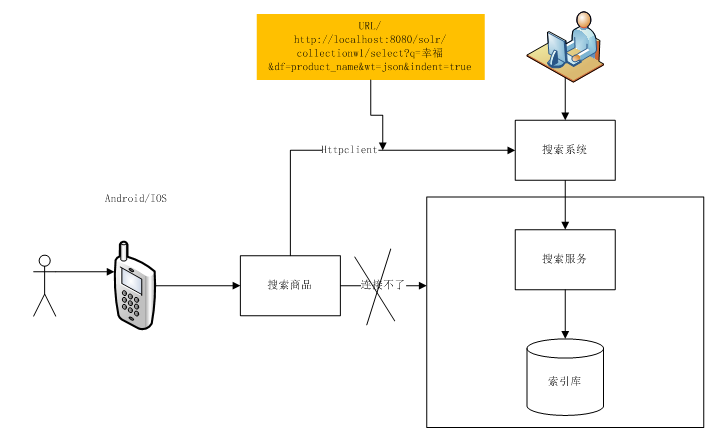
HttpClint

# HttpClient



## HttpClient简介

HttpClient 是 Apache Jakarta Common 下的子项目，可以用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且它支持 HTTP 协议最新的版本和建议。

HttpClient 提供的主要的功能

（1）实现了所有 HTTP 的方法（GET,POST,PUT,DELETE等）

（2）支持自动转向

（3）支持 HTTPS 协议

（4）支持代理服务器等

我们所用的solrj其实就封装了HttpClient

## 学习httpclient

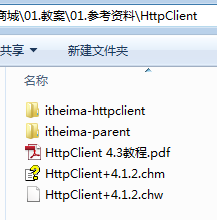
http状态码：



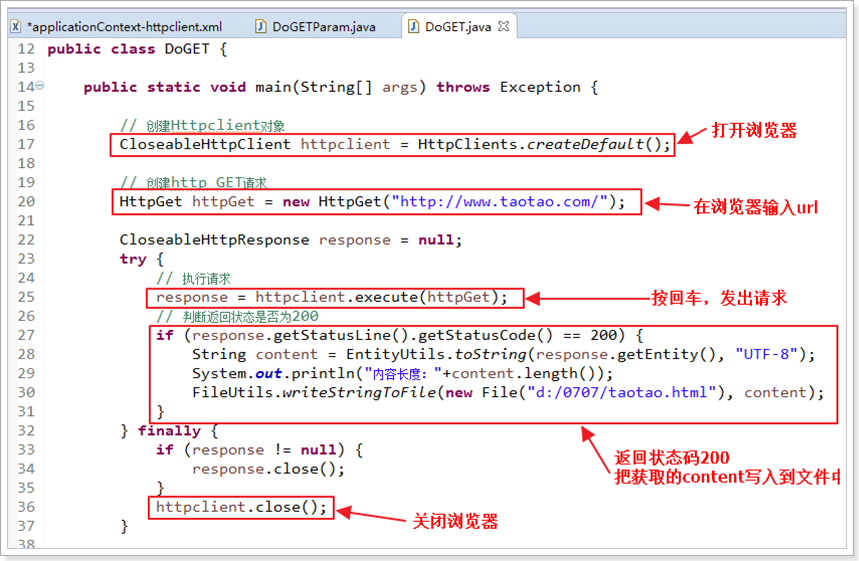
常用的：200 201 202 400 404 405 406 415 500 501 502 503

|  |  |
| --- | --- |
| 200 OK | 请求已成功，请求所希望的响应头或数据体将随此响应返回。 |
| 201 Created | 请求已经被实现，而且有一个新的资源已经依据请求的需要而建立，且其 URI 已经随Location 头信息返回 |
| 202 Accepted | 服务器已接受请求，但尚未处理 |
| 400 Bad Request | 1、语义有误，当前请求无法被服务器理解  2、请求参数有误 |
| 403 Forbidden | 服务器已经理解请求，但是拒绝执行它 |
| 404 Not Found | 请求失败，请求所希望得到的资源未被在服务器上发现 |
| 405 Method Not Allowed | 请求行中指定的请求方法不能被用于请求相应的资源 |
| 406 Not Acceptable | 请求的资源的内容特性无法满足请求头中的条件，因而无法生成响应实体 |
| 415 Unsupported Media Type | 对于当前请求的方法和所请求的资源，请求中提交的实体并不是服务器中所支持的格式，因此请求被拒绝 |
| 500 Internal Server Error | 服务器遇到了一个未曾预料的状况，导致了它无法完成对请求的处理 |
| 501 Not Implemented | 服务器不支持当前请求所需要的某个功能 |
| 502 Bad Gateway | 作为网关或者代理工作的服务器尝试执行请求时，从上游服务器接收到无效的响应 |
| 503 Service Unavailable | 由于临时的服务器维护或者过载，服务器当前无法处理请求，服务器宕机 |

通过此工程进行学习：



### 无参数的GET请求



### 带参数的GET请求

|  |
| --- |
| **public** **class** DoGETParam {  **public** **static** **void** main(String[] args) **throws** Exception {  // 创建Httpclient对象  CloseableHttpClient httpclient = HttpClients.*createDefault*();  // 定义请求的参数  URI uri = **new** URIBuilder("http://www.baidu.com/s").setParameter("wd", "java").build();  System.***out***.println(uri);  // 创建http GET请求  HttpGet httpGet = **new** HttpGet(uri);  CloseableHttpResponse response = **null**;  **try** {  // 执行请求  response = httpclient.execute(httpGet);  // 判断返回状态是否为200  **if** (response.getStatusLine().getStatusCode() == 200) {  String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");  System.***out***.println(content);  }  } **finally** {  **if** (response != **null**) {  response.close();  }  httpclient.close();  }  }  } |

### 无参数的POST请求

|  |
| --- |
| Public static void main(String[] args) throws Exception {  // 创建Httpclient对象  CloseableHttpClient httpclient = HttpClients.createDefault();  // 创建http POST请求  HttpPost httpPost = new HttpPost("http://www.oschina.net/");  // 在请求中设置请求头,设置请求头，跳过开源中国的访问限制  httpPost.setHeader("User-Agent", "");  CloseableHttpResponse response = null;  try {  // 执行请求  response = httpclient.execute(httpPost);  // 判断返回状态是否为200  if (response.getStatusLine().getStatusCode() == 200) {  String content = EntityUtils.toString(response.getEntity(), "UTF-8");  System.out.println(content);  }  } finally {  if (response != null) {  response.close();  }  httpclient.close();  }  } |

### 带参数的POST请求

|  |
| --- |
| **public** **class** DoPOSTParam {  **public** **static** **void** main(String[] args) **throws** Exception {  // 创建Httpclient对象  CloseableHttpClient httpclient = HttpClients.*createDefault*();  // 创建http POST请求  HttpPost httpPost = **new** HttpPost("http://www.oschina.net/search");    // 设置2个post参数，一个是scope、一个是q  List<NameValuePair> parameters = **new** ArrayList<NameValuePair>(0);  parameters.add(**new** BasicNameValuePair("scope", "project"));  parameters.add(**new** BasicNameValuePair("q", "java"));  // 构造一个form表单式的实体  UrlEncodedFormEntity formEntity = **new** UrlEncodedFormEntity(parameters);  // 将请求实体设置到httpPost对象中  httpPost.setEntity(formEntity);  CloseableHttpResponse response = **null**;  **try** {  // 执行请求  response = httpclient.execute(httpPost);  // 判断返回状态是否为200  **if** (response.getStatusLine().getStatusCode() == 200) {  String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");  System.***out***.println(content);  }  } **finally** {  **if** (response != **null**) {  response.close();  }  httpclient.close();  }  }  } |

# 使用HttpClient调用接口

## 编写返回对象

**public class** HttpResult {

// 响应的状态码

**privateint** code;

// 响应的响应体

**private** String body;

get/set…

}

## 封装HttpClient

**public class** ApiService {

**private** CloseableHttpClient httpClient;

**public** ApiService() {

**this**.httpClient = HttpClients.*createDefault*();

}

/\*\*

\* 带参数的get请求

\*

\* **@param** url

\* **@param** map

\* **@return**

\* **@throws** Exception

\*/

**public** HttpResult doGet(String url, Map<String, Object>map) **throws** Exception {

// 1.创建URIBuilder

URIBuilder uriBuilder = **new** URIBuilder(url);

// 2.设置请求参数

**if** (map != **null**) {

// 遍历请求参数

**for** (Map.Entry<String, Object>entry : map.entrySet()) {

// 封装请求参数

uriBuilder.setParameter(entry.getKey(), entry.getValue().toString());

}

}

// 3.创建请求对象httpGet

HttpGet httpGet = **new** HttpGet(uriBuilder.build());

// 4.使用httpClient发起请求

CloseableHttpResponse response = **this**.httpClient.execute(httpGet);

// 5.解析返回结果，封装返回对象httpResult

// 获取状态码

**int**code = response.getStatusLine().getStatusCode();

// 获取响应体

// 使用EntityUtils.toString方法必须保证entity不为空

String body;

**if** (response.getEntity() != **null**) {

body = EntityUtils.*toString*(response.getEntity(), "UTF-8");

} **else** {

body = **null**;

}

**returnnew** HttpResult(code, body);

}

/\*\*

\* 不带参数的get

\*

\* **@param** url

\* **@return**

\* **@throws** Exception

\*/

**public** HttpResult doGet(String url) **throws** Exception {

**returnthis**.doGet(url, **null**);

}

/\*\*

\* 带参数的post请求

\*

\* **@param** url

\* **@param** map

\* **@return**

\* **@throws** Exception

\*/

**public** HttpResult doPost(String url, Map<String, Object>map) **throws** Exception {

// 1. 声明httppost

HttpPost httpPost = **new** HttpPost(url);

// 2.封装请求参数，请求数据是表单

**if** (map != **null**) {

// 声明封装表单数据的容器

List<NameValuePair>parameters = **new** ArrayList<NameValuePair>();

**for** (Map.Entry<String, Object>entry : map.entrySet()) {

// 封装请求参数到容器中

parameters.add(**new** BasicNameValuePair(entry.getKey(), entry.getValue().toString()));

}

// 创建表单的Entity类

UrlEncodedFormEntity entity = **new**UrlEncodedFormEntity(parameters, "UTF-8");

// 3. 把封装好的表单实体对象设置到HttpPost中

httpPost.setEntity(entity);

}

// 4. 使用Httpclient发起请求

CloseableHttpResponse response = **this**.httpClient.execute(httpPost);

// 5. 解析返回数据，封装HttpResult

// 状态码

**int**code = response.getStatusLine().getStatusCode();

// 响应体内容

String body = **null**;

**if** (response.getEntity() != **null**) {

body = EntityUtils.*toString*(response.getEntity(), "UTF-8");

}

**returnnew** HttpResult(code, body);

}

/\*\*

\* 不带参数的post请求

\*

\* **@param** url

\* **@return**

\* **@throws** Exception

\*/

**public** HttpResult doPost(String url) **throws** Exception {

**returnthis**.doPost(url, **null**);

}

//doPut

//dodelete

}

这样：如果手机客户端，需要搜索，则我们只需要提供给它这个APIService类，再告诉它调用的URL，以及各参数的含义。手机客户端调用这些方法，返回的JSON数据自己做处理便可以了。这样就实现了系统之间的通信。

# Cookies相关

## 响应中的Set-Cookie

服务器生成Cookie，会产生一个Set-Cookie报头，放在HTTP报文中一起回传客户端。不是唯一，一个Set-Cookie报头对应一条Cookie。浏览器一般允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB

## cookie作用

web程序基于[http协议](https://www.baidu.com/s?wd=http%E5%8D%8F%E8%AE%AE&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao) [http协议](https://www.baidu.com/s?wd=http%E5%8D%8F%E8%AE%AE&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)本身是无状态的 那么你两次访问一个页面的时候 服务器怎么认识是张三还是李四呢 这时候就用到了cookie 他就像是个凭据 这也是cookie最主要的作用 其他的一些数据也可以保存在里面

## cookie:

　　1.定义:什么是cookie? 　　cookie就是存储在客户端的一小段文本

　　2.cookie是一门客户端的技术,因为cookie是存储在客户端浏览器中的

　　3.cookie的作用:是为了实现客户端与服务器之间状态的保持

　　4.cookie 技术不安全,不要使用cookie保存敏感信息

　　5.cookie默认 在浏览器关闭之后,就立即实现失效.如果想指定cookie的过期时间,需要通过使用expires属性实现.在服务器响应返回响应头时

　　　　写入cookie的过期时间. 即响应头设置 set-cookie:[expires=new.Date(Date.now() +10 \*1000)]   10S后过期

原理:由于http协议是无状态的.传统服务器只能被动响应请求.当服务器获取到请求,并为了能够区分每一个客户端,需要客户端发送请求时发送一个标识符(cookie),

也因此为了提供这个标识符,产生了cookie技术.我们在请求头(Request Headers)中添加了标识符(cookie). 每次发送请求,都会把这个cookie随同其它报文一起发送给服务器.

服务器根据报文中cookie,进行区分客户端浏览器.