《移动终端开发技术》

电子教案

第五单元 网络编程的最佳实践

所属专业（教研室）：计算机软件技术

制定人：陈媛媛

合作人：

制定时间：2018 年 2 月

日照职业技术学院

单元标题	网络编程的最佳实践	**单元教学学时**	6 课时
		在整体设计中的位置	第 30 次
授课班级		**上课地点**	一体化教室
上课时间	周　　月　　日第　　节		
教学目标	**能力目标**	**知识目标**	**素质目标**
	能够熟练运用工具类、Java 回调机制优化项目。	1、理解工具类； 2、掌握静态方法的格式； 3、掌握 Java 回调机制的使用方法。	1、养成积极主动学习意识； 2、养成勤于动手的习惯。
教学重点、难点	教学重点：静态方法的格式 教学难点：Java 回调机制的使用		
教学方法	采用反转课堂教学模式，课前学生学习微课了解知识点，课上采用教师引导、演示，学生分组练习、讨论等教学方法。 运用多媒体、AndroidStudio 开发环境、实训助手、教学平台等辅助授课。		
课前需掌握的知识点	使用 HttpURLConnection 访问网络 使用 OkHttp 访问网络		
教学任务分解	任务一、编写工具类 任务二、使用 Java 回调机制		
教学总结			

教学过程

一、情景导入

目前我们已经掌握了 HttpURLConnection 和 HttpClient 的用法，知道了如何发起 HTTP 请求，以及解析服务器返回的数据，但也许你还没有发现，之前我们的写法其实是很有问题的。因为每一个应用程序很有可能会在很多地方都使用到网络功能，而发送 HTTP 请求的代码基本都是相同的，如果我们每次都去编写一遍发送 HTTP 请求的代码，这显然是非常差劲的做法。那大家思考一下该如何解决呢？

二、复习

如何使用使用 HttpURLConnection 和 OkHttp 访问网络？

三、知识讲解

1、HttpURLConnection

任务一、编写工具类

通常情况下我们都应该将这些通用的网络操作提取到一个公共的类里，并提供一个静态方法，当想要发起网络请求的时候只需简单地调用一下这个方法即可。比如使用如下的写法：

```
public class HttpUtil {
      public static String sendHttpRequest(String address) {
        HttpURLConnection connection = null;
          try {
            URL url = new URL(address);
            connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
            connection.setConnectTimeout(8000);
            connection.setReadTimeout(8000);
            connection.setDoInput(true);
            connection.setDoOutput(true);
            InputStream in = connection.getInputStream();
          BufferedReader reader = new BufferedReader(new InputStreamReader
(in));
           StringBuilder response = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                response.append(line);
              }
              return response.toString();
            } catch (Exception e) {
                e.printStackTrace();
               return e.getMessage();
                } finally {
              if (connection != null) {
```

```
                connection.disconnect();
            }
            }
        }
    }
```

以后每当需要发起一条 HTTP 请求的时候就可以这样写：

```
String address = "http://www.baidu.com";
String response = HttpUtil.sendHttpRequest(address);
```

在获取到服务器响应的数据后我们就可以对它进行解析和处理了。但是需要注意，网络请求通常都是属于耗时操作，而 sendHttpRequest() 方法的内部并没有开启线程，这样就有可能导致在调用 sendHttpRequest() 方法的时候使得主线程被阻塞住。

你可能会说，很简单嘛，在 sendHttpRequest() 方法内部开启一个线程不就解决这个问题了吗？其实不是像你想象中的那么容易，因为如果我们在 sendHttpRequest() 方法中开启了一个线程来发起 HTTP 请求，那么服务器响应的数据是无法进行返回的，所有的耗时逻辑都是在子线程里进行的，sendHttpRequest() 方法会在服务器还没来得及响应的时候就执行结束了，当然也就无法返回响应的数据了。

那么遇到这种情况应该怎么办呢？其实解决方法并不难，只需要使用 Java 的回调机制就可以了，下面让我们来学习一下回调机制到底是如何使用的。

任务二、使用 Java 回调机制

首先需要定义一个接口，比如将它命名成 HttpCallbackListener，代码如下所示：

```
public interface HttpCallbackListener {
    void onFinish(String response);
    void onError(Exception e);
}
```

可以看到，我们在接口中定义了两个方法，onFinish() 方法表示当服务器成功想要我们请求的时候调用，onError() 表示当进行网络操作出现错误的时候调用。这两个方法都带有参数，onFinish() 方法中的参数代表着服务器返回的数据，而 onError() 方法中的参数记录着错误的详细信息。

接着修改 HttpUtil 中的代码，如下所示：

```
public class HttpUtil {
    public static void sendHttpRequest(final String address, final HttpCallbackListener
listener) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                HttpURLConnection connection = null;
                try {
                    BufferedReader reader = null;
```

```
                URL url = new URL(address);
                connection = (HttpURLConnection) url.openConnection();
                connection.setRequestMethod("GET");
                connection.setConnectTimeout(8000);
                 connection.setReadTimeout(8000);
                 connection.setDoInput(true);
                 connection.setDoOutput(true);
                 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);
                  }
                  if (listener != null) {
                     //回掉 onFinish 方法
                    listener.onFinish(response.toString());
                   }
                } catch (Exception e) {
                   if (listener != null) {
                      //回掉 onError()方法
                     listener.onError(e);
                    }
                 } finally {
                  if (connection != null) {
                   connection.disconnect();
                  }
                }
            }
        }).start();
    }
```

我们首先给 sendHttpRequest() 方法添加了一个 HttpCallbackListener 参数，并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。注意子线程中是无法通过 onFinish() 方法中，如果出现了异常就将异常原因传入到 onError() 方法中。

现在 sendHttpRequest() 方法接收两个参数了，因此我们在调用它的时候还需要将 HttpCallbackListener 的实例传入，如下所示：

```
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
            }
    });

    修改 MainActivity
public void onClick(View view) {
    if(view.getId()==R.id.send_request){
        //sendRequestWithOkHttp();
       // sendRequestWithHttpUrlConnection();
        //最佳实践
       HttpUtil.sendHttpRequest("https://www.baidu.com", new HttpCallbackListener() {
            @Override
            public void onFinish(String response) {
              showResponse(response);
            }

            @Override
            public void onError(Exception e) {
                 e.printStackTrace();
            }
        });
         }
}
```

这样的话，当服务器成功响应的时候我们就可以在 onFinish() 方法里对响应数据进行处理了，类似地，如果出现了异常，就可以在 onError() 方法里对异常情况进行处理。如此一来，我们就巧妙地利用回调机制将响应数据成功返回给调用方了。

另外需要注意的是，onFinish() 方法和 onError() 方法最终还是在子线程中运行的，因此我们不可以在这里执行任何的 UI 操作，如果需要根据返回的结果来更新 UI，则仍然要使用 《后台默默的劳动者，探究服务》 中我们学习的异步消息处理机制。

2、OkHttp

在 HttpUtil 中加入一个 sendOkHttpRequest()方法。

```
  public class HttpUtil {
...
  public static void sendOkHttpRequest(String address,okhttp3.Callback callback){
    OkHttpClient client=new OkHttpClient();
    Request request=new Request.Builder().url(address).build();
    client.newCall(request).enqueue(callback);

}
```

修改 MainActivity

```
public void onClick(View view) {
    if(view.getId()==R.id.send_request){
        //sendRequestWithOkHttp();
        // sendRequestWithHttpUrlConnection();
        //最佳实践
              //最佳实践 使用 OkHttp
        HttpUtil.sendOkHttpRequest("http://www.baidu.com", new okhttp3.Callback() {
            @Override
            public void onFailure(Call call, IOException e) {

            }
            @Override
            public void onResponse(Call call, Response response) throws IOException {
                String responseData=response.body().string();
                showResponse(responseData);
            }
        });
    }
}
```

可以看到，在 sendOkHttpRequest()方法中又一个 okhttp3.Callback 的参数，这个是 OkHttp 库中自带的一个回掉接口。然后在 client.newCall()之后没有调用 execute()方法，而是调用了 enqueue()方法，并把 okhttp3.Callback 的参数传入。OkHttp 在 enqueue()方法的内部已经帮我们开好子线程了，然后会在子线程中执行 HTTP 请求，并将最终的请求结果回调到 okhttp3.Callback 当中。

四、演示作品

各小组演示汇报作品，教师与其他小组总结评价作品，各小组课后完善作品并提交到教学平台。

五、知识巩固

总结知识点，使用教学平台中的随堂练习题巩固本所学知识。

拓展作业	使用工具类及 Java 回调机制完成访问网络功能。
教学后记	