网络编程

2018年5月2日　　10:58

HTTP协议：客户端向服务器发出一条HTTP请求，服务器收到请求后会返回一些数据给客户端，然后客户端再对这些数据进行解析和处理。

- HttpURLConnection

1. 获取到HttpURLConnection实例：

   URL url = new URL（“http：//www.baidu.com”）

   HttpURLConnection connection = （HttpURLConnection） url.openConnection（）；

2. 设置HTTP请求所使用的方法
   - GRT：从服务器那里获取数据 connection.setRequestMethod（“GET”）
   - POST：提交数据给服务器
3. 自由定制（设置连接超时、读取超时）

   connection.setConnectTimeout（8000）；

   connection.setReadTimeout（8000）；
4. 调用getInputStream（），获取到服务器的输入流

   InputStream in = connection.getInputStream（）；
5. 调用disconnect（）方法将这个HTTP连接关闭

   connection.disconnect（）；

```
try{
   URL url = new URL("http://www.baidu.com");
  connection = (HttpsURLConnection) url.openConnection();
  connection.setRequestMethod("GET");
  connection.setConnectTimeout(8000);
  connection.setReadTimeout(8000);
  InputStream in = connection.getInputStream();
  //下面对获取到的输入流进行读取
  reader = new BufferedReader(new InputStreamReader(in));
  StringBuilder response = new StringBuilder();
  String line;
  while((line=reader.readLine()) != null){
       response.append(line);
       }
  showResponse(response.toString());
  }catch(Exceptione){
       e.printStackTrace();
  }finally{
  if(reader != null){
  try{
     reader.close();
  }catch(IOExceptione){
     e.printStackTrace();
  }
  }
   if(connection != null){
       connection.disconnect();
    }
  }
```

```
}
}).start();
```

控件：ScrollView，在手机屏幕的空间不够时，以滚动的形式查看屏幕外的那部分内容

- 网络通信库OkHttp

项目主页地址： https://github.com/square/okhttp

1、创建一个OkHttpClient实例

OkHttpClient client = new OkHttpClient（）；

2、要发起一条HTTP请求，就创建一个Request对象

Request request = new Request.Builder().#################.build();

3、调用OkHttpClient的newCall（）方法来创建一个Call对象，并调用execute（）方法来发送请求并获取服务器返回的数据

Response response = client.newCall（request）.execute（）；

其中Response对象就是服务器返回的数据

String responseData = response.body（）.string（）；//得到返回的具体内容

- 发起一条POST请求
  - 构建一个RequestBody对象来存放待提交的参数

    RequestBody requestBody = new FormBody.Builder（）

    .add（“username”， “admin”）

    .add（“password”， “123456”）

    .build（）；
  - 在Request.Builder中调用post（）方法，并将RequestBody对象传入

    Request request = new Request.Builder（）

    .url（“http：//www.baidu.com”）

    .post（requestBody）

    .build（）；
  - 调用execute（）方法

在网络上传输数据时最常用的格式有两种：XML和JSON

```
<apps>
    <app>
        <id>1</id>
        <name>Google Maps</name>
        <version>1.0</version>
    </app>
    <app>
        <id>2</id>
        <name>Chrome</name>
        <version>2.1</version>
    </app>
    <app>
        <id>3</id>
        <name>Google Play</name>
        <version>2.3</version>
    </app>
</apps>
```

.xml文件

```
[{"id":"5","version":"5.5","name":"Angry Birds"},
{"id":"6","version":"7.0","name":"Clash of Clans"},
{"id":"7","version":"3.5","name":"Hey Day"}]
```

.json文件

- 解析XML格式数据
  - Pull解析方式：parseXMLWithPull（responseData）；
    1) 获取到一个xmlPullParserFactory实例
    2) 得到xmlPullParser对象
    3) 调用xmlPullParser的setInput（）方法将服务器返回的XML数据设置进去开始解析
    4) 通过getEventTpye（）得到当前的解析事件
    5) while循环不断进行解析
  - SAX解析方式：parseXMLWithSAX（responseData）；
    1) 创建一个SAXParserFactory对象
    2) 获取XMLReader对象
    3) 将contentHandler实例设置到XMLReader
    4) 调用parse（）开始执行解析

    新建一个类继承自DefaultHandler，并重写父类的5个方法

```
public class MyHandler extends DefaultHandler {
    1. @Override //开始XML解析时调用
       public void startDocument() throws SAXException {
       }
    2. @Override //开始解析某个节点的时候调用
       public void startElement(String uri, String localName, String qName,
       Attributes attributes) throws SAXException {
       }
    3. @Override //获取节点中内容的时候调用
       public void characters(char[] ch, int start, int length) throws
       SAXException {
       }
    4. @Override //完成解析某个节点的时候调用
       public void endElement(String uri, String localName, String qName) throws
       SAXException {
       }
    5. @Override //完成整个XML解析的时候调用
       public void endDocument() throws SAXException {
       }
}
```

- 解析JSON格式数据

  Json体积小，在网络上传输的时候省流量，但语义性差，不如XML直观
  - 使用JSONObject：parseJSONwithJSONObject（responseData）；
    1) 将服务器定义的JSON数组返回的数据传入到一个JSONArray对象中
    2) 循环便利JSONArray（每一个元素都是JSONObject对象）
    3) 调用getString（）方法将数据取出
  - 使用GSON：parseJSONWithGSON（responseData）

  添加GSON库的依赖（compile ‘com.google.code.gson:gson:2.7’）

GSON库可以将一段JSON格式的字符串自动映射成一个对象，从而不用手动去编写代码进行解析

```
Gson gson = new Gson();
//Person person = gson.fromJson(jsonData, Person.class);
//List<Person> people = gson.fromJson(jsonData, new TypeToken<List<Person>>().getType());
```

当需要发起一条HTTP请求的时候，可以将通用的网络操作提取到一个公共的类里（静态方法）

```
public class HttpUtil {
    public static Sting sendHttpRequest(String address){
        HttpURLConnection connection = null;
        ......
        }
```

注意！！网络请求通常是属于耗时操作，而sendHttpRequest()方法的内部并没有开启线程（主线程阻塞）

回调机制：

✓ 定义一个接口，命名为HttpCallbackListener

```
public interface HttpCallbackListener{
    void onFinish(String response); //当服务器成功响应请求的时候调用
                                    服务器返回的数据
    void onError(Exception e); //当进行网络操作出现错误的时候调用
                               错误的详细信息
}
```

~~~~~~

```
public class HttpUtil {
    public static String sendHttpRequest(final String address, final HttpCallbackListener
    listener ){
        new Thread(new Runnable(){
        @override
        public void run(){
            HttpURLConnection connection = null;
            ......
            }).start();
        }
//调用sendHttpRequest()方法
HttpUtil.sendHttpRequest(address, new HttpCallbackListener() {
    @Override
    public void onFinish(String response) {
        // 在这里根据返回内容执行具体的逻辑
    }
    @Override
    public void onError(Exception e) {
        // 在这里对异常情况进行处理
        }
    });
//在方法的内部开启了一个子线程，然后在子线程里去执行具体的网络操作，没有return语句返回数据，
```
~~~~~~

而是用到HttpCallbackListener的onFonish（）方法和onError（）方法。

```
public class HttpUtil {
    public static void sendOkHttpRequest（String address，okhttp3.Callback callback）{
        OkHttpClient client = new OkHttpClient（）；                    OkHttp库中自带的回调接口
        Request request = new Request.Builder（）
                .url（address）
                .build（）；
        client.newCall（request）.enqueue（callback）；  //在内部开启子线程
        }
    }
    //调用sendHttpRequest（）方法
    HttpUtil.sendOkHttpRequest(address, new okhttp3.Callback() {
        @Override
        public void onResponse(Call call，Response response) throws IOException {
            // 得到服务器返回的具体内容
            String responseData = response.body（）.string（）；
        }
        @Override
        public void onFailure (Call call，IOException e) {
            // 在这里对异常情况进行处理
            }
        }）；
```

注意！！！不管是使用HttpURLConnection 还是OkHttp，最终的回调接口都还是在子线程中运行的，因此不可以在这里执行任何的UI操作，但可以使用runOnUiThread（）方法来进行线程转换。