网络编程的最佳实践

1. 实训目的

（1）能够创建工具类；

（2）掌握 java 回调函数的使用。

2. 实训要求

（1）编写工具类 HttpUtil；

（2）定义接口 HttpCallbackListener；

（3）定义抽象方法；

（4）完成实验报告。

3. 实训指导

（1）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();
}
}
}
}
```

（2）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();
}
```

（3）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。