Tomcat对于请求参数的编码和解码问题

对于以tomcat作为web服务器来开发web应用的猿猿们来说，编码问题应该算是个比较头疼的问题了吧，虽然有一些“最佳实践”的方式，但是仅从servlet提供的外部接口来看tomcat实现所表现的结果的确会引起一些歧异。这些问题也困扰我许久，老师上课给我们总结了n种情况下要怎么处理，但是懒惰是猿猿们的天性好不？懒的记啊……..为求以不变应万变，我干脆直接调试了一把tomcat的源码并总结了tomcat解析请求参数的具体过程，吐血奉献……

**关于URL编码和解码：**

浏览器向服务端发送的信息在必要的时候会进行URL编码，比如会对地址栏里中文等特殊字符的请求参数进行编码，post请求在请求体里提交的数据也会进行相应的编码。假设浏览器当前的编码为UTF-8时，那么“中文”两个字会被编码为：%E4%B8%AD%E6%96%87。具体的编码方式没有深究，但是可以看出点端倪：“中文”两个字的UTF-8编码后字节数据的十六进制表示为E4B8ADE69687，可以看到URL编码似乎就是把编码后的字节映射成字符串的形式，并以“%”作为每个字节的开头。就像上面的第一个字节E4(11100100)映射成“%E4”

。经过URL编码后可以保证浏览器传给服务端的数据的正确性。URL**编码就是把字节映射成字符串。**

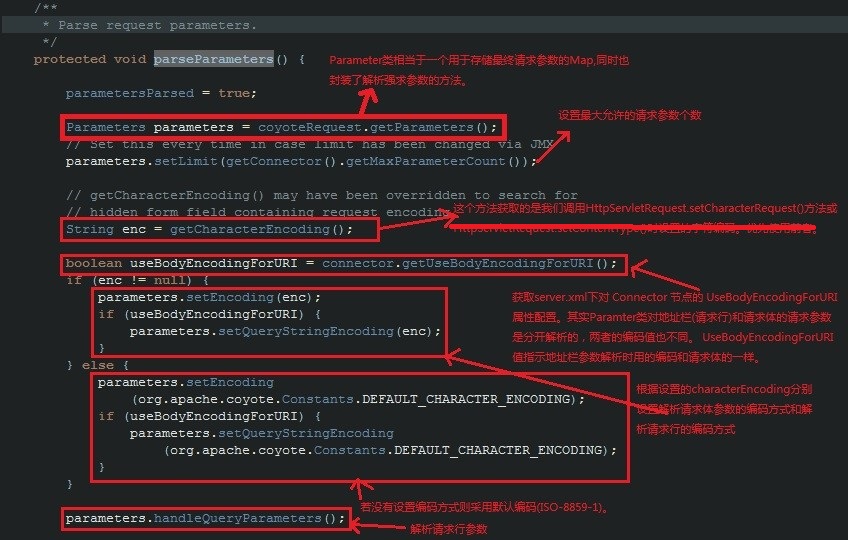
URL解码就是编码的逆过程，比如要把上面的“%E4”字符串进行URL解码的结果就是去掉%，然后把”E4”映射成字节11100100。**URL解码就是把字符串映射成字节。**

**情况一：请求参数由浏览器端传递给tomcat服务器**

1. **GET请求**

以请求地址<http://host/web/index.html?name=中文> 为例，并且假设浏览器当前编码方式为UTF-8。那么浏览器发送给浏览器的请求地址实际为经过URL编码的：<http://host/web/index.html?name=%E4%B8%AD%E6%96%87>

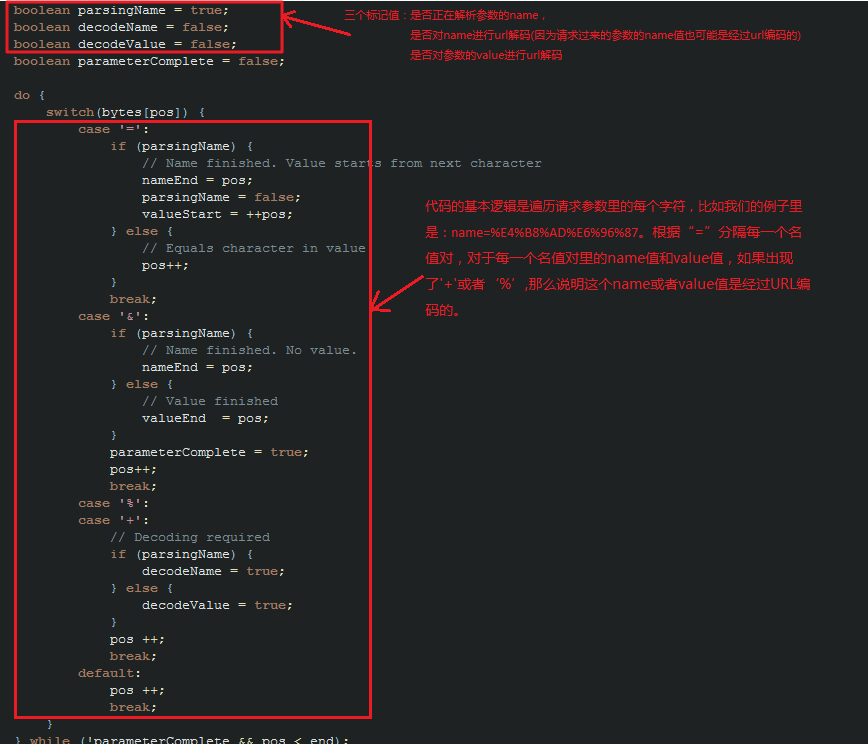
Tomcat接受到请求后需要逐个解析请求参数的名(name)和值(value)，解析的主要逻辑在org.apache.catalina.connector. parseParameters()方法中:



org.apache.tomcat.util.http.Parameters类的主要作用是存储最终解析后的名值对和封装了具体解析方法。上面的代码主要是设置Parameter类解析请求体参数时采用的编码方式，请求行和请求体是分别解析的，两者的编码方式也不一样（这就是为什么我们调用request.setCharacterEncoding()方法对请求行参数不管用的原因）。可以通过配置server.xml文件里http协议连接节点Connector的UseBodyEncodingForURI属性为true使解析请求行参数的编码和解析请求体的编码一致。如果我们调用过HttpServletRequest.setCharacterEncoding()方法，则请求体解析时采用的编码方式为我们设置的编码。如果我们没有设置就采用默认的编码ISO-8859-1。

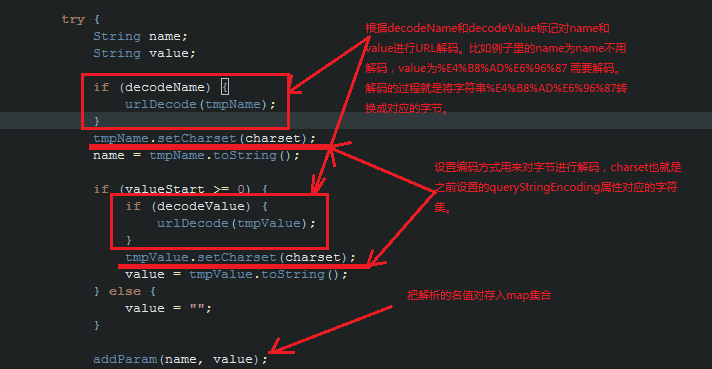
设置完请求体编码方式后就调会用Parameters.handleQueryParameters();方法来解析请求行参数. handleQueryParameters()内部首先会检查之前是否设置了queryStringEncoding（或者说调用过setQueryStringEncoding），如果没有就采用ISO-8859-1。可以通过设置server.xml下http协议Connector节点的URIEcoding属性来设置请求行采用的编码方式，或者像上面说的设置UseBodyEncodingForURI属性来使请求行的编码方式和请求体一样，那么在设置请求体编码的同时也会设置请求行。请求行参数的解析方式的具体逻辑在Parameters类的一个私有

**processParameters(byte bytes[], int start, int len, Charset charset)**方法中:



processParameters方法会先根据请求参数字符串(比如我们的例子里是name=%E4%B8%AD%E6%96%87)里的’&’字符分隔每个名值对，对于每一个名值对里的name和value值，用’=’分割name值和value值(比如例子里的name值是name，value值是%E4%B8%AD%E6%96%87)，分析name和value值，如果里面出现了’+’或者’%’字符，说明这个name值或者value值是经过URL编码的，并且分别用两个局部变量decodeName和decodeValue两个标记量进行标记，以便后面的操作判断是否对name值和value值进行URL解码。

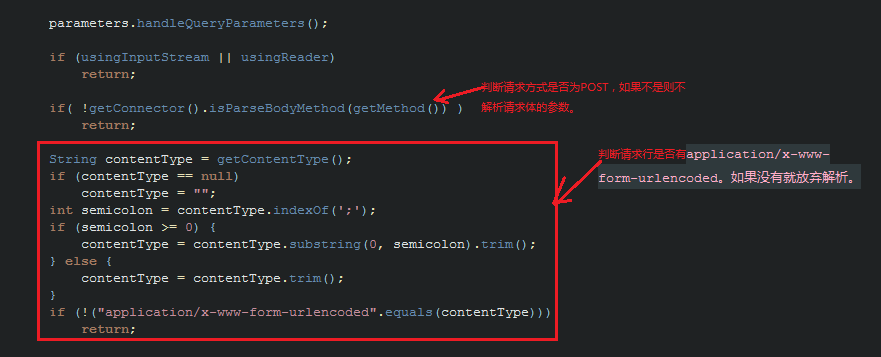
获取了每个名值对后就分别对name和value进行解码了，下面是主要的代码逻辑：



首先是根据decodeValue和decodeName两个标记值分别对name和value进行URL解码，我们的例子中name为name，不需要URL解码，value为%E4%B8%AD%E6%96%87 需要URL解码，解码的结果就是把对应的字符串转换成字节数组（开头讲过了）。URL解码后就相当于获得了浏览器端经过字符集编码后传过来的name和value值的原始字节数据，比如说我们的name仍然是”name”经过浏览器端编码后的原始字节数据（应为不需要经过URL解码），而value是”中文”经过浏览器编码后的字节数据：11100100 10111000……获取到name和value的原始字节数据后就可以通过之前设置的queryStringEncoding对他们进行解码了。如果我们之前从没在server.xml里设置URIEncoding或UseBodyEncodingForURI属性那么就会采用默认的编码ISO-8859-1进行解码，那么此时就会出现乱码，因为浏览器的实际编码方式为UTF-8。但是要是我们设置了URIEncoding为UTF-8或者是调用了HttpServletRequest.setCharacterEncoding(“UTF-8”)方法并且设置了UseBodyEncodingForURI属性，那么就不会出现乱码了。最后是把解析的结果放入map集合中，我们通过request.getParameter(“name”)方法就是从这里获取的。

1. **POST请求**

解析请求体的参数逻辑基本和解析请求行的一致，除了**采用的编码方式不同**（请求体采用request.setCharacterEncoding()方法设置的编码方式进行解析）外，在解析之前还要做相关判断，不过这里影响不大:

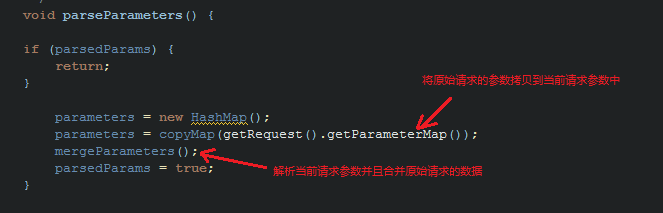


最终也是调用Parameters类的**processParameters(byte bytes[], int start, int len, Charset charset)**方法对请求体的数据进行解析。

**情况二：由服务器内部调用转发或包含方法时传递的参数**

以request.getRequestDispatcher("/path?name=中文").forward(request, response);为例。

这种情况和浏览器端传递数据主要有两个不同，一个是解析时采用的编码方式不同，另一个就是不会像浏览器一样事先对请求参数进行URL编码，比如”name=中文” 不会被编码为name=%E4%B8%AD%E6%96%87这种形式。这种方式获取请求参数的主要代码逻辑在org.apache.catalina.core.ApplicationHttpRequest.parseParameters()和mergeParameters()方法里:



parseParameters()方法做主要是先获取原始请求的请求参数(经过解析的)，然后调用mergeParameters()方法解析当前请求参数并将两者的请求参数合并到当前的请求参数map集合中。下面是mergeParameters ()方法的代码逻辑：



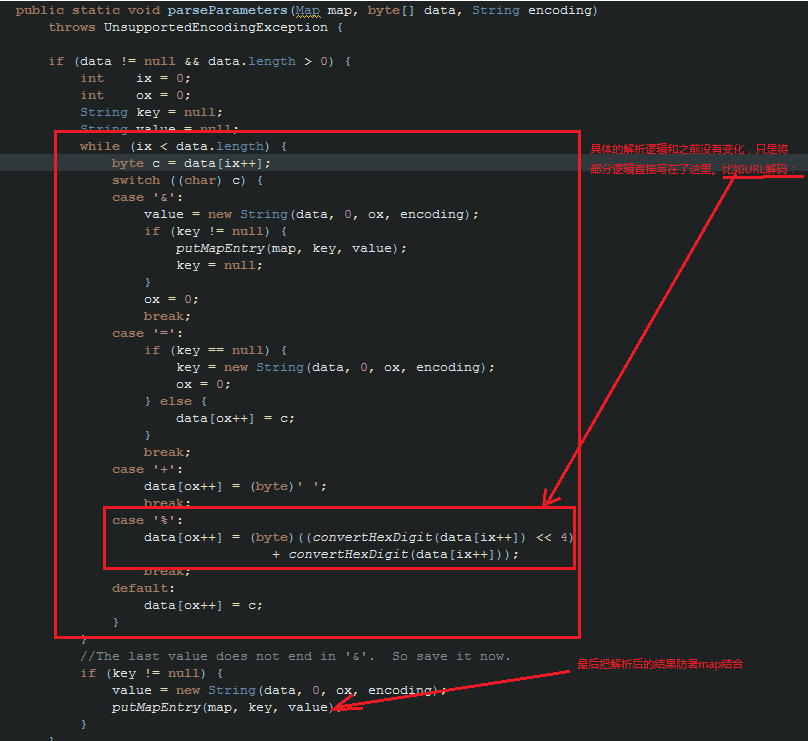
mergeParameters()方法先获取原始请求的characterEncoding属性作为解析参数时的编码方式，如果没有则采用默认编码ISO-8859-1。

最终的解析方式在[org](eclipse-javadoc:%E2%98%82=Tomcat6/java%3Corg).[apache](eclipse-javadoc:%E2%98%82=Tomcat6/java%3Corg.apache).[catalina](eclipse-javadoc:%E2%98%82=Tomcat6/java%3Corg.apache.catalina).[util](eclipse-javadoc:%E2%98%82=Tomcat6/java%3Corg.apache.catalina.util).RequestUtil的parseParameters(Map map, String data, String encoding)和parseParameters(Map map, byte[] data, String encoding)方法里,前者首先会根据之前设置的编码将请求字符串进行编码，如我们的例子会调用

” name=中文“.getBytes(encoding);如果没有指定编码方式，那么会调用

” name=中文“.getBytes()，采用系统默认的编码方式，很可能不是ISO-8859-1。

然后把获取的字节数组传递给第二个方法进行解析，下面是第二个方法的代码逻辑：



解析逻辑和之前解析浏览器传过来的参数一致，只是把部分逻辑比如URL解码固化在当前逻辑里。唯一要注意的也就是开头强调的两点：**1.解析使用的编码方式是request的chatacterEncoding属性，如果没有就采用ISO-8859-1；2.在解析前不会先对请求字符串进行URL编码(当然如果手动进行编码就另说了)，而是直接采用当前的chatacterEncoding编码成字节数组**。所以这种情况会出现乱码往往是之前没有设置chatacterEncoding属性导致的。(因为如果没有设置，那么实际编码是系统默认的编码，而解码时采用了ISO-8859-1)。

**最佳实践：**

1. **调用response.setContenType()方法设置编码输出编码方式，和浏览器的解码和编码方式。**
2. **在server.xml的HTTP协议的Connector节点里设置URIEncoding属性(和1一致)或设置UseBodyEncodingForURI为true。**
3. **调用request.setCharacterEncoding()方法设置和上面一样的编码方式。**