PDF 阅读器开发报告

马啸远 3140104199

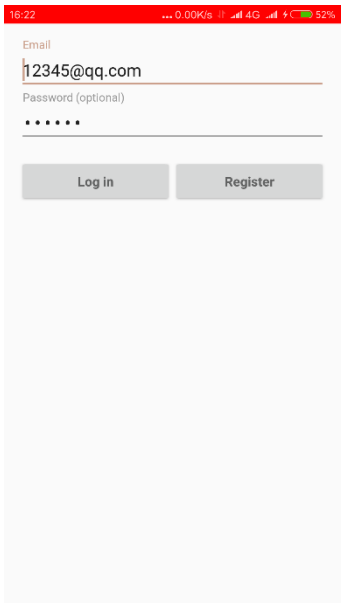
庞博 3140103364

智能终端软件开发

目录

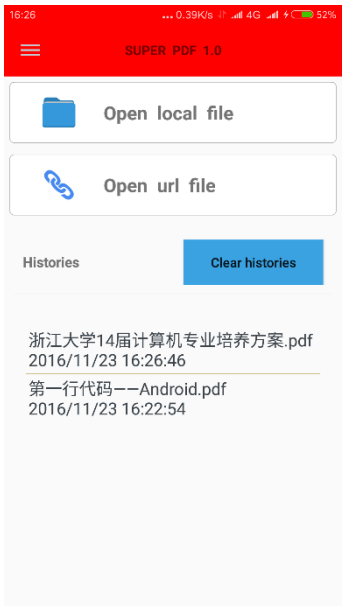
一、应用功能介绍

我们的应用《Super PDF 1.0》是基于开源库 **PSPdfKit** 实现的，基本实现了打开本地 pdf 文件、打开在线 pdf 文件，用多种方式编辑 pdf，创建小组、加入小组进行多人在线协作编辑等功能。下面通过图示介绍一下这款应用的功能和使用方法。



这是打开应用后首先出现的登录界面。

用户名必须是以前注册过的邮箱。没有注册过的，也可以进行注册。

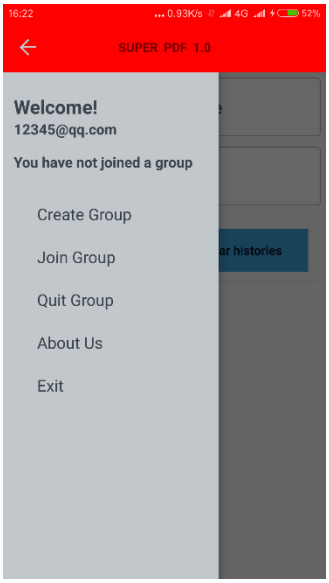


登录进入主界面。

最上面的“Open local file”可以打开在线的 pdf 文件。下面的“Open url file”用来打开一个在线的 pdf 文件，用户需要输入一个在线 pdf 的 URL，类似下面这个链接：

http://59.78.108.56/msta/MYCOURSES/dytjfxkj.pdf

再向下是打开文件的历史记录，可以看到打开过 2 个文件，并且还记录了最近一次编辑



的时间。通过点击历史记录，可以直接打开对应的文件。

点击“Clear histories”可以清除所有历史记录。

在主界面的左侧有一个侧滑菜单。

最上面显示了当前登录用户。

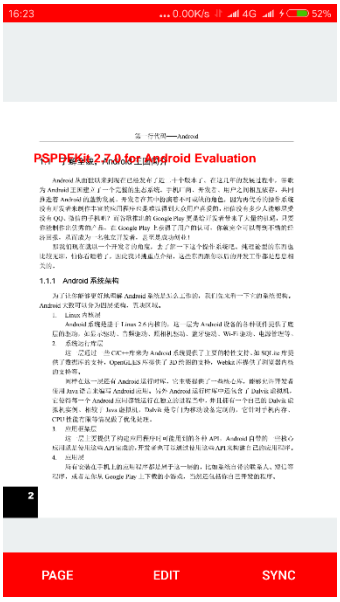
由于目前还没有加入任何组，所以显示“You have not joined a group”。如果加入一个组，这里的文字会变成关于组的信息。

再往下依次是：创建一个组、加入组、退出当前组、关于我们、退出应用。

创建组，需要用户绑定一个本地的 pdf 文件，作为共同编辑的源文件，然后用户会获得一个拥有唯一 id 号的组。

加入组，需要用户输入一个已存在的组的 id。

退出组，如果是组内的成员退出，直接退出即可；如果是创建者退出了组，那么这个组会被删除，其他组内成员也都退出组。



接下来是关于 PDF 的编辑。

打开 PDF 文件。

左下角的按钮“PAGE”，可以用来跳转，输入页码即可。如果输入负数回到第一页，页数过大就到最后一页。

中间的“EDIT”按钮用来对 PDF 进行编辑。

右下角的“SYNC”按钮，用于在线协同编辑时记录的同步。点击后包含两个动作，把自己新添加的注释、涂鸦等编辑记录上传，以及把其他人添加的从服务器下载并加入 PDF 中。

编辑 PDF。

在编辑模式下，可以在页面上添加文字、高亮文本、涂鸦、加入注释。还可以选择颜色。

以上就是应用的功能简介。具体的使用方法也可以参考我们录制的小视频。

二、技术要点

本章将对应用开发中的一些技术要点、设计方式进行详解。

1. Util 类

`Util` 类类似于一个工具类，里面放了一些在其他类中都可能用到了的常数、方法等。

```
private static final String HOST = "http://222.205.46.130:3000";
public static final String URL_ACCOUNT = HOST + "/accounts";
public static final String URL_ANNOTATION = HOST + "/annotations";
public static final String URL_GROUP = HOST + "/groups";


public static final int REQUEST_OPEN_DOCUMENT = 1;
public static final int REQUEST_ASK_FOR_PERMISSION = 2;
public static final int REQUEST_LOGIN = 3;
public static final int REQUEST_CREATE_GROUP = 4;
```

上面显示了一些常数，包括服务器的主机名，还有几个 request_code，用来在 intent 隐式打开活动后通过这个参数获得子活动传回的数据。

```
public static void showOpenFileDialog(@NonNull Activity activity, int requestCode){
    Intent intent = new Intent(Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT ?
            Intent.ACTION_OPEN_DOCUMENT : Intent.ACTION_GET_CONTENT);
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    intent.setType("*/*");

    activity.startActivityForResult(intent, requestCode);

}
```

这个方法用来调用系统中打开文件系统的活动。可以看到，在 intent 添加了打开文件的 action，安卓系统中自带的活动将会响应，然后向主活动返回文件的 URI，而获得这个数据就要用到刚才介绍的 request_code。

```
public static String getStringFromInputStream(InputStream is) throws IOException{

    StringBuilder sb = new StringBuilder();
    BufferedReader br = new BufferedReader(new InputStreamReader(is));
    String line = null;

    while ( (line = br.readLine()) != null ){
        sb.append(line);
    }

    is.close();
    br.close();
    return sb.toString();
}
```

这个方法是从输入流中得到字符串。在从网络端口获取数据时需要用到。

```
//获取当前时间
public static String getTime() {
    Date now = new Date();
    return getTimeSimple(now);
}

public static String getTimeSimple(Date date){
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    return dateFormat.format(date);
}
```

这两个方法可以用来获取格式化的系统时间，在历史记录的保存和显示中会用到。

```
public static void userLogin(@NonNull Activity activity) {
    Intent intent = new Intent(activity, LoginActivity.class);

    activity.startActivityForResult(intent, REQUEST_LOGIN);
}
```

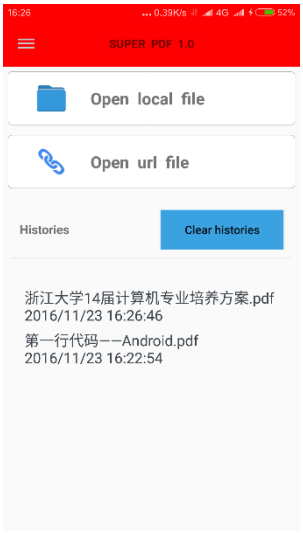
这个方法顾名思义，就是提供了用户登录的接口。其实就是通过 intent 启动了登录这个活动。之后这个活动会传回登录用户的信息。

```
public static String objectToJson(Object obj){
    try{
        return mapper.writeValueAsString(obj);
    }
    catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
}

public static Object jsonToObject(String json, Class cls){
    try{
        Object obj = mapper.readValue(json, cls);
          Log.v(LOG_TAG, "content = " + json);
        return obj;
    }
    catch (IOException ex){
        ex.printStackTrace();
        return null;
    }
}
```

这两个类用于“序列化”和“反序列化”，通过把类转变成 JSON 和把 JSON 转变会类可以方便地传送数据。

2. 历史记录的实现



历史记录的显示利用了安卓视图中的 ListView。List 中的每个子项的布局就是简单的字符串，包括文件名和最新编辑的时间。

历史记录用到了自己定义的 HistoryManager 类，用来对历史进行管理。

下面对这个类进行详细介绍。

```
private List<History> mHistories;
private ArrayList<String> mHistoryListData = new ArrayList<>();
```

mHistories 容器存放了所有的 history—这是个自定义的类，这里就不展开介绍了。MHistoryListData 是一个 ArrayList，用来在之后作为显示界面中 ListView 的数据适配器的源数据。

```
public void initHistories(Context context) {
    SharedPreferences pref = context.getSharedPreferences("history",
            MODE_PRIVATE);

    int i = 0;
    String formedHistory = pref.getString("history" + i, null);
    while (formedHistory != null) {
        String[] tmp = formedHistory.split("\\$");
        String time = tmp[0];
        String pdfName = "";
        int length = tmp.length;
        for(int j = 1; j < length; j++) {
            pdfName += tmp[j];
        }

        Log.d("test", time + " " + pdfName);

        addHistory(time, pdfName);

        i++;
        formedHistory = pref.getString("history" + i, null);
    }
}
```

initHistories()方法用来在应用刚打开时读取存储在本地的文件，初始化第一个容器。数据的存取用的是安卓的 SharedPreferences，它可以方便地存取键值对，因此使用起来相当方便。这个方法就是获取所有键值对，然后构建 history 对象加入容器。

```
public void saveHistories(Context context) {
    SharedPreferences.Editor editor = context.getSharedPreferences("history", MODE_PRIVATE).edit();
    editor.clear();

    int i = 0;
    for(History history : mHistories) {
        Log.d("test", history.getFormedHistory());

        editor.putString("history" + i, history.getFormedHistory());
        i++;
    }

    editor.commit();
}
```

SaveHistories()方法用来在应用退出时将新的历史容器中的数据写到本地文件。而 getFormedHistory 是 History 类中的方法，用来得到一个格式化的字符串。

```
public void addHistory(String time, String pdfName) {
    //delete same pdf history
    int i = 0;
    for(History history : mHistories) {
        if(history.getPDFName().equals(pdfName)) {
            int size = mHistories.size();
            mHistories.remove(i);
            mHistoryListData.remove(size - i - 1);
            break;
        }
        i++;
    }

    History history = new History(time, pdfName);
    mHistories.add(history);
    mHistoryListData.add(0, history.getListviewHistory());
}
```

AddHistory()方法用来添加新的历史记录，每次打开新的 PDF 文件都会调用它。首先要先判断打开的这个文件是否已经存在，如果存在，先从容器中删除它，因为时间更新了。

还有一些其他的方法，这里不用一一介绍。

还有要注意的问题就是编码问题，URI 得到的字符串是编码过的，因此中文无法正常显示，所以要解码后再显示：

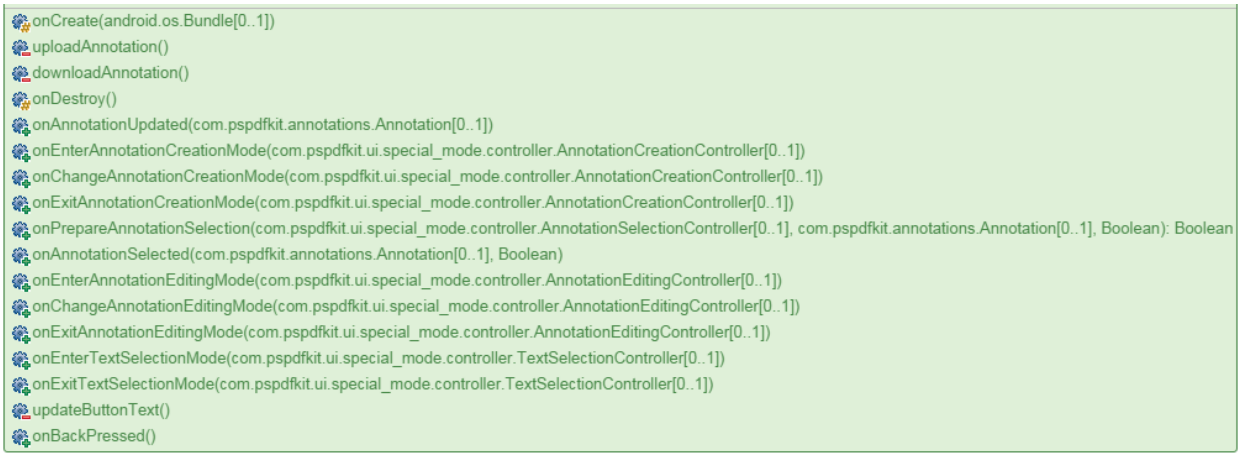
```
URLDecoder.decode(mPDFName, "utf-8");
```

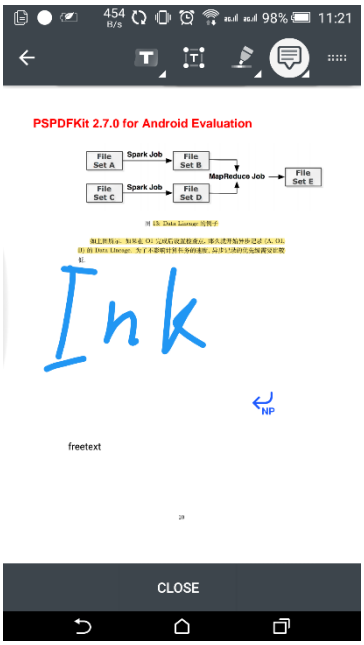
3. PDF 解析渲染+添加标记

由于 PDF 标准过于复杂，我们使用了一个商业的 PDF 解析绘制 SDK（试用）—PSPDFKit 2.7 For Android 来实现 App 的 PDF 阅读功能。

在具体实现中，主要把 PDF 阅读，编辑等与 PDF 相关的操作统一实现在 PDFViewActivity 中。该类的具体设计如下所示：

成员方法：





该 activity 中的 fragment 成员是 PSPDFKit 提供的 PSPDFFragment 类的一个实例。正如类名所表示的，它继承自 Android 的 Fragment 类，可以放在 Activity 中，并且封装了对 PDF 的大部分操作。

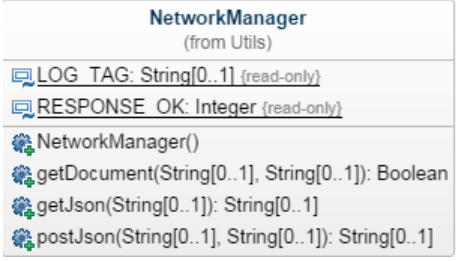
在 PDFViewActivity 打开后，在其中调用 getIntent().getParcelableExtra(EXTRA_URI) 获取得到 MainActivity 传入的 pdf 文件 URI，并通过调用 PSPDFFragment 类的静态方法 PSPDFFragment.newInstance()来生成一个 fragment 实例，由它进行 PDF 的渲染。

在 PDF 编辑方面，同样是利用 SDK 提供的几种标记（Annotation 对象），包括文本框（FreeTextAnnotation），笔记注释（NoteAnnotation），墨迹注释（InkAnnotaition），高亮（HighlightAnnotation），如右图所示。

标记的添加通过 SDK 的另一个控件工具栏，即 PDFViewActivity 中的 annotationCreationToolbar 成员来管理。在点击该工具栏的某个图标选择标记种类之后，在 Fragment（即 PDF 显示的区域内）中点击或者滑动，Fragment 即可处理该事件，并产生相应的标记。产生的 Annotation 对象更准确的说应该是 Annotation 类的某个子类的对象，由用户添加的标记类型决定，该对象中包含关于该 Annotation 的若干属性，包括创建者，创建时间等。

4. 多人在线阅读及标记同步

首先，由于需要实现客户端与服务端的通信，为了尽可能高的代码复用性，我们设计了

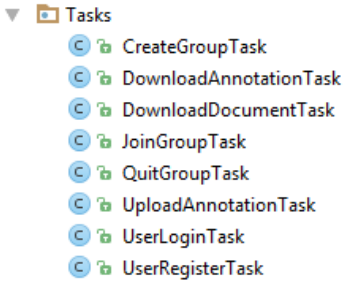


一个类 NetworkManager 专门用于处理网络请求的收发：

由于通信数据均采用 JSON，因此该类主要提供了 getJson 与 postJson 两个成员方法，

分别用于发送 GET 与 POST 请求，需要的参数为请求的 URL，而 postJson 还需要第二个参数即需要添加在 Request Body 中的 JSON 数据。实现上主要依赖于 Android SDK 提供的 HttpUrlConnection 类。

其次，所有的网络 IO 都必须异步执行。由于每个请求（登录、创建组、标记同步等）返回的 Json 数据类型不一定完全一致，因此针对不同的请求任务，我们都实现了一个继承



自 AsyncTask 的 Task 类，用于执行异步任务。

这些类都需要重载 AsyncTask 的 onPreExecute，doInBackground，onPostExecute 等接口，主要任务在 doInBackground 函数中完成（也只有这个函数会开启额外的线程执行）。每个 Activity 中需要进行某个 Task 时，就实例化其中的某个 Task 对象，重载 onPostExecute 接口以便执行更新该 Activity 的操作，再调用其 Execute 函数开始执行操作。

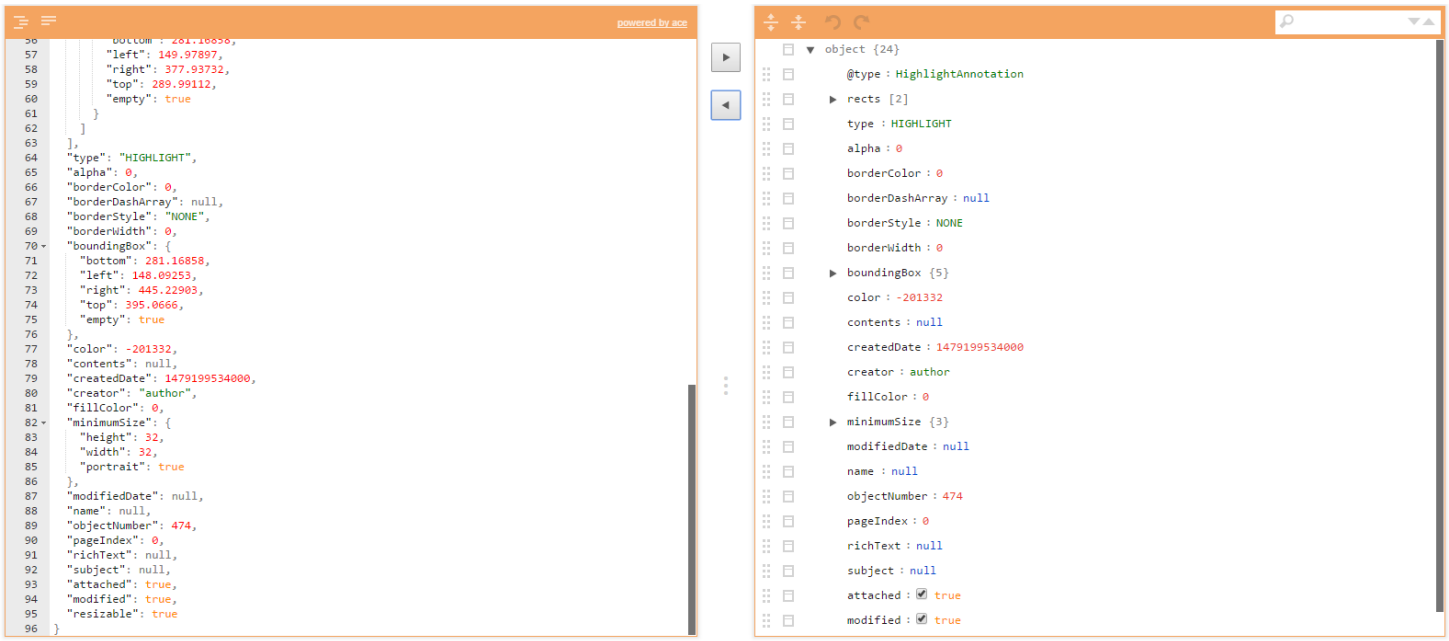
1）标记同步

该功能可以细分为两个阶段。首先是客户端上传标记，需要在客户端获取某一页上的所有 Annotation 对象，将其序列化为 JSON 后上传到服务器。第二阶段是客户端从服务器下载他人所上传的标记，需要客户端发送同步请求，服务端接收后，将服务器上的标记筛选出符合同步条件的标记（JSON 数据）发送给客户端，客户端将 JSON 反序列化成 Annotation 对象，再用 SDK 的 Fragment 提供的 AddAannotation 方法将该对象添加到本地 PDF 文件上。该功能是该 App 实现的关键，也是最难的一部分，由于涉及，因此会分阶段重点描述。

要将标记同步到服务器端，必须先在客户端获取该标记，最初步的想法就是在标记产生的时候捕获并上传。该 SDK 为其 Fragment 类提供了多个接口，以便用户能够自己实现某些事件的 Listenner 以便自己处理事件，包括 AnnotationUpdate（标记被修改时触发），TextSelect（选中某些文字时触发）等，但唯独没有提供监听 Annotation 产生的 Listenner。不过，SDK 也提供了遍历某一页上所有 Annotation 的接口，这样一来我们可以设置一个 Button，在用户点击的时候遍历该页上的 Annotation 对象，逐一序列化并上

传到服务器。

在序列化过程中，使用的是开源的 Java 对象序列化 SDK，Jackson。它提供了较为灵活的序列化、反序列化方式，包括自定义的序列化、反序列化函数，在序列化时自动加入对象类型，以及忽略某个类中的属性等功能。一个 Annotation 对象序列化之后得到的数据如下所示：



Jackson 的默认序列化方式可用性还是比较高的，但序列化的数据必须保证能够反序列化。但由于存在 4 种类型的标记，并且它们都继承自同一个抽象类，因此反序列化时需要能够从 JSON 中获取类型信息。可能由于这些对象中的成员类型比较复杂，在使用 Jackson 默认的反序列化函数时出现了许多问题，因此我们自己实现了一个反序列化器 AnnotationDeserializer，并设置 Jakcson 在反序列化 Annotation 子类对象时必须调用该反序列化器来进行反序列化。对于其他几个较为复杂的类我们也实现了几个反序列化器。

获取 Annotation 对象的数据之后，还需要上传到服务器。但是为了服务器筛选方便，上传时还需要注明该 Annotation 的创建者（Creator），以及创建者所在的小组（Group）ID。因此，实现了 AnnotationData 类，该类用于封装以上信息以便服务器解析。

由于所有数据传输时统一用 JSON 格式，因此 AnnotationData 封装了一个 Annotation 及相关属性之后，客户端需要把 AnnotationData 序列化后的数据通过 POST 请求传输给服务器，服务器只需要将 JSON 中的各个属性（Creator，Group，AnnotationJson）读出来

之后，存到数据库中即可。具体的服务器实现细节在后面会描述。

客户端需要从服务器同步标记时，需要先发送 POST 请求给服务器，发送的数据包括当前小组的 ID 以及该用户的 ID，仍然用 AnnotationData 来做封装（虽然 Annotation 的信息是空的）并序列化发送给服务器。服务器接收到请求后，从 JSON 解析出小组 ID 及用户 ID，从数据库中筛选出该用户没有同步过的 Annotation 数据，并把结果放在一个数组形式的 JSON 中返回。

客户端接收到请求返回的 JSON 数据后，以 List<Annotation>的形式对 JSON 进行反序列化（Annotation 对象的反序列化调用自定义的反序列化器），并将获得的 Annotation 数组中的 Annotation 逐个添加到用户当前浏览的 PDF 中。

2）多人在线阅读

该功能可以主要分为两个子功能，包括创建组与加入组。

创建组时，客户端首先让创建者从文件管理器中选择一个 PDF 文件作为需要分享的 PDF，再随机产生一个组的 ID。利用这两个数据以及 PDF 文件名，创建者 ID 共四项数据创建一个 Group 对象，将该对象序列化为 JSON 后放在 POST 请求中同步到服务器。

由于 PDF 文件以其特有的标准而非普通文本存储，因此在序列化过程中转化为 String 时会变成乱码，因此在序列化之前需要把 PDF 文件的数据编码为网络传输中常用的 BASE64 编码。相应的反序列化时也需要进行解码。编码与解码均使用 JDK 提供的 Base64 类实现。创建组成功后，用户能在侧滑菜单中看到自己的组号。

在加入组时，用户需要输入目标组号，并发送 GET 请求。服务端会返回该组创建者上传的 PDF 文件数据，客户端下载后解码 BASE64 字符串获得实际的 PDF 文件数据写入文件，之后只需要把文件 URI 以及用户 ID，组 ID 等信息传给 PDFViewActivity 即可。

为了避免不必要的操作，PDFViewActivity 用 Boolean 变量 isOnline 来记录用户是在线阅读还是离线阅读本地 PDF。只有 PDFViewActivity 获得了从 MainActivity 中传入的 GroupID 时这个变量值才为真，也只有这个值为真时才会执行标记的同步等操作。

三、服务器端实现

1. 服务器架构

服务器端主要采用 node.js+express4 框架，MVC 架构实现。服务器端需要实现以下几个功能：

- 用户的注册与登录
- 在线小组（Group）的创建，加入，删除操作
- 加入小组后用户标记的上传与下载同步

针对以上功能，我们需要利用 Controller 提供以下 URL 接口：

登录：POST http://hostname/accounts

注册：POST http://hostname/accousnts/register

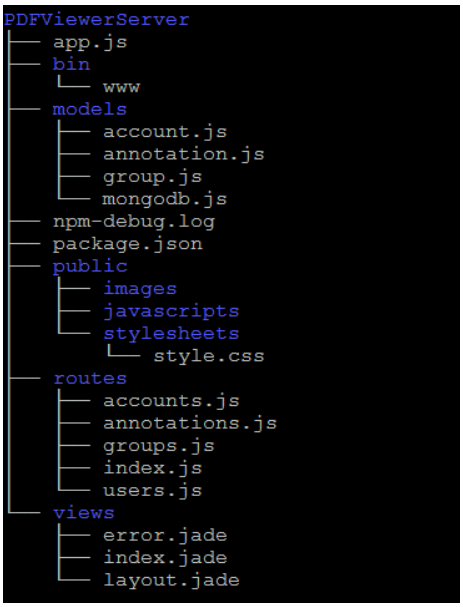
上传标记：POST http://hostname/annotations

下载标记：POST http://hostname/annotations/sync

创建小组：POST http://hostname/groups

加入小组：GET http://hostname/groups/:group_id

2. 源代码结构



3. 数据库设计：

在 Model 的实现中，使用 MongoDB 作为后端存储，以便能够直接利用 nodejs 的 mongoose 模块操作数据库而不需要写 Query 操作数据库。数据库中需要存储的对象包括 Account，Group 以及 Annotation 几种数据类型，与客户端中相应类的定义一致。具体定义如下：

```
var AccountSchema = new Schema({
    id : String,  // 用户登录ID
    currentGroupId : String, // 用户当前组号，默认为null
    password : String // 密码，BASE64保存
})

var AnnotationSchema = new Schema({
    id : String,  // Annotation的ID，由数据库生成
    groupId : String,  // 创建者组号
```

```
    account : String,  // 创建者
    jsonData : String, // Annotation对象序列化数据
    hasSync : Array // 数组，元素为已经拥有该Annotation的用户ID
})

var GroupSchema = new Schema({
    id : String,        // 组的ID，由客户端随机生成
    pdfData : String, // 该组创建者所浏览的PDF文件数据，BASE64编码
    fileName : String,// 该组创建者所浏览的文件名
    creator : String   // 创建者的ID
})
```

4. Annotation 相关操作

上传标记的服务器处理程序较为简单，只需要把 Request Body 中的 JSON 数据读出 id，groupid 等信息生成 Annotation 对象存入数据库即可。

对于用户下载标记的请求，需要在 Annotation 表中进行查询，查询的方式是先找出表中所有同一个 Group 内的 Annotation，再从结果中找出该用户没有同步过（hasSync）的 Annotatoin 作为结果返回。

返回之前还要更新数据库中的 Annotation 的 hasSync 数组，加入该次同步 Annotation 的用户 ID，以便下一次不再将这些 Annotation 发送给用户。

```
var result = []
            var thisAccount = req.body.account
            for(var i in objs){
                if( objs[i].hasSync.indexOf(thisAccount) == -1 ){
                    result.push(objs[i])
                    objs[i].hasSync.push(thisAccount)
                    Annotation.update({ _id : objs[i]._id }, { hasSync :
```

```
objs[i].hasSync }, { hasSync : true },
                                                    function  (err)
{
                                                        if( err )


res.send(err)
                                                    })
                }
            }
```

5. Group 相关操作

创建组时，先解析客户端发送来的组的 ID，PDF 文件等数据，构造出一个 group 对象。然后需要在数据库中找是否已经存在过同样 ID 的组(客户端有可能产生相同的随机数)，若没有则将该对象保存到 Group 表中，返回成功信息。

加入组则比较简单，客户端通过 GET 请求并将组号作为 URL 参数，服务端解析得到组号，在 Group 表中查找并返回找到的 Group 对象给客户端。

删除组时，客户端同样需要发送 POST 请求，包含了客户端用户的 ID 和组号。由于只有创建者能删除组，因此服务端先根据 ID 找出组号对应的 Group 对象，之后检查该 Group 的创建者与发送请求的用户 ID 是否相同，相同则可以将 Group 表中对应的对象删除。

四、服务端配置步骤

作业打包文件中的 PDFViewerServer 为服务器工程目录。 服务器现已部署在腾讯云主机上,可通过 http://123.206.216.203:3000 进行访问。

Windows:

1. 从 nodejs 官方网站 https://nodejs.org/ 下载安装包（推荐 6.9.1）并安装
2. 将 PDFViewerServer 文件夹解压到任意目录
3. 进入解压后的目录，运行 npm install 安装依赖包
4. 依赖包安装成功后，运行 npm start 启动服务器，默认监听端口为 3000

   输出类似下图信息则启动成功

Linux（Ubuntu 为例）:

同样先将 PDFViewerServer 解压到任意目录

1. sudo apt-get update && sudo apt-get upgrade
2. sudo apt-get install nodejs（其实推荐安装 nvm，便于切换不同的 node 版本，但是过程比较繁琐…具体过程在此 https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-an-ubuntu-14-04-server ）
3. sudo apt-get install npm
4. cd PDFViewerServer
5. npm install
6. npm start

启动成功的输出与 Windows 相同

五、分工情况

马啸远：前端 UI 设计与实现

庞博：PDF 编辑同步，服务器端实现