**java中的URLEncoder和URLDecoder类**

http://www.sina.com.cn 2008年07月07日 10:53  [IT168.com](http://www.it168.com" \t "_blank)

　　【IT168 技术文档】

　　/\*

　　网页中的表单使用POST方法提交时，数据内容的类型是 application/x-www-form-urlencoded，这种类型会：

　　1.字符"a"-"z"，"A"-"Z"，"0"-"9"，"."，"-"，"\*"，和"\_" 都不会被编码;

　　2.将空格转换为加号 (+) ;

　　3.将非文本内容转换成"%xy"的形式,xy是两位16进制的数值;

　　4.在每个 name=value 对之间放置 & 符号。

　　\*/

　　URLEncoder类包含将字符串转换为application/x-www-form-urlencoded MIME 格式的静态方法。

　　web设计者面临的众多难题之一便是怎样处理不同操作系统间的差异性。这些差异性能引起URL方面的问题：例如，一些操作系统允许文件名中含有 空格符，有些又不允许。大多数操作系统不会认为文件名中含有符号“#”会有什么特殊含义;但是在一个URL中，符号“#”表示该文件名已经结束，后面会紧 跟一个fragment(部分)标识符。其他的特殊字符，非字母数字字符集，它们在URL或另一个操作系统上都有其特殊的含义，表述着相似的问题。为了解 决这些问题，我们在URL中使用的字符就必须是一个ASCII字符集的固定字集中的元素，具体如下：

　　1.大写字母A-Z

　　2.小写字母a-z

　　3.数字 0-9

　　4.标点符 - \_ . ! ~ \* ' (和 ,)

　　诸如字符: / & ? @ # ; $ + = 和 %也可以被使用，但是它们各有其特殊的用途，如果一个文件名包括了这些字符( / & ? @ # ; $ + = %)，这些字符和所有其他字符就应该被编码。

　　编码过程非常简单，任何字符只要不是ASCII码数字，字母，或者前面提到的标点符，它们都将被转换成字节形式，每个字节都写成这种形式：一个 “%”后面跟着两位16进制的数值。空格是一个特殊情况，因为它们太平常了。它除了被编码成“%20”以外，还能编码为一个“+”。加号(+)本身被编码 为%2B。当/ # = & 和?作为名字的一部分来使用时，而不是作为URL部分之间的分隔符来使用时，它们都应该被编码。

　　WARNING这种策略在存在大量字符集的异构环境中效果不甚理想。例如：在U.S. Windows 系统中, é 被编码为 %E9. 在 U.S. Mac中被编码为%8E。这种不确定性的存在是现存的URI的一个明显的不足。所以在将来URI的规范当中应该通过国际资源标识符(IRIs)进行改善。

　　类URL并不自动执行编码或解码工作。你能生成一个URL对象，它可以包括非法的ASCII和非ASCII字符和/或%xx。当用方法 getPath() 和toExternalForm( ) 作为输出方法时，这种字符和转移符不会自动编码或解码。你应对被用来生成一个URL对象的字符串对象负责，确保所有字符都会被恰当地编码。

　　幸运的是，java提供了一个类URLEncoder把string编码成这种形式。Java1.2增加了一个类URLDecoder它能以这种形式解码string。这两个类都不用初始化：

　　public class URLDecoder extends Object

　　public class URLEncoder extends Object

　　一、URLEncoder

　　在java1.3和早期版本中，类java.net.URLEncoder包括一个简单的静态方法encode( )， 它对string以如下规则进行编码：

　　public static String encode(String s)

　　这个方法总是用它所在平台的默认编码形式，所以在不同系统上，它就会产生不同的结果。结果java1.4中，这个方法被另一种方法取代了。该方法要求你自己指定编码形式：

　　public static String encode(String s, String encoding) throws UnsupportedEncodingException

　　两种关于编码的方法，都把任何非字母数字字符转换成%xx(除了空格，下划线(\_)，连字符(?),句号(。),和星号(\*))。两者也都编码 所以的非ASCII字符。空格被转换成一个加号。这些方法有一点过分累赘了;它们也把“~”，“‘”，“()”转换成%xx，即使它们完全用不着这样做。 尽管这样，但是这种转换并没被URL规范所禁止。所以web浏览器会自然地处理这些被过分编码后的URL。

　　两中关于编码的方法都返回一个新的被编码后的string，java1.3的方法encode( ) 使用了平台的默认编码形式，得到%xx。这些编码形式典型的有：在 U.S. Unix 系统上的ISO-8859-1, 在U.S. Windows 系统上的Cp1252,在U.S. Macs上的MacRoman，和其他本地字符集等。因为编码解码过程都是与本地操作平台相关的，所以这些方法是令人不爽的，不能跨平台的。

　　这就明确地回答了为什么在java1.4中这种方法被抛弃了，转而投向了要求以自己指定编码形式的这种方法。尽管如此，如果你执意要使用所在平 台的默认编码形式，你的程序将会像在java1.3中的程序一样，是本地平台相关的。在另一种编码的方法中，你应该总是用UTF-8，而不是其他什么。 UTF-8比起你选的其他的编码形式来说，它能与新的web浏览器和更多的其他软件相兼容。

　　例子7-8是使用URLEncoder.encode( ) 来打印输出各种被编码后的string。它需要在java1.4或更新的版本中编译和运行。

　　Example 7-8. x-www-form-urlencoded strings

　　import java.net.URLEncoder;

　　import java.net.URLDecoder;

　　import java.io.UnsupportedEncodingException;

　　public class EncoderTest {

　　public static void main(String[] args) {

　　try {

　　System.out.println(URLEncoder.encode("This string has spaces","UTF-8"));

　　System.out.println(URLEncoder.encode("This\*string\*has\*asterisks","UTF-8"));

　　System.out.println(URLEncoder.encode("This%string%has%percent%signs", "UTF-8"));

　　System.out.println(URLEncoder.encode("This+string+has+pluses","UTF-8"));

　　System.out.println(URLEncoder.encode("This/string/has/slashes","UTF-8"));

　　System.out.println(URLEncoder.encode("This"string"has"quote"marks", "UTF-8"));

　　System.out.println(URLEncoder.encode("This:string:has:colons","UTF-8"));

　　System.out.println(URLEncoder.encode("This~string~has~tildes","UTF-8"));

　　System.out.println(URLEncoder.encode("This(string)has(parentheses)", "UTF-8"));

　　System.out.println(URLEncoder.encode("This.string.has.periods","UTF-8"));

　　System.out.println(URLEncoder.encode("This=string=has=equals=signs", "UTF-8"));

　　System.out.println(URLEncoder.encode("This&string&has&ersands","UTF-8"));

　　System.out.println(URLEncoder.encode("Thiséstringéhasé non-ASCII characters","UTF-8"));

　　// System.out.println(URLEncoder.encode("this中华人民共和国","UTF-8"));

　　} catch (UnsupportedEncodingException ex) {throw new RuntimeException("

Broken VM does not support UTF-8");

　　}

　　}

　　}

　　下面就是它的输出。需要注意的是这些代码应该以其他编码形式被保存而不是以ASCII码的形式，还有就是你选择的编码形式应该作为一个参数传给编译器，让编译器能据此对源代码中的非ASCII字符作出正确的解释。

　　% javac -encoding UTF8 EncoderTest %

　　java EncoderTest

　　This+string+has+spaces

　　This\*string\*has\*asterisks

　　This%25string%25has%25percent%25signs

　　This%2Bstring%2Bhas%2Bpluses

　　This%2Fstring%2Fhas%2Fslashes

　　This%22string%22has%22quote%22marks

　　This%3Astring%3Ahas%3Acolons

　　This%7Estring%7Ehas%7Etildes

　　This%28string%29has%28parentheses%29

　　This.string.has.periods

　　This%3Dstring%3Dhas%3Dequals%3Dsigns

　　This%26string%26has%26ampersands

　　This%C3%A9string%C3%A9has%C3%A9non-ASCII+characters

　　特别需要注意的是这个方法编码了符号，“\” ,&,=,和：。它不会尝试着去规定在一个URL中这些字符怎样被使用。由此，所以你不得不分块编码你的URL,而不是把整个URL一次传给这个 方法。这是很重要的，因为对类URLEncoder最通常的用法就是查询string，为了和服务器端使用GET方法的程序进行交互。例如，假设你想编码 这个查询sting，它用来搜索AltaVista网站：

　　pg=q&kl=XX&stype=stext&q=+"Java+I/O"&search.x=38&search.y=3

　　这段代码对其进行编码：

　　String query = URLEncoder.encode( "pg=q&kl=XX&stype=stext&q=+"Java+I/O"&search.x=38&search.y=3");System.out.println(query);

　　不幸的是，得到的输出是:

　　pg%3Dq%26kl%3DXX%26stype%3Dstext%26q%3D%2B%22Java%2BI%2FO%22%26search.x%3D38%26search.y%3D3

　　出现这个问题就是方法URLEncoder.encode( ) 在进行盲目地编码。它不能区分在URL或者查询string中被用到的特殊字符(象前面string中的“=”，和“&”)和确实需要被编码的字 符。由此，所以URL需要像下面这样一次只编码一块：

　　String query = URLEncoder.encode("pg");

　　query += "=";

　　query += URLEncoder.encode("q");

　　query += "&";

　　query += URLEncoder.encode("kl");

　　query += "=";

　　query += URLEncoder.encode("XX");

　　query += "&";

　　query += URLEncoder.encode("stype");

　　query += "=";

　　query += URLEncoder.encode("stext");

　　query += "&";

　　query += URLEncoder.encode("q");

　　query += "=";

　　query += URLEncoder.encode(""Java I/O"");

　　query += "&";

　　query += URLEncoder.encode("search.x");

　　query += "=";

　　query += URLEncoder.encode("38");

　　query += "&";

　　query += URLEncoder.encode("search.y");

　　query += "=";

　　query += URLEncoder.encode("3");

　　System.out.println(query);

　　这才是你真正想得到的输出：

　　pg=q&kl=XX&stype=stext&q=%2B%22Java+I%2FO%22&search.x=38&search.y=3

　　例子7-9是一个QueryString类。在一个java对象中，它使用了类URLEncoder来编码连续的属性名和属性值对，这个java对象被用来发送数据到服务器端的程序。

　　当你在创建一个QueryString对象时，你可以把查询string中的第一个属性对传递给类QueryString的构造函数，得到初始 string。如果要继续加入后面的属性对，就应调用方法add()，它也能接受两个string作为参数，能对它们进行编码。方法getQuery( )返回一个属性对被逐个编码后得到的整个string。

　　Example 7-9. -The QueryString class

　　package com.macfaq.net;

　　import java.net.URLEncoder;

　　import java.io.UnsupportedEncodingException;

　　public class QueryString {

　　private StringBuffer query = new StringBuffer();

　　public QueryString(String name, String value) {

　　encode(name, value);

　　}

　　public synchronized void add(String name, String value) {

　　query.append('&');

　　encode(name, value);

　　}

　　private synchronized void encode(String name, String value) {

　　try {

　　query.append(URLEncoder.encode(name, "UTF-8"));

　　query.append('=');

　　query.append(URLEncoder.encode(value, "UTF-8"));

　　} catch (UnsupportedEncodingException ex) {

　　throw new RuntimeException("Broken VM does not support UTF-8");

　　}

　　}

　　public String getQuery() {

　　return query.toString();

　　}

　　public String toString() {

　　return getQuery();

　　}

　　}

　　利用这个类，现在我们就能对前面那个例子中的string进行编码了:

　　QueryString qs = new QueryString("pg", "q");

　　qs.add("kl", "XX");

　　qs.add("stype", "stext");

　　qs.add("q", "+"Java I/O"");

　　qs.add("search.x", "38");

　　qs.add("search.y", "3");

　　String url = "http://www.altavista.com/cgi-bin/query?" + qs;

　　System.out.println(url);

　　二、URLDecoder

　　与URLEncoder 类相对应的URLDecoder 类有两种静态方法。它们解码以x-www-form-url-encoded这种形式编码的string。也就是说，它们把所有的加号(+)转换成空格符，把所有的%xx分别转换成与之相对应的字符：

　　public static String decode(String s) throws Exception

　　public static String decode(String s, String encoding) // Java 1.4 throws

UnsupportedEncodingException

　　第一种解码方法在java1.3和java1.2中使用。第二种解码方法在java1.4和更新的版本中使用。如果你拿不定主意用哪种编码方式，那就选择UTF-8吧。它比其他任何的编码形式更有可能得到正确的结果。

　　如果string包含了一个“%”，但紧跟其后的不是两位16进制的数或者被解码成非法序列，该方法就会抛出 IllegalArgumentException 异常。当下次再出现这种情况时，它可能就不会被抛出了。这是与运行环境相关的，当检查到有非法序列时，抛不抛出 IllegalArgumentException 异常，这时到底会发生什么是不确定的。在Sun's JDK 1.4中，不会抛出什么异常，它会把一些莫名其妙的字节加进不能被顺利编码的string中。这的确令人头疼，可能就是一个安全漏洞。

　　由于这个方法没有触及到非转义字符，所以你可以把整个URL作为参数传给该方法，不用像之前那样分块进行。例如：

　　String input = "http://www.altavista.com/cgi-bin/" + "query?

pg=q&kl=XX&stype=stext&q=%2B%22Java+I%2FO%22&search.x=38&search.y=3";

　　try {

　　String output = URLDecoder.decode(input, "UTF-8");

　　System.out.println(output);

　　}

|  |  |
| --- | --- |
|  | |
|  | |