**[浅谈HTTP中GET、POST用法以及它们的区别](https://www.cnblogs.com/williamjie/p/9099940.html)**

**浅谈HTTP中GET、POST用法以及它们的区别**

HTTP定义了与服务器交互的不同方法，最基本的方法有4种，分别是GET，POST，PUT，DELETE。URL全称是资源描述符。我们可以这样认为： 一个URL地址，它用于描述一个网络上的资源，而HTTP中的GET，POST，PUT，DELETE就对应着对这个资源的 查，改，增，删 4个操作。到这里，大家应该有个大概的了解了，GET一般用于获取/查询资源信息，而POST一般用于更新资源信息。那么，除了上面说的四种方法，HTTP还有其它方法么？其实HTTP中定义了以下几种请求方法：

* GET方法；
* POST方法；
* PUT方法；
* DELETE方法。
* HEAD方法；
* TRACE方法；
* OPTIONS方法；

1. Get是最常用的方法，通常用于请求服务器发送某个资源，而且应该是安全的和幂等的。

(1). 所谓安全是指该操作用于获取信息而非修改信息。换句话说，GET 请求一般不应产生副作用。就是说，它仅仅是获取资源信息，就像数据库查询一样，不会修改和增加数据，不会影响资源的状态。

　　注意：这里安全的含义仅仅是指是非修改信息。

(2). 幂等是指对同一个URL的多个请求应该返回同样的结果。这里我再解释一下幂等这个概念：

幂等（idempotent、idempotence）是一个数学或计算机学概念，常见于抽象代数中。   
幂等有以下几种定义：   
　　对于单目运算，如果一个运算对于在范围内的所有的一个数多次进行该运算所得的结果和进行一次该运算所得的结果是一样的，那么我们就称该运算是幂等的。比如绝对值运算就是一个例子，在实数集中，有abs(a)=abs(abs(a))。   
　　对于双目运算，则要求当参与运算的两个值是等值的情况下，如果满足运算结果与参与运算的两个值相等，则称该运算幂等，如求两个数的最大值的函数，在实数集中便是幂等的 ，即max(x,x) = x。

1. POST方法向服务器提交数据，比如完成表单数据的提交，将数据提交给服务器处理。
2. PUT方法是让服务器用请求的主体部分来创建一个由所请求的URL命名的新文档；如果那个文档存在的话，就用这个主体来代替它。
3. DELETE方法就是请求服务器删除指定URL所对应的资源。但是，客户端无法保证删除操作一定会被执行，因为HTTP规范允许服务器在不通知客户端的情况下撤销请求。
4. HEAD方法与GET方法的行为很类似，但服务器在响应中只返回实体的主体部分。这就允许客户端在未获取实际资源的情况下，对资源的首部进行检查，

使用HEAD，我们可以更高效的完成以下工作：   
①. 在不获取资源的情况下，了解资源的一些信息，比如资源类型；   
②. 通过查看响应中的状态码，可以确定资源是否存在；   
③. 通过查看首部，测试资源是否被修改。

1. TRACE方法会在目的服务器端发起一个“回环”诊断，我们都知道，客户端在发起一个请求时，这个请求可能要穿过防火墙、代理、网关、或者其它的一些应用程序。这中间的每个节点都可能会修改原始的HTTP请求，TRACE方法允许客户端在最终将请求发送服务器时，它变成了什么样子。由于有一个“回环”诊断，在请求最终到达服务器时，服务器会弹回一条TRACE响应，并在响应主体中携带它收到的原始请求报文的最终模样。这样客户端就可以查看HTTP请求报文在发送的途中，是否被修改过了。
2. OPTIONS方法用于获取当前URL所支持的方法。若请求成功，则它会在HTTP头中包含一个名为“Allow”的头，值是所支持的方法，如“GET, POST”。

GET和POST的区别：

1. GET请求的数据会附在URL之后（就是把数据放置在HTTP协议头中），以？分割URL和传输数据，参数之间以&相连，如：login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0%E5%A5%BD。如果数据是英文字母或数字，则原样发送；如果是空格，转换为+；如果是中文或其他字符，则直接把字符串用BASE64加密，得出如：%E4%BD%A0%E5%A5%BD，其中％XX中的XX为该符号以16进制表示的ASCII码值。而与之对应的，POST把提交的数据放置在HTTP包的包体中，文章最下面将会有代码示例。
2. POST的安全性要比GET的安全性高。注意：这里所说的安全性和上面GET提到的“安全”不是同个概念。上面“安全”的含义仅仅是不作数据修改，而这里安全的含义是真正的Security的含义。比如：通过GET提交数据，用户名和密码将明文出现在URL上，因为：(1)登录页面有可能被浏览器缓存，(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用GET提交数据还可能会造成Cross-site request forgery攻击（CSRF,跨站请求伪造，也被称为：one click attack/session riding）。

例：银行网站A，它以GET请求来完成银行转账的操作，如：[http://www.mybank.com/Transfer.php?toBankId=11&money=1000](http://www.mybank.com/Transfer.php?toBankId=11&money=1000" \t "_blank)   
危险网站B，它里面有一段HTML的代码如下：< img src=<http://www.mybank.com/Transfer.php?toBankId=11&money=1000/>>

首先，你登录了银行网站A，然后访问危险网站B，噢，这时你会发现你的银行账户少了1000块……   
为什么会这样呢？原因是银行网站A违反了HTTP规范，使用GET请求更新资源。在访问危险网站B之前，你已经登录了银行网站A，而B中的< img …/>以GET的方式请求第三方资源（这里的第三方就是指银行网站A了。原本这是一个合法的请求，但这里被不法分子利用了），所以你的浏览器会带上你的银行网站A的Cookie发出Get请求，去获取资源“<http://www.mybank.com/Transfer.php?toBankId=11&money=1000>”，结果银行网站服务器收到请求后，认为这是一个更新资源操作（转账操作），所以就立刻进行转账操作……   
为了杜绝上面的问题，银行改用POST请求完成转账操作。

GET和POST的误区：   
误区一：POST可以比GET提交更多更长的数据？

由于使用GET方法提交数据时，数据会以&符号作为分隔符的形式，在URL后面添加需要提交的参数，有人会说，浏览器地址栏输入的参数是有限的，而POST不用再地址栏输入，所以POST就比GET可以提交更多的数据。难道真的是这样的么？

而实际上，URL不存在参数上限的问题，HTTP协议规范没有对URL长度进行限制。这个限制是特定的浏览器及服务器对它的限制。IE对URL长度的限制是2083字节(2K+35)。对于其他浏览器，如Netscape、FireFox等，理论上没有长度限制，其限制取决于操作系统的支持。所以POST也是没有大小长度限制的，HTTP协议规范也没有进行大小限制。起限制作用的是服务器的处理能力。总归一句话，这个限制是针对所有HTTP请求的，与GET、POST没有多少关系。

注意：   
上面大概说了一下HTTP规范中GET和POST的一些原理性的问题。但在实际的做的时候，很多人却没有按照HTTP规范去做，导致这个问题的原因有很多，比如说：

1. 很多人贪方便，更新资源时用了GET，因为用POST必须要用到FORM（表单），这样会麻烦一点。
2. 对资源的增，删，改，查操作，其实都可以通过GET/POST完成，不需要用到PUT和DELETE。
3. 早期的Web MVC框架设计者们并没有有意识地将URL当作抽象的资源来看待和设计，所以导致一个比较严重的问题是传统的Web MVC框架基本上都只支持GET和POST两种HTTP方法，而不支持PUT和DELETE方法。
4. 大家都觉得使用GET很方便，毕竟使用POST要用到Form。但是使用GET方法时，浏览器会缓存你的地址等信息，留下历史记录和Cookie。而对于POST方法，则不会进行缓存。以后在开发中，一定要分清楚GET和POST的使用场合

创建和更新某个URL代表的资源的时候，是用HTTP的PUT还是POST：

摘抄自：[http://www.cnblogs.com/shanyou/archive/2011/10/17/2215930.html](http://www.cnblogs.com/shanyou/archive/2011/10/17/2215930.html" \t "_blank)

其实，用PUT还是POST，不是看这是创建还是更新资源的动作，这不是风格的问题，而是语义的问题。REST是一种风格，但是还是依赖于HTTP协议。在HTTP中，PUT被定义为idempotent的方法，POST则不是，这是一个很重要的区别。

“Methods can also have the property of “idempotence” in that (aside from error or expiration issues) the side-effects of N > 0 identical requests is the same as for a single request.”

上面的话就是说，如果一个方法重复执行多次，产生的效果是一样的，那就是idempotent的。

举一个简单的例子，假如由一个博客系统提供一个Web API，模式是 <http://superblogging/blogs/post/>{blog-name} 。很简单，将{blog-name}替换为我们的blog名字，往这个URI发送一个HTTP PUT或者POST请求，HTTP的body部分就是博文，这是一个很简单的REST API例子。我们应该用PUT方法还是POST方法？取决于这个REST服务的行为是否是idempotent的，假如我们发送两个[http://superblogging/blogs/post/Sample](http://superblogging/blogs/post/Sample" \t "_blank)请求，服务器端是什么样的行为？如果产生了两个博客帖子，那就说明这个服务不是idempotent的，因为多次使用产生了副作用；如果后一个请求把第一个请求覆盖掉了，那这个服务就是idempotent的。前一种情况，应该使用POST方法，后一种情况，应该使用PUT方法。

也许你会觉得这两个方法的差别没什么大不了的，用错了也不会有什么问题，但是你的服务一放到internet上，如果不遵从HTTP协议的规范，就可能给自己带来麻烦。比如，没准Google Crawler也会访问你的服务，如果让一个不是indempotent的服务可以用indempotent的方法访问，那么你服务器的状态可能就会被Crawler修改，这是不应该发生的。

使用GET：

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

new AsyncTask<String, Void, Void>() { //网络加载是耗时操作，放在异步任务中

@Override

protected Void doInBackground(String... params) { //相当于线程中的run方法，执行后台操作

try {

URL u = new URL(params[0]);

URLConnection conn = u.openConnection(); //获取URL的互联网连接

InputStream is = conn.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr); //包裹到BufferedReader中

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

br.close();

isr.close();

is.close();

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

}.execute("http://apis.juhe.cn/oil/region?key=3f73d0ea9d7c33d288fdc16f5257c1a5&format=2&city=%E5%8C%97%E4%BA%AC%E5%B8%82");

//execte(URL)使用GET方法时，填入的URL是带有信息的，例子中使用的是聚合数据提供的加油站数据

}

使用POST:

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

new AsyncTask<String, Void, Void>() {

@Override

protected Void doInBackground(String... params) { //相当于线程中的run方法，执行后台操作

try {

URL u = new URL(params[0]);

HttpURLConnection conn = (HttpURLConnection) u.openConnection();

//使用POST，需将URLConnection转换成HttpURLConnection

//使用之前，需要先对conn进行配置

conn.setDoInput(true);

conn.setDoOutput(true); //设置成true，conn才能向服务器输出数据

conn.setRequestMethod("POST"); //请求方式设置为POST

OutputStreamWriter osw = new OutputStreamWriter(conn.getOutputStream()); //把数据传到服务器

BufferedWriter bw = new BufferedWriter(osw);

bw.write("key=3f73d0ea9d7c33d288fdc16f5257c1a5&format=2&city=%E5%8C%97%E4%BA%AC%E5%B8%82"); //要传递的数据

bw.flush();

InputStream is = conn.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

br.close();

isr.close();

is.close();

} catch (IOException e) {

e.printStackTrace();

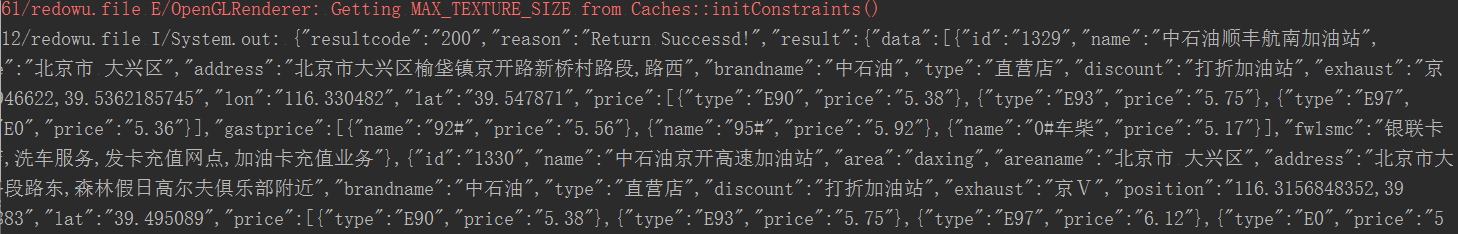
}

return null;

}

}.execute("http://apis.juhe.cn/oil/region");

}

上述两段代码获取到的数据结果一样，如下图：   


从图片可以看出，我们已经成功获取到数据

本文部分内容转载自下面的博文，但是原博文的部分内容是不正确的，我对其做了总结和修改。因此读者在前往原文观看时，先擦亮眼睛哈，多看看，多瞧瞧。还有，如果我也错了，请大家一定要指正：   
<http://www.jellythink.com/archives/806>   
<http://www.cnblogs.com/shanyou/archive/2011/10/17/2215930.html>   
<http://www.cnblogs.com/hyddd/archive/2009/03/31/1426026.html>