1 word转html

项目地址：<http://git.zhihuishu.com/coder/doctrans.git>

* 1. 预备知识

参考博客：<https://www.cnblogs.com/ct-csu/p/8178932.html>

word文档有两种存储格式：doc和docx

doc：习惯上称为Word2003，使用二进制储存数据，这个不是讨论的重点.

docx：word2007，使用xml来存储数据和格式.

明明是docx结尾的文档，怎么成了xml格式了？

很简单，你随便选择一个docx文件，右键使用压缩工具打开，就能得到一个这样的目录结构：



所以docx其实只是一个压缩文件。

要了解更多OpenXML相关知识，可以进入poi官网或者查阅附件中的官方手册《Office Open XML Part 1》

* 1. 技术选择

由于我们只使用Java作为开发语言，所以只讨论Java的相关技术实现。Apache poi和docx4j都可以用来解析OpenXML（docx，pptx，xslx），在这里我们选择了前者，虽然从功能介绍来看后者看似更吸引人，但我们对poi还是更熟悉一些。不过docx4j的文档还是值得一看的，它介绍了一些OpenXML的相关概念。

docx4j项目地址：<https://github.com/plutext/docx4j>

docx4j相比于poi的优势：<https://stackoverflow.com/questions/15013837/apache-poi-or-docx4j-for-dealing-with-docx-documents>

poi并没有提供转换html的功能，我们还需要使用借助xdocreport。Xdocreport简介摘抄：

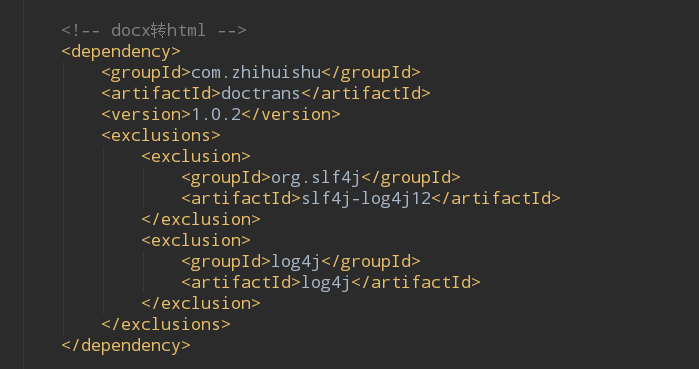
XDocReport means XML Document reporting. It's Java API to merge XML document created with MS Office (docx) or OpenOffice (odt), LibreOffice (odt) with a Java model to generate report and convert it if you need to another format (PDF, XHTML...).

xdocreport项目地址：<https://github.com/opensagres/xdocreport>

* 1. 程序设计
     1. 如何在web项目中使用doctrans

①配置环境（具体依赖的环境下一小节介绍）

②导入依赖的jar包

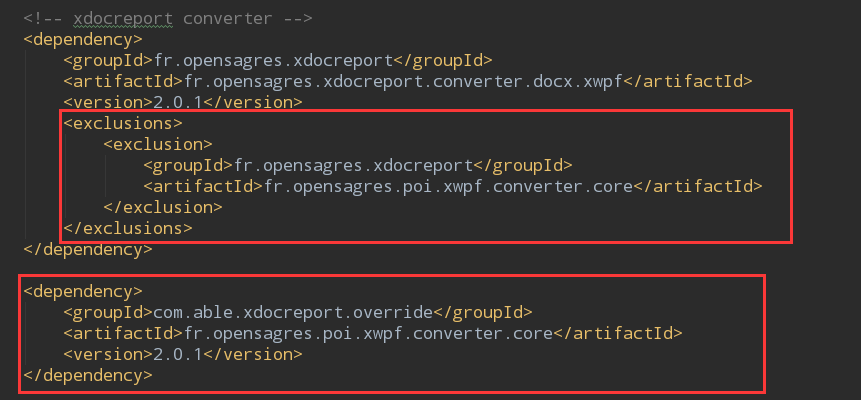


③调用核心转换方法



* + 1. doctrans项目依赖环境

doctrans的设计初衷是可以以jar包的形式导入任意项目中使用，但实际设计的过程中遇到了一些问题，这些问题使得项目必须配置相关依赖环境才能正常使用导入功能。doctrans项目的pom.xml文件中有几个依赖包需要注意：



这段配置导入xdocreport包后排除了core依赖包，同时又从公司的maven仓库导入了另一份core依赖包（注意groupId是com.able...）。这是因为源码core模块存在一个小BUG，没有捕获一个小异常，我对这个BUG进行简单修复后将core模块发布到了公司的maven仓库，所以**项目的运行环境必须能连接公司的Nexus服务器**。

word转html后图片必须上传到公网能够访问的OSS服务器，下面的依赖包跟图片上传相关，所以**项目的运行环境必须能够正常访问公司的OSS服务器**。

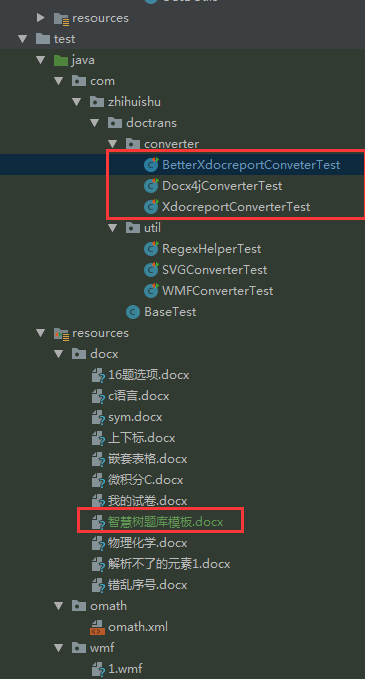


word中存在wmf格式的矢量图，但目前没有找到较好的wmf转png相关java类库，所以这一块功能需要借助C#服务器来完成。下图这个类将wmf图片作为参数通过http请求C#服务器将图片转换为png格式，所以**项目的运行环境必须正确配置wmf图片转换服务器的host**。如果只有wmf图片转换出了问题，大多是这个原因导致的。

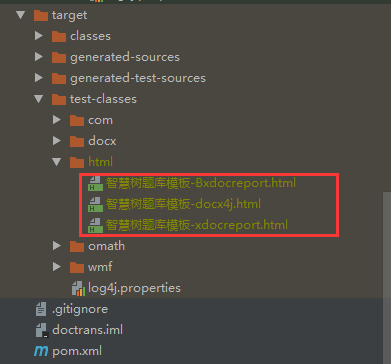


* + 1. doctrans项目测试类

可以在doctrans项目中运行测试类来查看转换效果。有3个较为重要的测试类：BetterXdocreportConveterTest、XdocreportConveterTest、Docx4jConverterTest，BetterXdocreportConveterTest测试的是我们实际使用BetterXdocreportConveter类的转换效果；XdocreportConveterTest测试的是直接使用Java API转换wmf格式图片的效果；Docx4jConverter没有对图片进行处理，Docx4jConverterTest主要用来比较docx4j和xdocreport两者的转换效果。

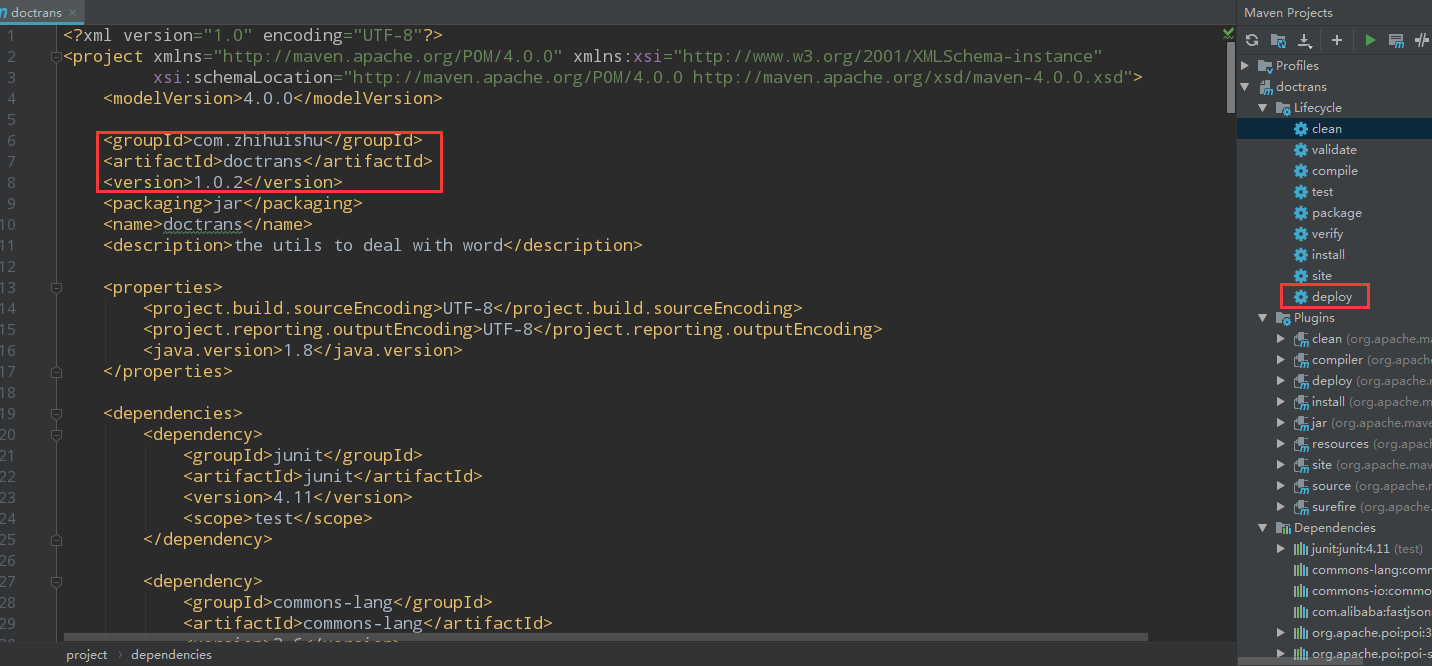


运行测试类后，html格式文件生成在如下位置，可打开查看转换效果。

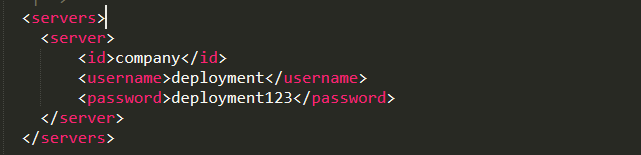


* + 1. doctrans项目部署

运行测试类并确认转换无误后，修改项目的版本号，通过maven的deploy插件自动将项目打包上传到公司的nexus服务器，以供其它项目依赖。



本地maven配置文件还需要有这一项，也就是连接nexus服务器的认证。



* + 1. doctrans项目核心设计

AbstractDocxConverter提供了word转html的一些通用方法，如wmf转png、omath转png等。其convert()方法是抽象的，供docx4j和xdocreport提供不同的实现。BetterXdocreportConveter相对于XdocreportConverter只做了一处改进，其覆盖了父类的convertWMFToPNG()方法，采用了上传wmf图片到C#服务器来进行png转换的方式。整个程序的核心流程可参阅XdocreportConverter中的convert()方法实现。

xdocreport提供的html转换方法存在很多瑕疵，除了上述提到的小BUG外，它还不能解析docx文档中的矢量图、公式等。其实根本不可能完美的做到docx转html，包括word自带的转换工具面对复杂的文档时也束手无策。在此，我们只解决最基本且核心的问题，也就是想办法转换docx文档中的矢量图、公式。对于一些复杂的样式问题，xdocreport没能解决的我们也无从下手。

要让xdocreport解析docx文档中的矢量图、公式，首先想到的方法是修改扩展它的源码，但这么做成本太高，需要把xdocreport的核心源码都过一遍，如果不小心还容易改出BUG。在这里我们使用了最简单暴力的方法，在调用xdocreport的转换方法之前，先遍历document，找出并提取矢量图、公式的元数据，将元数据转换为png格式的图片，然后把这块区域替换为一个字符串形式的占位符。上一步的工作完成后就可以调用xdocreport的转换方法了，得到html字符串，此时html字符串中包含了我们之前设置的占位符，最后将占位符替换为png图片即可。

提取矢量图、公式需要我们对OpenXML有一定的了解，博客<https://www.cnblogs.com/ct-csu/p/8178932.html>提供了很好的示例，项目中的大多地方也都是参照博客和附件中的官方手册《Office Open XML Part 1》来提取所需元素的。

wmf转png前面已经提到过，直接交给C#服务来做就好，现在主要说明下公式转png的实现。首先要知道什么是MathML，MathML是一个用于描述数学公式、符号的一种XML标记语言。omath是在office2007之后版本所编辑的公式对象，是office为了配合Office Open Xml制定的数学标记语言。我们要先把omath转换为MathML，office在安装目录中提供了转换所需的xslt工具OMML2MML.XML，直接将其拷贝到项目中使用。得到MathML后，使用JEuclid（https://www.oschina.net/p/jeuclid）渲染成png图片。

* + 1. doctrans项目总结

docx转html本身就是一件不合常理的事，对稍微复杂一些的文档转换效果往往不是很理想，甚至令人大失所望。我们依赖的是xdocreport，要是它不给力我们也束手无策，但不可否认xdocreport确实很了不起，必须具有完备的OpenXML还有web前端相关知识才能写出这样的类库，就算真的具备这样的能力，开发所花费的时间也是不可估量的。要是对xdocreport抱有怀疑态度，可以尝试使用docx4j，甚至考虑使用C#相关的类库。

xdocreport是有很多瑕疵的，与其这么说倒不如怪docx文档格式千变万化，不可能把所有的情况都考虑全。做到这一步已经是使出了浑身解数，再进行优化的话需要查阅大量的资料，而且最终能不能做出来还不一定。我们都不是这方面的专家（或许WPS的员工才是？），后续真的不建议把时间浪费在这了。

2 题目识别

项目地址：<http://git.zhihuishu.com/coder/AT_homeworkExam_h5.git>

主要文件：importQuestion.vue

/src/assets/js/questionRecognition/parser.js

/src/assets/js/questionRecognition/util.js

2.1 识别难点

有了html后并不代表万事俱备，识别也是一个大难点。识别的主要矛盾在于，简单的内容容易识别，但要保持原汁原味的html样式又会加大识别难度，这是不好取舍的。例如大量的”<div>”、”<p>”冗杂在一起，你怎么去确定什么时候换行呢？”<img>”标签为悬浮样式时，你又怎么确定图片是挂在那一道题呢？或许只有设计过浏览器渲染的人才能很好的解决类似问题，我对前端一窍不通，所以采取了最暴力的方法（这不值得效仿，实际出来的效果也是差强人意，如果有更好的方法应该去尝试下）。

2.2 我的设计

首先把不重要的元素全都干掉，除了”<img>”、”<table>”需要保持原汁原味外，其它的元素基本就是直接删掉这么简单。”<div>”和”<p>”元素是需要注意的，它们决定了该如何换行，我用”<br>”来表示换行，换两行就用两个”<br>”，多个”<div>”或者”<p>”标签组合在一起时怎么判断换行？这个是仅凭我的臆想加上实战经验来定的。经过上述的暴力处理，html变成了只包含4种元素的伪html：img、br、table、text。格式是变得简单了，但样式也已经面目全非，例如”<span style="font-family:'Symbol';white-space:pre-wrap;">^%^%%(!$@#</span>”，去掉了span标签后，原本定义的字体也失效了，显示出来的字符完全不一样。

针对伪html的处理相对简单一些，只需要根据产品定义的识别规则，配合正则表达式进行解析就好，遇到”<br>”就是换行，遇到”<img>”或者”<table>”保持不动就行。

顺便说下错题标红和文本编辑框也是个坑，我甚至都回想不起来是怎么完成这块的。文本编辑框飘忽不定，往里面输入回车有时会创建一块新的div，有时仅仅是加上一个br标签。

2.3 题目识别总结

可能是我一开始的解决思路有问题，又或者这类问题需要更高端的解决方案，总之我感觉做出来的效果不是很好，可扩展性也很差。总之这一块要是有充足的时间，并且有较好的解决思路，可以重构一下。可以对比一下转换后的html和实际识别出来的题目，两者之间的差距就是优化空间。