## [浅谈HTTP中Get与Post的区别](http://www.cnblogs.com/hyddd/archive/2009/03/31/1426026.html)

2009-03-31 14:51 by hyddd, 263290 阅读, 74 评论, [收藏](http://www.cnblogs.com/hyddd/archive/2009/03/31/1426026.html), [编辑](http://i.cnblogs.com/EditPosts.aspx?postid=1426026)

|  |
| --- |
| **HTTP** |
| * [Persistence](https://en.wikipedia.org/wiki/HTTP_persistent_connection) * [Compression](https://en.wikipedia.org/wiki/HTTP_compression) * [HTTPS](https://en.wikipedia.org/wiki/HTTPS) |
| [**Request methods**](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) |
| * [OPTIONS](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) * [GET](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) * [HEAD](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) * [POST](https://en.wikipedia.org/wiki/POST_%28HTTP%29) * [PUT](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) * [DELETE](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) * [TRACE](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) * [CONNECT](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) * [PATCH](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) |
| [**Header fields**](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields) |
| * [Cookie](https://en.wikipedia.org/wiki/HTTP_cookie) * [ETag](https://en.wikipedia.org/wiki/HTTP_ETag) * [Location](https://en.wikipedia.org/wiki/HTTP_location) * [HTTP referer](https://en.wikipedia.org/wiki/HTTP_referer) * [DNT](https://en.wikipedia.org/wiki/Do_Not_Track) * [X-Forwarded-For](https://en.wikipedia.org/wiki/X-Forwarded-For) |
| [**Status codes**](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes) |
| * [301 Moved Permanently](https://en.wikipedia.org/wiki/HTTP_301) * [302 Found](https://en.wikipedia.org/wiki/HTTP_302) * [303 See Other](https://en.wikipedia.org/wiki/HTTP_303) * [403 Forbidden](https://en.wikipedia.org/wiki/HTTP_403) * [404 Not Found](https://en.wikipedia.org/wiki/HTTP_404) * [451 Unavailable For Legal Reasons](https://en.wikipedia.org/wiki/HTTP_451) |

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。

**1**.根据HTTP规范，GET用于信息获取，而且应该是安全的和幂等的。

　　(1).所谓安全的意味着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，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).幂等的意味着对同一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。  
　　幂等有一下几种定义：  
　　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，有在在实数集中幂等，即max(x,x) = x。

看完上述解释后，应该可以理解GET幂等的含义了。

　　但在实际应用中，以上2条规定并没有这么严格。引用别人文章的例子：比如，新闻站点的头版不断更新。虽然第二次请求会返回不同的一批新闻，该操作仍然被认为是安全的和幂等的，因为它总是返回当前的新闻。从根本上说，如果目标是当用户打开一个链接时，他可以确信从自身的角度来看没有改变资源即可。

**2**.根据HTTP规范，POST表示可能修改变服务器上的资源的请求。继续引用上面的例子：还是新闻以网站为例，读者对新闻发表自己的评论应该通过POST实现，因为在评论提交后站点的资源已经不同了，或者说资源被修改了。

　　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**.另外一个是，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。

 　　\* 简单解释一下MVC：MVC本来是存在于Desktop程序中的，M是指数据模型，V是指用户界面，C则是控制器。使用MVC的目的是将M和V的实现代码分离，从而使同一个程序可以使用不同的表现形式。

　　以上3点典型地描述了老一套的风格（没有严格遵守HTTP规范），随着架构的发展，现在出现REST(Representational State Transfer)，一套支持HTTP规范的新风格，这里不多说了，可以参考《RESTful Web Services》。

　　说完原理性的问题，我们再从表面现像上面看看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/数字，原样发送，如果是空格，转换为+，如果是中文/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。

　　POST把提交的数据则放置在是HTTP包的包体中。

**2**."GET方式提交的数据最多只能是1024字节，理论上POST没有限制，可传较大量的数据，IIS4中最大为80KB，IIS5中为100KB"？？！

　　以上这句是我从其他文章转过来的，其实这样说是错误的，不准确的：

　　(1).首先是"GET方式提交的数据最多只能是1024字节"，因为GET是通过URL提交数据，那么GET可提交的数据量就跟URL的长度有直接关系了。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。

　　注意这是限制是整个URL长度，而不仅仅是你的参数值数据长度。[见参考资料5]

　　(2).理论上讲，POST是没有大小限制的，HTTP协议规范也没有进行大小限制，说“POST数据量存在80K/100K的大小限制”是不准确的，POST数据是没有限制的，起限制作用的是服务器的处理程序的处理能力。

　　对于ASP程序，Request对象处理每个表单域时存在100K的数据长度限制。但如果使用Request.BinaryRead则没有这个限制。

　　由这个延伸出去，对于IIS 6.0，微软出于安全考虑，加大了限制。我们还需要注意：

　　　　 1).IIS 6.0默认ASP POST数据量最大为200KB，每个表单域限制是100KB。  
　　　　 2).IIS 6.0默认上传文件的最大大小是4MB。  
　　　　 3).IIS 6.0默认最大请求头是16KB。  
　　IIS 6.0之前没有这些限制。[见参考资料5]

　　所以上面的80K，100K可能只是默认值而已(注：关于IIS4和IIS5的参数，我还没有确认)，但肯定是可以自己设置的。由于每个版本的IIS对这些参数的默认值都不一样，具体请参考相关的IIS配置文档。

**3**.在ASP中，服务端获取GET请求参数用Request.QueryString，获取POST请求参数用Request.Form。在JSP中，用request.getParameter(\"XXXX\")来获取，虽然jsp中也有request.getQueryString()方法，但使用起来比较麻烦，比如：传一个test.jsp?name=hyddd&password=hyddd，用request.getQueryString()得到的是：name=hyddd&password=hyddd。在PHP中，可以用$\_GET和$\_POST分别获取GET和POST中的数据，而$\_REQUEST则可以获取GET和POST两种请求中的数据。值得注意的是，JSP中使用request和PHP中使用$\_REQUEST都会有隐患，这个下次再写个文章总结。

**4**.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。

　　总结一下，Get是向服务器发索取数据的一种请求，而Post是向服务器提交数据的一种请求，在FORM（表单）中，Method默认为"GET"，实质上，GET和POST只是发送机制不同，并不是一个取一个发！

　　纯属hyddd个人总结，如有错漏请指出。:>

# [HTTP POST GET 本质区别详解](http://blog.csdn.net/csj50/article/details/5687850)

分类： [HTML](http://blog.csdn.net/csj50/article/category/364987)2010-06-22 23:39 9008人阅读 [评论](http://blog.csdn.net/csj50/article/details/5687850#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/csj50/article/details/5687850#report)

[服务器](http://www.csdn.net/tag/%e6%9c%8d%e5%8a%a1%e5%99%a8)[浏览器](http://www.csdn.net/tag/%e6%b5%8f%e8%a7%88%e5%99%a8)[url](http://www.csdn.net/tag/url)[encoding](http://www.csdn.net/tag/encoding)[soap](http://www.csdn.net/tag/soap)[mvc](http://www.csdn.net/tag/mvc)

**一 原理区别**

    一般在浏览器中输入网址访问资源都是通过GET方式；在FORM提交中，可以通过Method指定提交方式为GET或者POST，默认为GET提交

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(**个人认为这是GET和POST的本质区别，也是协议设计者的本意，其它区别都是具体表现形式的差异** )。

　　根据HTTP规范，GET用于信息获取，而且应该是安全的和幂等的 。

　　1.所谓安全的意味着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，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.幂等的意味着对同一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，有在实数集中幂等，即max(x,x) = x 。

看完上述解释后，应该可以理解GET幂等的含义了。

　　但在实际应用中，以上2条规定并没有这么严格。引用别人文章的例子：比如，新闻站点的头版不断更新。虽然第二次请求会返回不同的一批新闻，该操作仍然被认为是安全的和幂等的，因为它总是返回当前的新闻。从根本上说，如果目标是当用户打开一个链接时，他可以确信从自身的角度来看没有改变资源即可。

　　根据HTTP规范，POST表示可能修改变服务器上的资源的请求 。继续引用上面的例子：还是新闻以网站为例，读者对新闻发表自己的评论应该通过POST实现，因为在评论提交后站点的资源已经不同了，或者说资源被修改了。

　　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.另外一个是，早期的但是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 。还有一个较为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。

　　\* 简单解释一下MVC：MVC本来是存在于Desktop程序中的，M是指数据模型，V是指用户界面，C则是控制器。使用MVC的目的是将M和V的实现代码分离，从而使同一个程序可以使用不同的表现形式。

　　以上3点典型地描述了老一套的风格（没有严格遵守HTTP规范），随着架构的发展，现在出现REST(Representational State Transfer)，一套支持HTTP规范的新风格，这里不多说了，可以参考《RESTful Web Services》。

**二 表现形式区别**

   搞清了两者的原理区别，我们再来看一下他们实际应用中的区别：

    为了理解两者在传输过程中的不同，我们先看一下HTTP协议的格式：

    HTTP请求：

<request line>          //http请求行

<headers>               //http请求消息报头

<blank line>             // 回 车 换行

[<request-body>]     //http请求正文

在HTTP请求中，第一行必须是一个请求行（request line），用来说明请求类型、要访问的资源以及使用的HTTP版本。紧接着是一个首部（header）小节，用来说明服务器要使用的附加信息。在首部之后是一个空行，再此之后可以添加任意的其他数据[称之为主体（body）]。

GET与POST方法实例：  
GET /books/?sex=man&name=Professional HTTP/1.1  
Host: www.wrox.com  
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)  
Gecko/20050225 Firefox/1.0.1  
Connection: Keep-Alive  
  
POST / HTTP/1.1  
Host: www.wrox.com  
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)  
Gecko/20050225 Firefox/1.0.1  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 40  
Connection: Keep-Alive  
（----此处空一行----）  
name=Professional%20Ajax&publisher=Wiley

有了以上对HTTP请求的了解和示例，我们再来看两种提交方式的区别：

   （1）GET提交，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，多个参数用&连接；例 如：login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0 %E5%A5%BD。如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/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。

　　POST提交：把提交的数据放置在是HTTP包的包体中。上文示例中红色字体标明的就是实际的传输数据

     因此，GET提交的数据会在地址栏中显示出来，而POST提交，地址栏不会改变

(2)传输数据的大小：首先声明：HTTP协议没有对传输的数据大小进行限制，HTTP协议规范也没有对URL长度进行限制。

而在实际开发中存在的限制主要有：

GET:特定浏览器和服务器对URL长度有限制，例如 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。

因此对于GET提交时，传输数据就会受到URL长度的 限制。

POST:由于不是通过URL传值，理论上数据不受 限。但实际各个WEB服务器会规定对post提交数据大小进行限制，Apache、IIS6都有各自的配置。

(3)安全性：

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

（4）Http get,post,soap协议都是在http上运行的  
1）get：请求参数是作为一个key/value对的序列（查询字符串）附加到URL上的  
查询字符串的长度受到web浏览器和web服务器的限制（如IE最多支持2048个字符），不适合传输大型数据集同时，它很不安全  
2）post：请求参数是在http标题的一个不同部分（名为entity body）传输的，这一部分用来传输表单信息，因此必须将Content-type设置为:application/x-www-form- urlencoded。post设计用来支持web窗体上的用户字段，其参数也是作为key/value对传输。  
但是：它不支持复杂数据类型，因为post没有定义传输数据结构的语义和规则。  
3）soap：是http post的一个专用版本，遵循一种特殊的xml消息格式  
Content-type设置为: text/xml   任何数据都可以xml化

**三 HTTP响应**  
1．HTTP响应格式：

<status line>          //http响应状态行  
<headers>             //http响应消息报头  
<blank line>            //回车换行  
[<response-body>]  //http响应正文  
  
在响应中唯一真正的区别在于第一行中用状态信息代替了请求信息。状态行（status line）通过提供一个状态码来说明所请求的资源情况。   
  
HTTP响应实例：  
  
HTTP/1.1 200 OK  
Date: Sat, 31 Dec 2005 23:59:59 GMT  
Content-Type: text/html;charset=ISO-8859-1 Content-Length: 122

＜html＞  
＜head＞  
＜title＞Wrox Homepage＜/title＞  
＜/head＞  
＜body＞  
＜!-- body goes here --＞  
＜/body＞  
＜/html＞  
2．最常用的状态码有：  
  
◆200 (OK): 找到了该资源，并且一切正常。  
◆304 (NOT MODIFIED): 该资源在上次请求之后没有任何修改。这通常用于浏览器的缓存机制。  
◆401 (UNAUTHORIZED): 客户端无权访问该资源。这通常会使得浏览器要求用户输入用户名和密码，以登录到服务器。  
◆403 (FORBIDDEN): 客户端未能获得授权。这通常是在401之后输入了不正确的用户名或密码。  
◆404 (NOT FOUND): 在指定的位置不存在所申请的资源。

**四 完整示例：**

例子：

**HTTP GET**   
  
发送  
  
GET /DEMOWebServices2.8/Service.asmx/CancelOrder?UserID=string&PWD=string&OrderConfirmation=string HTTP/1.1  
Host: api.efxnow.com  
  
回复  
  
HTTP/1.1 200 OK  
Content-Type: text/xml; charset=utf-8  
Content-Length: length  
  
<?xml version="1.0" encoding="utf-8"?>  
<objPlaceOrderResponse xmlns="https://api.efxnow.com/webservices2.3">  
<Success>boolean</Success>  
<ErrorDescription>string</ErrorDescription>  
<ErrorNumber>int</ErrorNumber>  
<CustomerOrderReference>long</CustomerOrderReference>  
<OrderConfirmation>string</OrderConfirmation>  
<CustomerDealRef>string</CustomerDealRef>  
</objPlaceOrderResponse>  
  
  
  
**HTTP POST**   
  
发送  
  
POST /DEMOWebServices2.8/Service.asmx/CancelOrder HTTP/1.1  
Host: api.efxnow.com  
Content-Type: application/x-www-form-urlencoded  
Content-Length: length  
  
UserID=string&PWD=string&OrderConfirmation=string  
  
回复  
  
HTTP/1.1 200 OK  
Content-Type: text/xml; charset=utf-8  
Content-Length: length  
  
<?xml version="1.0" encoding="utf-8"?>  
<objPlaceOrderResponse xmlns="https://api.efxnow.com/webservices2.3">  
<Success>boolean</Success>  
<ErrorDescription>string</ErrorDescription>  
<ErrorNumber>int</ErrorNumber>  
<CustomerOrderReference>long</CustomerOrderReference>  
<OrderConfirmation>string</OrderConfirmation>  
<CustomerDealRef>string</CustomerDealRef>  
</objPlaceOrderResponse>  
  
  
  
**SOAP 1.2**   
  
发送  
  
POST /DEMOWebServices2.8/Service.asmx HTTP/1.1  
Host: api.efxnow.com  
Content-Type: application/soap+xml; charset=utf-8  
Content-Length: length  
  
<?xml version="1.0" encoding="utf-8"?>  
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">  
<soap12:Body>  
<CancelOrder xmlns="https://api.efxnow.com/webservices2.3">  
<UserID>string</UserID>  
<PWD>string</PWD>  
<OrderConfirmation>string</OrderConfirmation>  
</CancelOrder>  
</soap12:Body>  
</soap12:Envelope>  
  
回复  
  
HTTP/1.1 200 OK  
Content-Type: application/soap+xml; charset=utf-8  
Content-Length: length  
  
<?xml version="1.0" encoding="utf-8"?>  
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">  
<soap12:Body>  
<CancelOrderResponse xmlns="https://api.efxnow.com/webservices2.3">  
<CancelOrderResult>  
<Success>boolean</Success>  
<ErrorDescription>string</ErrorDescription>  
<ErrorNumber>int</ErrorNumber>  
<CustomerOrderReference>long</CustomerOrderReference>  
<OrderConfirmation>string</OrderConfirmation>  
<CustomerDealRef>string</CustomerDealRef>  
</CancelOrderResult>  
</CancelOrderResponse>  
</soap12:Body>  
</soap12:Envelope>

本文转自：http://hi.baidu.com/eveready/blog/item/585bbb30fbcee915eac4af27.html

  1. get是从服务器上获取数据，post是向服务器传送数据。   
get 和 post只是一种传递数据的方式，get也可以把数据传到服务器，他们的本质都是发送请求和接收结果。只是组织格式和数据量上面有差别，http协议里面有介绍  
  2. get是把参数数据队列加到提交表单的ACTION属性所指的URL中，值和表单内各个字段一一对应，在URL中可以看到。post是通过HTTP post机制，将表单内各个字段与其内容放置在HTML HEADER内一起传送到ACTION属性所指的URL地址。用户看不到这个过程。   
因为get设计成传输小数据，而且最好是不修改服务器的数据，所以浏览器一般都在地址栏里面可以看到，但post一般都用来传递大数据，或比较隐私的数据，所以在地址栏看不到，能不能看到不是协议规定，是浏览器规定的。  
3. 对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。   
没明白，怎么获得变量和你的服务器有关，和get或post无关，服务器都对这些请求做了封装  
  4. get传送的数据量较小，不能大于2KB。post传送的数据量较大，一般被默认为不受限制。但理论上，IIS4中最大量为80KB，IIS5中为100KB。   
post基本没有限制，我想大家都上传过文件，都是用post方式的。只不过要修改form里面的那个type参数  
  5. get安全性非常低，post安全性较高。   
如果没有加密，他们安全级别都是一样的，随便一个监听器都可以把所有的数据监听到，不信你自己下一个监听网络资源的软件，

 Get：是以实体的方式得到由请求URI所指定资源的信息，如果请求URI只是一个数据产生过程，那么最终要在响应实体中返回的是处理过程的结果所指向的资源，而不是处理过程的描述。  
  
Post：用来向目的服务器发出请求，要求它接受被附在请求后的实体，并把它当作请求队列中请求URI所指定资源的附加新子项，Post被设计成用统一的方法实现下列功能：  
1：对现有资源的解释  
2：向电子公告栏、新闻组、邮件列表或类似讨论组发信息。  
3：提交数据块  
4：通过附加操作来扩展数据库  
  
从上面描述可以看出，Get是向服务器发索取数据的一种请求；而Post是向服务器提交数据的一种请求，要提交的数据位于信息头后面的实体中。

1、Get是用来从服务器上获得数据，而Post是用来向服务器上传递数据。   
2、Get将表单中数据的按照variable=value的形式，添加到action所指向的URL后面，并且两者使用“?”连接，而各个变量之间使用“&”连接；Post是将表单中的数据放在form的数据体中，按照变量和值相对应的方式，传递到action所指向URL。   
3、Get是不安全的，因为在传输过程，数据被放在请求的URL中，而如今现有的很多服务器、代理服务器或者用户代理都会将请求URL记录到日志文件中，然后放在某个地方，这样就可能会有一些隐私的信息被第三方看到。另外，用户也可以在浏览器上直接看到提交的数据，一些系统内部消息将会一同显示在用户面前。Post的所有操作对用户来说都是不可见的。   
4、Get传输的数据量小，这主要是因为受URL长度限制；而Post可以传输大量的数据，所以在上传文件只能使用Post（当然还有一个原因，将在后面的提到）。   
5、Get限制Form表单的数据集的值必须为ASCII字符；而Post支持整个ISO10646字符集。默认是用ISO-8859-1编码   
6、Get是Form的默认方法。  
以下的比较非常非常使用：  
做java的web开发有段日子了，有个问题老是困扰着我，就是乱码问题，基本上是网上查找解决方案（网上资料真的很多），都是一大堆的介绍如何解决此类的乱码问题，但是没几个把问题的来龙去脉说清楚的，有时候看了些文章后，以为自己懂了，但是在开发中乱码问题又像鬼魂一样出来吓人，真是头大了！这篇文章是我长时间和乱码做斗争的一些理解的积累，还希望有更多的朋友给出指点和补充。   
  form有2中方法把数据提交给服务器，get和post,分别说下吧。   
（一）get提交   
  1.首先说下客户端（浏览器）的form表单用get方法是如何将数据编码后提交给服务器端的吧。   
    
    对于get方法来说，都是把数据串联在请求的url后面作为参数，如：http://localhost:8080/servlet?msg=abc   
（很常见的一个乱码问题就要出现了，如果url中出现中文或其它特殊字符的话，如：http://localhost:8080 /servlet?msg=杭州，服务器端容易得到乱码），url拼接完成后，浏览器会对url进行URL encode，然后发送给服务器，URL encode的过程就是把部分url做为字符，按照某种编码方式（如：utf-8,gbk等）编码成二进制的字节码，然后每个字节用一个包含3个字符的字符串 "%xy" 表示，其中xy为该字节的两位十六进制表示形式。我这里说的可能不清楚，具体介绍可以看下java.net.URLEncoder类的介绍在这里。了解了 URL encode的过程，我们能看到2个很重要的问题，第一：需要URL encode的字符一般都是非ASCII的字符（笼统的讲），再通俗的讲就是除了英文字母以外的文字（如：中文，日文等）都要进行URL encode，所以对于我们来说，都是英文字母的url不会出现服务器得到乱码问题，出现乱码都是url里面带了中文或特殊字符造成的；第二：**URL encode到底按照那种编码方式对字符编码？这里就是浏览器的事情了，而且不同的浏览器有不同的做法，中文版的浏览器一般会默认的使用GBK，通过设置浏览器也可以使用UTF-8，可能不同的用户就有不同的浏览器设置，也就造成不同的编码方式，所以很多网站的做法都是先把url里面的中文或特殊字符用 javascript做URL encode，然后再拼接url提交数据，也就是替浏览器做了URL encode，好处就是网站可以统一get方法提交数据的编码方式。** 完成了URL encode，那么现在的url就成了ASCII范围内的字符了，然后以iso-8859-1的编码方式转换成二进制随着请求头一起发送出去。这里想多说几句的是，对于get方法来说，没有请求实体，含有数据的url都在请求头里面，之所以用URL encode，**我个人觉的原因是：对于请求头来说最终都是要用iso-8859-1编码方式编码成二进制的101010.....的纯数据在互联网上传送，如果直接将含有中文等特殊字符做iso-8859-1编码会丢失信息，所以先做URL encode是有必要的。**   
   2。服务器端（tomcat）是如何将数据获取到进行解码的。   
   第一步是先把数据用iso-8859-1进行解码，对于get方法来说，tomcat获取数据的是ASCII范围内的请求头字符，其中的请求url里面带有参数数据，如果参数中有中文等特殊字符，那么目前还是URL encode后的%XY状态，先停下，我们先说下开发人员一般获取数据的过程。通常大家都是request.getParameter("name")获取参数数据，我们在request对象或得的数据都是经过解码过的，而解码过程中程序里是无法指定，这里要说下，有很多新手说用 **request.setCharacterEncoding("字符集")可以指定解码方式，其实是不可以的**，看servlet的官方API说明有对此方法的解释：Overrides the name of the character encoding used in the body of this request. This method must be called prior to reading request parameters or reading input using getReader().可以看出对于get方法他是无能为力的。那么到底用什么编码方式解码数据的呢，这是tomcat的事情了，默认缺省用的是 iso-8859-1,这样我们就能找到为什么get请求带中文参数为什么在服务器端得到乱码了，原因是在客户端一般都是用UTF-8或GBK对数据 URL encode，这里用iso-8859-1方式URL decoder显然不行，在程序里我们可以直接   
Java代码   
1. new String(request.getParameter("name").getBytes("iso-8859-1"),"客户端指定的URL encode编码方式")   
还原回字节码，然后用正确的方式解码数据，网上的文章通常是在tomcat里面做个配置   
Xml代码   
1. <Connector port="8080" protocol="HTTP/1.1" maxThreads="150" connectionTimeout="20000" redirectPort="8443" URIEncoding="GBK"/>    
这样是让tomcat在获取数据后用指定的方式URL decoder，URL decoder的介绍在这里   
（一）post提交   
1.客户端（浏览器）的form表单用post方法是如何将数据编码后提交给服务器端的。   
  在post方法里所要传送的数据也要URL encode，那么他是用什么编码方式的呢？   
   在form所在的html文件里如果有段**<meta http-equiv="Content-Type" content="text/html; charset=字符集（GBK，utf-8等）"/>，那么post就会用此处指定的编码方式编码。**一般大家都认为这段代码是为了让浏览器知道用什么字符集来对网页解释，所以网站都会把它放在html代码的最前端，尽量不出现乱码，其实它还有个作用就是**指定form表单的post方法提交数据的 URL encode编码方式**。从这里可以看出对于get方法来数，浏览器对数据的URL encode的编码方式是有浏览器设置来决定，（可以用js做统一指定），而post方法，开发人员可以指定。   
2。服务器端（tomcat）是如何将数据获取到进行解码的。   
如果用tomcat默认缺省设置，也没做过滤器等编码设置，那么他也是用iso-8859-1解码的，但是request.setCharacterEncoding("字符集")可以派上用场。   
  
**我发现上面说的tomcat所做的事情前提都是在请求头里没有指定编码方式，如果请求头里指定了编码方式将按照这种方式编码。**   
   有2篇文章推荐下，地址分别是   
深入浅出URL编码：[http://www.cnblogs.com/yencain/articles/1321386.html](http://www.cnblogs.com/yencain/articles/1321386.html" \t "_blank)；   
表单用post方法提交数据时乱码问题：[http://wanghuan8086.javaeye.com/blog/173869](http://wanghuan8086.javaeye.com/blog/173869" \t "_blank)   
  
用post很重要的在form所在的html文件里如果有段<meta http-equiv="Content-Type" content="text/html; charset=字符集（GBK，utf-8等）"/>   
**强烈建议使用post提交**

1. get是从服务器上获取数据，post是向服务器传送数据。搜索  
2. get是把参数数据队列加到提交表单的ACTION属性所指的URL中，值和表单内各个字段一一对应，在URL中可以看到。post是通过HTTP post机制，将表单内各个字段与其内容放置在HTML HEADER内一起传送到ACTION属性所指的URL地址。用户看不到这个过程。  
3. 对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。  
4. get传送的数据量较小，不能大于2KB。post传送的数据量较大，一般被默认为不受限制。但理论上，IIS4中最大量为80KB，IIS5中为100KB。  
5. get安全性非常低，post安全性较高。但是执行效率却比Post方法好。   
  
建议：  
1、get方式的安全性较Post方式要差些，包含机密信息的话，建议用Post数据提交方式；  
2、在做数据查询时，建议用Get方式；而在做数据添加、修改或删除时，建议用Post方式；

**两种最常用的 HTTP 方法是：GET 和 POST。**

**什么是 HTTP？**

超文本传输协议（HTTP）的设计目的是保证客户机与服务器之间的通信。

HTTP 的工作方式是客户机与服务器之间的请求-应答协议。

web 浏览器可能是客户端，而计算机上的网络应用程序也可能作为服务器端。

举例：客户端（浏览器）向服务器提交 HTTP 请求；服务器向客户端返回响应。响应包含关于请求的状态信息以及可能被请求的内容。

**两种 HTTP 请求方法：GET 和 POST**

在客户机和服务器之间进行请求-响应时，两种最常被用到的方法是：GET 和 POST。

* **GET** - 从指定的资源请求数据。
* **POST** - 向指定的资源提交要被处理的数据

**GET 方法**

请注意，查询字符串（名称/值对）是在 GET 请求的 URL 中发送的：

/test/demo\_form.asp?name1=value1&name2=value2

有关 GET 请求的其他一些注释：

* GET 请求可被缓存
* GET 请求保留在浏览器历史记录中
* GET 请求可被收藏为书签
* GET 请求不应在处理敏感数据时使用
* GET 请求有长度限制
* GET 请求只应当用于取回数据

**POST 方法**

请注意，查询字符串（名称/值对）是在 POST 请求的 HTTP 消息主体中发送的：

POST /test/demo\_form.asp HTTP/1.1

Host: w3schools.com

name1=value1&name2=value2

有关 POST 请求的其他一些注释：

* POST 请求不会被缓存
* POST 请求不会保留在浏览器历史记录中
* POST 不能被收藏为书签
* POST 请求对数据长度没有要求

**比较 GET 与 POST**

下面的表格比较了两种 HTTP 方法：GET 和 POST。

|  |  |  |
| --- | --- | --- |
|  | **GET** | **POST** |
| 后退按钮/刷新 | 无害 | 数据会被重新提交（浏览器应该告知用户数据会被重新提交）。 |
| 书签 | 可收藏为书签 | 不可收藏为书签 |
| 缓存 | 能被缓存 | 不能缓存 |
| 编码类型 | application/x-www-form-urlencoded | application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。 |
| 历史 | 参数保留在浏览器历史中。 | 参数不会保存在浏览器历史中。 |
| 对数据长度的限制 | 是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。 | 无限制。 |
| 对数据类型的限制 | 只允许 ASCII 字符。 | 没有限制。也允许二进制数据。 |
| 安全性 | 与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。  在发送密码或其他敏感信息时绝不要使用 GET ！ | POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。 |
| 可见性 | 数据在 URL 中对所有人都是可见的。 | 数据不会显示在 URL 中。 |

**其他 HTTP 请求方法**

下面的表格列出了其他一些 HTTP 请求方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| HEAD | 与 GET 相同，但只返回 HTTP 报头，不返回文档主体。 |
| PUT | 上传指定的 URI 表示。 |
| DELETE | 删除指定资源。 |
| OPTIONS | 返回服务器支持的 HTTP 方法。 |
| CONNECT | 把请求连接转换到透明的 TCP/IP 通道。 |