10f, 100,true,"归档章",
page);
            //3、添加公文文件
            packet.file(new Common(null, "pdf", new FileInputStream("D:\\a.pdf")));
            //4、请求转换服务
            ha.convert(packet, new
FileOutputStream("D:\\convert_ofd\\attach3.ofd"));
            packet.close();
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            try {
                //关闭 ha
                ha.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }

    }

    public static void main(String[] args) throws PackException, IOException,
ConvertException {

         //3-1:定义添加归档章对象
         AddArchiveSeal ad = new AddArchiveSeal();
        //3-2:执行添加归档章方法
         ad.addArchiveSeal();
         System.out.println("......");
    }
}
```

1.4.9.4. 接口约束

0820 以后版本可用。

将归档模板放在准备的 build75/template 文件夹下



1.4.10. 添加语义

1.4.10.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 invoke 接口实现将原文件转换为 ofd 文件并添加语义。

1.4.10.2. 接口定义

```
E invoke(String type,
         String module,
         String method,
         Pair<String,String>... pairs);
```

参数说明详见下表：

添加语义接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	String 类型	调用类型	null
2	module	String 类型	调用模块	"dom"
3	method	String 类型	调用方法名称	"MergeCustomTages"
4	pairs	Pair<String,String> 类型	自定义参数	new Pair<String, String>("arg", JBTagesXml2Json.jbArchivesXmlToJson(new File("D:\\公文\\1.xml"))); //1.xml 为语义文件

返回值说明详见下表：

添加语义接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

1.4.10.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.packet.AddCustomTag
//功能说明: 将原文转换为 ofd 文件并添加语义
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
//添加语义
public class AddCustomTag {
```

```
    public void addCustom() {

          //1、定义 http 代理对象，请求访问转换服务(war 包版)
         HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
         //1、定义代理对象，请求访问 springboot 转换服务
        //AtomAgent ha = new AtomAgent("http://127.0.0.1:8091");
        //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

        //定义数据包
        Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);

        try {

            //2、添加公文文件
            packet.file(new Common(null, "ofd", new FileInputStream("D:\\公文
\\cs.ofd")));
            //3、添加语义
            packet.invoke(null, "dom", "MergeCustomTages", new Pair<String,
String>("arg", JBTagesXml2Json.jbArchivesXmlToJson(new File("D:\\公文\\1.xml"))));
            //4、请求转换服务
            ha.convert(packet, new
FileOutputStream("D:\\convert_ofd\\addCustom.ofd"));

        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }finally {
            try {
                //关闭 ha
                packet.close();
                ha.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }

    }

    public static void main(String[] args) throws PackException, IOException,
ConvertException {
```

```
        //3-1:定义添加语义对象
        AddCustomTag ad = new AddCustomTag();
        //3-2:执行添加语义方法
        ad.addCustom();
        System.out.println("结束......");
    }
}
```

1.4.10.4. 接口约束

单个语义内容就显示单个，多个语义内容时显示属性和值的字段。

1.4.11. 文件高压缩

1.4.11.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 optimize 接口实现将原文件转换为 ofd 文件并进行压缩处理。

1.4.11.2. 接口定义

```
Packet optimize(boolean opCompress);
```

参数说明详见下表：

文件高压缩接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	opCompress	boolean 类型	是否压缩	true
				false

返回值说明详见下表：

文件高压缩接口返回值说明表

序号	返回类型	返回值含义
1	Packet	高压缩后文件数据包

1.4.11.3. 接口示例

```
//java 代码调用
```

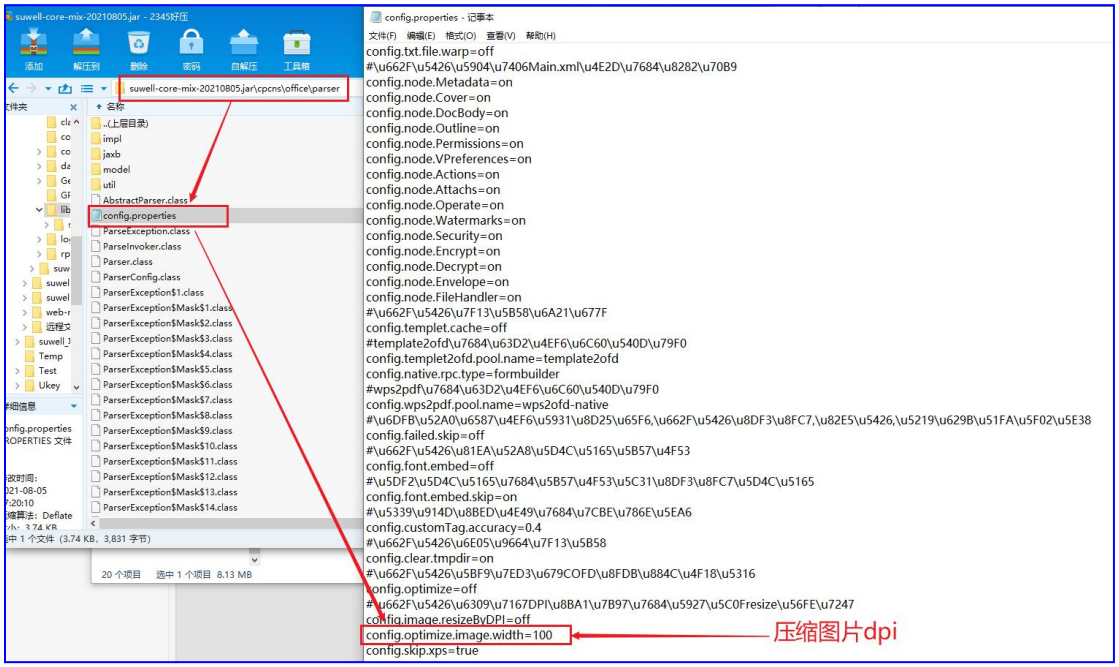
```
//参考示例：com.springboot.api.HighSolidity
//功能说明：将原文件转换为 ofd 文件格式并进行压缩
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class HighSolidity{



    //文件类型
    public static String[] array = { "pdf","doc", "docx", "wps", "xls", "xlsx", "csv",
"ppt", "pptx", "rtf", "txt", "jpg", "tif", "tiff", "png" };
     //1、定义 http 代理对象，请求访问转换服务(war 包版)
    static HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问 springboot 转换服务
     //static AtomAgent ha = new AtomAgent("http://127.0.0.1:8091");
     //static HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //方法调用
    public static void main(String[] args) {
        //输入文件路径
        String fileInPath="D:\\2222\\";
        //轻量 OFD 输出文件路径
        String fileOutPathQ="D:\\out\\";

        //获取文件夹中所有文件
        File[] fs = new File(fileInPath).listFiles(new FileFilter() {
            @Override
            public boolean accept(File pathname) {
                String extension =
FilenameUtils.getExtension(pathname.getName()).toLowerCase();
                System.out.println("extension:"+extension);
                return pathname.isFile()&&Arrays.asList(array).contains(extension);
            }
        });

        for (File f : fs) {
            try {
                 System.out.println("压缩文件：  "+f.getName()+"......开始......");
                 //获取文件类型
              String format = f.getName().substring(f.getName().lastIndexOf(".") +
1);
              //定义数据包
              Packet pa = new Packet(PackType.COMMON, Target.OFD);
              //数据包添加文件
              pa.file(new Common("test", format, new FileInputStream(f)));
              //执行文件压缩
```

```
                pa.optimize(true);
                //执行添加元数据，添加自定义元数据
                pa.metadata(Const.Meta.CUSTOM_DATAS, "a=自定义数据 1,b=自定义 2");
                //定义文件输出流
                OutputStream out =new
FileOutputStream(fileOutPathQ+f.getName()+".ofd");
                //调用转换服务，执行转换
                  ha.convert(pa, out);

                  out.flush();
                  out.close();
                  pa.close();
                  System.out.println("压缩文件: "+f.getName()+"......结束......");


              }catch(Exception e) {
                   e.printStackTrace();
              }

        }

    }
}
```

1.4.11.4. 接口约束

注：需要大文件查看效果。

压缩图片 dpi 配置：进入转换服务部署包的 lib 目录下，修改 suwell-core-mix-xxxxxxxx.jar



【备注】：通过修改压缩图片的 dpi 大小从而改变转换后 ofd 文件的大小

1.4.12. 拆分页面

1.4.12.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 commonSplit 接口把文件转换为 ofd 文件后分割成若干个 ofd。转换完成的文件以压缩包类型（.zip）文件保存至设置的路径下。

1.4.12.2. 接口定义

```
Packet commonSplit(int... splitPoint);
```

参数说明详见下表：

文件拆分转换接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	splitPoint	Int 类型	拆分文件的页码,可传入多个参数。	1,3,5 //第 1 页，2-3 页，4-5 页，6-最后一页拆分，共拆分为 4 个文件，以压缩包文件形式返回

返回值说明详见下表：

文件拆分转换接口返回值说明表

序号	返回类型	返回值含义
1	Packet	拆分后文件数据包

1.4.12.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.SplitFile_1
//功能说明: 将原文件拆分页面为多个文件，并转换为 ofd 文件格式，返回拆分后 ofd 文件的压缩包
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

    //1、定义转换服务 http 代理对象(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
```

```
//HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

//2、定义方法，执行文件拆分转换操作
 private void splitFile(){
      try {
          //2-1、定义数据包对象，执行拆分,拆分结果:1-2,3,4-5,6-7,8-11
          Packet packet = Packet.commonSplit(2,3,5,7);
          //2-2、定义通用文件对象，源文件数据,即要操作的文件
     Common common = new Common("","docx",new FileInputStream("D:\\测试文件.docx"));
          //2-3、数据包添加源文件
          packet.file(common);
          //2-4、请求转换服务，执行文件拆分转换，转换结果为拆分后 ofd 文件压缩包，
         ha.convert(packet,
                     new FileOutputStream("D:\\convert_ofd\\splitFile_1.zip"));
           packet.close();
          } catch (Exception e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
          }finally {
               try {
                   //2-5、关闭 ha
                   ha.close();
               } catch (IOException e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
               }
          }
  }
  //3、调用方法
  public static void main(String[] args) {
      //3-1:定义拆分文件对象
     SplitFile_1 splitFile = new SplitFile_1();
     //3-2:执行拆分文件方法
     splitFile.splitFile();
   }
```

1.4.12.4. 接口约束

无。

1.4.13. 裁切页面

1.4.13.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 clip 接口把文件转换为 ofd 文件并剪切页面。转换完成的文件保存至设置的路径下。

1.4.13.2. 接口定义

```
Packet clip(float x,
            float y,
            float w,
            float h,
            int... pages);
```

参数说明详见下表：

页面剪切转换接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	x	float 浮点类型	剪切的 x 坐标	80f
2	y	float 浮点类型	剪切的 y 坐标	80f
3	w	float 浮点类型	剪切的宽度	80f
4	h	float 浮点类型	剪切的高度	80f
5	pages	int [] 类型	执行操作的页数	1,2

返回值说明详见下表：

页面剪切转换接口返回值说明表

序号	返回类型	返回值含义
1	Packet	文件数据包

1.4.13.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.ClipFile_2
//功能说明: 将原文件页面剪切并转换为 OFD 文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

public class ClipFile_2 {
```

```
    //1、定义转换服务 http 代理对象(war 包版)
   HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问 springboot 转换服务
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent  ha = new HTTPAgent("http://127.0.0.1:8090");


     //2、定义方法，执行文件剪切转换操作
     private void clipFile(){
           try {
             //2-1、定义数据包对象
             Packet packet =Packet.common();
             //2-2、定义公共文件对象，获取源文件
             Common common = new Common("","doc",new FileInputStream("D:\\1.doc"));
             //2-3、数据包添加源文件
             packet.file(common);
             //2-4、执行方法剪切页面
             packet.clip(80f, 80f, 80f, 80f, 1,2);
             //2-5、请求转换服务，执行文件剪切转换
             ha.convert(packet,new FileOutputStream("D:\\clipFile_1.ofd"));
             packet.close();
             } catch (Exception e) {
                 // TODO Auto-generated catch block
                 e.printStackTrace();
             }finally {
                  try {
                     //2-5、关闭 ha
                     ha.close();
                  } catch (IOException e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
                 }
             }
     }
     //3、调用方法
     public static void main(String[] args) {
        //3-1:定义剪切文件对象
        ClipFile_2 clipFile = new ClipFile_2();
        //3-2:执行剪切文件方法
        clipFile.clipFile();
        System.out.println("结束...");
     }
}
```

1.4.13.4. 接口约束

无。

1.4.14. 插入页面

1.4.14.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 insertPage 接口把文件转换为 ofd 文件并插入页面。转换完成的文件保存至设置的路径下。

1.4.14.2. 接口定义

```
Packet insertPage(InputStream source,
                  int srcStart,
                  int srcEnd,
                  int index);
```

```
Packet insertPage(InputStream source,
                  int srcStart,
                  int srcEnd,
                  int index,
                  String format);
```

参数说明详见下表：

插入页面文件转换接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	source	InputStream 类型	待插入的文档，必须是 OFD 格式/PDF 格式	80
2	srcStart	int 类型	待插入的范围，起始位置	0
3	srcEnd	int 类型	待插入的范围，结束位置	2
4	index	int 类型	插入的位置，0 表示第一页前，1 表示第 1 页后，如超过源文件页数插入到源文件最后面	1/0/3

5	format	String 类型	待插入文档的格式,必须与 source 参数的文件类型一致	"pdf"
				"ofd"

返回值说明详见下表：

插入页面文件转换接口返回值说明表

序号	返回类型	返回值含义
1	Packet	文件数据包

1.4.14.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.InsertPageFile_3
//功能说明: 将指定页面插入到原文件并转换为 OFD 文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

//插入页面
public class InsertPageFile_3 {

     //1、定义转换服务 http 代理对象(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问 springboot 转换服务
     //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
     //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

     //2、定义方法，执行文件插入转换操作
      private void insertPage(){
            try {
               //2-1、定义数据包对象
              Packet packet =Packet.common();
              //2-2、定义公共文件对象，获取源文件
              Common common = new Common("","pdf",new FileInputStream("D:\\cs.pdf"));
              //2-3、数据包添加源文件
              packet.file(common);
              //2-4、执行方法插入页面
              //packet.insertPage(new FileInputStream("D:/cs.ofd"), 0, 1, 1);
              packet.insertPage(new FileInputStream("D:/a.pdf"), 0, 1, 2,"pdf");
              //2-5、请求转换服务，执行文件插入转换
              ha.convert(packet,new FileOutputStream("D:\\insertFile.ofd"));
             packet.close();
          } catch (Exception e) {
                 // TODO Auto-generated catch block
                 e.printStackTrace();
```

```
        }finally {
                try {
                    //2-5、关闭 ha
                    ha.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
    }

    //3、调用方法
    public static void main(String[] args) {
        //3-1:定义插入文件方法对象
        InsertPageFile_3 insert = new InsertPageFile_3();
        //3-2:执行插入文件方法
        insert.insertPage();
        System.out.println("结束...");
    }

}
```

1.4.14.4. 接口约束

无。

1.4.15. 合并书签

1.4.15.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 invoke 接口把两个 OFD 文件书签合并。

1.4.15.2. 接口定义

```
Packet invoke(InvokePair invokePair);
```

参数说明详见下表：

合并书签接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	invokePair	InvokePair<K, V>类型	调用参数对象（存储键值对）	InvokePair.SIGNATURE .build(InvokePair.SIGNAM.RemoveSignature) .add("SignType", Arrays.asList("LockSign"))

返回值说明详见下表：

合并书签接口返回值说明表

序号	返回类型	返回值含义
1	Packet	文件数据包

1.4.15.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.MargeBook
//功能说明: 将 OFD 文件书签合并
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class MargeBook {

    public static void main(String[] args) {

       //1、定义转换服务 http 代理对象(war 包版)
       HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
       //1、定义代理对象，请求访问 springboot 转换服务
        //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
        //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

        Packet packet = new Packet(Const.PackType.COMMON, Target.OFD);
        try {
          //原文件 1
            packet.file(new Common("test", "ofd", new
FileInputStream("D:\\test.ofd")));
            //带书签的文件 2
            String name = packet.presetEntry(PackEntry.wrap(new
File("D:\\test2.ofd"))).value();//test.ofd 的书签往 test2.ofd 里面合
            //调用方法执行合并书签

    packet.invoke(InvokePair.DOM.build(InvokePair.DM.MergeBookMark).add("FileName",
name));
            //转换输出合并后的文件
            ha.convert(packet, new FileOutputStream("D:\\13-out.ofd"));
            //packet.pack(new FileOutputStream("D:/ofd/packet-51.zip"));
```

```
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                packet.close();
                ha.close();
                System.out.println("结束......");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

1.4.15.4. 接口约束

21.09.03 以后版本 API 可用。

说明：1.ofd 为原文件，2.ofd 为书签文件，3.ofd 为书签合并后文件。3.ofd 内容为 1.ofd，书签信息为 1.ofd+2.ofd 书签（如果有 1.ofd 与 2.ofd 有同名书签，同名书签仅保留 1.ofd 文件中的书签以及定位，如果 2.ofd 页码大于 1.ofd 页码，那超过 1.ofd 页码的书签只显示书签标题，点击书签标题不做任何跳转）

1.4.16. 删除附件

1.4.16.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 removeAttach 接口把文件转换为 ofd 文件并删除附件。转换完成的文件保存至设置的路径下。

1.4.16.2. 接口定义

```
Packet removeAttach(Const.RemoveAttachType type,
                    String... attach);
```

参数说明详见下表：

删除附件接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	Const.RemoveAttachType 类型	删除附件的类型	Const.RemoveAttachType.Name //根据名称删除
				Const.RemoveAttachType.Format //根据附件格式删除
				Const.RemoveAttachType.All
2	attach	String 类型	删除附件的名称或者格式（可以为空）	null
				"pdf"
				"cs" //注意不要加文件格式

返回值说明详见下表：

删除附件接口返回值说明表

序号	返回类型	返回值含义
1	Packet	文件数据包

1.4.16.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.DelAttach_4
//功能说明：将 OFD 文件指定附件删除
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

//删除附件
public class DelAttach_4 {

     //1、定义转换服务 http 代理对象(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问 springboot 转换服务
     //AtomAgent ha = new AtomAgent("http://127.0.0.1:8091");
     //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8091");

     //2、定义方法，执行文件拆分转换操作
      private void removeAttach(){
```

```
            try {
               //2-1、定义数据包对象
              Packet packet =Packet.common();
              //2-2、定义公共文件对象，获取源文件
              Common common = new Common("","ofd",new
FileInputStream("D:\\removeFile1.ofd"));
              //2-3、数据包添加源文件
                  packet.file(common);
                  //2-4、执行方法删除所有附件
                  packet.removeAttach(Const.RemoveAttachType.All,null);
                  //2-4_2、执行方法根据文件格式删除附件
                  //packet.removeAttach(Const.RemoveAttachType.Format,"pdf");
                  //2-4_3、执行方法根据文件名称删除附件
                  //packet.removeAttach(Const.RemoveAttachType.Name,"config");

                  //2-5、请求转换服务，执行文件剪切转换
                  ha.convert(packet,new FileOutputStream("D:\\removeFile4.ofd"));
                 packet.close();
              } catch (Exception e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
              }finally {
                   try {
                      //2-5、关闭 ha
                      ha.close();
                   } catch (IOException e) {
                      // TODO Auto-generated catch block
                      e.printStackTrace();
                  }
              }
      }

      //3、调用方法
      public static void main(String[] args) {
         //3-1:定义插入文件对象
         DelAttach_4 deleAtt = new DelAttach_4();
         //3-2:执行插入文件方法
         deleAtt.removeAttach();
         System.out.println("结束...");
      }

}
```

1.4.16.4. 接口约束

无。

1.4.17. 删除注释

1.4.17.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 removeAnnot 接口把文件转换为 ofd 文件并删除注释。转换完成的文件保存至设置的路径下。

1.4.17.2. 接口定义

```
Packet removeAnnot(String name, int... pages);
```

参数说明详见下表：

删除注释接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	name	String 类型	注释类型	见接口约束
2	pages	int [] 类型	删除注释的页码	1

返回值说明详见下表：

删除注释接口返回值说明表

序号	返回类型	返回值含义
1	Packet	文件数据包

1.4.17.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.DelAnnot_5
//功能说明: 将 OFD 文件指定注释删除
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
```

```
//删除注释
public class DelAnnot_5 {

    //1、定义转换服务 http 代理对象(war 包版)
   HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
   //1、定义代理对象，请求访问 springboot 转换服务
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8091");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8091");

     //2、定义方法，执行文件拆分转换操作
     private void removeAnnot(){
           try {
              //2-1、定义数据包对象
             Packet packet =Packet.common();
             //2-2、定义公共文件对象，获取源文件
             Common common = new Common("","ofd",new FileInputStream("D:\\cs.ofd"));
             //2-3、数据包添加源文件
             packet.file(common);
             //2-4、执行方法删除注释（批注）
             packet.removeAnnot("AddRevisions",1);

             //2-5、请求转换服务，执行删除注释
             ha.convert(packet,new FileOutputStream("D:\\removeAnnot4.ofd"));
            packet.close();
           } catch (Exception e) {
                 // TODO Auto-generated catch block
                 e.printStackTrace();
           }finally {
                  try {
                      //2-5、关闭 ha
                      ha.close();
                  } catch (IOException e) {
                      // TODO Auto-generated catch block
                      e.printStackTrace();
                  }
             }
     }

     //3、调用方法
     public static void main(String[] args) {
        //3-1:定义删除注释方法对象
        DelAnnot_5 deleAnnot = new DelAnnot_5();
        //3-2:执行删除注释方法
        deleAnnot.removeAnnot();
```

```
        System.out.println("结束...");
    }

}
```

1.4.17.4. 接口约束

序号	注释类型	Type
1	高亮（注释）	Highlight
2	下划线（注释）	Underline
3	波浪线（注释）	Squiggly
4	删除线（注释）	Strikeout
5	区域高亮(修订)	RectHighlight
6	批注(修订)	AddRevisions
7	删除(修订)	DelRevisions
8	插入(修订)	InsertLine
9	替换(修订)	Replace
10	移动(修订)	Movedout
11		MovedIn
12	增加间距(修订)	IncreaseSpace
13	缩小间距(修订)	ReduceSpace
14	后移(修订)	MoveBack
15	前移(修订)	MoveForward
16	接续(修订)	ContinueLine
17	对调(修订)	Exchange
18	另起段(修订)	AnotherParagraph
19	切换字体(修订)	SwitchFont
20	签字笔\软笔	Tablet
21	文本框 （注释）	FreeText
22	注释框（注释）	NoteBox

1.4.18. 删除锁定签名

1.4.18.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 invoke 接口删除 ofd 文件锁定签名。转换完成的文件保存至设置的路径下。

1.4.18.2. 接口定义

```
Packet invoke(InvokePair invokePair);
```

参数说明详见下表：

删除锁定签名接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	invokePair	InvokePair<K, V>类型	调用参数对象（存储键值对）	InvokePair.SIGNATURE .build(InvokePair.SIGNAM.RemoveSignature) .add("SignType", Arrays.asList("LockSign"))

返回值说明详见下表：

删除锁定签名接口返回值说明表

序号	返回类型	返回值含义
1	Packet	文件数据包

1.4.18.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.DelLockSign
//功能说明: 删除 ofd 文件锁定签名
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class DelLockSign {

    public static void main(String[] args) {

        //1、定义转换服务 http 代理对象(war 包版)
       HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
       //1、定义代理对象，请求访问 springboot 转换服务
       //AtomAgent ha = new AtomAgent("http://127.0.0.1:8091");
       //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8091");

       Packet packet = new Packet(Const.PackType.COMMON, Target.OFD);

        try {

            //数据包添加原文件
            packet.file(new Common("test", "ofd", new
FileInputStream("D:/addSign2.ofd")));
```

```
            //执行方法删除锁定签名
            packet.invoke(InvokePair.SIGNATURE
                    .build(InvokePair.SIGNAM.RemoveSignature)
                    .add("SignType", Arrays.asList("LockSign")));

            //调用服务，将删除锁定签名的文件输出
            ha.convert(packet, new FileOutputStream("D:/addSign222.ofd"));
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                packet.close();
                ha.close();
                System.out.println("结束.....");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }
}
```

1.4.18.4. 接口约束

仅支持删除锁定的签名，如签名未锁定，不会删除。

1.4.19. WPS 文件转换 OFD 设置修订者信息

1.4.19.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 setArgument 接口设置 wps 转 ofd 设置修订状态。转换完成的文件保存至设置的路径下。

1.4.19.2. 接口定义

```
Common setArgument(Argument argument);
```

参数说明详见下表：

WPS 文件转换 OFD 设置修订者信息接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	argument	Argument 类型	自定义参数	new CustomArgument(new Gson().toJson(map))

返回值说明详见下表：

WPS 文件转换 OFD 设置修订者信息接口返回值说明表

序号	返回类型	返回值含义
1	Common 类型	原文件包装对象

1.4.19.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.Revise
//功能说明: wps 转 ofd 设置修订状态
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
//WPS 文件转换修订者信息
public class Revise {

    public static void main(String[] args) {

       //1、定义转换服务 http 代理对象(war 包版)
       HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
       //1、创建 Http 代理对象，连接请求转换服务地址
        //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
        //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

        //Packet packet = new Packet("wps.template", Target.OFD);
        Packet packet = new Packet(Const.PackType.COMMON, Target.OFD);

        try {

            //自定义参数
            Map<String, Object> map = new HashMap<String, Object>();
            //设置修订状态
            map.put("revise", "Original"); //原始状态
            //map.put("revise", "Original Show Markup");//显示标记的原始状态
            //map.put("revise", "Final Show Markup");//显示标记的最终状态
            //map.put("revise", "Final");//最终状态
            //在批注框中显示修订者信息
            map.put("revise", "MixedRevisions");
```

```
            //自定义参数
            CustomArgument customArgument = new CustomArgument(new
Gson().toJson(map));
            //设置原文件并自定义参数
            Common common = new Common(null, "doc", new FileInputStream("D:\\修
订.doc")).setArgument(customArgument);
            //添加文件
            packet.file(common);

            //packet.pack(new FileOutputStream("D:\\convert\\Final.zip"));
            ha.convert(packet, new FileOutputStream("D:\\convert\\Final5.ofd"));
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                packet.close();
                ha.close();
                System.out.println("......");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

1.4.19.4. 接口约束

参数对应 wps 中修订显示状态：

1.4.20. html 转 OFD 自定义转换参数

1.4.20.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口

（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），通过调用 WebArgument 类中的方法，在 html 网页转换 ofd 时，可自定义参数。可设置的参数有纸张大小、背景、边距、页码等（其中 html 网页转换走 html2ofd 引擎；分页重新排版走 api 引擎，添加页码走 api 引擎）。

1.4.20.2. 接口定义

1、WebArgument 类中方法（设置纸张大小、页边距、背景）
详情见 14.7 WebArgument
2、PageNumber 类中方法（设置纸张页码）
详情见 14.6 PageNumber
3、设置添加自定义参数到源文件

```
Common setArgument(Argument argument);
```

参数说明详见下表：

HTML 文件转 OFD 文件自定义参数接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	argument	Argument 类型	自定义参数	new WebArgument().setPaperMargin(Const.PaperMargin.DEFAULTMARGIN).setPaperBackgrounds(Const.Opinion.NO);//自定义 html 文件转 ofd 文件参数，设置纸张边距，设置是否打印背景色

返回值说明详见下表：

HTML 文件转 OFD 文件自定义参数接口返回值说明表

序号	返回类型	返回值含义
1	Common 类型	原文件包装对象

1.4.20.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.Html_2
//功能说明: html 文件转换为 ofd 文件，可以自定义文件相关参数
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

   //1、定义服务代理对象，请求访问转换服务（war 包版）
   HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
```

```
     //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义方法，执行 html 自定义参数
    private void html() {
            try {
              //2-1、定义数据包对象
              Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
              //2-2、定义通用文件对象，源文件数据，即要操作的文件
          Common common = new Common(null, "html", new FileInputStream("D:\\测试.html"));
              //2-3、定义设置 web 参数的对象 webArgument，需要动态传值时，用到此对象
              WebArgument webArgument = new WebArgument();

              //2-4、调用 webArgument 的方法设置参数
              //1）设置纸张边距， 1、Const.PaperMargin.NOMARGIN：无边距
                                  2、Const.PaperMargin.DEFAULTMARGIN：默认边距
              webArgument.setPaperMargin(Const.PaperMargin.DEFAULTMARGIN);
              //2）设置是否打印背景，1、no-不打印  2、yes-打印
              webArgument.setPaperBackgrounds(Const.Opinion.NO);
              //3）设置纸张大小
              //3-1）定义 自定义参数 对象，定义页边距，顶部边距为 20
              Pair pair_topMargin = new Pair<Const.WebMargin,
Float>(Const.WebMargin.TopMargin, 30f);
              //3-2）定义 自定义参数 对象，定义页边距，底部边距为 25
              Pair pair_bottomMargin = new Pair<Const.WebMargin,
Float>(Const.WebMargin.BottomMargin, 35f);
              //3-3）设置纸张大小
      webArgument.setWebPageSize(Const.PageSize.A4,pair_topMargin,pair_bottomMargin);


              //2-5、源文件数据添加自定义的 web 参数.(纸张大小、是否打印背景、页边距)
              common.setArgument(webArgument);

              //2-6、添加页码
              //1-1)定义偶数页页码对象
              PageNumber pageNum_even = new PageNumber();
              //1-2)定义奇数页页码对象
              PageNumber pageNum_odd = new PageNumber();

              //2-1）设置页码范围：EVEN--偶数，添加偶数页。
              pageNum_even.setPageRange("EVEN");
              //2-2)设置页码范围：ODD--奇数，添加奇数页。
              pageNum_odd.setPageRange("ODD");

              //3-1)设置偶数页页码的参数
```

```
//设置偶数页的页码 x 方法位置: 右侧
pageNum_even.setX("Right");
//设置偶数页的页码 y 方向位置: 下方
pageNum_even.setY("Bottom");
//设置偶数页的边距: 左 0  上 0 右 30 下 30
pageNum_even.setPagePadding("0 0 10 30");
//设置偶数页页码的字体: 宋体
pageNum_even.setFontName("宋体");
//设置偶数页页码的字体大小: 12 float 类型
pageNum_even.setFontSize(12F);
//定义偶数页页码的字体颜色
pageNum_even.setForeColor("#000000");
//定义偶数页页码的显示格式: - 1 -
pageNum_even.setPageNumberFormat("- ${PageNumber} -");
//定义偶数页页码的起始页码: 2
pageNum_even.setStartNumber(2);
//定义偶数页页码的间隔: 2
pageNum_even.setStep(2);

//3-2)设置奇数页页码的参数
//设置奇数页的页码 x 方法位置: 右侧
pageNum_odd.setX("Right");
//设置奇数页的页码 y 方向位置: 下方
pageNum_odd.setY("Bottom");
//设置奇数页的边距: 左 0  上 0 右 30 下 30
pageNum_odd.setPagePadding("0 0 30 30");
//设置奇数页页码的字体: 宋体
pageNum_odd.setFontName("宋体");
//设置奇数页页码的字体大小: 12 float 类型
pageNum_odd.setFontSize(12f);
//定义奇数页页码的字体颜色
pageNum_odd.setForeColor("#000000");
//定义奇数页页码的显示格式: - 1 -
pageNum_odd.setPageNumberFormat("- ${PageNumber} -");
//定义奇数页页码的起始页
pageNum_odd.setStartNumber(1);
//定义奇数页页码的间隔: 2
pageNum_odd.setStep(2);

//4)数据包添加页码
//数据包添加奇数页。参数: 是否删除旧页码, 自定义的页码
packet.addPageNumber(false, pageNum_even);
//数据包添加偶数页.参数: 是否删除旧页码, 自定义的页码
packet.addPageNumber(false, pageNum_odd);
```

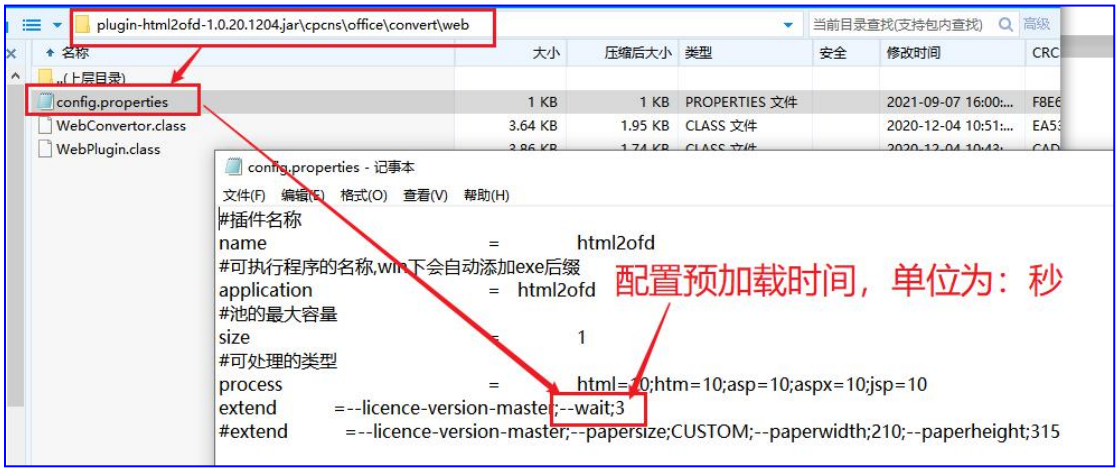
```
            //2-7、数据包添加源文件
            packet.file(common);
            //2-8、请求转换服务，执行 html 自定义参数转换 ofd 方法
            ha.convert(packet, new FileOutputStream("D:\\convert_ofd\\html_2.ofd"));
            packet.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally{
            try {
                //2-8、关闭 http 代理对象
                ha.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
    //3、调用方法
    public static void main(String[] args) {
        //3-1:定义 html 自定义参数对象
        Html_2 html = new Html_2();
        //3-2:执行 html 转换自定义参数方法
        html.html();
     }
```

1.4.20.4. 接口约束

转换服务代码转 html 网址为 OFD 空白页解决方案：转换 html 网址时预加载 http 资源的配置项。（Common common = new Common(null, "html", new URI("https://www.baidu.com"));）

修改 convert-mix-boot-*/rpc-x2y/html2ofd/（针对 spring-boot 版本的转换服务）目录下

或者 build*/plugin/html2ofd（针对 build 版本的转换服务）目录下 plugin-html2ofd-1.0.20.1204.jar 的 config.properties 配置文件，如下图所示：



【备注】：可根据网页资源内容的多少设置相应的预加载时间。

1.4.21. 公文加附件合并转版并添加“附件”标识

1.4.21.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）。通过调用自定义 assembleWatermark 接口，将文件合并转版（公文正文+公文附件合并展示）并在附件文档的第一页添加“附件”标识，多个附件标识为：“附件 1”、“附件 2”...“附件 n”。

1.4.21.2. 接口定义

```
E invoke(String type,
         String module,
         String method,
         Pair<String,String>... pairs);
```

参数说明详见下表：

添加“附件”标识接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	String 类型	调用类型	null
2	module	String 类型	调用模块	"dom"
3	method	String 类型	调用方法名称	"HandOfficialDoc"

4	pairs	Pair<String,String> 类型	自定义参数	new Pair<String, String>("arg", GSON.toJson(map)); //map 为自定义参数集合，具体查看示例

返回值说明详见下表：

添加“附件”标识接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

自定义接口，设置“附件”标识的参数：

```
void assembleWatermark(Map<String, Object> data,
                       boolean isAdd);
```

参数说明详见下表：

公文合并转版设置“附件”标识参数接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	data	Map 类型	添加附件标识的信息参数集合，参数名"Watermark"，参数值为 map 集合，例：data.put("Watermark", map);	map.put("Moveable", true);是否支持移动 （false：移动 true：不移动）
				map.put("FontSize", 16);//标签的字体大小
				map.put("FontName", "方正黑体_GBK");//标签的字体样式
				map.put("ForeColor", "#eca7bd");//标签的字体颜色
				map.put("X", 20);//标签的 X 坐标
				map.put("Y", 20);//标签的 Y 坐标起始位置
2	isAdd	boolean 类型	是否添加附件标识	true/false

返回值说明详见下表：

公文合并转版设置“附件”标识参数接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.4.21.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.AssembleWatermark_5
//功能说明：将文件与附件合并转换为 ofd 文件，同时附件第一页添加“附件”标识
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
```

```
//1-1、定义 http 代理对象，请求访问转换服务(war 包版)
static HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
//1-1、定义代理对象，请求访问转换服务(Springboot 版)
//AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
//HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

//1-2、定义静态常量 WATERMARK 和 PAGENUMBER
private static final String WATERMARK = "Watermark", PAGENUMBER = "PageNumber";
//1-3、定义静态常量，解析 json
private static final Gson GSON = new Gson();

//2、定义静态方法 assembleWatermark()，添加“附件 n”水印标签
 static void assembleWatermark(Map<String, Object> data,boolean isAdd) {
         //2-1、判断是否添加“附件 n”标识，不添加直接跳出
        if(!isAdd) {
             return;
         }
        //2-2、定义 map 集合对象，存储“附件 n”标识的相关参数
         Map<String, Object> wtm = new HashMap<String, Object>();
         //2-3、定义“附件 n”标识的相关参数
         //是否支持移动 （false：移动  true：不移动）
         wtm.put("Moveable", true);
         //标签的字体大小
         wtm.put("FontSize", 16);
         //标签的字体样式
         wtm.put("FontName", "方正黑体_GBK");
         //标签的字体颜色
         wtm.put("ForeColor", "#eca7bd");
         //标签的位置坐标 x
         wtm.put("X", 20);
         //标签的位置坐标 y
         wtm.put("Y", 20);
         //2-3、追加定义的水印数据
         data.put(WATERMARK, wtm);
   }
   //3、方法调用
   public static void main(String[] args) {
       try {
             //3-1、定义数据包
             Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
             //3-2、设置数据包文件不合并
             packet.setMerge(false);
              //URLEncoder.encode(arg0)
```

```
                //3-3、添加公文文件
            packet.file(new Common(null, "pdf", new FileInputStream("D:\\a.pdf")));
                //3-4、添加附件
         packet.file(new Common(null, "wps", new URI("file:///D:/公文附件001.wps")));
        packet.file(new Common(null, "doc", new URI("file:///D:/公文附件002.doc")));

                //3-5、定义map集合，存储“附件n”标识相关参数
                Map<String, Object> map = new HashMap<String, Object>();
                //添加"附件"标签，true-添加附件标签  false-不添加附件标签
                assembleWatermark(map, true);//
                //是否骑马钉，true-骑马钉  false-不骑马钉
                map.put("SaddleStitch", true);//
                //是否是函文件，true-是函文件   false-不是函文件
                map.put("OfficeLetter", false);
                //空白页是否添加页码，true-添加页码   false-不添加页码
                map.put("BlankPageNumber", true);
                //是否包含版记，true-包含版记  false-不包含版记
                map.put("VersionNote", false);
                //3-6、数据包调用HandOfficialDoc方法处理添加标识
                packet.invoke(null, "dom", "HandOfficialDoc", new Pair<String,
String>("arg", GSON.toJson(map)));
                //packet.pack(new FileOutputStream("D://4.zip"));//文件打包
                //3-7、请求文件转换服务，转换并输出ofd文件
                ha.convert(packet,new
FileOutputStream("D:\\convert_ofd\\assembleWatermark_5.ofd"));
        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }finally {
            try {
                //3-8、关闭ha
                ha.close();
                System.out.println("转换结束---");
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    }
```

1.4.21.4. 接口约束

无。

1.4.22. 公文加附件合并转版并添加“页码”标识

1.4.22.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）。通过调用自定义 assemblePageNumber 接口对合并的文件（公文正文+公文附件合并展示）的附件中添加页码，添加页码顺序为：从附件文件的第一页衔接正文文件最后一页添加页码，横板页码自动旋转+-90°。

1.4.22.2. 接口定义

```
E invoke(String type,
         String module,
         String method,
         Pair<String,String>... pairs);
```

参数说明详见下表：

添加“页码”标识接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	String 类型	调用类型	null
2	module	String 类型	调用模块	"dom"
3	method	String 类型	调用方法名称	"HandOfficialDoc"
4	pairs	Pair<String,String> 类型	自定义参数	new Pair<String, String>("arg", GSON.toJson(map)); //map 为自定义参数集合，具体查看示例

返回值说明详见下表：

添加“页码”标识接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

自定义方法，设置“页码”标识参数：

```
void assemblePageNumber( Map<String, Object> data,
                         boolean isAdd);
```

参数说明详见下表：

公文合并转版设置“页码”标识参数接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	data	Map 类型	添加页码标识的信息参数集合，参数名"PageNumber"，参数值为map集合，例：data.put("PageNumber", list);List集合存储map,奇数页和偶数页	map.put("Moveable", true);是否支持移动（false：移动 true：不移动）
				map.put("PageRange", "ODD");//奇数页"ODD"、偶数页"EVEN"
				map.put("PagePadding", "25 275 30 15");//页码位置坐标
				map.put("PageNumberFormat", "— ${PageNumber} —");//格式化页码
				map.put("ForeColor", "#000000");//页码颜色
				map.put("Orientation", "Portrait");//页码显示基准界面 Portrait Landscape
				map.put("AutoAdaptOrientation", "Clockwise");//页码旋转方向，顺时针 Clockwise，逆时针?
				map.put("FontName", "宋体");//页码字体
				map.put("FontSize", 14);//页码字号
2	isAdd	boolean 类型	是否添加附件标识	true/false

返回值说明详见下表：

公文合并转版设置“页码”标识参数接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.4.22.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.AssemblePageNumber_6
//功能说明：将文件与附件合并转换为 ofd 文件，同时附件添加“页码”标识
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

    //1-1、定义 http 代理对象，请求访问转换服务(war 包版)
     static HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //static AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //static HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");
```

```
//1-2、定义静态常量 WATERMARK 和 PAGENUMBER
private static final String WATERMARK = "Watermark", PAGENUMBER = "PageNumber";
//1-3、定义静态常量，解析 json
private static final Gson GSON = new Gson();

//2、定义静态方法 assembleWatermark()，添加“页码”水印标签
static void assemblePageNumber(Map<String, Object> data,boolean isAdd) {
    //2-1、判断是否添加“页码”标识，不添加直接跳出
    if(!isAdd) {
        return;
    }
    //2-2、添加奇数页页码标识，定义“页码”标识参数
    List<Map<String, Object>> lpnm = new ArrayList<Map<String, Object>>();
    //定义 map 集合，存储“页码”标识参数信息
    Map<String, Object> map_pageNum = new HashMap<String, Object>();
    //是否支持移动
    map_pageNum.put("Moveable", true);
    //奇数页 ODD
    map_pageNum.put("PageRange", "ODD");
    //页码的具体位置坐标
    map_pageNum.put("PagePadding", "175 275 30 265");
    //页码格式
    map_pageNum.put("PageNumberFormat", "—— ${PageNumber} ——");
    //页码颜色
    map_pageNum.put("ForeColor", "#000000");
    //页码显示基准界面 Portrait Landscape
    map_pageNum.put("Orientation", "Portrait");
    //页码旋转方向
    map_pageNum.put("AutoAdaptOrientation", "Clockwise");
    //页码字体
    map_pageNum.put("FontName", "宋体");
    //页码字号
    map_pageNum.put("FontSize", 14);
   //添加参数集合数据到 list
    lpnm.add(map_pageNum);

    //偶数页页码
    map_pageNum = new HashMap<String, Object>();
    //是否支持移动
    map_pageNum.put("Moveable", true);
    //偶数页 EVEN
    map_pageNum.put("PageRange", "EVEN");
    //页码的具体位置坐标
```

```
        map_pageNum.put("PagePadding", "25 275 30 15");
        //页码格式
        map_pageNum.put("PageNumberFormat", "--- ${PageNumber} ---");
        //页码颜色
        map_pageNum.put("ForeColor", "#000000");
        //页码显示基准界面 Portrait Landscape
        map_pageNum.put("Orientation", "Portrait");
        //页码旋转方向
        map_pageNum.put("AutoAdaptOrientation", "Clockwise");
        //页码字体
        map_pageNum.put("FontName", "宋体");
        //页码字号
        map_pageNum.put("FontSize", 14);
        //添加参数集合数据到 list
        lpnm.add(map_pageNum);
        //定义“页码”标识水印参数集合
       data.put(PAGENUMBER, lpnm);
    }
   //3、方法调用
   public static void main(String[] args){
          try {
                //3-1、定义数据包对象，定义源文件类型，目标文件类型。
                Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
                //3-2、设置文件不合并
                packet.setMerge(false);
                 //URLEncoder.encode(arg0)
                 //3-3、添加公文文件
                 packet.file(new Common(null, "wps", new FileInputStream("D:\\公文正
文.wps")));

                 //3-4、添加附件
                 packet.file(new Common(null, "wps", new URI("file:///D:\\公文合并
001.wps")));
                 packet.file(new Common(null, "doc", new URI("file:///D:\\公文附件
002.doc")));
                 //3-5、定义附件的参数
                 Map<String, Object> map = new HashMap<String, Object>();
                 //添加"页码"标签,true-添加页码标签  false-不添加页码标签
                 assemblePageNumber(map, true);
                 //是否骑马钉,true-骑马钉  false-不骑马钉
                 map.put("SaddleStitch", false);
                 //是否是函文件,true-是函文件   false-不是函文件
                 map.put("OfficeLetter", false);
                 //空白页是否添加页码,true-添加页码   false-不添加页码
```

```
                map.put("BlankPageNumber", false);
                //是否包含版记,true-包含版记  false-不包含版记
                map.put("VersionNote", true);
                //3-6、数据包调用 officedoc 处理器处理添加标识
                packet.invoke(null, "dom", "HandOfficialDoc", new Pair<String,
String>("arg", GSON.toJson(map)));

//3-7、请求转换服务，执行文件转换
                ha.convert(packet,
new FileOutputStream("D:\\convert_ofd\\assemblePageNumber_6.ofd"));
                packet.close();
       } catch (Exception e) {
            e.printStackTrace();
        }finally {
            try {
                //3-8、关闭 ha
                ha.close();
                System.out.println("转换结束---");
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
   }
```

1.4.22.4. 接口约束

无。

1.4.23. OFD 文件转为图片

1.4.23.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）。通过调用 setArgument 方法设置 ofd 文件转换后图片的格式。

1.4.23.2. 接口定义

```
Common setArgument(Argument argument);
```

参数说明详见下表：

OFD 转图片设置图片格式接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	argument	Argument 类型	自定义参数	new ImageArgument().setTargetFormat(ImageArgument.Type.jpg); //jpg 格式或 png 格式

返回值说明详见下表：

OFD 转图片设置图片格式接口返回值说明表

序号	返回类型	返回值含义
1	Common 类型	原文件包装对象

1.4.23.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.OfdConvertImage_7
//功能说明：将 ofd 文件转换为图片格式文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

    //1、定义 http 代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

     //2、定义方法，执行文件拆分转换操作
     private void ofdConvertImage(){
         try {
             //2-1 转为 TIFF 文件示例
              Packet packet1 = new Packet(Const.PackType.IMAGE, Target.TIFF);
              Common common1 = new Common(null, "ofd",  -1  , new
FileInputStream("D:\\convert_ofd\\测试.ofd"));
             packet1.file(common1);
             ha.convert(packet1, new
FileOutputStream("D:\\convert_ofd\\OfdConvertImage_71.tiff"));



             //2-2 转为 bmp 图片文件示例
             Packet packet2 = new Packet(Const.PackType.IMAGE, Target.IMAGE);
             Common common2 = new Common(null, "ofd",  -1  , new
FileInputStream("D:\\convert_ofd\\测试.ofd"));
             packet2.file(common2);
             ha.convert(packet2, new
```

```
FileOutputStream("D:\\convert_ofd\\OfdConvertImage_72.zip"));



            //2-3 转为 jpg 图片文件示例
            Packet packet3 = new Packet(Const.PackType.IMAGE, Target.IMAGE);
            Common common3 = new Common(null, "ofd",  -1  , new
FileInputStream("D:\\convert_ofd\\测试.ofd"));
            common3.setArgument(new
ImageArgument().setTargetFormat(ImageArgument.Type.jpg));
            packet3.file(common3);
            ha.convert(packet3, new
FileOutputStream("D:\\convert_ofd\\OfdConvertImage_73.zip"));

            //2-4 转为 png 图片文件示例
            Packet packet4 = new Packet(Const.PackType.IMAGE, Target.IMAGE);
            Common common4 = new Common(null, "ofd",  -1  , new
FileInputStream("D:\\convert_ofd\\测试.ofd"));
            common4.setArgument(new
ImageArgument().setTargetFormat(ImageArgument.Type.png));
            packet4.file(common4);
            ha.convert(packet4, new
FileOutputStream("D:\\convert_ofd\\OfdConvertImage_74.zip"));
            packet4.close();
        } catch (Exception e) {
                    e.printStackTrace();
        } finally {
                    try {
                        //2-5、关闭 ha
                        ha.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
        }
    }

    //3、方法调用
    public static void main(String[] args){
            //3-1:定义 ofd 转图片对象
            OfdConvertImage_7 ofdConvertImage = new OfdConvertImage_7();
            //3-2:执行 ofd 转图片方法
            ofdConvertImage.ofdConvertImage();
   }
```

1.4.23.4. 接口约束

无。

1.4.24. OFD 转图片支持自定义图片参数

1.4.24.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）。通过 packet.invoke()方法调用"MergeImage"添加图片自定义参数。将 ofd 文件转换为图片格式。

1.4.24.2. 接口定义

```
E invoke(String type,
         String module,
         String method,
         Pair<String,String>... pairs);
```

参数说明详见下表：

OFD 转图片自定义图片参数接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	String 类型	调用类型	null
2	module	String 类型	调用模块	null
3	method	String 类型	调用方法名称	"MergeImage" //合并图片

4	pairs	Pair 类型	自定义参数	//定义图片类型
				new Pair<String, String>("type", "png");
				//定义图片像素
				new Pair<String, String>("dpi", "96");
				//定义图片步长，每 3 页生成一个图片
				new Pair<String, String>("step", "3");

返回值说明详见下表：

OFD 转图片自定义图片参数接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

1.4.24.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.OfdConvertImage_7_2
//功能说明: 将 ofd 文件转换为图片格式文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

    //1、定义转换服务 http 代理对象(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //2、定义方法，执行文件拆分转换操作
     private void ofdConvertImage(){
         try {
              //2-1、ofd 转图片，按页分割
              //定义数据包拆分 ofd
              Packet packet_split= new Packet("merge.ofd2image", Target.MERGE);
              //数据包添加原文件
     packet_split.file(new Common(null, "ofd", new FileInputStream("D:\\测试.ofd")));
              //定义生成图片的类型（支持 png、jpg、bmp）
              Pair pair_imgType = new Pair<String, String>("type", "png");
              //定义生成图片的 dpi
              Pair pair_imgDpi = new Pair<String, String>("dpi", "96");
              //定义生成每张图片的页数,每 3 页生成一张图片
              Pair pair_imgPage = new Pair<String, String>("step", "3");
              //传入自定义参数调用合并图片接口
packet_split.invoke(null, null, "MergeImage",pair_imgType,pair_imgDpi,pair_imgPage);
              //通过代理请求转换服务，将 ofd 转换为图片，得到 zip 格式的图片集合
              ha.convert(packet_split, new
FileOutputStream("D:\\convert_ofd\\OfdConvertImage_7_2-1.zip"));



              //2-2、ofd 转图片，不分割
              //定义数据包
              Packet packet_unSplit= new Packet("merge.ofd2image", Target.IMAGE);
              //数据包添加原文件
     packet_unSplit.file(new Common(null, "ofd", new FileInputStream("D:\\测试.ofd")));
              //通过代理请求转换服务，将 ofd 转换为图片，得到 zip 格式的图片集合
              ha.convert(packet_unSplit, new
FileOutputStream("D:\\convert_ofd\\OfdConvertImage_7_2-3.png"));
             packet_split.close();
            packet_unSplit.close();
```

```
        } catch (Exception e) {
                    e.printStackTrace();
        } finally {
                    try {
                        //关闭 ha
                        ha.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
        }
    }

   //3、方法调用
   public static void main(String[] args){
            //3-1:定义 ofd 转图片对象
             OfdConvertImage_7_2 ofdConvertImage = new OfdConvertImage_7_2();
             //3-2:执行 ofd 转图片方法
             ofdConvertImage.ofdConvertImage();
   }
```

1.4.24.4. 接口约束

无。

1.4.25. OFD 文件自定义页码转图片

1.4.25.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）。通过 packet.invoke()方法调用"MergeImage"设置指定页码转图片。将 ofd 文件转换为图片格式。

1.4.25.2. 接口定义

```
E invoke(String type,
         String module,
         String method,
         Pair<String,String>... pairs);
```

参数说明详见下表：

OFD 指定页码生成图片接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	String 类型	调用类型	null
2	module	String 类型	调用模块	null
3	method	String 类型	调用方法名称	"MergeImage" //合并图片
4	pairs	Pair 类型	自定义参数	//定义要生成图片的页码，页码从 0 开始，需要-1，示例为 3，4，6 页生成一张图 new Pair<String, String>("retain", "2,3,5");

返回值说明详见下表：

OFD 指定页码生成图片接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

1.4.25.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.OfdConvertImage_7_3
//功能说明：将 ofd 文件转换为图片格式文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

    //1、定义转换服务 http 代理对象（war 包版）
    HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

     //2、定义方法，执行文件拆分转换操作
     private void ofdConvertImage(){
         try {
             //2-1、定义数据包拆分 ofd,生成 tif 类型
             Packet packet_tiff= new Packet("merge.ofd2image", Target.TIFF);
             //定义数据包拆分 ofd,生成 jpg,png,bmp 类型
             Packet packet_img= new Packet("merge.ofd2image", Target.IMAGE);

             //2-2、数据包添加原文件
             packet_tiff.file(new Common(null, "ofd", new FileInputStream("D:\\测
试.ofd")));
             packet_img.file(new Common(null, "ofd", new FileInputStream("D:\\测
试.ofd")));
```

```
            //2-3、自定义转换后图片参数
            //指定页数（3，4，6 页）生成一个图片（长图）
            Pair pair_imgSplit = new Pair<String, String>("retain", "2,3,5");
            //定义生成图片的 dpi
            Pair pair_imgDpi = new Pair<String, String>("dpi", "96");

            //2-4、传入自定义参数调用合并图片接口
            packet_tiff.invoke(null, null, "MergeImage",pair_imgSplit,pair_imgDpi);
            packet_img.invoke(null, null, "MergeImage",pair_imgSplit,pair_imgDpi);

            //2-5、通过代理请求转换服务，将 ofd 转换为图片
            ha.convert(packet_tiff, new
FileOutputStream("D:\\convert_ofd\\OfdConvertImage_7_3.tiff"));
            ha.convert(packet_img, new
FileOutputStream("D:\\convert_ofd\\OfdConvertImage_7_3.bmp"));

         } catch (Exception e) {
                    e.printStackTrace();
        } finally {
                    try {
                        //关闭 ha
                        ha.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
        }
    }

   //3、方法调用
   public static void main(String[] args){
          //3-1:定义 ofd 转图片对象
            OfdConvertImage_7_3 ofdConvertImage = new OfdConvertImage_7_3();
            //3-2:执行 ofd 转图片方法
            ofdConvertImage.ofdConvertImage();
  }
```

1.4.25.4. 接口约束

无。

1.4.26. 关键字盖章

1.4.26.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），通过 packet.invoke()方法调用"Sign"在待转换的文件中进行查找关键字盖章。

1.4.26.2. 接口定义

```
E invoke(String type,
         String module,
         String method,
         Pair<String,String>... pairs);
```

参数说明详见下表：

OFD 调用查找关键字盖章接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	String 类型	调用类型	null
2	module	String 类型	调用模块	"signature"
3	method	String 类型	调用方法名称	"Sign" //签章
4	pairs	Pair 类型	自定义参数	new Pair<String, String>("arg", GSON.toJson(sealInfo)); //sealInfo 为 map 集合，是签章的自定义数据。

返回值说明详见下表：

OFD 调用查找关键字盖章接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

1.4.26.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.KeyWordSeal_8
//功能说明: 根据 ofd 文件关键字签章
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

  //1、定义转换服务 http 代理对象（war 包版）
```

```
  HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
  //1、定义代理对象，请求访问转换服务(Springboot 版)
  //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
  //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


//定义关键字签章方法
private static void  keyWordSeal() {
       try {

           Packet  packet = new Packet(Const.PackType.COMMON, Target.OFD);
           Common common = new Common("","docx",new FileInputStream("D:\\测试文
件.docx"));
           packet.file(common);

           //定义盖章信息
           Map<String,Object> sealInfo = new HashMap<String, Object>();
           //签章名称
           sealInfo.put("OesName", "Suwell_SDK");
            //签章 ID
            sealInfo.put("SealId", "2d7c5856-c55f-47c1-8b3c-466bddc60f46");
            //签章密码
             sealInfo.put("Password", "123456");
             //是否开启关键字盖章
             sealInfo.put("KeyWordSeal", true);
             //定义关键字信息
              Map<String,Object> keyWordInfo = new HashMap<String, Object>();
             //关键字
             keyWordInfo.put("Text", "盖章区域");
             //第几个关键字盖章: 从 0 开始
             keyWordInfo.put("Index", 0);
             //盖章在第一个关键字信息
             sealInfo.put("SearchParam", keyWordInfo);

              //调用签章方法
             packet.invoke(null, "signature", "Sign",new Pair<String, String>("arg",
GSON.toJson(sealInfo)));
             //通过 http 代理请求转换服务，生成带附件并添加签章的 OFD 文件
             ha.convert(packet, new
FileOutputStream("D:\\convert_ofd\\keyWordSeal_8.ofd"));
             packet.close();
       } catch (Exception e) {
                      e.printStackTrace();
        } finally {
```

```
                    try {
                        //关闭 ha
                        ha.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
        }
    }

    //3、方法调用
    public static void main(String[] args){
            //3-1:定义关键字盖章对象
             KeyWordSeal_8  keyWordSeal = new KeyWordSeal_8();
            //3-2:执行 ofd 文件关键字盖章方法
            keyWordSeal.keyWordSeal();
    }
```

1.4.26.4. 接口约束

无。

1.4.27. 关键字遮盖

1.4.27.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），通过 packet.invoke()方法在待转换的文件中进行查找关键字遮盖。

1.4.27.2. 接口定义

```
Packet invoke(InvokePair invokePair);
```

参数说明详见下表：

OFD 调用查找关键字遮盖接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	invokePair	InvokePair 类型	调用参数对象（存储键值对）	InvokePair.SAFEMARK.build(InvokePair.SAM.KeywordsProtection)

返回值说明详见下表：

OFD 调用查找关键字遮盖接口返回值说明表

序号	返回类型	返回值含义
1	Packet	Packet 数据包对象

1.4.27.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.api.KeyWrodCover
//功能说明: 根据 ofd 文件关键字遮盖
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
//关键字遮盖
public class KeyWrodCover {

	public static void main(String[] args) {

		//1、定义转换服务 http 代理对象（war 包版）
		HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
		//1、定义代理对象，请求访问 springboot 转换服务
		//HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");
		//AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");

		//定义数据包
		Packet packet = new Packet(Const.PackType.COMMON, Target.OFD);


		try {

			//添加原文件
			packet.file(new Common("test", "ofd", new
FileInputStream("D:\\00004.ofd")));
			//调用方法执行关键字遮盖
			packet.invoke(InvokePair.SAFEMARK
					.build(InvokePair.SAM.KeywordsProtection) //关键字遮盖
					//关键字 必要参数 可以传多个关键字
				.add("Keywords", Arrays.asList("数科网维","9 月 29 日","公告"))
					//用户 必要参数 随便填  可以传多个用户
					.add("Users", Arrays.asList("1", "2"))
					//关键字遮盖页码  传多个页码会搜索多页的关键字  如果要搜索所有页
就传空字符串
					.add("Range", "1,2")
					//加密方式  不传 为添加预遮罩注释  暂时这里为默认 base64
					//.add("EncryptType", "Base64")
					//遮盖样式
```

```
                        .add("Appearance", InvokePair.build()
                                .add("Fill", true)//是否填充
                                .add("FillColor", "#FF44AA") //填充颜色
                                .add("Stroke", true)//是否描边
                                .add("StrokeColor", "#0044BB")//描边颜色
                                .add("Transparency", 0)//注释透明度  0-100    0 是不
透明    100 是全透明
                        ));


                //调用转换服务执行关键字遮盖
                ha.convert(packet, new FileOutputStream("D:\\convert\\cs3.ofd"));
                //打包输出文件
                //packet.pack(new FileOutputStream("D:/ofd/packet-50.zip"));


            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                try {
                    packet.close();
                    ha.close();
                    System.out.println("结束......");
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

        }
}
```

1.4.27.4. 接口约束

无。

1.4.28. 关键字遮盖 2

1.4.28.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），通过

invoke 方法在待转换的文件中进行查找关键字遮盖。

1.4.28.2. 接口定义

```
E invoke(String type,
         String module,
         String method,
         Pair... pairs);
```

参数说明详见下表：

OFD 调用查找关键字遮盖接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	String 类型	调用类型	null
2	module	String 类型	调用模块	"safemark"
3	method	String 类型	调用方法名称	"KeywordsProtection" //关键字遮盖
4	pairs	Pair 类型	自定义参数	new Pair<String, String>("arg", GSON.toJson(keywordInfo)); //keywordInfo 为 map 集合，存储关键字及页码范围参数

返回值说明详见下表：

OFD 调用查找关键字遮盖接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

1.4.28.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.KeyWrodCover2
//功能说明：根据 ofd 文件关键字遮盖
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
//关键字遮盖 2
public class KeyWrodCover2 {

       private static final Gson GSON = new Gson();

      public static void main(String[] args) {

            //1、定义转换服务 http 代理对象（war 包版）
```

```
            HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
            //1、定义代理对象，请求访问 springboot 转换服务
            HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");
            //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");

            //定义数据包
            Packet packet = new Packet(Const.PackType.COMMON, Target.OFD);


            try {

                //添加原文件
                packet.file(new Common(null, "ofd", new
FileInputStream("D:\\13-out.ofd")));

                //定义关键字及所属范围 1
                Map<String,Object> map = new HashMap<String,Object>();
                //要遮盖的关键字
                map.put("Keywords", new String[] {"关键字 1","关键字 2"});
                //要遮盖的关键字范围（第一页），传空为所有页
                map.put("Range", "1");


                //定义关键字及所属范围 2
                Map<String,Object> map2 = new HashMap<String,Object>();
                //要遮盖的关键字
                map2.put("Keywords", new String[] {"关键字 1","关键字 2"});
                //要遮盖的关键字范围:第一页的"关键字 1" 和 第二页的"关键字 2"进行遮盖
                map2.put("Range", "1,2");


                packet.invoke(null,"safemark", "KeywordsProtection", new
Pair<String,String>("args",GSON.toJson(map2)));
                //调用转换服务执行关键字遮盖
                ha.convert(packet, new
FileOutputStream("D:\\convert\\keyWord1.ofd"));
                //打包输出文件
                //packet.pack(new FileOutputStream("D:/ofd/packet.zip"));


            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                try {
```

```
                    packet.close();
                    ha.close();
                    System.out.println("结束......");
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

        }
}
```

1.4.28.4. 接口约束

无。

1.4.29. 签章脱密

1.4.29.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 agent,packet 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）,通过 packet.invoke()方法调用"DecryptSeal"将包含签章的 ofd 文件进行脱密处理，脱密后的 ofd 文件签章变为签章图片，不可进行验证操作。

1.4.29.2. 接口定义

```
E invoke(String type,
         String module,
         String method,
         Pair<String,String>... pairs);
```

参数说明详见下表：

OFD 调用方法进行签章脱密接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	String 类型	调用类型	null
2	module	String 类型	调用模块	"signature"
3	method	String 类型	调用方法名称	"DecryptSeal" //脱密章

4	pairs	Pair 类型	自定义参数	new Pair<String, String>("arg", GSON.toJson(sealInfo)); //sealInfo 为 map 集合，存储签章脱密参数

返回值说明详见下表：

OFD 调用方法进行签章脱密接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

1.4.29.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.DecryptSeal_9
//功能说明：将 ofd 文件签章脱密
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

//1、定义转换服务 http 代理对象（war 包版）
HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
//1、定义代理对象，请求访问转换服务(Springboot 版)
//AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
//HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");



//定义签章脱密方法
private static void  decryptSeal() {
       try {
          //2-1、定义数据包
         Packet  packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
         //2-2、数据包添加原文件
          packet.file(new Common(null, "ofd", new FileInputStream("D:\\测试.ofd")));
          //2-3、定义 map 集合，存储签章自定义参数
          Map<String, Object> sealInfo = new HashMap<String, Object>();
         //脱密后签章图片是否模糊显示（false--不模糊显示  true--模糊显示）
          sealInfo.put("SealAtomised", false);
          //2-4、调用"DecryptSeal"方法执行签章脱密
         packet.invoke(null, "signature", "DecryptSeal",new Pair<String, String>("arg",
GSON.toJson(sealInfo)));
          //2-5、通过代理对象，请求转换服务
         ha.convert(packet, new FileOutputStream("D:\\convert_ofd\\脱密.ofd"));
     } catch (Exception e) {
                    e.printStackTrace();
```

```
    } finally {
                    try {
                        //关闭 ha
                        ha.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
        }
    }

   //3、方法调用
   public static void main(String[] args){
            //3-1:定义 ofd 文件签章脱密对象
            DecryptSeal_9 decryptSeal= new DecryptSeal_9();
            //3-2:执行 ofd 文件签章脱密方法
            decryptSeal.decryptSeal();
  }
```

1.4.29.4. 接口约束

无。

1.4.30. 文件脱密

1.4.30.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent（Springboot 版对应 AtomAgent/HTTPAgent,war 包版对应 HTTPAgent),packet 接口,调用 decryptSeal 接口实现 OFD 文件签章，红头，图片脱密。

1.4.30.2. 接口定义

```
Packet decryptSeal(boolean decryptSeal,
                   boolean decryptContent,
                   boolean decryptImage);
```

参数说明详见下表：

签章脱密接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	decryptSeal	boolean	是否把 章 黑化	true :脱密 false :不脱密

序号	参数名称	参数类型	参数含义	内容举例
2	decryptContent	boolean	是否把 红头 黑化	true :脱密
				false :不脱密
3	decryptImage	boolean	是否把 图片 黑化	true :脱密
				false :不脱密

返回值说明详见下表：

签章脱密接口返回值说明表

序号	返回类型	返回值含义
1	Packet 类型	packet 对象，脱密后的文件数据包

1.4.30.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.packet.DecryptSeal
//功能说明: OFD 文件脱密（签章，红头，图片）
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
public class DecryptSeal {

    //1、定义 http 代理对象，连接请求转换服务（war 包版）
    HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
   //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


    //2、定义方法实现 签章脱密
    private void decryptSeal() {
        try {

        //2-1、定义数据包对象
        Packet packet = new Packet(Const.PackType.COMMON, Const.Target.OFD);
        //2-2、定义通用文件对象，源文件，即待脱密的文件
        Common common = new Common("ofd_1", "ofd", new FileInputStream("D:\\55.ofd"));
        //2-3、数据包添加源文件
        packet.file(common);
        //2-4、调用脱密方法（章是否脱密，红头是否脱密，图片是否脱密）
        packet.decryptSeal(true, true, true);
        //2-5、请求转换服务
        ha.convert(packet, new FileOutputStream("D:\\convert\\decryptSeal_cs.ofd"));
        packet.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                //2-6、关闭资源
                ha.close();
            } catch (IOException e) {
```

```
                e.printStackTrace();
            }
        }
    }

    //3、方法调用用
    public static void main(String[] args) {

        DecryptSeal ofdDecryptSeal = new DecryptSeal();
        ofdDecryptSeal.decryptSeal();
       System.out.println("转换结束-------------------");

    }

}
```

1.4.30.4. 接口约束

build75-20210331 以后版本可用

1.4.31. 调 OES 盖章

1.4.31.1. 接口描述

实现该功能需引用 agent*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用第三方签章 OES 库进行后台盖章。转换完成的文件保存至设置的路径下。

1.4.31.2. 接口定义

```
E seal(String provider, String oesClass, SealInfo info);
```

参数说明详见下表：

添加盖章接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	provider	String 类型	安全服务提供者标示（不能为空）	"Suwell_SDK"
2	oesClass	String 类型	服务提供者实现类的名称（可以为空）	Null
3	info	SealInfo 自定义类型	印章的图像信息（可能为空）	SealInfo info=new SealInfo(SealInfo.TYPE_ALL,

序号	参数名称	参数类型	参数含义	内容举例
			具体见 1.6.4	"wstoes", 105, 105, 30, 30);

返回值说明详见下表：

添加盖章接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

```
//已过时
Packet seal(CA ca, InputStream esl, SealInfo info);
```

参数说明详见下表：

添加盖章接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	Ca	CA 类型	封装的 CA 证书(不能为空)	new CA("MD5", "SHA1withRSA", new FileInputStream(new File("C:\\Program Files (x86)\\Suwell\\Reader_Pro\\plugins\\seal\\Suwell\\ROOTCA.pfx")), "123456", new FileInputStream(new File("C:\\Program Files (x86)\\Suwell\\Reader_Pro\\plugins\\seal\\Suwell\\ROOTCA.cer")));
2	esl	InputStream 输出流	标准签章输入流（可以为空）	new FileInputStream(new File("C:\Program Files (x86)\Suwell\Reader_Pro\plugins\seal\Suwell\OfdSeal.esl"))
3	info	SealInfo 自定义类型	印章的图像信息（可以为空）	SealInfo info=new SealInfo(SealInfo.TYPE_ALL, "wstoes", 105, 105, 30, 30);

返回值说明详见下表：

添加盖章接口返回值说明表

序号	返回类型	返回值含义
1	Packet	数据包对象

1.4.31.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.packet.AddSeal_C
//功能说明: 调用 OES 库实现的接口完成服务端盖章
```

```
    //1、定义代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8080/convert-issuer");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

Packet w = new Packet("Native.parser", Const.Target.OFD);
        try {
             SealInfo info=new
SealInfo(SealInfo.NativeType.Normal,"2d7c5856-c55f-47c1-8b3c-466bddc60f46", 10, 10,
45, 45, "123456", 0);
             //添加签章拓展参数(json 格式字符串)
      info.getExtend().put(Const.SignatureExtendKey.Param_GetCertList, "json 扩展");
             //设置签章用户名
             //info.setUserID("username");
           w.file(new Common("签章测试", "ofd", new FileInputStream(new
File("D:/convert/tar/doc_1.ofd"))));

             //参数 1: 签章提供者名
             //参数 2: 调用 C 的签章时，此项可设为 null
            w.seal("Suwell_SDK", null,info);
            ha.convert(w,new FileOutputStream("D:/convert/tar/doc_2.ofd"));

        } catch (PackException | IOException | ConvertException e) {
            e.printStackTrace();
        }
    }finally{
                    try {

                         w.close();
                         ha.close();
                     } catch (IOException e) {
                         e.printStackTrace();
                     }

    }
```

1.4.31.4. 接口约束

添加扩展参数SignatureExtendKey类型说明：

Param_GetCertList	//扩展参数适用于 OES-GetCertList
Param_GetDigestMethod	//扩展参数适用于 OES-GetDigestMethod

Param_GetSeal	//扩展参数适用于 OES-GetSealI
Param_GetSealImage	//扩展参数适用于 OES-GetSealImage
Param_GetSealList	//扩展参数适用于 OES-GetSealList
Param_GetSignMethod	//扩展参数适用于 OES-GetSignMethod
Param_RawSign	//扩展参数适用于 OES-RawSign
Param_Sign	//扩展参数适用于 OES-Sign
Param_TimeStamp	//扩展参数适用于 OES-TimeStamp

1.4.32. URI 获取源文转换

1.4.32.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet,agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 file 接口进行转换，转换完成的文件保存至设置的路径下。可用于实现本地网页文件转换及 ftp 上的源文件转换。

1.4.32.2. 接口定义

```
//添加一个文件
packet.file(Common common);
Common common= new Common(String title, String format, URI uri);
```

参数说明详见下表：

单文件转换实现本地网页转为 OFD 文件接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	title	String 类型	文件标题	示例
2	format	String 类型	文件类型	html 或 doc 等
3	uri	URI 类型	待转换源文件位置	new URI("file:///D:/ofd/预览.html") 或 URI uri=new URI("ftp://administrator:cs123456@172.16.14.43:22\\test\\111.doc");

返回值说明详见下表：

单文转换本地网页转为 OFD 文件接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.4.32.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.packet.html2OfdTest
//功能说明：转换 ftp 上的文件和本地的网页文件


    //1、定义代理对象，连接及调用转换服务（war 包版）
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

//转换本地的网页文件
private void localHtml() {
        try {
            Packet packet = new Packet(PackType.COMMON,Target.OFD);
            URI uri = new URI("C://Users//admin//Desktop//2.html");
            packet.file(new Common("1","html",uri));
            ha.convert(packet,new FileOutputStream("D://666.ofd"));
             packet.close();
       } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ConvertException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (PackException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }



//转换 ftp 上的文件
private void ftpFile() {
        try {
            Packet packet = new Packet(PackType.COMMON,Target.OFD);
            URI uri=new
URI("ftp://administrator:cs123456@172.16.14.43:22\\test\\111.doc");
            packet.file(new Common("1","docx",uri));
            ha.convert(packet,new FileOutputStream("D://888.ofd"));
            packet.close();
       } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
```

```
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

1.4.32.4. 接口约束

无。

1.4.33. ocr 文字识别并生成双层 ofd

1.4.33.1. 接口描述

实现该功能需引用 agent*.jar 和 packet*.jar，初始化 Agent 接口（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），调用 convert 接口将 OFD 版式文件中的内容实现 ocr 文字识别，生成双层 ofd。转换完成的文件保存至设置的路径下。注意:需要安装指定节点引擎 ocr(详见”接口约束”)

1.4.33.2. 接口定义

```
packet.file(new Common(String title, String format,int dpi,InputStream data)));
```

参数说明详见下表：

单文 OFD 文件实现 ocr 文字识别，生成双层 ofd 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例

1	title	String 类型	文件标题	packet.file(new Common(null, "jpg", 600,new FileInputStream(new File("D:\\ofd\\wxtp.jpg"))));
				或
				packet.file(new Common("1", "pdf", 600,new FileInputStream(new File("D:\\ofd-muti\\33.pdf"))));

2	format	String 类型	文件类型(pdf 或 ofd 或图片类型)	
3	dpi	int 类型	清晰度参数	
4	data	FileInputStream 类型	待转换源文件	

5	out	FileOutputStream 输出流	本地文件输出路径	new FileOutputStream("D:\\ofd\\wxtp-1.ofd")

返回值说明详见下表：

单文 OFD 文件实现 ocr 文字识别，生成双层 ofd 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.4.33.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.packet.OCR
//功能说明：在 OFD 文件实现 ocr 文字识别，生成双层 ofd

     //1、定义代理对象，请求访问转换服务（war 包版）
     HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
     //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

   private void Convert(){
        // 实现 ocr 文字识别，生成双层 ofd
        long beginName = System.currentTimeMillis();
        try {
            Packet packet =new Packet("custom.ofdocr", Target.OFD);
            //待转换文件：图片文件或扫描生成的 pdf 文件或单层 ofd 文件
            packet.file(new Common(null, "jpg",600,new FileInputStream(new
File("D:\\ofd\\wxtp.jpg"))));
            ha.convert(packet,new FileOutputStream("D:\\ofd\\wxtp-1.ofd"));
            packet.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

```
    }

    public static void main(String[] args){
        Convert_1 Convert = new Convert_1();
        Convert.Convert();

    }
```

1.4.33.4. 接口约束

注：需使用指定的节点引擎，并进行配置

在 build75(转换服务节点)的 plugin 文件夹中放入对应平台的 ocr 工具包，修改 build75\plugin 中的 program.ini 文件。

1.4.34. 简单加密信封（已过时）

1.4.34.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）,packet 接口，调用 envelope 接口将文件转换为 OFD 版式文件并添加密码、添加有效期等，加工后的 ofd 文件需要输入密码才能打开，转换完成的文件保存至设置的路径下。

1.4.34.2. 接口定义

```
Packet envelope(Const.EnvelopeType type,
                String password,
                Map<Const.EnvelopePerm,Boolean> perms,
                Pair<Const.EnvelopeMeta,Object>... meta);
```

参数说明详见下表：

加密信封接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	Const.EnvelopeType	文件的封装信封类型	EnvelopeType.Custom
2	password	String 字符串类型（1-255）	密码	123456
3	perms	Map 类型	封装信封的权限	Map<EnvelopePerm, Boolean> perms

序号	参数名称	参数类型	参数含义	内容举例
				= new HashMap<>(); perms.put(EnvelopePerm.Copy, true);
4	meta	Pair 类型	元数据信息,扩展类型可以写多个	Pair<EnvelopeMeta, Object> meta = new Pair<Const.EnvelopeMeta, Object>(EnvelopeMeta.DocTitle, "测试添加密码");

返回值说明详见下表：

加密信封接口返回值说明表

序号	返回类型	返回值含义
1	Packet	Packet 文件包

1.4.34.3. 接口示例

1：文件加密

```
//java 代码调用
//参考示例: com.springboot.packet.AddValidPeriod2
//功能说明: 为文件添加密码


    //1、定义代理对象，请求访问转换服务（war 包版）
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

private void demo() {
        try {
    Packet packet =new Packet(PackType.COMMON, Target.OFD);
    //设置信封标题
    Pair<EnvelopeMeta, Object> meta = new Pair<Const.EnvelopeMeta,
Object>(EnvelopeMeta.DocTitle, "测试添加密码");
    //设置拷贝权限为 true，一共三个权限：print 打印，saveas 另存为，Copy 文本复制
    Map<EnvelopePerm, Boolean> perms = new HashMap<>();
    perms.put(EnvelopePerm.Copy, true);
    //调用加密信封进行设置对应参数，
    packet.envelope(EnvelopeType.Custom,"123456" ,perms,meta);
    packet.file(new Common("测试", "ofd", new FileInputStream("D:/ofd/5.ofd")));
    ha.convert(packet, new FileOutputStream("D:/www.ofd"));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ConvertException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (PackException e) {
```

```
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

2：文件加有效期

```
//java 代码调用
//参考示例：com.springboot.packet.AddValidPeriod
//功能说明：为文件设置阅读有效期


     //1、定义代理对象，请求访问转换服务(war 包版)
     HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
     //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

private void demo() {
        try {
    Packet packet =new Packet(PackType.COMMON, Target.OFD);
    //定义有效期起始时间
    Pair<EnvelopeMeta, Object> meta = new Pair<Const.EnvelopeMeta,
Object>(EnvelopeMeta.StartDate, "2020-02-20") ;
    //定义有效期结束时间
    Pair<EnvelopeMeta, Object> met1 = new Pair<Const.EnvelopeMeta,
Object>(EnvelopeMeta.EndDate, "2020-02-23") ;
   //设置拷贝权限为 true，一共三个权限：print 打印，saveas 另存为，Copy 文本复制
    Map<EnvelopePerm, Boolean> perms = new HashMap<>();
    perms.put(EnvelopePerm.Copy, true);
    //调用加密信封进行设置对应参数，
    packet.envelope(EnvelopeType.Custom,"123456" ,perms,meta,met1);
    packet.file(new Common("测试", "ofd", new FileInputStream("D:/ofd/5.ofd")));
    ha.convert(packet, new FileOutputStream("D:/www.ofd"));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ConvertException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (PackException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
```

```
            }
        }
    }
```

1.4.34.4. 接口约束

无。

1.4.35. 转换并嵌入字体

1.4.35.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），packet 接口，调用 embedFont 接口将设置转换文档时嵌入字体，加工后的 ofd 文件会将依赖的字库嵌入到文档中，转换完成的文件保存至设置的路径下。

1.4.35.2. 接口定义

```
viod embedFont(boolean exclude, String... names)
```

参数说明详见下表：

嵌入字体名接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	exclude	boolean	是否是排除嵌入指定字体,如果是 false,则是包含嵌入指定字体	True
2	names	String 字符串数组	指定的字体名称.如果参数为 null 时, exclude 为 true,则嵌入所有字体; 为 false 时,则忽略操作	"宋体" 或 new String[] {"黑体","宋体"} 或 null

返回值说明详见下表：

嵌入字体接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.4.35.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.packet.EmbedFont
//功能说明：为文件嵌入字体

    //1、定义代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    private void demo() {
         try {
              Packet packet =new Packet(PackType.COMMON, Target.OFD);
              //设置嵌入字体为 true
              packet.embedFont(true);
              //设置嵌入字体为 false，则嵌入指定字体
             packet.embedFont(true,"宋体");

              //嵌入多种字体，如黑体和宋体
              //packet.embedFont(true, new String[] {"黑体","宋体"});

              //嵌入所有字体
              //packet.embedFont(true, null);
              //添加需要嵌入字体的文件
              packet.file(new Common("1", "ofd", new FileInputStream(new
File("D:/ofd/5.ofd"))));
              //处理后文件保存地址
              ha.convert(packet, new FileOutputStream("D:/testembe.ofd"));
               packet.close();
          } catch (IOException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
         } catch (ConvertException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
         } catch (PackException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
         } finally {
              try {
                  ha.close();
              } catch (IOException e) {
                  e.printStackTrace();
              }
         }
     }
```

1.4.35.4. 接口约束

无

1.4.36. 套版的复合转换

1.4.36.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）,packet 接口,使用通用转换类型，调用 packet 接口将模板 ofd，数据 xml 信息进行封装，再调用 convert 接口，实现将文档复合转换，转换完成的文件保存至设置的路径下。

1.4.36.2. 接口定义

```
Void Template(String title, FileInputStream temp, FileInputStream data);
```

参数说明详见下表：

套版的复合转换接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	title	String 字符串类型（1-255）	套版转换的名称	套转生成 ofd
2	temp	FileInputStream 输入文件流	模板	InputStream temp=new FileInputStream(new File("E:\\sk2.ofd"));
3	data	FileInputStream 输入文件流	数据	InputStream data=new FileInputStream(new File("E:\\1.xml"));

返回值说明详见下表：

套版的复合转换接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.4.36.3. 接口示例

1：单页转换（java 套转）

```
//java 代码调用
```

```
//参考示例：com.springboot.packet.PacketTemplate
//功能说明：  实现套版转换

    //1、定义代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

private void demo() {
        try {
              //使用通用转换类型
              Packet packet =new Packet(PackType.COMMON, Target.OFD);
              //提交模板文件
              InputStream temp=new FileInputStream(new File("E:\\sk2.ofd"));
              //提交数据源 xml
              InputStream data=new FileInputStream(new File("E:\\1.xml"));
              //调用模板接口将模板与数据打包放入待转换包中
              Template t = new Template("套转生成 ofd", temp, data);
              packet.data(t);
              //调用转换接口生成合并后的 ofd 文件
              ha.convert(packet, new FileOutputStream("D:\\ofd\\套转.ofd"));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ConvertException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (PackException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

2：单页转换（template 引擎套转）

```
//java 代码调用
//参考示例：com.springboot.packet.PacketTemplat2ofd
//功能说明：  使用 template 类型转换

    //1、定义代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
```

```
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    private void demo() {
        try {
                 Packet packet =new Packet("common.template", Target.OFD);
                 InputStream temp=new FileInputStream(new
File("D:\\003-CC123456717110023 - 模板.ofd"));
                InputStream data=new FileInputStream(new File("D:\\data.xml"));
                Template t = new Template("复杂套转", temp, data);
                packet.data(t);
                ha.convert(packet,new FileOutputStream("D:\\xpath1.ofd"));
                        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ConvertException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (PackException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

3：多个单页套转并合并（template 引擎套转）

```
//java 代码调用
//参考示例：com.springboot.packet.PacketTemplat2ofd2
//功能说明：使用 template 类型转换的多页转换
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

    //1、定义代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
   //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    private void demo() {
        try {
              //设置转换类型，可以使用通用转换类型，也可以使用 PackType.COMMON 转换类型
              Packet packet =new Packet("common.template", Target.OFD);
              //提交模板文件
              InputStream temp=new FileInputStream(new File("E:\\sk1.ofd"));
              //提交数据源 xml
```

```
            InputStream data=new FileInputStream(new File("E:\\sk1.xml"));
            //调用模板接口将模板与数据打包放入待转换包中
            Template t = new Template("套转生成 ofd1 页", temp, data);
            //------设置第二页模板以及数据
            //提交模板文件
            InputStream temp2=new FileInputStream(new File("E:\\sk2.ofd"));
            //提交数据源 xml
            InputStream data2=new FileInputStream(new File("E:\\sk2.xml"));
            //调用模板接口将模板与数据打包放入待转换包中
            Template t2 = new Template("套转生成 ofd2 页", temp2, data2);
            packet.data(t);
            packet.data(t2);
            //调用转换接口生成合并后的 ofd 文件
            ha.convert(packet, new FileOutputStream("D:\\ofd\\套转.ofd"));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ConvertException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (PackException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

4：带续页和尾页的多页转换（template 引擎套转）

```
//java 代码调用
//参考示例: com.springboot.packet.PacketTemplat2ofdnext
//功能说明: 通过 template2ofd 转换引擎进行套版带续页转换
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

    //1、定义代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    private void demo() {
        try {
              Packet packet =new Packet("common.template", Target.OFD);
              Map<String, InputStream> files=new HashMap<String, InputStream>();
              //续页。必须与模板里的定义相同。
```

```
                files.put("Files\\20-republic-add.ofd",new FileInputStream(new
File("D:\\Files\\20-republic-add.ofd")));
                //尾页
                files.put("Files\\20-republic-tu-add.ofd",new FileInputStream(new
File("D:\\Files\\20-republic-tu-add.ofd")));
                //设置主页模板与数据源 xml 文件
                Template t=new Template("test",new
FileInputStream("D:\\Files\\co-general.ofd"),new
FileInputStream("D:\\Files\\data.xml"),files);
                packet.data(t);
                //调用转换接口生成合并后的 ofd 文件
                ha.convert(packet,new FileOutputStream("D:\\SDK5.ofd"));      } catch
(IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ConvertException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (PackException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

注意：首页模板中需指定好续页模板，续页模板需指定好对应的尾页模板，模板的设置请参见《数科模板设计器手册》。

5：套转并生成图片（template 引擎套转）

```
//java 代码调用
//参考示例：com.springboot.packet.PacketTemplateofd2img
//功能说明：实现将实现证照合成并同时生成缩略图
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务


    //1、定义代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");


private void demo() {
        try {
            //设置转换类型为 ofdmiximage(ofd 转换为图片)
            Packet packet =new Packet("custom.ofdmiximage", Target.OFD);
            //提交模板文件
            InputStream temp=new FileInputStream(new File("D:\\1.ofd"));
```

```
            //提交数据源 xml
            InputStream data=new FileInputStream(new File("D:\\1.xml"));
            //调用模板接口将模板与数据打包放入待转换包中
            Template t = new Template("套转生成 ofd", temp, data);
            packet.data(t);
            //调用转换接口生成合并后的 ofd 文件以及缩略图
            ha.convert(packet, new FileOutputStream("D:\\ofd\\ofd 加缩略图.zip"));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ConvertException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (PackException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

6：模板在服务中的套转

```
//java 代码调用
//参考示例：com.springboot.packet.PacketTemplateofd3
//功能说明：实现将实现证照合成并同时生成缩略图
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

    //1、定义代理对象，请求访问转换服务(war 包版)
    HTTPAgent ha = new HTTPAgent("http://localhost:8088/convert-issuer/");

private void demo() {
        try {
    //使用 template 转换类型
            Packet packet =new Packet("common.template", Target.OFD);
            //上传数据 xml，并声明使用模板的名称
            PackEntry file1 =PackEntry.wrap(new FileInputStream("D:\\1.xml"));
            packet.data(file1, "1.ofd");
            //调用转换接口生成合并后的 ofd 文件
            ha.convert(packet, new FileOutputStream("D:\\ofd\\套转.ofd"));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ConvertException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (PackException e) {
            // TODO Auto-generated catch block
```

```
            e.printStackTrace();
        } finally {
            try {
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

注意：模板在服务中的套装中，需要将模板文件放到转换服务指定的 template(模板文件)存放处例如：D:\tomcat\webapps\convert-issuer\WEB-INF\Convert\template

1.4.36.4. 接口约束

无

1.4.37. 文件 md5 校验

1.4.37.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent（Springboot 版对应 AtomAgent/HTTPAgent,war 包版对应 HTTPAgent),PackEntry 接口,调用 setHash 接口设置系统中已有的文档 md5 值,打包时与新上传的文档的 MD5 值进行校验对比。如不匹配，会直接报错：Hash is not match。

1.4.37.2. 接口定义

```
void setHash(byte[] hash);
```

参数说明详见下表：

设置文件 Hash 值接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	hash	byte[]	文件的MD5值字节数组	Hex.decodeHex("1fbc02e98dfc013da817019752cfe192".toCharArray())

返回值说明详见下表：

设置文件 Hash 值接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.4.37.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.packet.PacketMD5
//功能说明：校验文档 md5

public class PacketMD5 {

    //1、定义代理对象，请求访问 springboot 转换服务
    AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    private void demo() {
        try {
            Packet packet = new Packet(PackType.COMMON,Target.OFD);
            //上传文件获取 MD5 值
            PackEntry file =
            PackEntry.wrap(new File("D:\\公文附件
001.wps"),MessageDigest.getInstance("MD5"));
            //设置业务系统中已经计算好的 MD5 值与新上传文件的 MD5 值进行对比

    file.setHash(Hex.decodeHex("1fbc02e98dfc013da817019752cfe191".toCharArray()));
            //添加文件
            packet.file(new Common("File1","wps",100,file));
            //文件打包输出
            packet.pack(new FileOutputStream("d:\\md5__4.zip"));

        } catch (IOException e) {
            e.printStackTrace();
        } catch (PackException e) {
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                ha.close();
                System.out.println(".....");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

```
    public static void main(String[] args){
         PacketMD5 officeToOFD = new PacketMD5();
         officeToOFD.demo();
     }
}
```

1.4.37.4. 接口约束

注:此方法在 packet-wrapper-1.10.18.0918.jar 以后版本的 jar 包可使用;

1.4.38. 导出 OFD 文件注释到另一个 OFD 文件中

1.4.38.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent,packet 接口，调用 graftingAnnotation 接口实现导出当前 OFD 文件注释到新的 OFD 文件中。注意:需要安装指定节点引擎 ofdannot(详见”接口约束”)

1.4.38.2. 接口定义

```
Packet graftingAnnotation(PackEntry ofd);
```

参数说明详见下表:

导出 OFD 文件注释到另一个 OFD 文件中接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	ofd	PackEntry	带注释的数据包	PackEntry.wrap(new FileInputStream("D:\\1\\test.ofd"))

返回值说明详见下表:

导出 OFD 文件注释到另一个 OFD 文件返回值说明表

序号	返回类型	返回值含义
1	Packet 类型	packet 对象，导出注释后的文件数据包

1.4.38.3. 接口示例

```
//java 代码调用
//参考示例: com.springboot.annot.AnnotExportImport
```

```
//功能说明：OFD 文件注释导出到另一个 OFD 文件中
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务
package com.http.annot;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import com.suwell.ofd.custom.agent.HTTPAgent;
import com.suwell.ofd.custom.wrapper.Const.PackType;
import com.suwell.ofd.custom.wrapper.Const.Target;
import com.suwell.ofd.custom.wrapper.PackEntry;
import com.suwell.ofd.custom.wrapper.Packet;
import com.suwell.ofd.custom.wrapper.model.Common;

public class AnnotExportImport {


    public static void main(String[] args) {
        //1、定义代理对象，请求访问转换服务（war 包版）
         HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
        //1、定义代理对象，请求访问转换服务(Springboot 版)
        //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
        //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

         Packet packet = new Packet(PackType.COMMON,Target.OFD);

        try {
               //需要被导入注释的 OFD 文件
               packet.file(new Common(null, "ofd", new
FileInputStream("D:\\1\\test1.ofd")));
               //带注释的 OFD 文件
               packet.graftingAnnotation(PackEntry.wrap(new
FileInputStream("D:\\1\\test.ofd")));
               //最终生成的 OFD 文件
               ha.convert(packet, new FileOutputStream("D:\\3.ofd"));
               packet.close();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }finally {
                try {
                    ha.close();
                    System.out.println("结束......");
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

        }
}
```

1.4.38.4. 接口约束

注：需使用指定的节点引擎，并进行配置

在 build75(转换服务节点)的 plugin 文件夹中放入对应平台的工具包 ofdannot，修改 build75\plugin 中的 program.ini 文件。

1.4.39. 流式文件（docx 文件）预盖章

1.4.39.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent），packet 接口，调用 packet.invoke()方法调用"ApplySign"实现流式 docx 文件插入签章图片预盖章，转换为盖章的 OFD 文件。注意:需要按步骤操作 docx 文档(详见”接口约束”)

1.4.39.2. 接口定义

```
E invoke(String type,
         String module,
         String method,
         Pair<String,String>... pairs);
```

参数说明详见下表：

流式文件预盖章转 OFD 文件应用签章接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	String 类型	调用类型	null
2	module	String 类型	调用模块	"signature"
3	method	String 类型	调用方法名称	"ApplySign"
4	pairs	Pair 类型	自定义参数	//定义要应用签章的签章信息（印章名称，印章密码，印章 ID） new Pair<String, String>("args", GSON.toJson(map));

返回值说明详见下表：

流式文件预盖章转 OFD 文件应用签章接口返回值说明表

序号	返回类型	返回值含义
1	E extends MainXML	Packet 对象

1.4.39.3. 接口示例

```
//java 代码调用
```

```
//参考示例：com.springboot.api.AddStampSeal_11
//功能说明：docx 文件插入签章图片预盖章，转换为签章的 OFD 文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

//docx 文件预盖章,转换为 ofd 为已盖章文件
public class AddStampSeal_11 {

    //1、创建 Http 代理对象，连接请求转换服务地址
    static HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8080/convert-issuer/");
    //1、定义代理对象，请求访问转换服务(Springboot 版)
    //static  AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
    //static  HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

    //定义 gson 常量，用于存储签章信息 map 集合
    private static final Gson GSON = new Gson();
    //2、定义调用方法
    public static void main(String[] args) {
        //2-1、定义数据包对象
        Packet packet = new Packet(Const.PackType.COMMON, Target.OFD);
        try {
            //文件转换路径
            packet.file(new Common(null, "docx", new
FileInputStream("D:\\test.docx")));
            //定义 map 集合
            Map<String, Object> sm = new HashMap<String, Object>();
            //印章名称
            sm.put("OesName", "Suwell_SDK");
            //密码
            sm.put("Password", "123456");
            //印章 ID
            sm.put("SealId", "4d16b8af-88c1-4789-833b-a4db60ddb593");
            //3-6、数据包调用 ApplySign 方法添加预盖章
            packet.invoke(null, "signature","ApplySign",new Pair<String,
String>("arg", GSON.toJson(sm)));
//          packet.pack(new FileOutputStream("D:/ofd/packet-46.zip")); //打包
            //调用转换服务，转换为 OFD 文件
            ha.convert(packet, new FileOutputStream("D:\\ofd\\预盖章 docx.ofd"));
            //输出结果
            System.out.println("预盖章执行完成！！！");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                packet.close();
                ha.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

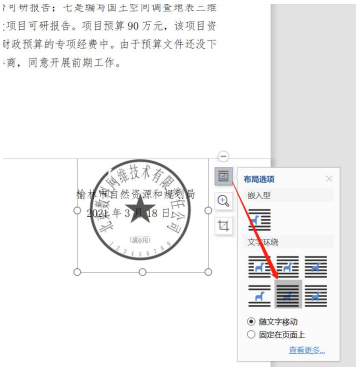
1.4.39.4. 接口约束

实现此功能需要按照如下流程操作：

新建一个 docx 文件，并使用 wps 软件打开该文件，在 docx 文件中需要盖章的位置插入一个签章图片（或拷贝图片到当前需要签章的位置），如下图 1.png

1.png

选择签章图片属性为“衬于文字下方”，如下图所示：



选中签章图片，点击 wps 工具栏中的插入->书签->填写书签名（必须是 PRE_SEAL_X 开头,X 可以设置任意数字）->添加如下图所示：

图片添加书签名称后，保存当前 docx 文件。调用上述接口实现 docx 文件预盖章转 OFD 文件实现预盖章操作。（注：Linux 平台转换服务可实现当前功能）

1.4.40. HTML 带 CSS 资源文件转换为 OFD 文件

1.4.40.1. 接口描述

实现该功能需引用 packet*.jar，初始化 agent（Springboot 版对应 AtomAgent/HTTPAgent，war 包版对应 HTTPAgent）,packet 接口(注意需要添加目标文件为 HTML 资源集合)，调用 ha.convert 接口实现将带有 CSS 资源的 HTML 文件目录转换为 OFD 文件。

1.4.40.2. 接口定义

```
packet.file(new Common(String title,
                       String format,
                       GetTogether together));
```

参数说明详见下表：

HTML 带 CSS 静态资源转 OFD 接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	title	String 类型	文件标题,可以为 null	packet.file(new Common(null, "html",GetTogether.build(new Pair<String, PackEntry>("singleTest.html",PackEntry.wrap(new File("D:\\jsapi\\singleTest.html"))))));
2	format	String 类型	文件类型(html)	
3	together	GetTogether	HTML 目标资源集合	
4	out	FileOutputStream 输出流	本地文件输出路径	new FileOutputStream("D:\\ofd\\html2ofd.ofd")

返回值说明详见下表：

HTML 带 CSS 静态资源转 OFD 接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.4.40.3. 接口示例

```
//java 代码调用
//参考示例：com.springboot.api.OfdToHtml_10
//功能说明：HTML 文件夹（带 css 静态资源）转 OFD 文件
//HTTPAgent/AtomAgent 为服务代理类，通过此类连接及调用转换服务

//转换包含 css 资源的 html 文件为 OFD 文件
public class OfdToHtml_10 {


    public static void main(String[] args) {

        //1、定义转换服务 http 代理对象（war 包版）
        HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8080/convert-issuer/");
         //1、定义代理对象，请求访问转换服务(Springboot 版)
         //AtomAgent ha = new AtomAgent("http://127.0.0.1:8090");
         //HTTPAgent ha = new HTTPAgent("http://127.0.0.1:8090");

        //2-1、定义数据包对象
         Packet packet = new Packet(PackType.COMMON, Target.OFD);
         try {
             //参数 1：html 文件名     参数 2：html 文件路径
             GetTogether g = GetTogether.build(new Pair<String,
PackEntry>("singleTest.html",
                     PackEntry.wrap(new File("D:\\jsapi\\singleTest.html"))));

             //html 资源包路径(传入 html 资源包名)
             String replacePath="D:\\jsapi\\";
             //html 资源包路径(传入 html 资源包详细路径)
             allfilelist(new File("D:\\jsapi\\"),g,replacePath);
             //数据包添加文件
             packet.file(new Common(null, "html",  g));

//           packet.pack(new FileOutputStream("D:/ofd/80.zip")); //打包
             ha.convert(packet, new FileOutputStream("D:\\ofd\\test_css.ofd"));

             } catch (Exception e) {
                 e.printStackTrace();
             } finally {
                 try {
                 packet.close();
             } catch (IOException e) {
                 e.printStackTrace();
         }
    }
}
    /**
     * 获取静态资源文件集合
     * @param file 要遍历的文件夹
     * @param g 追加静态资源类
     * @param replacePath 要截取的相对路径
     */
```

```
    static void allfilelist(File file, GetTogether g,String replacePath){
       //定义 list 集合，用于存储文件夹中的所有文件
        List<String> filenamelist=new ArrayList<String>();
        //获取文件夹中的子文件及文件夹集合
       File[] files = file.listFiles();
       //循环文件夹中的子文件集合 list
       for (int i = 0; i < files.length; i++) {
        //如果文件夹中的子文件是文件夹，递归调用当前方法
          if (files[i].isDirectory()) {
              allfilelist(files[i],g,replacePath);
          //如果文件夹中的子文件是文件
          } else{

                //将文件路径加入到 list 集合中
                  filenamelist.add(files[i].getPath());
                  //调用方法添加节点
                  g.addItem(new Pair<String,
PackEntry>(files[i].getPath().replace(replacePath,"").replace("\\", "/"),
PackEntry.wrap(new File(files[i].getPath()))));

                  System.out.println("输出 html 资源包文件:
"+files[i].getPath().replace(replacePath,"").replace("\\", "/"));
                  System.out.println("输出 html 资源包文件路径: "+files[i].getPath());
          }
       }
   }
}
```

1.4.40.4. 接口约束

210820 及以后版本。

1.5. 流式文件套模板转换

1.5.1. 流式文件套模板转换生成 ofd 功能

1.5.1.1. 接口描述

实现该功能需调用指定（具有该功能的）转换服务。根据提供的示例代码，调用指定方法，完成流式文件+xml 模板文件套转 ofd 文件功能。

1.5.1.2. 接口定义

1、流式文件与模板 xml 文件合成套转

```
String wpsFileTask(File moudelfile,
```

```
                    File xmlfile);
```

参数说明详见下表：

wps 文件模板合成转换接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	moudelfile	File 文件类型	待合成的 wps 文件	new File("D:\\docx_1.docx")
2	xmlfile	File 文件类型	模板文件 xml 文件	new File("D:\\模板样例.xml")

返回值说明详见下表：

wps 文件模板合成转换接口返回值说明表

序号	返回类型	返回值含义
1	String 字符串	请求服务后获取的响应信息，响应是否转换成功。示例：{"statusCode":0,"message":null,"data":"2021062808504627400001"}

2、查询合成转换状态

```
String reflushOfd(String ticket);
```

参数说明详见下表：

查询转换状态接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	ticket	String 类型	请求转换的文件标识，该参数由 wpsFileTask 方法返回值中 data 获取	“202106280850462740001”

返回值说明详见下表：

查询转换状态接口返回值说明表

序号	返回类型	返回值含义
1	String 字符串	查询到的文件转换状态及转换成功后文件下载路径。示例：{"statusCode":0,"message":null,"data":"http://192.168.221.134:8089/convert-issuer/download?ticket=20210629085028240001"}

3、合成转换成功后下载 OFD 文件

```
void saveUrlAs(String url,
```

```
            String filePath,
            String fileName,
            String method);
```

参数说明详见下表：

下载转换后的 OFD 文件接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	url	String 类型	转换后 ofd 文件访问路径，该参数由 reflushOfd 方法返回值 data 获取	"http://192.168.221.134:8089/convert-issuer/download?ticket=2021062908 50282400001"
2	filePath	String 类型	Ofd 文件保存到本地的自定义路径	“D:\\convert_ofd/”
3	fileName	String 类型	自定义的 ofd 文件保存名称,注意该名称最好随机，不能重名，否则覆盖	“test”
4	method	String 类型	请求下载文件的请求方法	"GET"

返回值说明详见下表：

下载转换后的 OFD 文件接口返回值说明表

序号	返回类型	返回值含义
1	无	无

1.5.1.3. 接口示例

```
//java 代码调用
//参考示例：见示例代码
//功能说明：将流式文件与 xml 模板文件合并转换为 ofd 文件


    //1、定义转换服务请求地址
     private static String issuerurl = "http://192.168.221.134:8089/convert-issuer/";
     //2、方法调用
    public static void main(String[] args) throws JsonParseException,
JsonMappingException, IOException {
        //2-1、调用 wps+xml 模板合成方法，其中传参一：待转换的流式文档，传参二：待合成的 xml
模板文件,获取响应信息 :{"statusCode":0,"message":null,"data":"202106280850462740000 1"}
        String body = wpsFileTask(new File("D:\\报告样例设计样例.docx"), new File("D:\\
告样例设计样例.xml"));
```

```
        //2-2、将响应信息转换为 ObjectNode 类型对象
        ObjectNode node = new ObjectMapper().readValue(body, ObjectNode.class);
        //2-3、判断响应值状态码是否为 0，0 为成功
        if (node.get("statusCode").toString()=="0"){
            //2-4、记录日志
            System.out.println("文件上传成功 执行查询方法");
            //2-5、获取响应信息返回的数据 data 值,即 ticket
            String id=node.get("data").toString().replace("\"", "");
            //2-6、循环查询文件转换状态
            while (true){
                 //2-7、查询转换执行状态
                 String reflushOfdBody =  reflushOfd(id);
                 //2-8、将响应结果转为 ObjectNode 类型对象
             ObjectNode reflushOfdNode = new ObjectMapper().readValue(reflushOfdBody,
  ObjectNode.class);
                 //2-9、日志记录状态码
                 System.out.println(reflushOfdNode.get("statusCode").toString());
                 //2-10、当状态码为 0 时，转换成功，执行文件下载
                 if (reflushOfdNode.get("statusCode").toString()=="0"){
                      //2-11、记录日志
                      System.out.println("下载准备就绪 执行下载方法");
                      //2-12、获取响应信息中的 data 值，定义该值为 ofd 保存的文件名
                      String name = node.get("data").toString().replace("\"", "");
                      //2-13、下载文件并将文件保存到指定位置，第二个参数
（"D:\\convert_ofd/"）需改为自己本机地址

DownloadURLFile.saveUrlAs(issuerurl+"/download?ticket="+name,"D:\\convert_ofd/","a_
"+name,"GET");
                      //下载保存文件后跳出循环
                      break;
                  }

             }

        }

    }
```

1.5.1.4. 接口约束

无。

1.6. 常用工具类

1.6.1. Group

1.6.1.1. 接口描述

集合类，继承 Pair 类

1.6.1.2. 接口定义

```
//构造函数
public class Group<K,V,T> extends Pair<K,V>
```

参数说明详见下表：

序号	参数名称	参数类型	参数含义	内容举例
1	key	K 类型	自定义类型	
2	value	V 类型	自定义类型	
3	type	T 类型	自定义类型	

1.6.2. MarkPosition

1.6.2.1. 接口描述

用于设置添加水印的位置区域等信息。

1.6.2.2. 接口定义

```
//构造函数
MarkPosition(float x,
             float y,
             float width,
             float height,
             int[] index
             Pair<Integer,Integer>... pages);
```

参数说明详见下表：

序号	参数名称	参数类型	参数含义	内容举例
1	x	float 类型	水印起始 X 坐标，单位毫米	10
2	y	float 类型	水印起始 Y 坐标，单位毫米	20

序号	参数名称	参数类型	参数含义	内容举例
3	width	float 类型	水印的宽度，单位毫米	20
3	height	float 类型	水印的高度，单位毫米 注意：如果宽高设置大于页面大小，则图片水印默认页面的一半大小	20
4	index	int[]类型	添加水印的页码集合.从0开始。所有页使用常量 MarkPosition.INDEX_ALL	new int[]{1,2,3} //2,3,4 页
5	pages	Pair<Integer, Integer>类型	添加水印页面的页码集合(连续页),区别 index 方式（1-3，6-8）	new Pair<Integer, Integer>(6, 8);//7-9 页

1.6.2.3. 接口方法

序号	方法名称	方法参数	方法返回值类型	方法说明
1	getX()	无	float 类型	获取水印 x 坐标
2	getY()	无	float 类型	获取水印 Y 坐标
3	getWidth()	无	float 类型	获取水印的宽度
4	getHeight()	无	float 类型	获取水印的高度
5	getIndex()	无	int[]类型	获取添加水印的页码集合(单页：1，3，4)
6	getPages()	无	Pair<Integer,Integer>[]类型	获取添加水印页面的页码集合,区别 index 方式（连续页：1-3，6-8）
7	isPattern()	无	boolean 类型	是否是图案
8	getTarget()	无	String 类型	获取水印
9	getxAlign()	无	Const.XAlign 类型	获取水印水平位置,左，中，右
10	getyAlign()	无	Const.YAlign 类型	获取水印垂直位置,上，中，下
11	isTile()	无	boolean 类型	获取水印是否平铺
12	getyStep()	无	Integer 类型	获取平铺水印 y 轴间距
13	getxStep()	无	Integer 类型	获取平铺水印 x 轴间距
14	getRotate()	无	Integer 类型	获取水印旋转角度
15	getMargin()	无	float[]类型	获取水印添加区域
16	setRotate(in	int rotate	MarkPosition 类型	设置水印旋转角度

序号	方法名称	方法参数	方法返回值类型	方法说明
	t rotate)			
17	setTarget(String target)	String target	无	设置水印内容
18	setTile(boolean tile)	boolean tile	MarkPosition 类型	设置水印是否平铺
19	setxAlign(Const.XAlign xAlign)	Const.XAlign xAlign	MarkPosition 类型	设置水印水平位置，left,right,center
20	setyAlign(Const.YAlign yAlign)	Const.YAlign yAlign	MarkPosition 类型	设置水印垂直位置，Top，bottom,middle
21	setxStep(int xStep)	int xStep	MarkPosition 类型	设置水印水平方向间距/步长
22	setyStep(int yStep)	int yStep	MarkPosition 类型	设置水印垂直方向间距/步长
23	setMargin(float left, float top, float right, float bottom)	float left float top float right float bottom	MarkPosition 类型	设置水印添加页面区域，（在设置的左,上,右,下,页边距以内添加水印）

1.6.3. TextInfo

1.6.3.1. 接口描述

用于设置添加文字水印的文字内容信息。

1.6.3.2. 接口定义

```
//构造函数
TextInfo(String text,
         String fontName,
         float fontSize,
         String color);
```

```
//构造函数（该构造函数已过时）
```

```
TextInfo(String text,
         String fontName,
         float fontSize,
         String color,
         int rotate,
         Const.XAlign xAlign,
         Const.YAlign yAlign);
```

参数说明详见下表：

序号	参数名称	参数类型	参数含义	内容举例
1	text	String 类型 (1-255)	水印内容	"水印水印"
2	fontName	String 类型 (1-255)	水印字体	"宋体"
3	fontSize	Float 类型	水印字号	20
4	Color	String 类型 (1-255)	水印的透明度及颜色	"#FFFF3030"
5	Rotate	Int 整型	旋转度数，只支持 45 度的倍数	90
6	xAlign	XAlign 自定义类型	水平定位方向。使用常量如：Const.XAlign.Center	Const.XAlign.Center
7	yAlign	YAlign 自定义类型	垂直定位方向。使用常量如：Const.YAlign.Middle	Const.YAlign.Middle

注意：文字水印中，XAlign，yAlign 设置与position中的x,y互斥，如果想用MarkPosition中的绝对位Const.XAlign.Absolute。

1.6.3.3. 接口方法

序号	方法名称	方法参数	方法返回值类型	方法说明
1	getCharSpace()	无	int 类型	获取水印字间距
2	getLineSpace()	无	int 类型	获取水印行间距
3	getOpacity()	无	int 类型	获取水印透明度
4	getText()	无	String 类型	获取水印的文本内容
5	isBold()	无	Boolean 类型	获取水印是否是粗体
6	isItalic()	无	boolean 类型	获取水印是否是斜体
7	setBold(boolean bold)	boolean bold	TextInfo 类型	设置文字水印是否加粗
8	setCharSpace(int charSpace)	int charSpace	TextInfo 类型	设置文字水印字间距

序号	方法名称	方法参数	方法返回值类型	方法说明
9	setItalic(boolean italic)	boolean italic	TextInfo 类型	设置文字水印是否倾斜
10	setLineSpace(int lineSpace)	int lineSpace	TextInfo 类型	设置文字水印行间距
11	setOpacity(int opacity)	int opacity	TextInfo 类型	设置文字水印透明度

1.6.4. SealInfo

1.6.4.1. 接口描述

用于设置电子签章的基本信息及位置区域等信息。

1.6.4.2. 接口定义

```
//构造函数
SealInfo(NativeType type,
         String sealID,
         float x,
         float y,
         float width,
         float height,
         String password,
          int pageIndex);
```

参数说明详见下表：

序号	参数名称	参数类型	参数含义	内容举例
1	type	NativeType 整型	签章位置。目前有五个常量 SealInfo.NativeType.Normal、SealInfo.NativeType.All 、SealInfo.NativeType.First SealInfo.NativeType.Last SealInfo.NativeType.Check	SealInfo.NativeType.Normal
2	sealID	String 类型	章的唯一标识。需要签章厂商提供	"2d7c5856-c55f-47c1-8b3c-466bddc60f46"
3	x	Float 类型	章在图像上的 x 坐标	10f
3	y	Float 类型	章在图像上的 y 坐标	10f

序号	参数名称	参数类型	参数含义	内容举例
4	width	Float 类型	章的显示宽度	200f
5	height	Float 类型	章的显示高度	200f
6	password	String 类型	签章密码	"123456"
7	pageIndex	Int 类型	添加签章的页码	1

1.6.4.3. 接口方法

序号	方法名称	方法参数	方法返回值类型	方法说明
1	getHeight()	无	float 类型	获取签章显示高度
2	getLocInfos()	无	SealInfo.LocInfo[] 类型	获取签章位置
3	getPageIndex()	无	int 类型	获取签章页码
4	getPassword()	无	String 类型	获取签章密码
5	getSealID()	无	String类型	获取签章 ID
6	getStampAnnot()	无	SealInfo.StampAnnot[]	获取签章水印
7	getType()	无	int 类型	获取签章类型
8	getTypeC()	无	String 类型	获取水印
9	getUserID()	无	String 类型	获取签章用户 ID
10	getWidth()	无	String 类型	获取签章显示宽度
11	getX()	无	float 类型	获取签章 x 轴坐标位置
12	getY()	无	float 类型	获取签章 y 轴坐标位置
13	setUserID(String userID)	String userID	SealInfo 类型	设置签章用户信息
14	getExtend()	无	Map<SignatureExtendKey, String>	获取签章拓展参数

添加扩展参数SignatureExtendKey类型说明：

Param_GetCertList	//扩展参数适用于 OES-GetCertList
Param_GetDigestMethod	//扩展参数适用于 OES-GetDigestMethod
Param_GetSeal	//扩展参数适用于 OES-GetSealI
Param_GetSealImage	//扩展参数适用于 OES-GetSealImage
Param_GetSealList	//扩展参数适用于 OES-GetSealList
Param_GetSignMethod	//扩展参数适用于 OES-GetSignMethod
Param_RawSign	//扩展参数适用于 OES-RawSign
Param_Sign	//扩展参数适用于 OES-Sign
Param_TimeStamp	//扩展参数适用于 OES-TimeStamp

1.6.5. SignInfo

1.6.5.1. 接口描述

用于设置电子签名的信息。

1.6.5.2. 接口定义

文字签名：

```
//构造函数
SignInfo(String certID,
         String password,
         TextInfo textInfo,
         MarkPosition position,
         boolean printable,
         boolean visible)
```

参数说明详见下表：

序号	参数名称	参数类型	参数含义	内容举例
1	certID	String 类型	签名 ID	"9a6fe709143103ff"
2	password	String 类型	密码，与签章密码一致	"123456"
3	textInfo	TextInfo 类型	文字签名	new TextInfo("我是签名","黑体", 20,"#000000")
4	position	MarkPosition 类型	签名位置	new MarkPosition(new int [] {1,3})
5	printable	boolean 类型	是否打印	true /false
6	visible	boolean 类型	是否显示	true /false

图片签名：

```
//构造函数()
SignInfo(String certID,
         String password,
         T image,
         MarkPosition position,
         String type,
         boolean printable, boolean visible)
```

参数说明详见下表：

序号	参数名称	参数类型	参数含义	内容举例
1	certID	String 类型	签章 ID	"9a6fe709143103ff"

序号	参数名称	参数类型	参数含义	内容举例
2	password	String 类型	密码，与签章密码一致	"123456"
3	image	PackEntry 类型	图片签名	PackEntry.wrap(new File("D:/2.png"))
4	position	MarkPosition 类型	签名位置	new MarkPosition(new int [] {1,3})
5	type	String 类型	类型	
6	printable	boolean 类型	是否打印	true /false
7	visible	boolean 类型	是否显示	true /false

1.6.5.3. 接口方法

序号	方法名称	方法参数	方法返回值类型	方法说明
1	getCertID()	无	String 类型	获取签名 ID
2	getPassword()	无	String 类型	获取签名密码
3	getTextInfo()	无	TextInfo 类型	获取签名文本对象
4	getPosition()	无	MarkPosition 类型	获取签名位置对象
5	isPrintable()	无	Boolean 类型	是否打印
6	isVisible()	无	Boolean 类型	是否显示
7	getType()	无	String 类型	获取签名类型
8	getImage()	无	PackEntry 类型	获取签名图片

1.6.6. PageNumber

1.6.6.1. 接口描述

用于设置转换文件页码自定义参数。

1.6.6.2. 接口定义

```
//构造函数
PageNumber();
```

1.6.6.3. 接口方法

序号	方法名称	方法参数	方法返回值类型	方法说明
1	getX()	无	String 类型	获取页码 x 轴坐标位置

序号	方法名称	方法参数	方法返回值类型	方法说明
2	getY()	无	String 类型	获取页码 y 轴坐标位置
3	getStep()	无	Integer 类型	获取页码步长（间距）
4	getStartNumber()	无	Integer 类型	获取页码起始编号
5	getRotate()	无	Float 类型	获取页码旋转角度
6	getPredefinedPosition()	无	String 类型	获取页码预定义的位置
7	getPageRange()	无	String 类型	获取页码范围：even,odd
8	getPagePadding()	无	String 类型	获取页码边距
9	getPageNumberFormat()	无	String 类型	获取页码格式
10	getOrientation()	无	String 类型	获取页码方向
11	getItalic()	无	Boolean 类型	获取页码是否斜体
12	getForeColor()	无	String 类型	获取页码颜色
13	getFontSize()	无	Float 类型	设置页码字体大小
14	getFontName()	无	String 类型	设置页码字体
15	getBold()	无	Boolean 类型	获取页码是否加粗
16	getAutoAdaptOrientation()	无	String 类型	获取自适应页码方向
17	setAutoAdaptOrientation(String autoAdaptOrientation)	String autoAdaptOrientation	无	设置页码自适应方向
18	setBold(Boolean bold)	Boolean bold	无	设置页码加粗
19	setFontName(String fontName)	String fontName	无	设置页码字体名称
20	setFontSize(Float fontSize)	Float fontSize	无	设置页码字体大小
21	setForeColor(String foreColor)	String foreColor	无	设置页码字体颜色
22	setItalic(Boolean italic)	Boolean italic	无	设置页码字体斜体
23	setOrientation(String orientation)	String orientation	无	设置页码方向
24	setPageNumberFormat(String pageNumberFormat)	String pageNumberFormat	无	设置页码显示格式
25	setPagePadding(String pagePaddin	String pagePadding	无	设置页码边距

序号	方法名称	方法参数	方法返回值类型	方法说明
	g)			
26	setPageRange(String pageRange)	String pageRange	无	设置页码范围：EVEN：偶数,ODD：奇数
27	setPredefinedPosition(String predefinedPosition)	String predefinedPosition	无	设置页码预定义的位置
28	setRotate(Float rotate)	Float rotate	无	设置页码旋转角度
29	setStartNumber(Integer startNumber)	Integer startNumber	无	设置页码的起始页码
30	setStep(Integer step)	Integer step	无	设置页码步长，间隔
31	setX(String x)	String x	无	设置页码 X 轴位置
32	setY(String y)	String y	无	设置页码 y 轴位置
33	toMap()	无	Map<String,String>	将 PageNumber 转为 map 集合

1.6.7. WebArgument

1.6.7.1. 接口描述

用于设置 HTML 转换 OFD 文件自定义参数。

1.6.7.2. 接口定义

```
//构造函数
WebArgument()
```

1.6.7.3. 接口方法

```
//设置纸张大小
WebArgument setWebPageSize(Const.PageSize webPageSize,
                           Pair<Const.WebMargin,Float>... margin)
```

参数说明详见下表：

Html 转 OFD 文件自定义参数设置纸张大小接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	webPageSize	Const.PageSize 类型	指定纸张	A0、A1、A2、A3、A4

			指定纸张的宽度，长度自适应	A0W、A1W、A2W、A3W、A4W
			自定义纸张大小	CUSTOM
			Web 页面有多大就打印多大的纸张，如果不指定 --papersize 或 --papersize 指定错误。默认自适应。WEB 有多大就打印多大的纸张。	ADAPTIVE
2	margin	Pair 类型	自定义可选参数，可以指定页边距，上边距	new Pair<Const.WebMargin, Float>(Const.WebMargin.TopMargin, 35f);
			自定义可选参数，可以指定页边距，下边距	new Pair<Const.WebMargin, Float>(Const.WebMargin.BottomMargin, 35f);
			自定义可选参数，可以指定页边距，左边距	new Pair<Const.WebMargin, Float>(Const.WebMargin.LeftMargin, 35f);
			自定义可选参数，可以指定页边距，右边距	new Pair<Const.WebMargin, Float>(Const.WebMargin.RightMargin, 35f);

返回值说明详见下表：

Html 转 OFD 文件自定义参数设置纸张大小接口返回值说明表

序号	返回类型	返回值含义
1	WebArgument 类型	自定义 html 转换文件的参数

```
//设置纸张大小 2
WebArgument setWebPageSize(int width,
                           int height,
                           Pair<Const.WebMargin,Float>... margin)
```

参数说明详见下表：

Html 转 OFD 文件自定义参数设置纸张大小接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	width	int 类型	自定义纸张宽度	210
2	height	int 类型	自定义纸张高度	258
3	margin	Pair 类型	自定义可选参数，可以指定页边距，上边距	new Pair<Const.WebMargin, Float>(Const.WebMargin.TopMargin, 35f);

			自定义可选参数，可以指定页边距，下边距	new Pair<Const.WebMargin, Float>(Const.WebMargin.BottomMargin, 35f);
			自定义可选参数，可以指定页边距，左边距	new Pair<Const.WebMargin, Float>(Const.WebMargin.LeftMargin, 35f);
			自定义可选参数，可以指定页边距，右边距	new Pair<Const.WebMargin, Float>(Const.WebMargin.RightMargin, 35f);

返回值说明详见下表：

Html 转 OFD 文件自定义参数设置纸张大小接口返回值说明表

序号	返回类型	返回值含义
1	WebArgument 类型	自定义 html 转换文件的参数

```
//设置纸张边距
WebArgument setPaperMargin(paperMargin paperMargin)
```

参数说明详见下表：

Html 转 OFD 文件自定义参数设置纸张边距接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	paperMargin	paperMargin 类型	无边距	NOMARGIN
			默认边距	DEFAULTMARGIN

返回值说明详见下表：

Html 转 OFD 文件自定义参数设置纸张边距接口返回值说明表

序号	返回类型	返回值含义
1	WebArgument 类型	自定义 html 转换文件的参数

```
//设置是否打印纸张背景
WebArgument setPaperBackgrounds(Const.Opinion opinion);
```

参数说明详见下表：

Html 转 OFD 文件自定义参数设置纸张背景接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	opinion	Const.Opinion 类型	打印背景	YES
			不打印背景	NO

返回值说明详见下表：

Html 转 OFD 文件自定义参数设置纸张背景接口返回值说明表

序号	返回类型	返回值含义
1	WebArgument 类型	自定义 html 转换文件的参数

```
//设置第一页上边距
WebArgument setFirstTop(float topMargin);
```

参数说明详见下表：

Html 转 OFD 文件自定义参数设置第一页上边距接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	topMargin	float 类型	第一页上边距	10f

返回值说明详见下表：

Html 转 OFD 文件自定义参数设置第一页上边距接口返回值说明表

序号	返回类型	返回值含义
1	WebArgument 类型	自定义 html 转换文件的参数

```
//设置是否横向转换
WebArgument setLandScape(Opinion opinion);
```

参数说明详见下表：

Html 转 OFD 文件自定义参数设置横向打印格式接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	opinion	Opinion 类型	设置转换后横向/纵向展示	Opinion.NO //纵向 Opinion.YES //横向

返回值说明详见下表：

Html 转 OFD 文件自定义参数设置横向打印格式接口返回值说明表

序号	返回类型	返回值含义
1	WebArgument 类型	自定义 html 转换文件的参数

1.6.8. ImageArgument

1.6.8.1. 接口描述

用于设置 OFD 转 Image 图片格式文件自定义参数。

1.6.8.2. 接口定义

```
//构造函数
ImageArgument();
```

1.6.8.3. 接口方法

```
//设置目标文件 Image 图片格式
ImageArgument setTargetFormat(ImageArgument.Type type);
```

参数说明详见下表：

OFD 转 Image 设置图片格式接口参数说明表

序号	参数名称	参数类型	参数含义	内容举例
1	type	ImageArgument.Type	JPG 图片格式	jpg

			PNG 图片格式	png

返回值说明详见下表：

OFD 转 Image 设置图片格式接口返回值说明表

序号	返回类型	返回值含义
1	ImageArgument 类型	自定义图片格式参数

1.7. 异步转换

注：异步转换用于非 springboot 版，springboot 版调用 1.8 批量转换（多线程提交）

Agent 接口的部分方法可完成下载工作，但是此上传及下载是一个同步的过程，容易造成排队，一般用于单文件的即时转换。大批量文件转换过程，建议使用后处理的方式完成下载。后处理接口可获取到转换完成的文件流，通过此接口实现文件的下载，即可节省文件传输时间，又可给集成方提供一个获取文件信息、下载文件到指定路径、更新数据库信息的节点。在进行异步转换，不要求即时结果的情况下，推荐此方式。后处理对应的 Agent 提交方式为 submit 方法（参见 1.5.4 批量转换文件并指定后处理接口）。

1.7.1. 批量提交文件并指定后处理接口

1.7.1.1. 接口描述

实现该功能需引用 packet*.jar,agent*.jar，初始化 packet、agnet 接口（该方法仅限 war 包版转换服务，仅 HTTPAgent 可调用），调用 fileHandler 和 submit 方法。submit 的提交方式，是异步处理的提方交式，配合着后处理方式可实现大批量文档的异步转换。后处理的接口及使用参见 1.5.2 后处理接口。

1.7.1.2. fileHandler 接口定义

```
Packet fileHandler(String name,
                   String fhClass,
                   Pair<String,String>...parameters)
```

参数说明详见下表：

序号	参数名称	参数类型	参数含义	内容举例
1	name	String 类型	后处理操作的名称(可以为空)	String name = "suwell-custom-handler-suwell-compa

				ny"
2	fhClass	String 类型	后处理操作类名称(可以为空)	String fhClass = "mycompany.handler.impl.SimpleFileMetaHandler2"
3	Pair<String,String>...parameters	Pair 类型	自定义的后处理参数（可以为空）	Pair<String,String> p1=new Pair(“1”,”2”);

返回值说明详见下表：

序号	返回类型	返回值含义
1	Packet 类型	Packet 对象

1.7.1.3. submit 接口定义

接口 1：

```
submit(InputStream packet)
```

接口 2：

```
submit(InputStream packet, String token, boolean raise)
```

接口 3：

```
submit(Packet packet)
```

参数说明详见下表：

序号	参数名称	参数类型	参数含义	内容举例
1	packet	InputStream 类型	打包文件流（接口 1，2）	ConvertAgent.*store*(pa)
2	token	String 类型	系统注册的标识（接口 2）。注册方式参见 3.多节点部署及调用。	“8D43C8”
3	raise	boolean 类型	是否将优先级+1（接口 2）	如转换节点 8D43C8，级别：5，如设置升级为 true,则优先级可看成 6
4	packet	Packet 类型	打包文件类（接口 3）	Packet pa =new Packet(PackType.COMMON,Target.OFD);

返回值说明详见下表：

序号	返回类型	返回值含义
1	无	无

1.7.1.4. 接口示例

```
//java 代码调用
//参考示例: com.http.submit.OfficeToOFD_submit1
//参考示例: com.http.submit.OfficeToOFD_submit2
//参考示例: com.http.submit.OfficeToOFD_submit3
//功能说明: 批量转换文件并指定后处理接口
package com.http.submit;
public class OfficeToOFD_ {

HTTPAgent ha = new HTTPAgent("http://localhost:8080/convert-issuer/");
    public void HATest() {

            try {
              //源文件
                File f=new File("D:\\test\\doc");
                int count =0;
                    File[] fs=f.listFiles();
                    for (File file : fs) {
                        Packet pkg = new Packet(PackType.COMMON, Target.OFD);
                        pkg.file(new Common("1", "doc", new FileInputStream(file)));
                            //参数 1: 后处理操作的名称
                            //参数 2: 后处理操作类的名称
                           //参数 3: 自定义的后处理参数
                 pkg.fileHandler("suwell-custom-handler-suwell-company",
                            "mycompany.handler.impl.SimpleFileMetaHandler2", null);
                        //调用 submit 进行文件提交, 参数为 pkg
                        //ha.submit(pkg);
                       //调用 submit 进行文件提交, 参数为流。
                        //ha.submit(ConvertAgent.store(pa));
                         //调用 submit 进行文件提交, 参数为流。
                        if(format.equals("doc")) {
                            //Tocken 指注册的系统标识。在 convert-issuer 服务的主页-系
  统管理-系统注册, 注册, 注册完成系统会分派 tocken,注册时需要设置优先级别。此级别会对应寻找
  级别匹配的节点。如 tocken 为 n1 的系统级别为 2, 则会找优先级为 2-5 或 0-5 的节点处理此文件。
                            //raise  指是否将系统级别+1。意思是如果 n1 级别是 2, 此项为
  true 时, n1 级别为 2+1, 也就是会去找 3-5 优先级的转换节点处理此文件。
                           ha.submit(ConvertAgent.store(pa), "8D43C8", false);
                           System.out.println("-----8D43C8----级别: 5---ini:[5-6)注:
  前开后闭----");
                        }else {
                           ha.submit(ConvertAgent.store(pa), "1823B8", false);
                           System.out.println("-----1823B8---级别: 2---ini:[2-4)注:
  前开后闭----");
```

```
                        }
                    }
        } catch (Exception e) {
            e.printStackTrace();
            } finally {
                try {
                    ha.close(); //文件一定要关闭
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
```

1.7.1.5. 接口约束

无。

1.7.2. 后处理接口实现及部署

1.7.2.1. 接口描述

接口全名为 cpcns.convert.IFileMetaHandler

getProviderInfo 方法

cpcns.defs.ProviderInfo getProviderInfo()

说明:返回实现提供者信息,该接口主要用于在内容服务中接入多个数据源。

返回：实现提供者标识

processFileAndMeta 方法

public String processFileAndMeta(String ticket, String type, String base64Meta, InputStream ofdContent);

说明：转换成功的任务后处理并返回状态代码

参数 ticket　任务标识

参数 type　任务类型（主要用于需要区分优先处理的任务）

参数 base64Meta 元数据 xml 文件形成的 base64 串,xml 的内部格式随应用商定。

参数 ofdContent 转换后文件内容流。

返回：处理状态标识。如果处理成功，返回"6000"，其他情况请自行返回错误码建议返回为 6 开头的四位数字。

processError 方法

public String processError(String ticket, String type, String base64Meta, InputStream zipContent, String errorCode, String errorMsg);

说明：转换失败的任务后处理汇报

参数 ticket 任务标识

参数 type 任务类型（主要用于需要区分优先处理的任务）

参数 base64Meta 元数据 xml 文件形成的 base64 串，xml 的内部格式随应用商定。

参数 zipContent 数据源包内容流。

参数 errorCode 错误代码

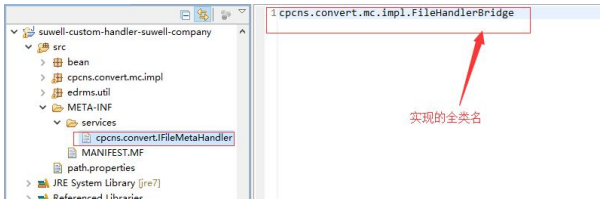
参数 errorMsg 错误信息

返回：处理状态标识。如果处理成功，返回"7000"，其他情况请自行返回错误码建议返回为 7 开头的四位数字。

1.7.2.2. 实现示例

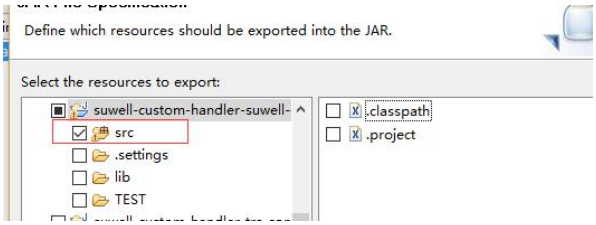
参见代码示例：suwell-custom-handler-suwell-company

注意：接口的注册。在项目的 META-INF 注册实现类的全类名。



1.7.2.3. 接口部署

- 将实现的程序，导出成 suwell-custom-handler-suwell-company.jar，导出时注意去掉除 src 以外的其它选项。



- 将导出的 jar 包放到转换节点 build*\\lib\\extension 里

1.7.2.4. 接口调用

参见 1.7.1.4 批量转换文件并指定后处理接口。

1.8. 批量转换(多线程提交)

1.8.1. 批量转换（多线程提交）

springboot 版本转换服务实现同步转换，可多线程提交转换。线程数建议为转换进程数到转换进程数的 2 倍之间（即 program.ini 中 count 值到 count 值的 2 倍之间），具体转换最佳效果可自主测试。代码示例仅供参考：多线程提交,具体效果与服务器配置，转换文件类型及文件大小相关，需要具体场景具体测试。

1.8.2. 代码示例

```
public class Check2 {
    //定义线程数: 该线程数建议为 build/plugin/program.ini 中 count 值的 2 倍（具体文件对应具体引擎），具体最佳转换效果需要测试验证
    static final int count = 4;
    //可以转换的文件类型集合
    static final List<String> FS = Arrays.asList("doc", "docx",
"wps","wpt","rtf","uot","uof","dot","dotx","docm","dotm","DOC","DOCX","et","ett","x
ls","xlsx","xlt","xlsm","uos","csv","xltx","dif","XLS","XLSX","dps","dpt","dwt","dw
g","ppt","pptx","pot","pps","pptm","potx","potm","ppsx","ppsm","uop","PPT","PPTX",
"xps","pdf","jpe","gif","jpeg","jpg","dib","tiff","tif","png","bmp","ceb","txt","of
d","sep","gd","gw","s72","s92","s10");

    static {
        AtomAgent.init(new DefaultConvertConfig() {

            @Override
```

```
            public int getMaxThreadCount(){
                return count;
            }
            @Override
            public int getConnectTimeoutInMissecond(String s,int i) {
                return 1000000;
            }
            @Override
            public int getReadTimeoutInMissecond(String s,int i) {
                return 1000000;
            }

        });
    }

    public static void main(String[] args) {
    //定义可重用固定线程数的线程池（线程数即定义的 count）
        ExecutorService executor = Executors.newFixedThreadPool(count);
        //定义转换代理对象，用于请求转换服务
        final ConvertAgent ca = new AtomAgent("http://127.0.0.1:8090");
        //获取指定文件夹中的可以转换的文件数组集合（获取文件夹中包含 FS 中所有文件格式的文件）
        File [] files = new File("D:\\2222\\").listFiles(new FilenameFilter() {
            @Override
            public boolean accept(File dir, String name) {
                String n = FilenameUtils.getExtension(name).toLowerCase();
                return FS.contains(n);
            }
        });
        //定义原子操作对象，用于获取当前处理的线程数，该对象线程安全
        final AtomicInteger count = new AtomicInteger(0);
        //获取当前系统时间
        long l = System.currentTimeMillis();
        //当文件数组不为空，即文件夹中有 FS 中包含类型的文件
        if(files!=null){
         //循环集合中的文件
            for (final File file : files){
              //将任务委托给
                executor.execute(new Runnable() {
                    @Override
                    public void run() {
                        try {
                            //输出当前正在转换的文件
                            System.out.println("开始转换 ".concat(file.getName()));
                            //定义转换后的输出 OFD 文件
```

```
                        File out = new File("D:\\ofd\\"+file.getName()+".ofd");
                        //定义数据包
                        Packet packet = new Packet(Const.PackType.COMMON,
Const.Target.OFD);
                        //数据包存储原文件
                        packet.file(new Common(file.getName(),
FilenameUtils.getExtension(file.getName()), 0, PackEntry.wrap(file)));
                        //定义转换后 OFD 文件的文件输出流
                        FileOutputStream outputStream = new FileOutputStream(out);
                        //转换代理对象请求转换服务，执行文件转换操作
                        ca.convert(packet,outputStream);
                        //关闭资源
                        IOUtils.closeQuietly(outputStream);
                        //输出当前转换完成的文件信息
                        System.out.println("转换完成 ".concat(file.getName())+" 第
"+count.incrementAndGet()+"个");
                    }catch (Exception e){
                        e.printStackTrace();
                    }

                }
            });
        }
    }
    //关闭线程池
    executor.shutdown();
    try {
     //设置主线程最长等待时间，超过该设置的等待时间将自动销毁
        executor.awaitTermination(3, TimeUnit.HOURS);
    }catch (Exception e){
     e.printStackTrace();
    }
    System.out.println("转换完毕 总耗时 : "+(System.currentTimeMillis()-l)+"ms");
    //关闭资源（http 请求转换服务代理对象）(它将无条件的关闭一个可被关闭的对象而不抛出任
何异常。)
    IOUtils.closeQuietly(ca);

  }
}
```

2. 附件1：公文元数据表

数据类型	ID	定义
公文元数据	GWBS	公文标识
	WZ	文种
	FH	份号
	MJHBMQX	密级和保密期限
	JJCD	紧急程度
	FWJGBZ	发文机关标志
	FWZH	发文字号
	QFR	签发人
	BT	标题
	ZSJG	主送机关
	FJSM	附件说明
	FWJGSM	发文机关署名
	CWRQ	成文日期
	FZ	附注
	CSJG	抄送机关
	YFJG	印发机关
	YFRQ	印发日期
	FBCC	发布层次
标准元数据	Const.Meta.DOC_ID.value()	DOCID
	Const.Meta.AUTHOR.value()	作者
	Const.Meta.MOD_DATE.value()	修改日期
	Const.Meta.TITLE.value()	标题
	Const.Meta.ABSTRACT.value()	摘要
	Const.Meta.SUBJECT.value()	主题
	Const.Meta.DOC_USAGE.value()	文档类型
	Const.Meta.KEYWORD.value()	关键字
	Const.Meta.KEYWORDS.value()	关键字集，用“，”分隔

3. 附件2：异常错误码对应说明

3.1. 转换服务异常错误码对应说明(211222 以前 war 版本)

异常标识	异常码对应说明
源包处理异常状态码及说明	
2000	数据源包错误：无法在服务器上找到指定的源数据包
2001	数据源包不是一个标准的 ZIP 文件
2002	无法从数据源包中找到 Main.xml 文件
2003	数据源包中的 XML 无法解析：XML 不符合规范或字

	符编码错误
2004	找不到数据处理器：Main.xml 中根节点的 Type 属性指定的转换处理器没有注册
2005	数据无法访问：引用的内部文件无法在 ZIP 中找到;使用了外部数据,但使用提供的 URL 无法获取内容
2006	无法获取模板：在数据源包或服务器上未能找到指定的模板文件
2007	附件无法访问
2008	无法找到有效的后处理实现
2020	数据不存在：无法从数据库中查询到指定的任务数据
转换器（引擎/节点）异常状态码及说明	
3000	转换器错误：转换处理器在执行过程中抛出了异常导致无法正常完成转换
3001	添加文件失败
3002	添加模板失败
3003	处理模版数据出错
3004	创建临时结果文件失败
3100	其他错误引起的转换错误
3101	转换结束但结果为空
3102	转换超时
3500	无法获取插件的对象池
3600	签名/签章失败
后处理异常状态码及说明	
4000	后处理错误:后处理接口在处理文件时出错
4001	后处理超时:后处理接口在处理文件时超时
5000	未知错误

3.2. 转换服务状态码对应说明(211222 以后 war 和 Springboot 统一)

状态码	是否处理完成	任务是否成功	状态码说明
1002	×	×	已进入队列
1003	×	×	正在处理
1010	√	√	处理完成
1011	√	×	处理失败
2000	√	×	数据源包错误
2001	√	×	数据源包不是一个标准的 ZIP 文件
2002	√	×	无法从数据源包中找到 Main.xml 文件
2003	√	×	数据源包中的 XML 无法解析
2004	√	×	找不到数据处理器
2005	√	×	数据无法访问
2006	√	×	无法获取模板

2007	√	×	附件无法访问
2008	√	×	无法找到有效的后处理实现
2020	√	×	数据不存在
3000	√	×	转换器错误
3001	√	×	添加文件失败
3002	√	×	添加模板失败
3003	√	×	处理模版数据出错
3004	√	×	创建临时结果文件失败
3100	√	×	其他错误引起的转换错误
3102	√	×	转换超时
3104	√	×	添加水印失败
3105	√	×	添加元数据失败
3106	√	×	文档操作失败
3107	√	×	格式不支持
3500	√	×	无法获取插件的对象池
3600	√	×	签名/签章失败
4000	√	×	后处理错误
4001	√	×	后处理超时
5000	√	×	未知错误

转换任务状态码查询路径：

http://ip:port/query?ticket=任务 ticket

Springboot 版本状态码查询 URL 示例：

http://127.0.0.1:8090/query?ticket=20220222-140926-250979980