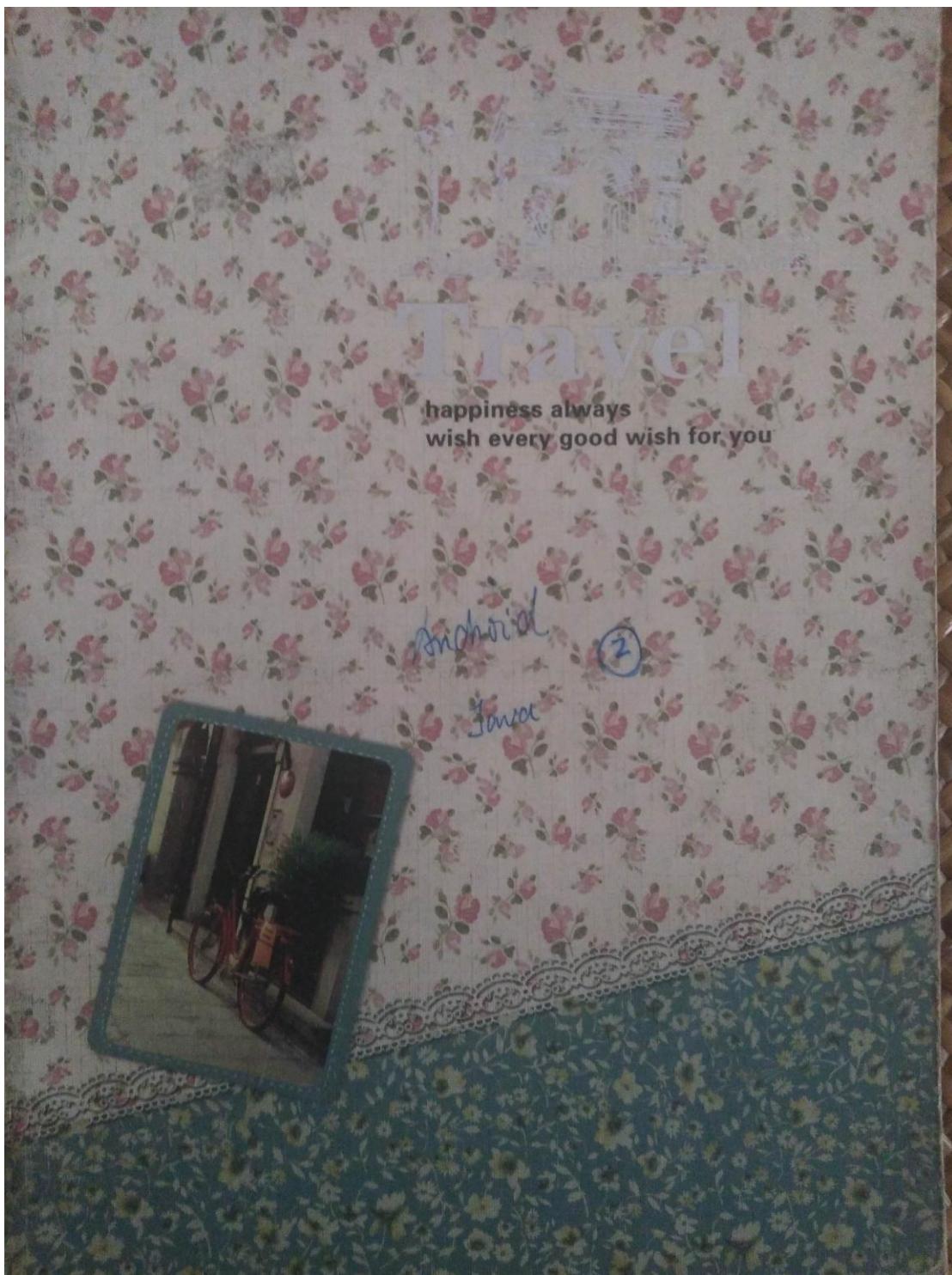


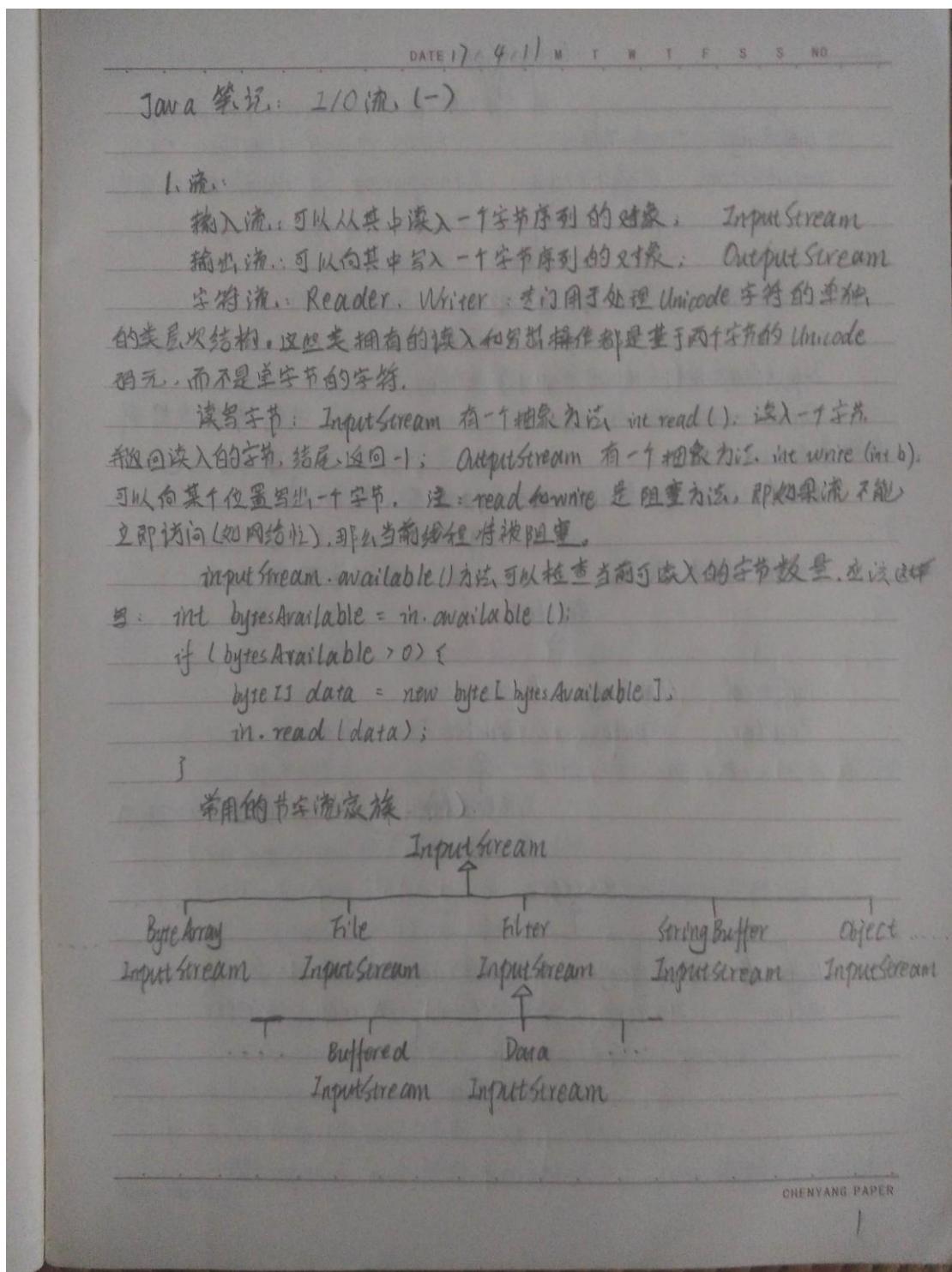
NoteBook-2

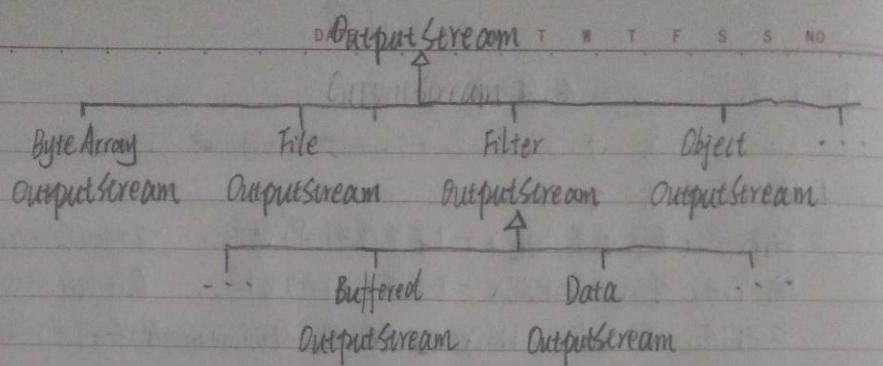


目录

201.Java I/O 流.....	3
205.Java I/O 流 文件操作.....	7
207.第一行代码 第九章 网络 解析 XML 笔记.....	9
209.第一行代码 第九章 网路 解析 json 笔记	11
210. 一位面试腾讯 Android 实习生的反思.....	12
211.第一行代码 第九章 网路 回调机制 笔记.....	13
212.Java 网络编程 Socket	14
215.第一行代码 第十章 多线程 Handler AsyncTask 笔记.....	17
219.第一行代码 第十章 Service 笔记	21
222.Hexo 首次搭建自己的博客	24
225.第一行代码 第十章 Service 下载实例 笔记	27
227.Java 连接数据库 MySQL	29
231.第一行代码 第十二章 MaterialDesign Toolbar 笔记	33
233.第一行代码 第十二章 MaterialDesign DrawableLayout、NavigationView	35
235.第一行代码 第十二章 MaterialDesign FAB、SnackBar、CoordinatorLayout	37
237.第一行代码 第十二章 MaterialDesign CardView AppBarLayout SwipeRefreshLayout.....	39
239.第一行代码 第十二章 MaterialDesign CollapsingToolbar、BottomNavigationView	41
242.Android 应用程序员必须掌握的 20 个技能.....	44
246.Android 群英传 第一章 系统架构 笔记.....	45
248.Android 群英传 第三章 Android 控件架构 笔记.....	46
250.Java 权限关键字 静态内部类与内部类.....	49
253.Android 群英传 第三章 View 测量 笔记.....	51
254.Android 群英传 第三章 ViewGroup 测量与绘制 笔记.....	52
255.Java volatile 与 synchronized 区别	53
256.Java synchronized final 死锁.....	54

201.Java I/O 流



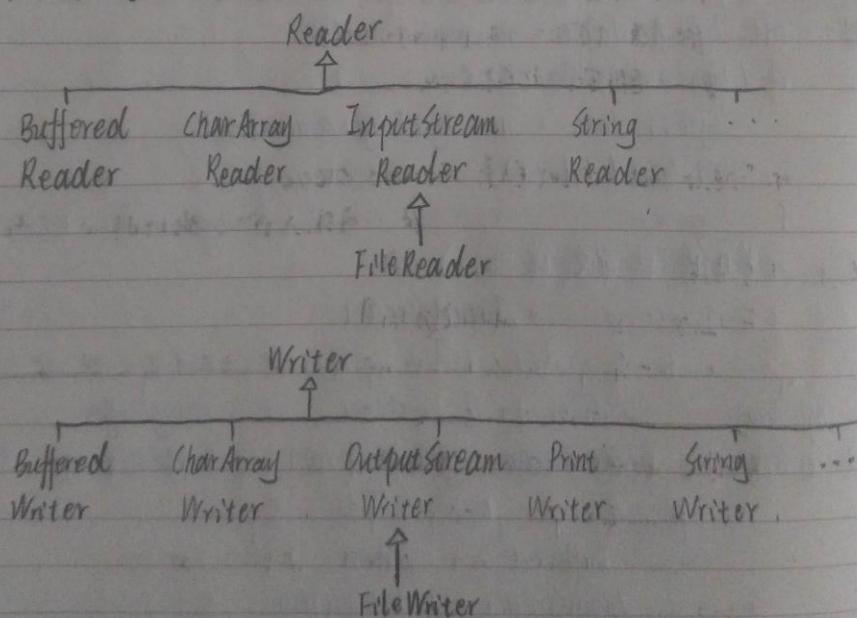


InputStream 和 OutputStream 是基类。

DataInputStream 和 DataOutputStream 可以以二进制格式读写所有基本 Java 类型。

FileInputStream 和 FileOutputStream 可以读写文件。

常见的字符流家族：



DATE / / / M T W T F S S NO

2. InputStream类的常用方法:

`int read (byte[] b);` 读入一个字节数组，返回实际读入的字节数。
遇到流的结尾时返回-1。最多读入b.length个字节。

`int read (byte[] b, int off, int len);`

b: 数据读入的数组；

off: 第一个读入字节应该被放置的位置在b中的偏移量。

len: 读入字节的最大数量

`int available ()`: 返回不阻塞情况下可获取的字节数。

3. OutputStream类的常用方法:

`void write (byte[] b);`

`void write (byte[] b, int off, int len);`

b: 数据写出的数组

off: 第一个写入字节在b中的偏移量

len: 写出字节的最大数量

4. FileInputStream 和 FileOutputStream

可以提供附着在一个磁盘文件上的输入流和输出流。只能对字节级别进行读写。

FileInputStream 的常用构造方法：

• `FileInputStream (String name)`: 传入文件名或完整文件路径。

• `FileInputStream (File file)`:

FileOutputStream 的常用构造方法：

• `FileOutputStream (String name)`:

• `FileOutputStream (String name, boolean append)`:

• `FileOutputStream (File file)`:

• `FileOutputStream (File file, boolean append)`:

参数append: true 数据添加到文件尾; false 代替同名的文件。

CHENYANG PAPER

DATE / / M T W T F S S NO

5. BufferedInputStream 和 BufferedOutputStream

流在默认情况下是不被缓冲区缓存的，请求一个数据块并将其置于缓冲区中会显得更加高效。

其构造方法如下：

BufferedInputStream (InputStream in);

BufferedOutputStream (OutputStream out);

例如 使用缓冲机制，以及用于文件的数据插入方法，可以这么写

DataInputStream din = new DataInputStream (

new BufferedInputStream (

new FileInputStream ("fileName")));

6. DataInput 和 DataOutput

DataOutput 接口定义了以二进制格式写数组、字符串、字符串等的方法：

writeChars, writeInt, writeDouble, writeUTF … 等等。如 writeInt() 将一个整数写出为 4 字节的二进制数量值。writeUTF 方法使用了修改版的 8 位 Unicode 转换格式写出字符串。

DataInput 接口定义了如下方法用来读回数据 如 .readChars, readInt, readDouble, readUTF … 等。

7. DataInputStream 和 DataOutputStream

读文件中的二进制数据：

DataInputStream dis = new DataInputStream (

new FileInputStream ("fileName"));

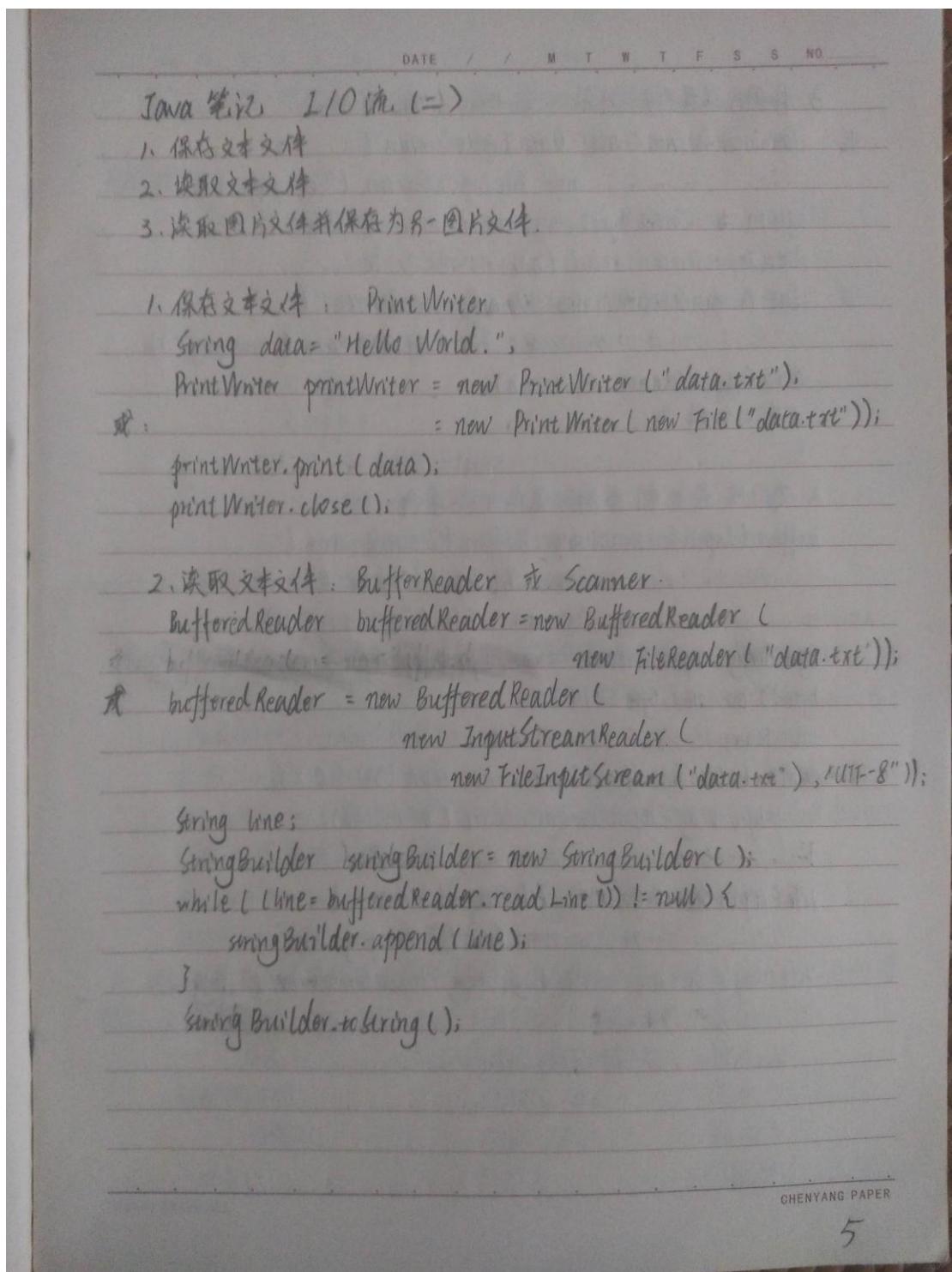
写二进制数据到文件：

DataOutputStream dos = new DataOutputStream (

new FileOutputStream ("fileName"));

CHENYANG PAPER

205.Java I/O 流 文件操作



DATE / / M T W T F S S NO.

3. 将图片保存为另一图片

```
读: dataInputStream = new DataInputStream (
    new FileInputStream ("xm.png"));
byte[] b = new byte[dataInputStream.available()];
dataInputStream.read (b);
写: dataOutputStream = new DataOutputStream (
    new FileOutputStream ("new.png"));
dataOutputStream.write (b);
//关闭各种流。
```

4. 使用缓存机制将图片保存为另一图片

```
bufferedInputStream = new BufferedInputStream (
    new FileInputStream (
        new File ("xm.png")));
byteArrayOutputStream = new ByteArrayOutputStream ();
byte[] b = new byte[1024];
int len;
while (len = bufferedInputStream.read (b)) > 0) {
    byteArrayOutputStream.write (b, 0, len);
}
fileOutputStream = new FileOutputStream (
    new File ("new.png"));
fileOutputStream.write (byteArrayOutputStream.toByteArray());
```

207. 第一行代码 第九章 网络 解析 XML 笔记

DATE / / M T W T F S S NO.

Android 第一行代码笔记：第九章. 网络-解析XML

文本内容为：<apps>

```
<apps>
    <app>
        <id>1</id>
        <name>Google Map</name>
        <version>1.0</version>
    </app>
    <app>
        <id>2</id>
        <name>Chrome</name>
        <version>2.0</version>
    </app>
</apps>
```

假设已经从网上获得了 XML 数据，并保存到字符串 xmlData 中。

1. Pull 解析：XmlPullParser.

获取 XmlPullParser 对象：

```
XmlPullParser parser = XmlPullParserFactory.newInstance().newPullParser();
```

设置好需要解析的数据：

```
parser.setInput(new StringReader(xmlData));
```

int eventType = parser.getEventType();

可以获取到当前的解析事件。（用整型表示各种事件，即各种标签）常用的值如下：

文档的开始：XmlPullParser.START_DOCUMENT	值为 0
结束：	.END_DOCUMENT 值为 1
标签的开始：Y	.START_TAG 值为 2
结束：	.END_TAG 值为 3
文本	.TEXT 值为 4

CHENYANG PAPER

DATE / M T W T F S S NO

String nodeName = xmlPullParser.getName();
可以读取到每个标签的名字。

解析的过程是，文档开始，从上往下，从左到右依次解析。
简单地说，就是顺序解析，当解析到文本时，就跳到下一个开始标签。如
<id></ids> 解析了文本数据 1 行，它的下一个
<name> Chrome </name> event type 就是 <name>
eventType = xmlPullParser.next(); 下一个事件

2. SAX 解析

自定义一个类继承 DefaultHandler，重写其中五个方法分别是

startDocument(): 开始解析的时候调用，通常做一些初始化操作。

startElement(String uri, String localName, String qName, Attributes attr):
开始解析某个节点时候调用，通常记录（保存）节点名：nodeName = localName；
characters(char[] ch, int start, int length):
解析节点中的内容时候调用，通常会根据节点名保存对应的信息。如
if (nodeName.equals("id")) stringbuilder.append(ch, start, len);

endElement(String uri, String localName, String qName): 完成某个节点的解析时候调用。

endDocument(): 完成整个 XML 解析时候调用

注意在获取节点内容时，characters 方法会调用多次，其中一些空白换行也被当作内容解析出来。

获取 XMLReader 对象

```
xmlReader = SAXParserFactory.newInstance().newSAXParser().getXMLReader();  
创建自定义的 DefaultHandler 对象
```

```
myHandler = new MyDefaultHandler();  
把 handler 设置到 xmlReader 中
```

```
xmlReader.setContentHandler(myHandler);
```

```
开始解析：xmlReader.parse(new InputSource(new StringReader(xmlData)));
```

209. 第一行代码 第九章 网路 解析 json 笔记

DATE / / M T W T F S S NO.

Android 笔记 - 第一行代码 第九章 网络(二)
——解析 json 数据

json 数据如下(省略其他属性)

```
{"user": {  
    "username": "xm",  
    "signature": "他很懒，什么也没写。",  
    "list-news-img": ["/images/newsimg/104-1.jpg",  
                      "/images/newsimg/104-2.jpg"]}}
```

1. JSONObject 解析。我 new JSONArray("list") → 必须和 json 数据中的相对应

```
JSONObject jsonObject = new JSONObject(jsonData);  
jsonObject = jsonObject.getJSONObject("user"); // 获取到 user 的对象  
String signature = jsonObject.getString("signature");  
JSONArray jsonArray = jsonObject.getJSONArray("list-news-img");  
for (int i = 0; i < jsonArray.length(); i++) {  
    String imagePath = jsonArray.get(i).toString();  
}
```

// 如果数组元素仍然是一个对象 {...}，则

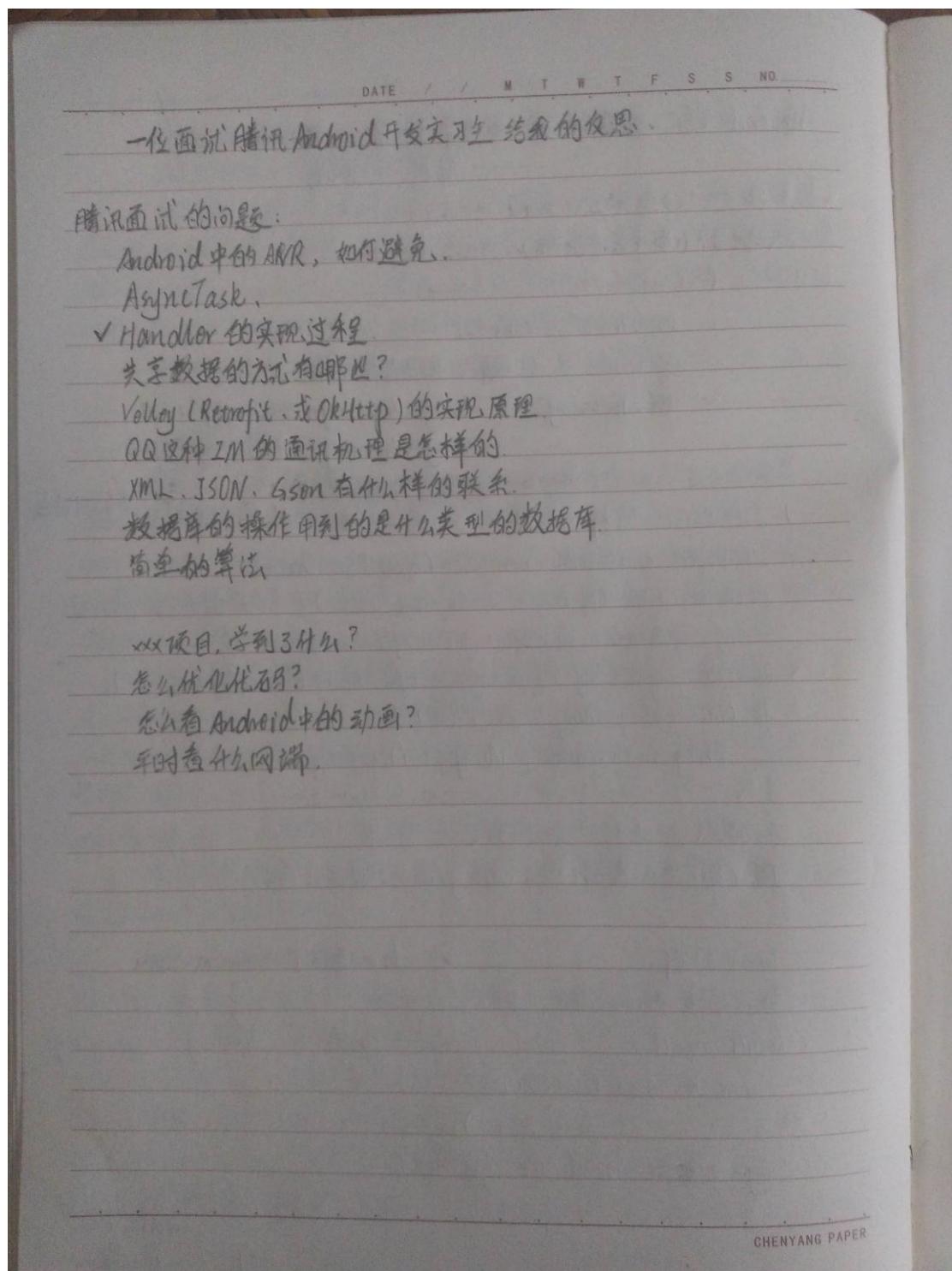
```
JSONObject jsonObject = jsonArray.getJSONObject(i);
```

2. Gson 解析。

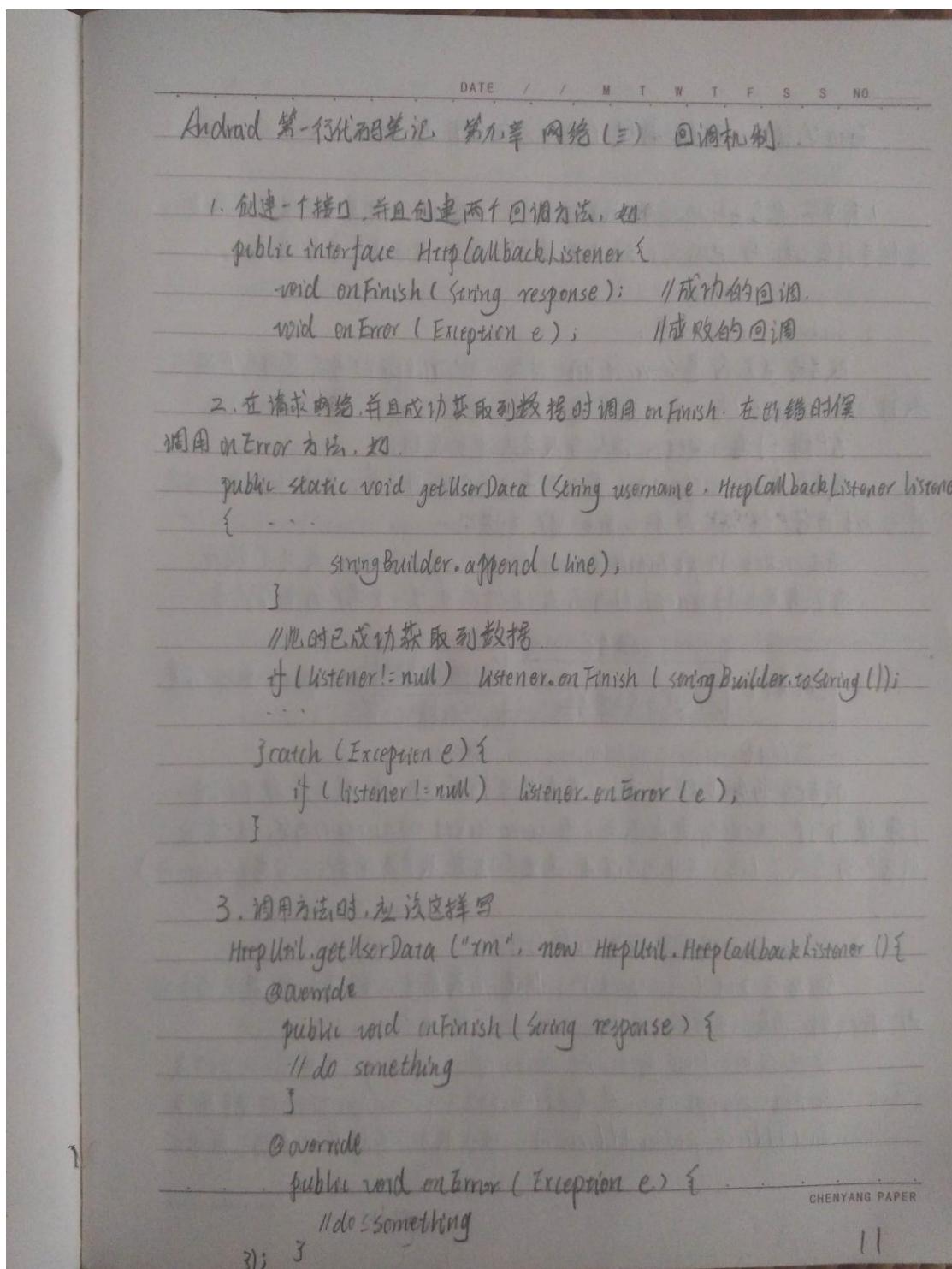
```
Gson gson = new Gson();  
Result result =  
    gson.fromJson(jsonData,  
                  Result.class);  
String signature = result.user.signature; }  
});
```

```
class User { String username;  
String signature;  
List<String> list-news-img;  
}  
class Result { User user;  
}  
类名可以任意  
相对应。  
CHEM&TECH PAPER
```

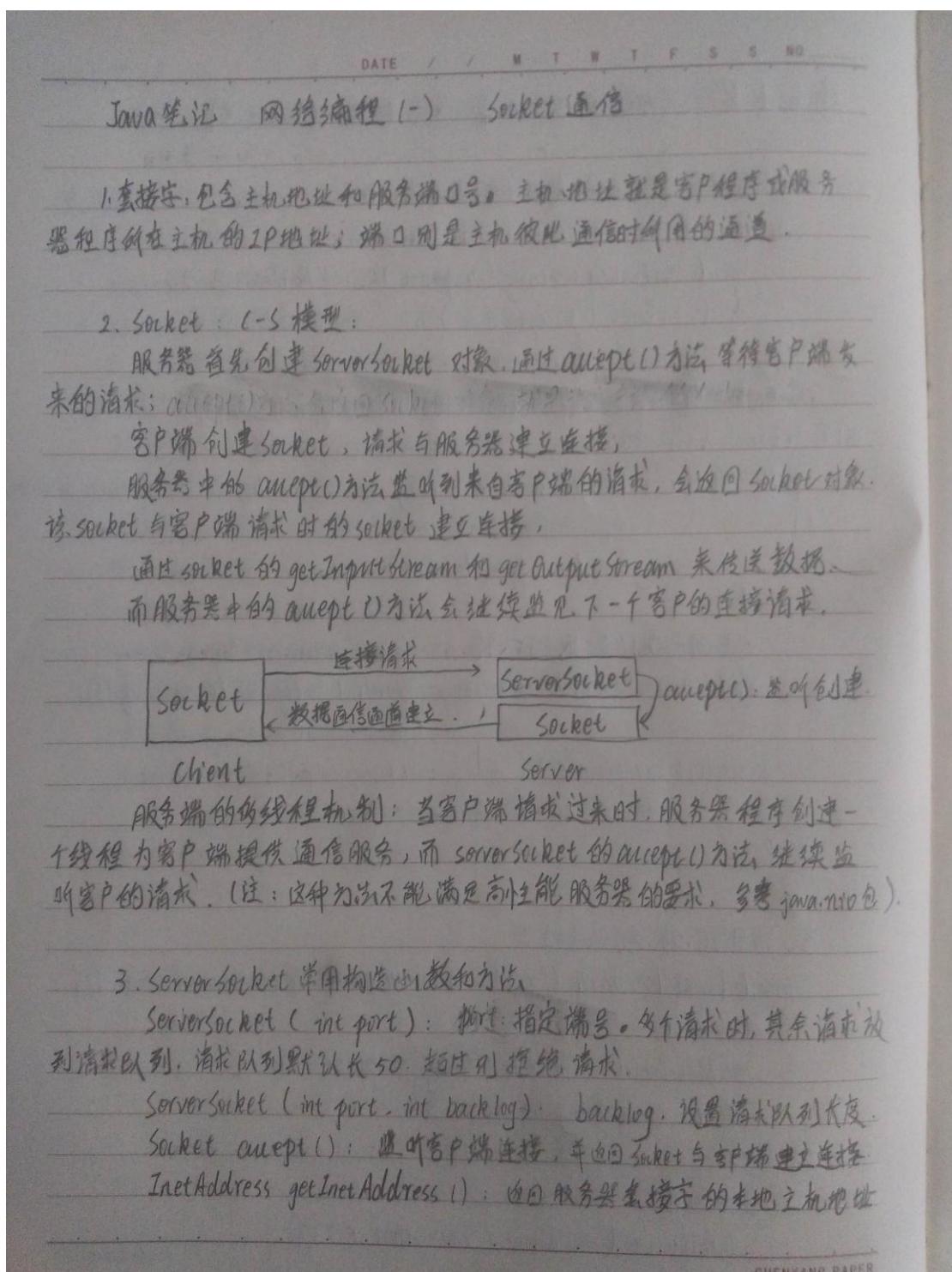
210. 一位面试腾讯 Android 实习生的反思



211. 第一行代码 第九章 网路 回调机制 笔记



212.Java 网络编程 Socket



DATE / / M T W T F S S NO.

4. Socket 的构造函数和常用方法

Socket (InetAddress address, int port): IP地址和端口号。

public OutputStream getOutputStream()

public int getPort(): 返回套接字的连接的远程主机端口。

public int getLocalPort(): 为因套接字所连接的本地主机端口。

5. 实例 1: 单客户端与服务器的通信。

// Server 端:

```
ServerSocket serverSocket = new ServerSocket(5400);
```

```
Socket clientSocket = serverSocket.accept();
```

```
BufferedReader in = new BufferedReader(
```

```
new InputStreamReader(
```

```
clientSocket.getInputStream()));
```

```
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
```

```
boolean done = false;
```

```
while (!done) { String line = in.readLine();
```

```
System.out.println("client:" + line);
```

```
out.println("server:" + line);
```

```
if (line.trim().equals("BYE")) done = true;
```

```
}
```

```
clientSocket.close();
```

// Client 端代码

```
Socket socket = new Socket("127.0.0.1", 5400);
```

```
BufferedReader bufferedReaderFromServer = new BufferedReader(
```

```
new InputStreamReader(socket.getInputStream()));
```

```
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
```

```
BufferedReader bufferedReaderFromKeyboard = new BufferedReader(
```

```
new InputStreamReader(System.in));
```

CHENYANG PAPER

DATE / / M T W T F S S N

```
boolean done = false;
while (!done) {
    String strKeyboard = bufferedReaderFromKeyboard.readLine();
    out.println(strKeyboard);
    String strServer = bufferedReaderFromServer.readLine();
    System.out.println(strServer);
    if (strKeyboard.equals("BYE")) done = true;
}
socket.close();
```

6 实例2：多客户端与服务器的通信

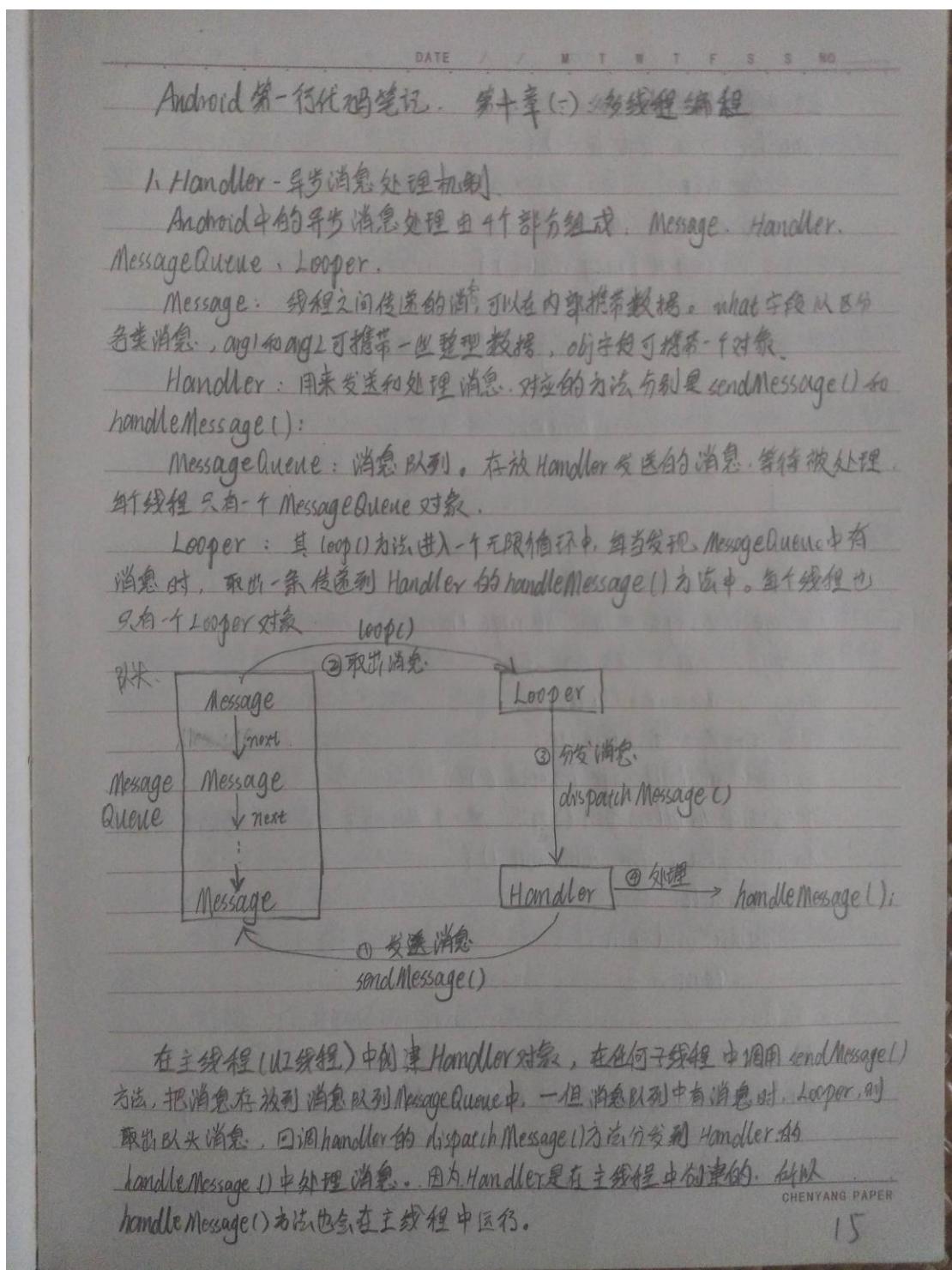
```
ServerSocket serverSocket = new ServerSocket(5400);
while (true) {
    Socket clientSocket = serverSocket.accept();
    new Thread(new Runnable() {
        @Override
        public void run() {
            // 代码与上相同
        }
    }).start();
}
```

客户端代码无变化。

7.

215. 第一行代码 第十章 多线程 Handler

AsyncTask 笔记



DATE / / M T W T F S S NO

在主线程中创建 Handler 对象

```
handler = new Handler () {  
    @Override  
    public void handleMessage (Message msg) {  
        switch (msg.what) {  
            case MESSAGE_OK :  
                String result = (String) msg.obj;  
                Toast ...  
                break;  
        }  
    }  
};
```

在任何子线程中发送消息

```
Message message = new Message ();  
message.what = MESSAGE_OK;  
String strMessage = "Hello";  
message.obj = strMessage;  
handler.sendMessage (message);
```

或者调用 handler.post() 方法，把一个 Runnable 对象发送到消息队列中。

```
handler.post (new Runnable () {
```

@Override

```
    public void run () {
```

Toast ...
 }
};

handler.post() 方法，把一个 Runnable 对象发送至消息队列中。

2. AsyncTask

AsyncTask的实现原理 简单地说就是 Handler + 线程池。Android 帮我们做了很好的封装，即便对异步消息处理机制完全不了解也可以十分简单地从子线程切换到主线程。

AsyncTask抽象类指定了3个泛型参数：Params, Progress, Result

Params：执行AsyncTask时传入，作为doInBackground() 的参数。

Progress：onProgressUpdate() 的传入参数，也是在doInBackground中调用publishProgress()的传入参数。

Result：doInBackground 的返回参数，作为onPostExecute()的传入参数。
这三个参数都是

继承 AsyncTask类，重写几个常用的方法。

onPreExecute()：后台方法执行之前调用，通常做一些界面初始化工作，如显示一个进度条对话框。

doInBackground(Params...)：这个方法在子线程中运行，做一些耗时的操作，如网络请求，方法里不可以进行UI操作，如需要实时更新UI，如更新进度条，可以调用publishProgress(Progress...):

onProgressUpdate(Progress...)：如果在doInBackground中调用了publishProgress()方法，那么该方法很快就被执行。该方法中的参数是publishProgress()传来的。

onPostExecute(Result...)：doInBackground()执行完毕后，返回的数据作为该方法的参数，并且可以更新UI，比如关闭进度条对话框。

其中除了doInBackground()是运行在子线程外，其余方法运行在主线程。

例如，一个AsyncTask类应该这样写：

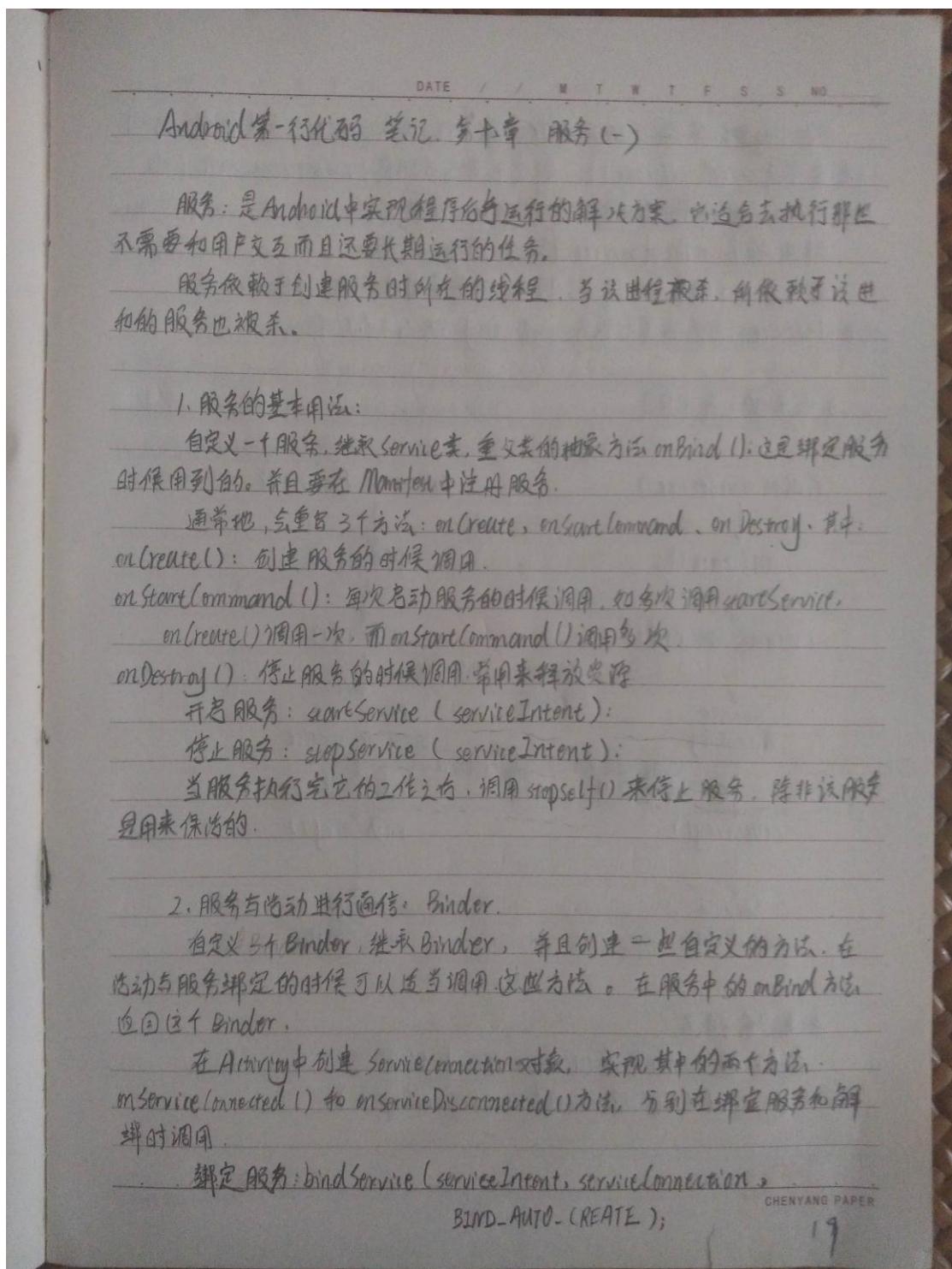
(见下页)

DATE / / M T W T F S S NO

```
class MyAsyncTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... urls) {
        String url = urls[0];
        String result = url; // 假设请求网络后返回的数据
        return result;
    }
    @Override
    protected void onProgressUpdate(Void... values) {
        super.onProgressUpdate(values); // 在这里根据值更新进度条
    }
    @Override
    protected void onPostExecute(String s) {
        Toast... // 更新UI的操作
    }
}
```

在线程中这样调用
new MyAsyncTask().execute("url address");

219. 第一行代码 第十章 Service 笔记



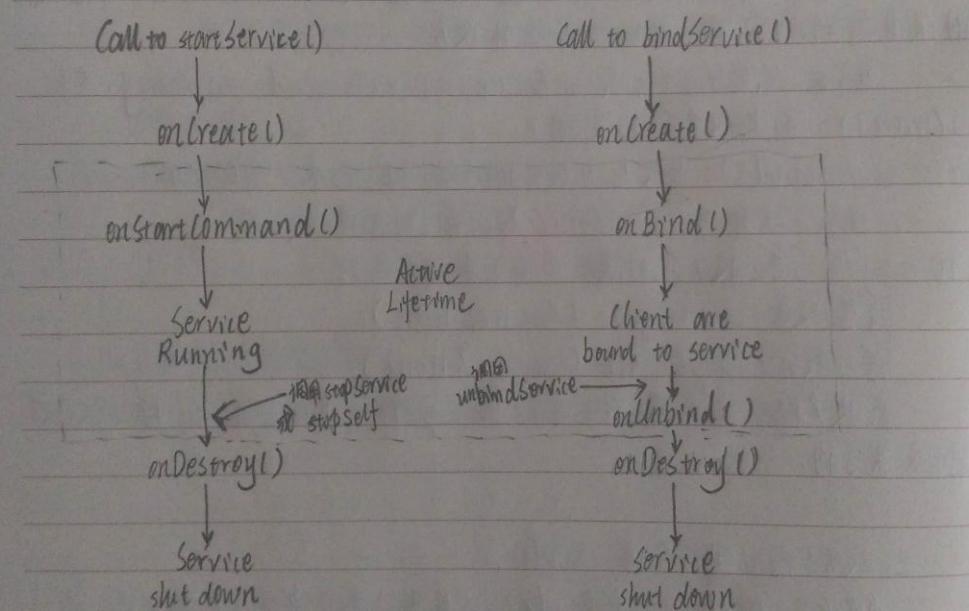
DATE / / M T W T F S S NO

其中，flag标志的值 BIND-AUTO-CREATE 表示，启动与服务绑定之后自动创建服务，执行 onCreate()。绑定服务后不执行 onStartCommand() 而执行 serviceConnection 的 onServiceConnected 方法。

解绑服务：unbindService(serviceIntent)；

如果启动一个服务后又解绑了服务，则需要同时调用 stopService() 和 unbindService() 方法来停止服务，onDestroy() 方法才会执行。

3. 服务的生命周期



4. 使用前台服务

在自己的 Service 类的 onCreate 方法中 调用：

startForeground (int notificationId, notificationObject)；

DATE / / M T W T F S S NO

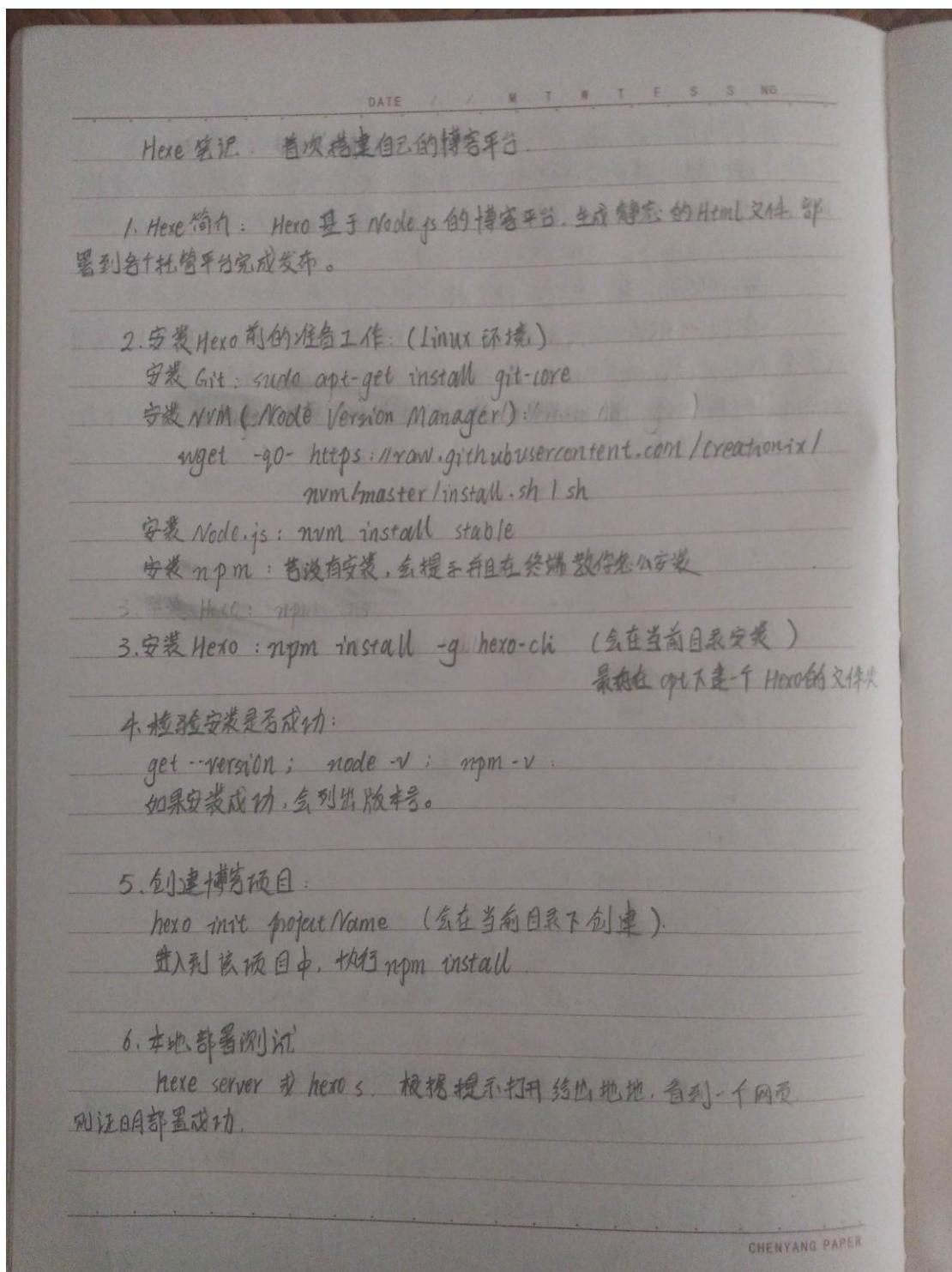
5. 使用 IntentService:

如果直接在服务中处理一些耗时的操作很容易 ARN，因此通常在服务中开启一个线程去处理。这种服务一但启动就会一直处于运行状态，直到调用 stopService 或 stopSelf。但也有可能忘了 stop。

IntentService 就是处理这种问题。

自定义一个服务继承 IntentService，重写默认的构造函数（无参），去掉用父类的构造函数。重写 onHandleIntent() 和 onDestroy() 方法，其中 onHandleIntent(): 当启动服务时调用该方法，处理一些具体的逻辑。这个方法运行在子线程中了。

222.Hexo 首次搭建自己的博客



DATE / / M T W T F S S NO

7. 修改主题：在该主题的主页(github)下会有详细的教程，但一般来说，都是通过一条命令完成修改。

git clone git@github.com:xxxxxx/themes/themeName

8. 依赖安装，在Hexo根目录下：

在Hexo的安装目录下，安装以下依赖。

Less: npm install hexo-renderer-less --save

Feed: npm install hexo-generator-feed --save

Json-content: npm install hexo-generator-json-content --save

QRCode: npm install hexo-helper-qrcode --save

9. 部署到github。

在github中创建Repository，名称必须是 userName.github.io

或 devamever.github.io

修改博客项目的 config.yml。

deploy:

type: git

repo: https://github.com/UserName/UserName.github.io.git

branch: master

部署: hexo deploy 或 hexo d

部署成功后在浏览器输入 https://UserName.github.io 就能看到你的博客了。

注意：在deploy时提示没有权限时，是因为github上没有添加SSH key。

创建SSH key: ssh-keygen -t rsa -C "emailAddress"

成功后在用户根目录下找到.ssh目录(是隐藏的)，其中 id_rsa 是私钥，id_rsa.pub 是公钥。

在github的Setting中找到SSH，添加SSH key。把公钥的内容复制进去，你就看到已经添加的key。

DATE / / M T W T F S S NO

10. 新建文章: hexo new articleName

会生成 source/_posts/articleName.md 文件，可以用 Markdown 语法编写该文件。

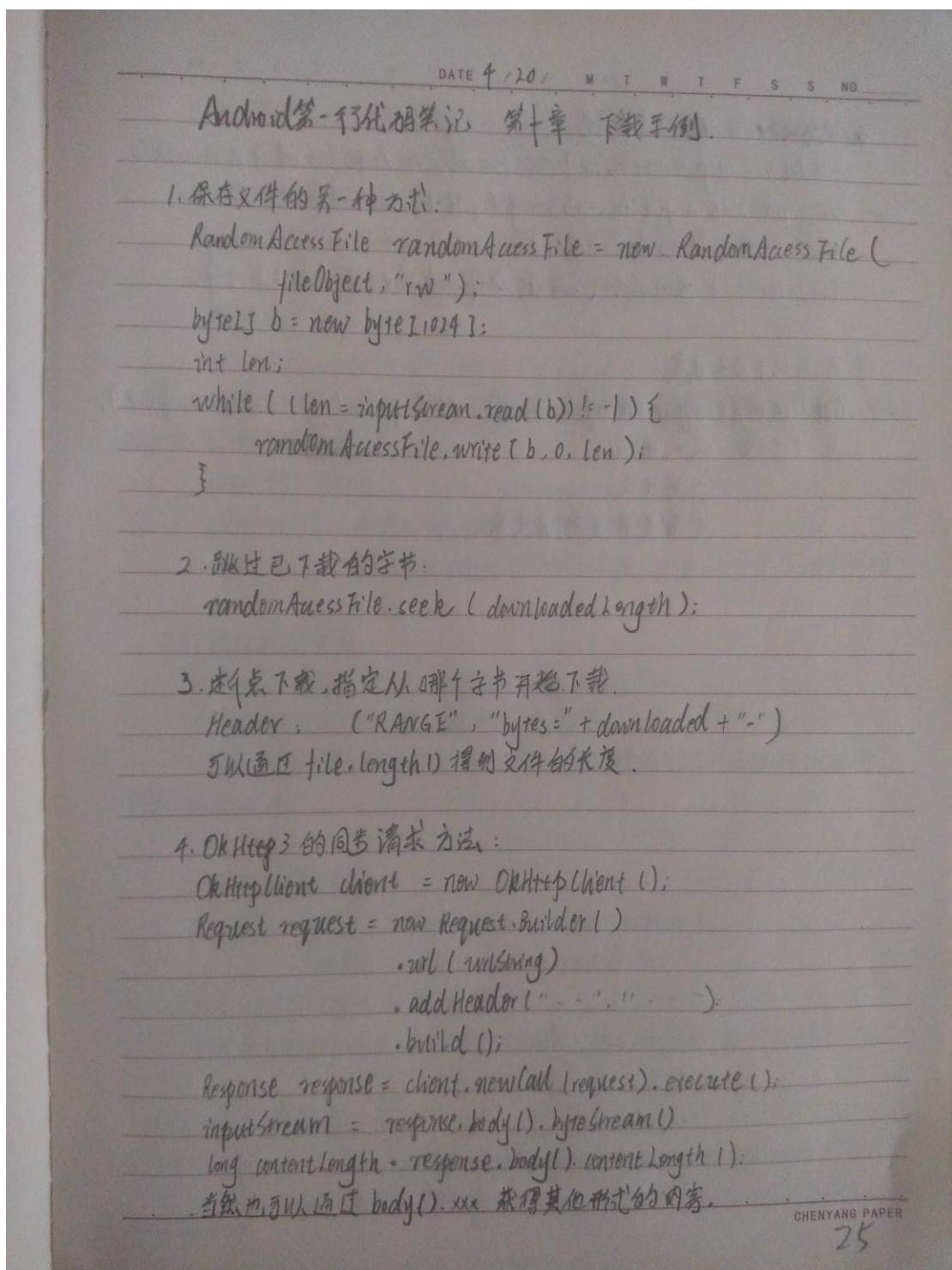
执行 hexo generate，生成静态 Html 文件到 public 中。

然后执行 hexo deploy 提交修改过的文件或新的文件。

CHENYANG PAPER

225. 第一行代码 第十章 Service 下载实例

笔记



DATE / / M T W T F S S NO

5. 根据文件的URL获得文件名

```
fileUrl = "http://192.168.23.1:8080/SocialServer/apk/social-0.0.01.apk";
```

```
String fileName = fileUrl.substring(fileUrl.lastIndexOf("/") + 1);
```

根据函数名就知道要干什么了。

lastIndexOf() 返回最后一个字符的索引，然后根据这个索引求字符串。

6. 在通知中显示进度

```
NotificationCompat.Builder builder (this).setProgress (100, intProgress, false);
```

第一个参数：最大进度，

第二个参数：当前进度

第三个参数：是否使用模糊进度条。

如：

禁

227. Java 连接数据库 MySQL

DATE / / M T W T F S S NO

Java 笔记 连接数据库 (MySQL)

1. 下载相关的驱动程序 JAR 文件。
如 Java 与 MySQL 的连接，可在 MySQL 的官网 下载

2. 数据库的 URL 格式：
如 "jdbc:mysql://127.0.0.1/dbname"
在连接数据库时，我们必须使用与数据库类型相关的参数，例如
如主机名、端口号和数据库名。

JDBC 的一般语法：jdbc:subprotocol:other stuff
subprotocol：连接到数据库的具体驱动程序。
other stuff：随 subprotocol 的不同而不同，应查阅供应商相关文档

3. 注册驱动器类
`Class.forName ("com.mysql.jdbc.Driver");`
字符串内容为驱动器所在包的全路径。
这条语句使得驱动器类被加载，由此将执行可以注册驱动器的
静态初始化器。

4. 连接到数据库
`Connection conn = DriverManager.getConnection (`
`sqlUrl, username, password);`
其中三个参数都是字符串类型参数。
连接成功时将返回 Connection 对象，用它来执行 SQL 语句操作。

5. 操作数据库
5.1 `String sql = "select nickname, phone from tuser where username='xm';"`
执行 SQL 命令之前首先要创建 Statement 对象，或 Statement 的子类。

CHENYANG PAPER

PreparedStatement 对象

```
statement = conn.createStatement();
preparedStatement = conn.prepareStatement(sql);
ResultSet resultSet = statement.executeQuery(sql);
```

或 = preparedStatement.executeQuery();

executeQuery() 方法可以执行 select 的查询语句。executeUpdate() 方法可以执行 insert、update 和 delete 之类的操作。也可以执行 create table 和 drop table 之类的数据定义语句。execute() 方法则可以执行任意的 SQL 语句。
且 executeQuery() 的查询结果保存在 ResultSet 对象中，可以通过 resultSet.next() 循环遍历每一行记录。

```
while (resultSet.next()) {
```

String nickname = resultSet.getString("nickname");

或。 = resultSet.getString(1);

}

可以根据列名或者列的序号。（如上例 SQL 语句中， nickname 为第 1 列， phone 为第 2 列）。数据库的列序从 1 开始，是查询结果中的列，而不是数据表中的列。

根据列的不同类型调用 resultSet 的不同访问器，如 getInt，getDouble。

注意，操作完数据库之后，ResultSet，Statement，Connection 都要关闭。

5.3 带占位符的 SQL 语句

```
String sql = "select * from tuser where username = ?";
```

```
PreparedStatement.setString(1, "xm");
```

第一个参数表示几个占位符，第二个参数表示值。

5.4 当 preparedStatement 执行了另一个 SQL 语句时，需要调用 clearParameters() 清除预备语句中的所有参数。

5.5 读取大对象 LOB 或二进制数据

resultSet.getBlob() 或 getClob 方法得到 Blob 或 Clob 对象，可以从 Blob 对象中调用 getBytes 或 getInputStream 获得字节数组或流。

DATE / / M T W T F S S NO

5.6 多结果的处理 (执行存储过程)

使用 execute() 的方法执行SQL语句

获取第一个结果集或更新计数

重复调用 getMoreResults 移动到下一个结果集

如果由多结果集构成的链的最后一项是结果集 executeToGetMoreResults
将返回 true, 而如果在链中的下一项不是更新计数, getUpdateCount() -1.

下面的循环可以遍历所有结果

```
boolean isResult = statement.execute(sql);
```

```
boolean done = false;
```

```
while (!done) {
```

```
    if (isResult) {
```

```
        ResultSet result = statement.getResultSet();
```

```
        // do something
```

```
    } else {
```

```
        int updateCount = statement.getUpdateCount();
```

```
        if (updateCount >= 0) // do something
```

```
        else done = true;
```

```
}
```

```
    if (!done) isResult = statement.getMoreResults();
```

```
}
```

5.7 获取自动生成键

```
statement.executeUpdate(insertSql, Statement.RETURN_GENERATED_KEYS);
```

```
resultSet = statement.getGeneratedKeys();
```

```
if (resultSet.next()) int key = resultSet.getInt(1);
```

当 execute() 或 executeUpdate() 执行的是插入的SQL语句并且第二个参数值为 Statement.RETURN_GENERATED_KEYS 时, 那么第一列中就是自动生成的键。

CHENYANG PAPER

DATE / / M T W T F S S ND

5.8 可更新的结果集

有时候我们需要从数据库中找到某些记录之后更新该记录，因此用到了可更新的结果集。

```
statement = conn.createStatement (ResultSet.TYPE_SCROLL_INSENSITIVE,  
                                ResultSet.CONCUR_UPDATABLE);
```

那么调用 executeQuery() 方法得到的就是可更新的 resultSet。

```
如: String newNickname = "Alicever";
```

```
String oldNickname = resultSet.getString ("nickname");
```

```
resultSet.updateString ("nickname", newNickname);
```

```
resultSet.updateRow (); 确保更新成功;
```

注意，并非所有查询都会返回可更新的结果集。例如查询涉及多个表的连接操作的结果集是不可更新的。如果查询一个表或是使用主键直接找表，它产生的结果集是可更新的。可用 resultSet 的 getConcurrency 方法确定是否可更新。

注意：resultSet 的 updateXXX 方法只是改变结果集中的行的值，非数据库中的值。因此要调用 updateRow 方法，把更新信息发送给数据库。

5.9 事务

我们就可以将一组语句构成一个事务，当所有语句都顺利执行之后，就提交 (commit)。否则，如果其中某个语句遇到错误，那么事务被回滚，就好像没有任何语句被执行过一样。这样可以确保数据完整性。

默认情况下，数据库连接处于自动提交模式，即每 SQL 语句一执行就会提交；要使用事务，就要关闭这个默认：conn.setAutoCommit (false);

然后多次调用 execute () & executeUpdate () 方法。

```
statement.executeUpdate (sql1);
```

```
statement.executeUpdate (sql2); ...
```

执行完且不出错时，调用 conn.commit () 方法提交事务；

如果出现错误，则调用 conn.rollback ()；程序将自动撤销自上
上提交以来的所有语句。

CHENYANG PAPER

231. 第一行代码 第十二章 MaterialDesign

Toolbar 笔记

DATE / / M T W T F S S NO

Android 第一行代码笔记 第十二章 MaterialDesign (-) Toolbar

1. Toolbar的强大之处在于，它不仅继承了ActionBar的所有功能，而且灵活性很高，可以配合其他控件来完成一些Material Design 的效果

2. 创建一个Android，默认是使用了包含ActionBar的主题，因此在应用图标栏中修改为NoActionBar的主题。
如：Theme.AppCompat.Light.NoActionBar，其中
colorPrimary 为 Toolbar 的颜色
colorPrimaryDark 为状态栏的颜色
colorAccent 为强调色，如默认的FAB

3. 在布局中使用 Toolbar 控件。

```
<android.support.v7.widget.Toolbar  
    android:id="@+id/toolbar"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="?attr/colorPrimary"  
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"  
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

其中 app 为新定义的命名空间。为什么要这样呢？因为 popupTheme 是 5.0 系统中新增的，使用 app 的话就可以兼容 5.0 以下的系统。
theme 为 Toolbar 设置成单独的主题，注意其菜单也会使用同样的主题。
popupTheme 单独设置成另外的主题，不修改 Toolbar 的主题。
在 Activity 中：setSupportActionBar(toolbar) 就能用了

4. Toolbar 联系菜单
首先定义菜单的资源文件。

CHENYANG PAPER

31

DATE / / M T W T F S S NO
<menu xmlns:android="http://schemas.android.com/apk/res/android">
 <item android:id="@+id/icon" android:title="title" android:showAsAction="always"/>

</menu>

showAsAction 用来指定按钮的显示方式，有3种值可选：

always：显示在 toolbar 中；

ifRoom：toolbar 有足够的空间才显示。

never：不显示，而显示为三个点，提示用户打开。

然后在 Activity 的 onCreateOptionMenu 中加载这个菜单：

getMenuInflater().inflate(R.menu.toolbar_menu, menu)。在 onCreateOptionsMenu 可以在 Activity 的 onOptionsItemSelected 方法中对菜单项进行监听。onOptionsItemSelected

5. 设置 Toolbar 的 Home 按钮及图标。

ActionBar actionBar = getSupportActionBar();

if (actionBar != null) {

 actionBar.setDisplayHomeAsUpEnabled(true);

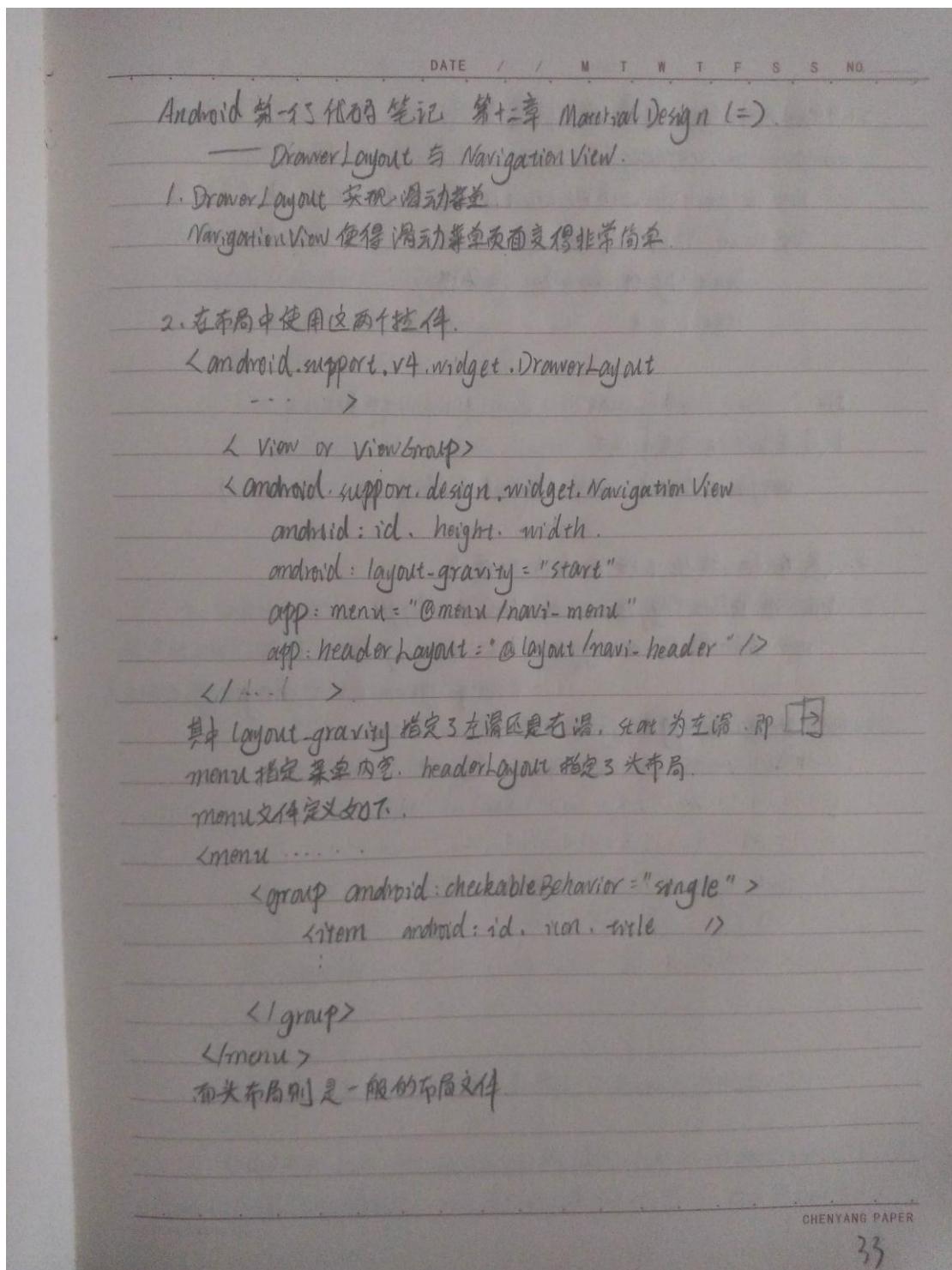
 actionBar.setHomeIndicatorEnabled(true);

}

为 Home 按钮的监听在 onOptionsItemSelected 中完成。

233. 第一行代码 第十二章 MaterialDesign

DrawerLayout、NavigationView



DATE / / M T W T F S S NO

3. 为 NavigationView 的菜单项设置监听.

```
navigationView.setNavigationItemSelectedListener (
```

```
new NavigationView.OnNavigationItemSelectedListener () {
```

```
@Override onNavigationItemSelected () {
```

```
drawerLayout.closeDrawers ();
```

```
return true;
```

```
}
```

});

设置菜单的初始选中状态.

```
navigationView.setCheckedItem (R.id.id_navi_menu_xx);
```

4. 设置 Toolbar 的 Home 按钮的动画特效.

```
ActionBarDrawerToggle actionBarDrawerToggle =
```

```
new ActionBarDrawerToggle (this, drawerLayout, toolbar,
```

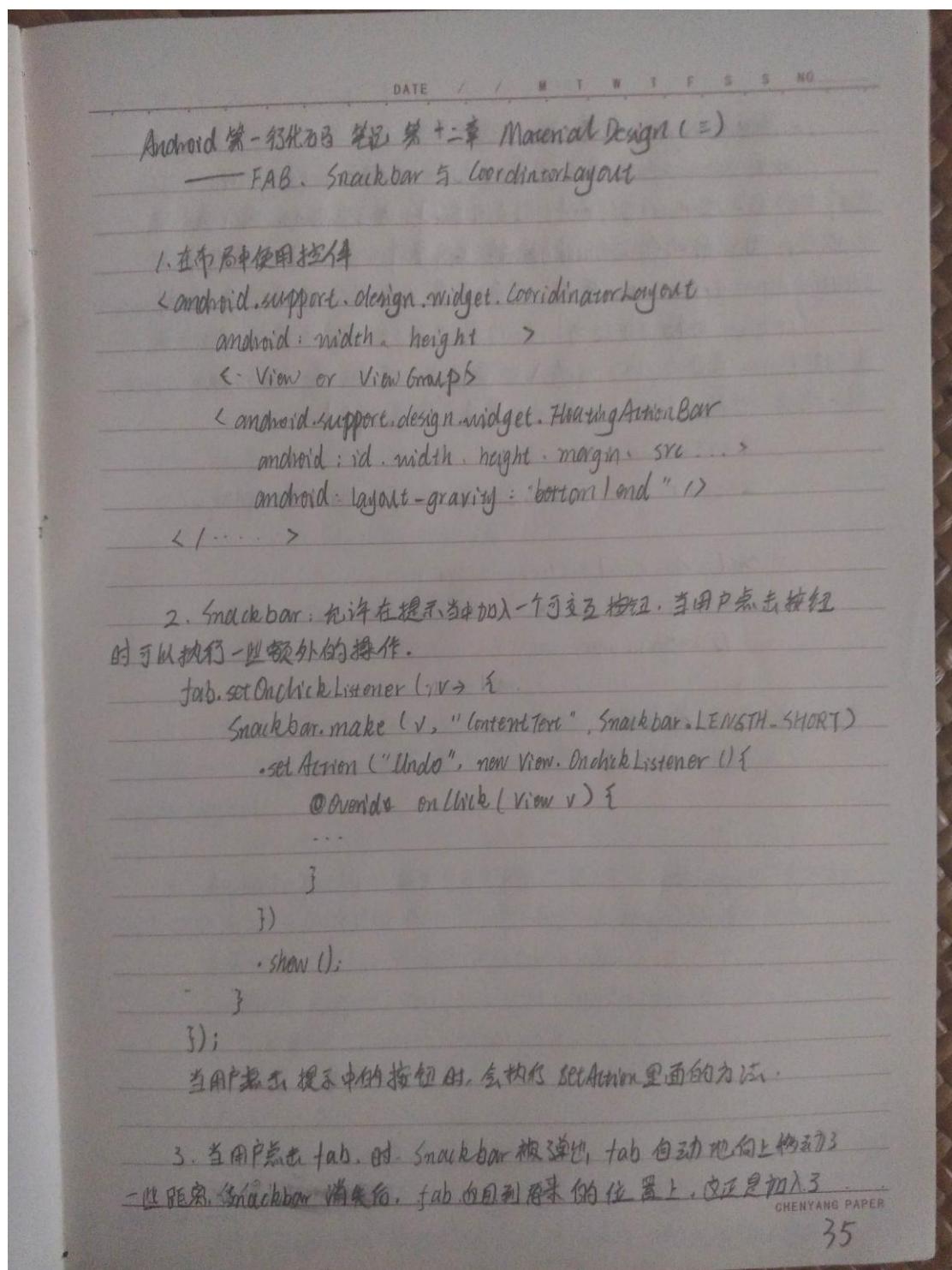
```
R.string.app_name, R.string.app_name);
```

```
actionBarDrawerToggle.syncState();
```

CHENYANG PAPER

235. 第一行代码 第十二章 MaterialDesign

FAB、SnackBar、CoordinatorLayout



DATE / / M T W T F S S NO

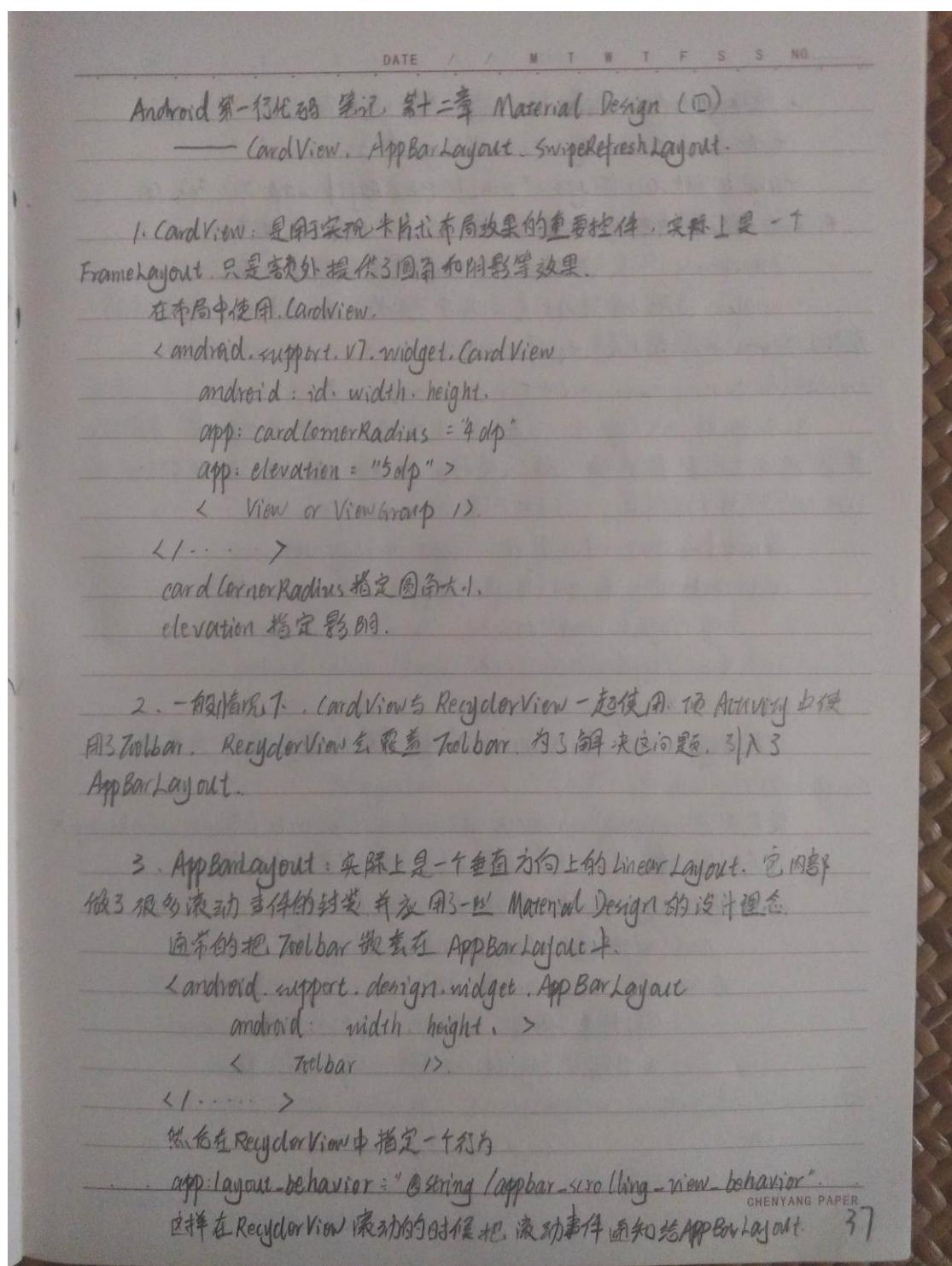
CoordinatorLayout 造成的。

CoordinatorLayout 其实就是一个加强版的 FrameLayout，它可以监听其所有子控件的各种事件（如点击事件），然后帮助我们做出最合理的响应。如它监听到了 Snackbar 的弹出事件，它会自动将内部的 Floating Action Button 向上偏移，从而确保不会被 Snackbar 遮挡到。

Snackbar 的 make() 方法中的第一个参数 用来指定 Snackbar 是基于那个 view 来触发的，如传入的是 fab，而 fab 是 CoordinatorLayout 的子控件，因此，Snackbar 的弹出事件会被监听到。

237. 第一行代码 第十二章 MaterialDesign

CardView AppBarLayout SwipeRefreshLayout



DATE / / M T W T F S S NO

1. 随 RecyclerView 的滚动，自动隐藏和显示 Toolbar。

在 Toolbar 中添加 layout_scrollFlags 属性。

app:layout_scrollFlags = "scroll|enterAlways|snap"

scroll：当向上滑动时，Toolbar 随一起向上滚动并隐藏。

enterAlways：向下滚动时，随着一起向下滚动并显示。

snap：当 Toolbar 还没完全隐藏或显示的时候，会根据当前滚动的距离，自动选择 隐藏还是显示。

5. SwipeRefreshLayout：用于实现下拉刷新功能的核心类。我们把想要实现下拉刷新功能的控件放置到 SwipeRefreshLayout 中，就可以迅速让这个控件支持下拉刷新。如下拉刷新的 RecyclerView。

<android.support.v4.widget.SwipeRefreshLayout>

 <android:id="id" width="height">

 app:layout_behavior >

 <RecyclerView />

</>

如果 RecyclerView 使用了 behavior 属性，要把该属性提到 外层的 SwipeRefreshLayout 中。

设置颜色：SwipeRefreshLayout.setColorsSchemeResources(R.color.colorPrimary);

设置刷新的监听，注意执行完任务后一定要结束刷新。

SwipeRefreshLayout.setOnRefreshListener (

 new SwipeRefreshLayout.OnRefreshListener () {

 @Override --- onRefresh() {

 //如新建一个线程，睡眠一秒，回到主线程

 SwipeRefreshLayout.setRefreshing (false);

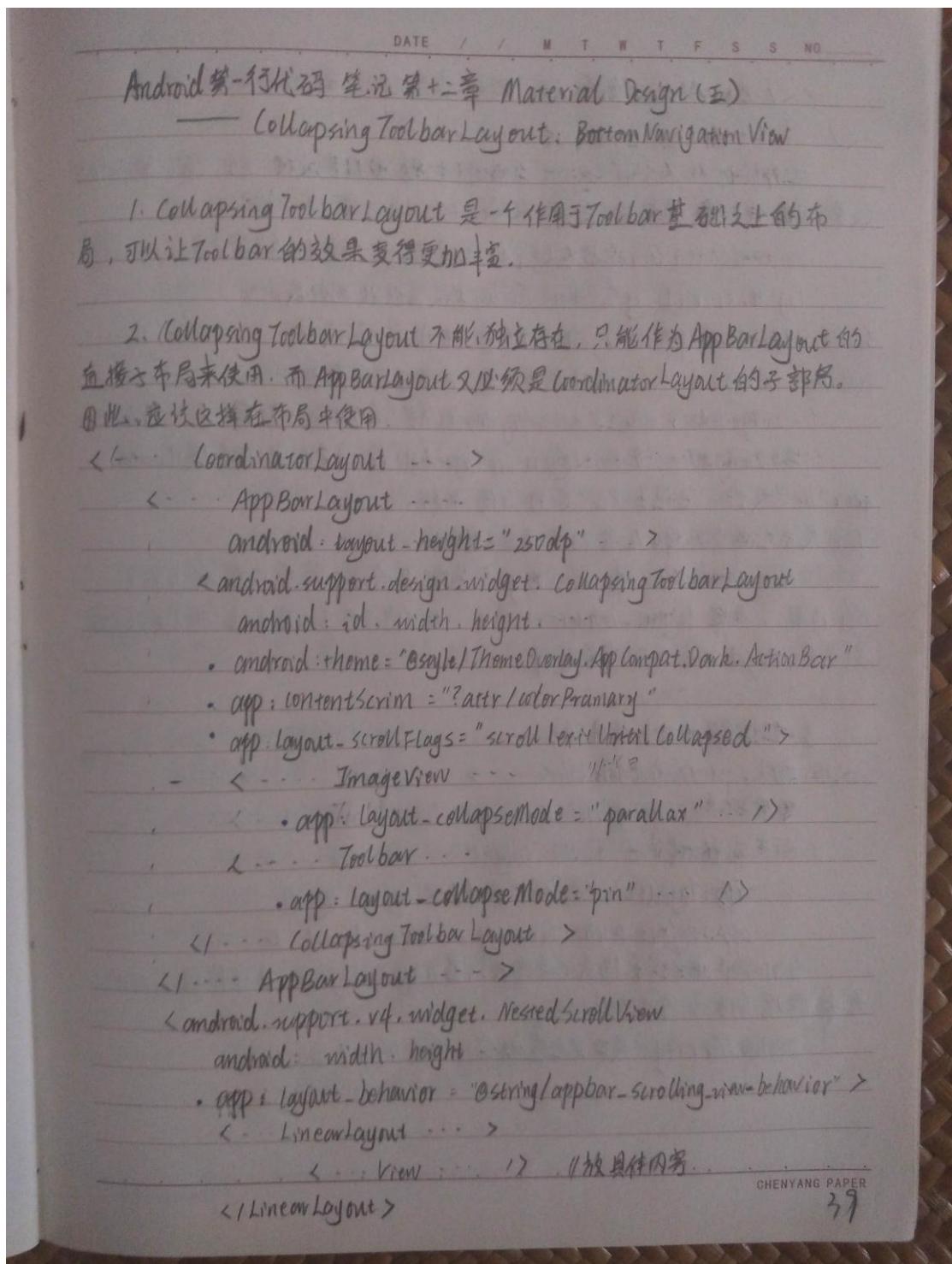
}

});

CHENYANG PAPER

239. 第一行代码 第十二章 MaterialDesign

CollapsingToolbar、BottomNavigationView



DATE / / M T W T F S S NO

<1... NestedScrollView >

</1... CoordinatorLayout >

在 CollapsingToolbarLayout 中指定了主题，因为要实现更加高级的 Toolbar 效果，因此将主题提到上一层中。

contentScrim 用于指定在处于折叠状态以及折叠之后的背景色。

layout_scrollFlag 在原来的 Toolbar 中设置也提到外层。

scroll：表示 CollapsingToolbarLayout 会随着内容详情一起滚动；

exitUntilCollapsed 表示 滚动完成折叠之后保留在外层上。

<ImageView> 用来显示 ToolbarLayout 的背景。<Toolbar> 就是折叠后的 Toolbar。

其中 ImageView 使用了 Layout_collapseMode 属性来指定折叠模式。

parallax：表示折叠过程中产生一定的错位偏移。

pin：表示在折叠过程中位置始终保持不变。

NestedScrollView 在 ScrollView 的基础上增加了嵌套响应滚动事件的功能，指定 layout_behavior 属性，使内容布局不会覆盖住 ToolbarLayout，就好像 RecyclerView 与 Toolbar 那样。

3. 在标题栏的边上使用 FAB。

<1... NestedScrollView ... >

<... FloatingActionButton >

android: id, width, height, margin, src ... >

app: layout_anchor = "@+id/appbar_layout"

app: layout_anchorGravity = "bottom | end" />

anchor：指定了一个锚点，将锚点设置为 AppBarLayout，这样按钮会出现在标题栏的区域内。

anchorGravity：表示定位在标题栏区域的右下角。

CHENYANG PAPER

DATE / / M T W T F S S NO

4. 设置透明状态栏 5.0 以后的特性

把与状态栏融合的控件，以及包含该控件的所有父控件都添

加 android:fitsSystemWindows = "true".

在 res 目录中新建 values-v21 的文件夹，在该文件夹下新建 styles.xml

3/14.

```
<style name = "TransparentActivityTheme" parent = "AppTheme">
```

```
    <item name = "android:statusBarColor">
```

```
        @android:color/transparent</item>
```

```
</style>
```

只有 API 21 及以上的系统才会读取该文件。

另外，在 values 目录下的 styles.xml 中，添加以下：

```
<style name = "TransparentActivityTheme" parent = "AppTheme"></style>
```

这里只有 5.0 以下的系统才会读取，因为不能设置状态栏为透明，所以

什么也不做。

最后在需要设置透明状态栏的 Activity 使用该主题。

5. BottomNavigationView

在布局中使用该控件。

```
<android.support.design.widget.BottomNavigationView
```

```
    android: id . width . height . background
```

```
    app:menu = "@menu/menufile" />
```

设置监听

```
bottomNavigationView.setOnNavigationItemSelectedListener (
```

```
    new BottomNavigationView.OnNavigationItemSelectedListener () {
```

```
        @Override ... onNavigationItemSelected (MenuItem item) {
```

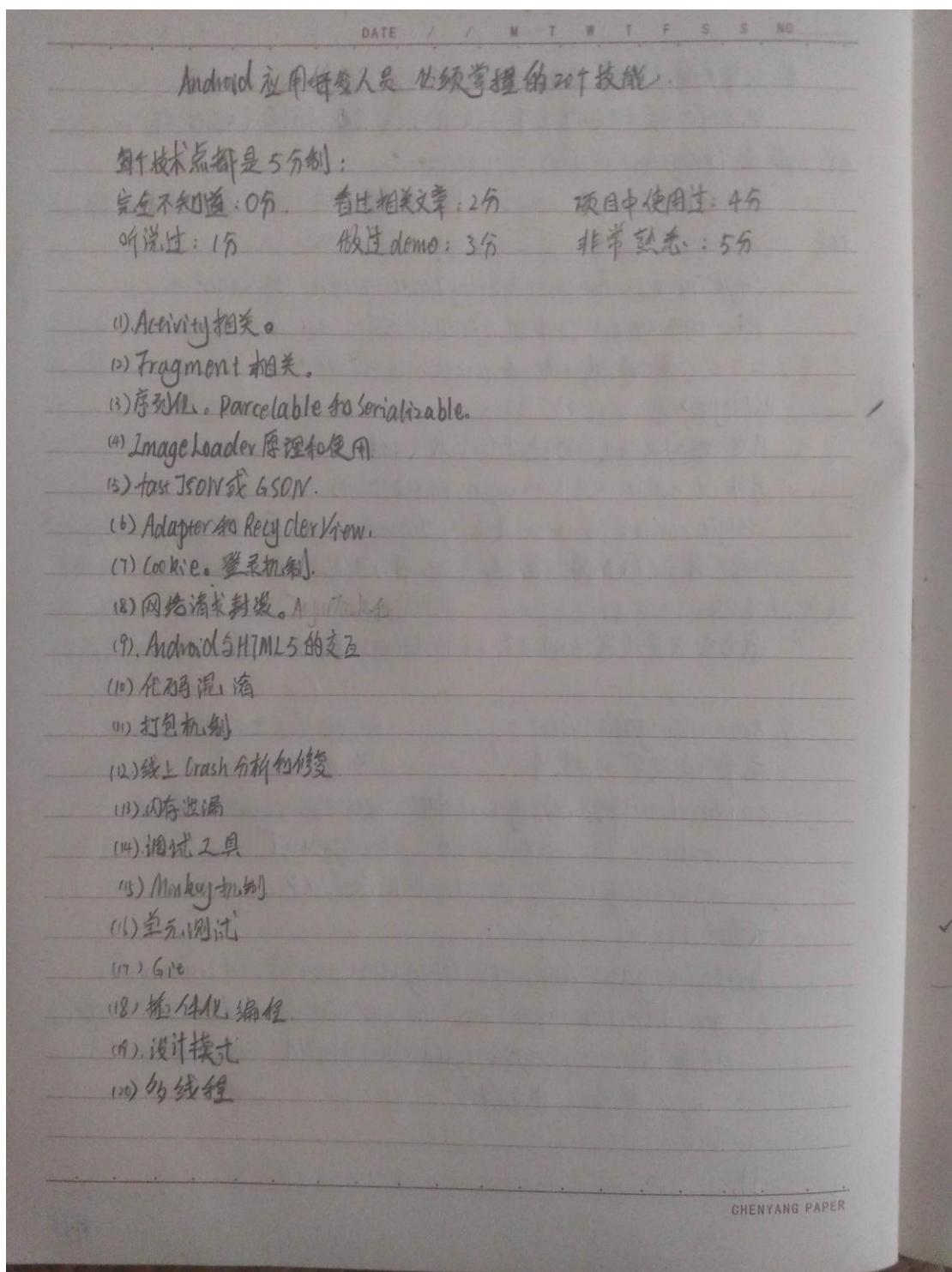
```
            // 根据 id 进行操作
```

```
        }
```

```
    };
```

CHENYANG PAPER

242.Android 应用程序员必须掌握的 20 个技能



246.Android 群英传 第一章 系统架构 笔记

DATE 5/3 M T W T F S S NO						
Android 群英传 笔记：第一章 系统架构						
1. Android 系统架构：Linux 内核层、库和运行时 Framework 层、应用层						
Linux 层：Android 最底层最核心的部分，包含了 Android 系统的核心服务，包括硬件驱动、进程管理、系统安全、等等。						
标准库和运行时：						
• Dalvik to ART：Android 运行环境虚拟机，每个 APP 都会分配虚拟机，保证互相之间不受干扰，并保持独立。运行时编译。ART 采用的是安装时进行编译。						
• 标准库：开发者在开源环境中可以使用的开发库						
Framework：构建应用程序用到的各种 API						
Application：						
2. Android APP 组件架构：四大组件。Activity、BroadcastReceiver、ContentProvider、Service。						
3. 四大组件协同工作：						
Activity 作为人机交互的第一界面，负责向用户展示信息和处理结果。						
ContentProvider 获取其他应用的信息。						
Service 从后台计算、下载、处理数据。						
BroadcastReceiver：获取广播信息。						
Intent：作为信息的载体，组件与组件之间通过 Intent 来通信，传递信息交换数据。						
4. 应用运行上下文对象：Context						
Activity、Service、Application 都是继承 Context，在创建 Context 实现类的时候就是创建 Context 的时机。Application Context 对象贯穿整个应用进程的生命周期，为应用全局提供了功能和环境支持。						

248.Android 群英传 第三章 Android 控件架构 笔记

DATE / / M T W T F S S ND

Android 群英传 笔记 第三章 (-) Android 控件架构

1. 控件分两类，即 ViewGroup 和 View。 ViewGroup 可作为父控件包含其他 ViewGroup 和 View，从而形成了一个控件树。上层控件负责下层控件的绘制和测量，并传递交互事件。

ViewParent

ViewGroup

View Group

View

View

2. findViewById() 方法就是在控件树中以树的深度优先遍历来查找对应元素。

3. 每棵树控件树的顶部，都有一个 ViewParent 对象，是整棵树控制核心，统一调度和分配交互管理事件。

4. 每个 Activity 都包含一个 Window 对象。
Window 对象通常由 PhoneWindow 来实现。

5. PhoneWindow 将一个 DecorView 设置为整个应用窗口的根 View。将要显示的具体内容呈现给 PhoneWindow。

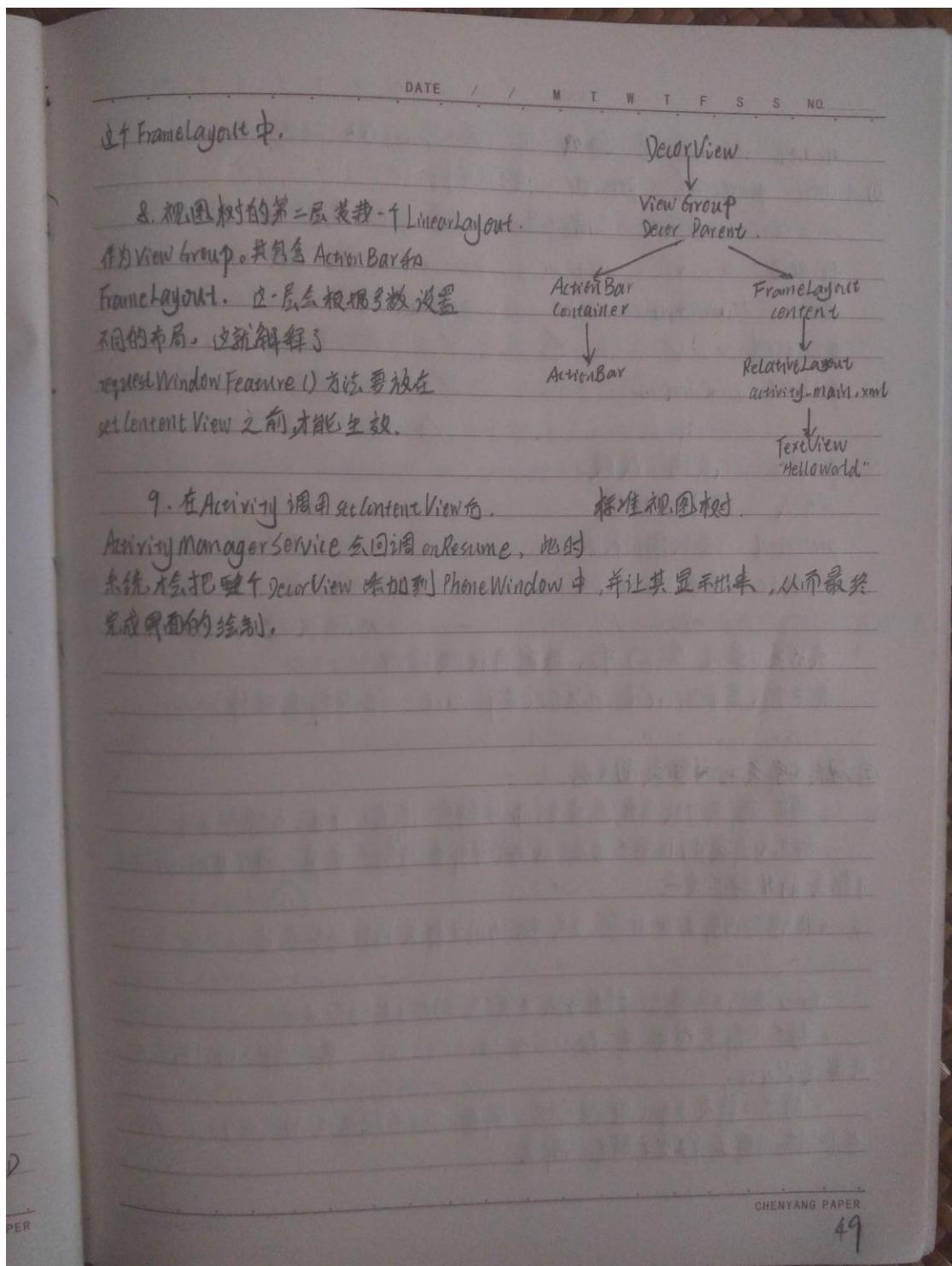
6. 所有的 View 的监听事件通过 WindowManagerService 来进行接收，并通过 Activity 对象来回调相应的 onClickListener。

7. DecorView 把屏幕分成 TitleView 和 Content View。Content View 是一个 ID 为 content 的 FrameLayout。setContentView 就是把我们做好的布局加到。

UI 界面架构图

CHENYANG PAPER

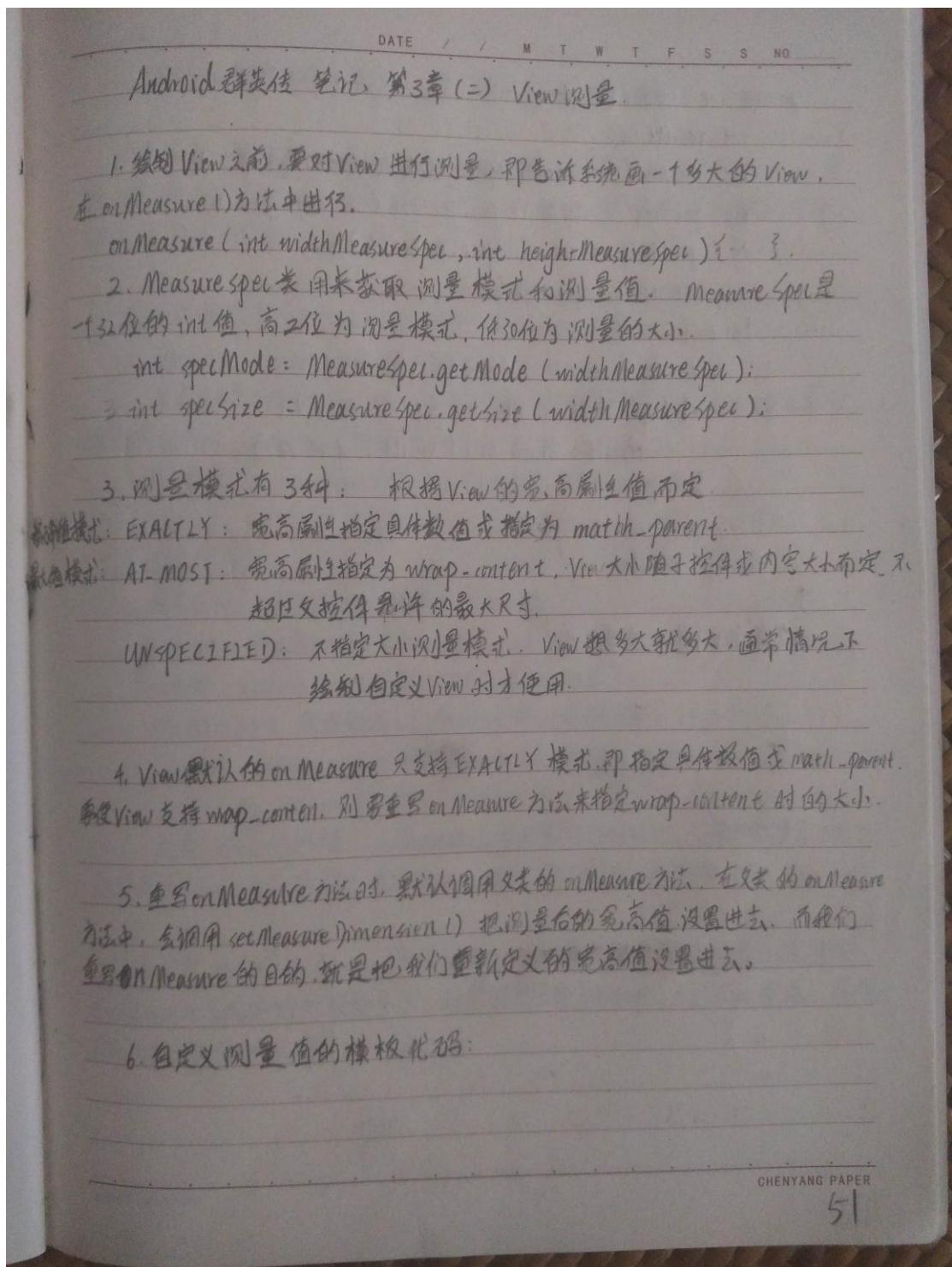
```
graph TD; Activity[Activity] --- PhoneWindow[PhoneWindow]; PhoneWindow --- DecorView[DecorView]; DecorView --- TitleView[TitleView]; DecorView --- ContentView[ContentView];
```



250. Java 权限关键字 静态内部类与内部类

	DATE	/	/	M	T	W	T	F	S	S	NO																																																
Java																																																											
① public, protected, private 的区别:																																																											
当前类 同一个包 子类 其他包																																																											
<table border="1" style="width: 100%;"><tr><td>private</td><td>✓</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>默认</td><td></td><td>✓</td><td>✓</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>protected</td><td></td><td>✓</td><td>✓</td><td>✓</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>public</td><td></td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td></td><td></td><td></td><td></td><td></td></tr></table>												private	✓											默认		✓	✓									protected		✓	✓	✓								public		✓	✓	✓	✓	✓					
private	✓																																																										
默认		✓	✓																																																								
protected		✓	✓	✓																																																							
public		✓	✓	✓	✓	✓																																																					
public: 公共访问权限																																																											
默认: 包访问权限																																																											
protected: 继承访问权限																																																											
private: 私有访问权限																																																											
类成员(字段、属性)可以被所有权限修饰																																																											
类不可以是 protected, 也不可以是 private, 但内部类除外。																																																											
② 静态内部类和内部类的区别																																																											
• 静态内部类可以有静态成员, 而非静态内部类则不能有静态成员																																																											
• 静态内部类的非静态成员可以访问外部类的静态变量, 而不可以访问 外部类的非静态变量。 <small>(非静态成员)</small>																																																											
• 非静态内部类的非静态成员可以访问外部类的非静态变量																																																											
Java语法规定: 静态方法不能直接访问非静态成员																																																											
• 静态内部类型可以是 class, interface, enum, 其他类型的内部类型 只能是 class.																																																											
• 静态内部类中可以再定义静态内部类, 可以无限深度的嵌套下去。不能 在非静态内部类中定义静态内部类。																																																											

251.Android 群英传 第三章 View 测量 笔记

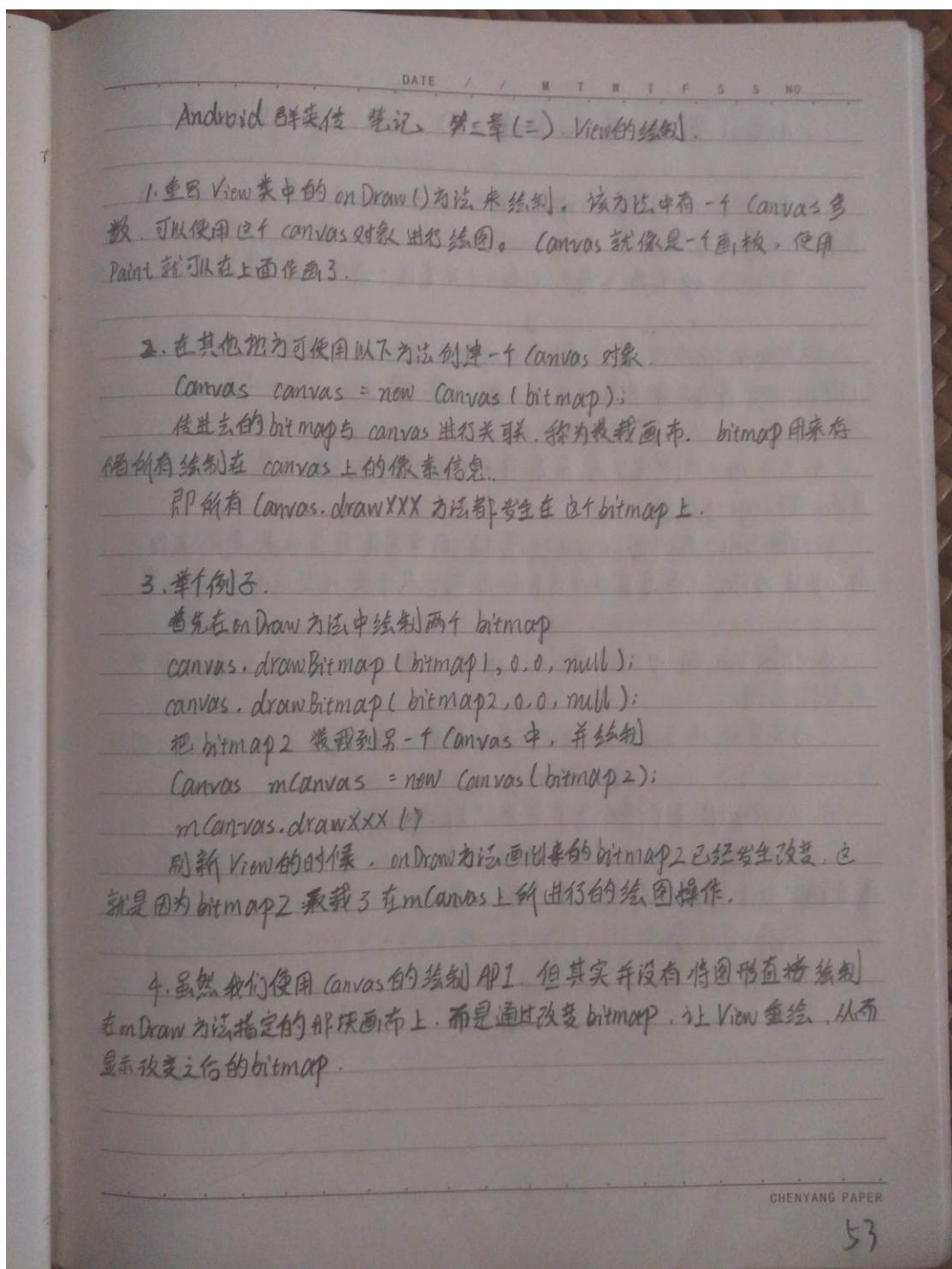


DATE / / M T W T F S S NO

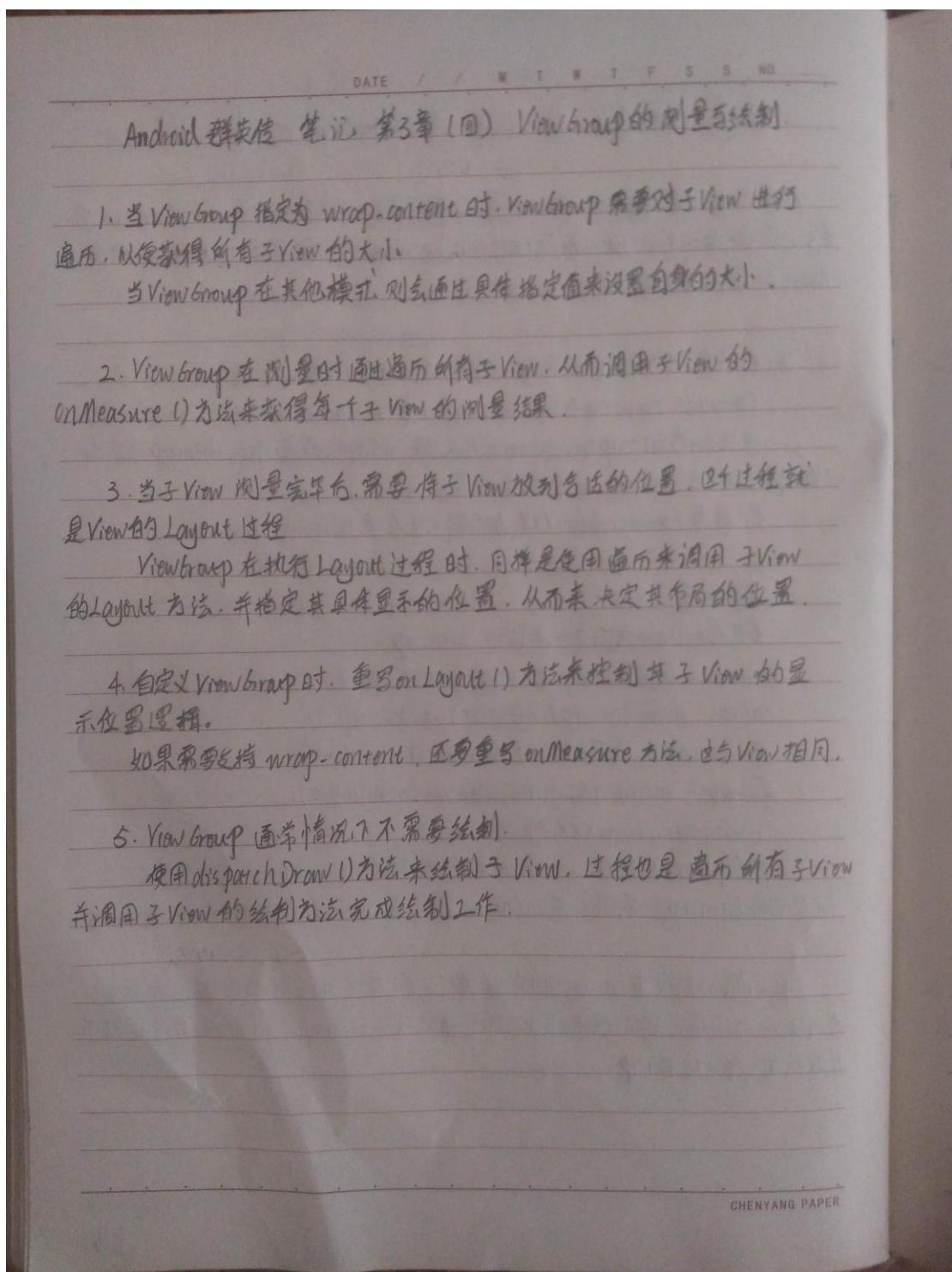
```
private int measureWidth (int measureSpec) {
    int result = 0;
    int specMode = MeasureSpec.getMode(measureSpec);
    int specSize = MeasureSpec.getSize(measureSpec);
    if (specMode == MeasureSpec.EXACTLY) {
        result = specSize;
    } else {
        result = 200; // 设置默认值，单位px;
        if (specMode == MeasureSpec.AT_MOST) {
            result = Math.min(result, specSize);
        }
    }
    return result;
}
```

对高度的测量也是差不多。

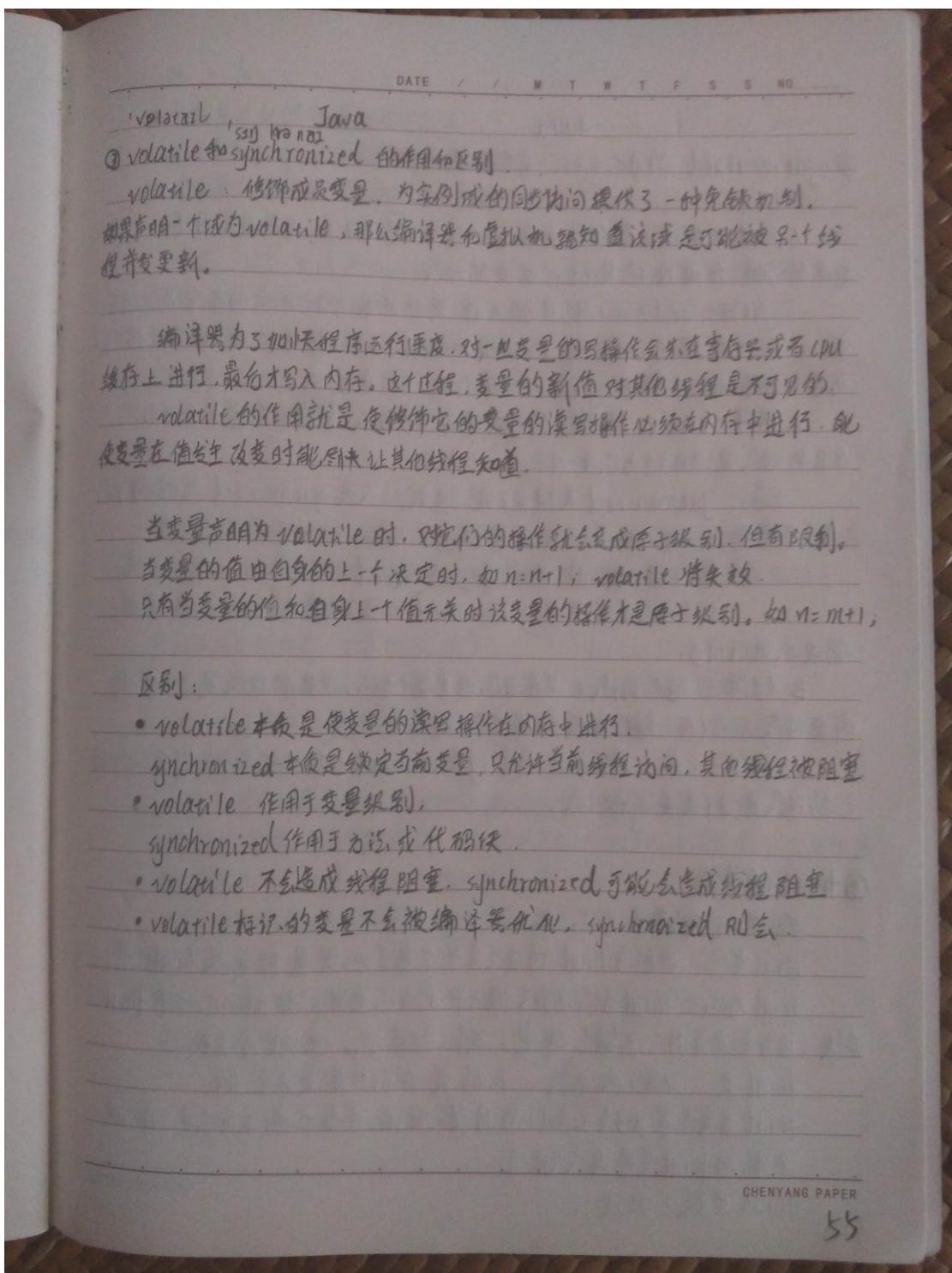
253.Android 群英传 第三章 View 测量 笔记



254.Android 群英传 第三章 ViewGroup 测量与绘制 笔记



255. Java volatile 与 synchronized 区别



256. Java synchronized final 死锁

DATE / / M T W T F S S NO.

JAVA

④ synchronized 的用途区别，是否能嵌套。

1. 修饰代码块。被修饰的代码块称为同步代码块，其作用范围是不~~了~~括起来的代码，作用的对象是这个代码块的对象。
synchronized (this) 同步代码块时，其他试图访问该对象的线程将被阻塞。
当一个线程访问对象的一个 synchronized (this) 同步代码块时，另一个线程仍可以访问该对象中的非 synchronized (this) 同步代码块。

2. 修饰一个方法。被修饰的方法称为同步方法，其作用范围是整个方法，作用的对象是调用这个方法的对象。
注意：synchronized 关键字不能继承，父类 synchronized 修饰的方法子类覆盖时必须加入 synchronized，才能调用父类的方法。
在定义接口方法时不能使用 synchronized。
构造方法不能使用 synchronized，但可以使用 synchronized 修饰块来进行同步。

3. 修饰一个静态方法。其作用范围是整个方法，作用的对象是这个类的所有对象。（因为静态方法属于类不属于对象）。

4. 修饰一个类。其作用范围是到 synchronized 语句括起来的部分，作用的对象是这个类的所有对象。

⑤ final 的作用

可以声明成员变量、方法、类以及本地变量

final 变量：声明为 final 的成员变量或者本地变量都叫做 final 变量，只读。

final 方法：final 方法不可以被子类的方法重写，final 方法比非 final 方法更快，因为在编译时已经静态绑定。不可以是构造方法，但可以被继承。

final 类：不能被继承。final 类通常功能是完整的。

final 成员变量必须在声明的时候初始化或者在构造方法中初始化。
不能对 final 变量再次赋值。

final 参数：只读。

CHENYANG PAPER

DATE / / M T W T F S S NO

本地变量必须在声明时赋值

在匿名类中所有变量都必须是final变量

接口中声明的所有变量本身是final的

final与abstract是反相关，final类就不可能是abstract的

将类、方法、变量声明为final能提高性能，JVM对其进行优化

对final的集合对象是引用不能被更改，可以更改集合中的内容

final类中的方法默认是final的。

父类private方法不能被子类方法覆盖，因此private方法默认是final

⑥ 死锁

若干进程竞争有限资源，加上推进顺序不当，从而构成无限期循环等待的局面。如果没有外力的作用，那么死锁涉及的各个进程将永远处于封锁状态。

产生死锁的条件：(同时满足)

(1) 互斥条件：每个资源不能同时被两个或以上的进程占有。

(2) 不可抢占条件：进程所获得的资源在未使用完毕之前，资源申请者不能强行夺取资源。

(3) 占有且申请条件：已经占有资源的进程又申请新的资源。

(4) 循环等待(环路等待)条件：如 $\{P_1, P_2, \dots, P_n\}$, P_1 等待 P_2 , P_2 等待 P_3 , P_3 等待 P_1 ，形成一个进程循环等待环。

对待死锁的策略：阻塞、预防、避免、检测与恢复。

预防：破坏必要条件其中之一

避免：使用银行家算法：资源分配算法、安全性算法。

检测与恢复：