JDK 中的 URLConnection 参数详解

针对 JDK 中的 URLConnection 连接 Servlet 的问题,网上有虽然有所涉及,但是只是说明了某一个或几个问题，是以 FAQ 的方式来解决的，而且比较零散，现在对这个类的使用就本人在项目中的使用经验做如下总结：

1:> URL 请求的类别:

分为二类,GET 与 POST 请求。二者的区别在于：

a:) get 请求可以获取静态页面，也可以把参数放在 URL 字串后面，传递给 servlet，

b:) post 与 get 的不同之处在于 post 的参数不是放在 URL 字串里面，而是放在 http 请求的正文内。

2:> URLConnection 的对象问题:

URLConnection 的对象,如下代码示例:

```
// 下面的 index.jsp 由<servlet-mapping>映射到
// 一个 Servlet(com.quantanetwork.getClientDataServlet)
// 该 Servlet 的注意点下边会提到
URL url = new URL("http://localhost:8080/TestHttpURLConnectionPro/index.jsp");

URLConnection rulConnection = url.openConnection();// 此处的 urlConnection 对象实际上是根据 URL 的
        // 请求协议(此处是 http)生成的 URLConnection 类
        // 的子类 HttpURLConnection,故此处最好将其转化
        // 为 HttpURLConnection 类型的对象,以便用到
        // HttpURLConnection 更多的 API.如下:

HttpURLConnection httpUrlConnection = (HttpURLConnection) rulConnection;
```

3:> HttpURLConnection 对象参数问题

```
// 设置是否向 httpUrlConnection 输出，因为这个是 post 请求，参数要放在
// http 正文内，因此需要设为 true, 默认情况下是 false;
httpUrlConnection.setDoOutput(true);

// 设置是否从 httpUrlConnection 读入，默认情况下是 true;
httpUrlConnection.setDoInput(true);

// Post 请求不能使用缓存
httpUrlConnection.setUseCaches(false);

// 设定传送的内容类型是可序列化的 java 对象
// (如果不设此项,在传送序列化对象时,当 WEB 服务默认的不是这种类型时可能抛 java.io.EOFException)
httpUrlConnection.setRequestProperty("Content-type", "application/x-java-serialized-object");
```

```
// 设定请求的方法为"POST"，默认是 GET
httpUrlConnection.setRequestMethod("POST");

// 连接，从上述第 2 条中 url.openConnection()至此的配置必须要在 connect 之前完成，
      httpUrlConnection.connect();
```

4:> HttpURLConnection 连接问题：

```
// 此处 getOutputStream 会隐含的进行 connect(即：如同调用上面的 connect()方法，
// 所以在开发中不调用上述的 connect()也可以)。
OutputStream outStrm = httpUrlConnection.getOutputStream();
```

5:> HttpURLConnection 写数据与发送数据问题：

```
// 现在通过输出流对象构建对象输出流对象，以实现输出可序列化的对象。
ObjectOutputStream objOutputStrm = new ObjectOutputStream(outStrm);

// 向对象输出流写出数据，这些数据将存到内存缓冲区中
objOutputStrm.writeObject(new String("我是测试数据"));

// 刷新对象输出流，将任何字节都写入潜在的流中（些处为 ObjectOutputStream）
objOutputStm.flush();

// 关闭流对象。此时，不能再向对象输出流写入任何数据，先前写入的数据存在于内存缓冲区中,
// 在调用下边的 getInputStream()函数时才把准备好的 http 请求正式发送到服务器
objOutputStm.close();

// 调用 HttpURLConnection 连接对象的 getInputStream()函数,
// 将内存缓冲区中封装好的完整的 HTTP 请求电文发送到服务端。
InputStream inStrm = httpConn.getInputStream(); // <===注意，实际发送请求的代码段就在
这里

// 上边的 httpConn.getInputStream()方法已调用,本次 HTTP 请求已结束,下边向对象输出流的输
出已无意义,
// 既使对象输出流没有调用 close()方法，下边的操作也不会向对象输出流写入任何数据.
// 因此，要重新发送数据时需要重新创建连接、重新设参数、重新创建流对象、重新写数据、
// 重新发送数据(至于是否不用重新这些操作需要再研究)
objOutputStm.writeObject(new String(""));
httpConn.getInputStream();
```

总结：a:) HttpURLConnection 的 connect()函数，实际上只是建立了一个与服务器的 tcp 连接，并没有实际发送 http 请求。

无论是 post 还是 get，http 请求实际上直到 HttpURLConnection 的 getInputStream() 这个函数里面才正式发送出去。

b:) 在用 POST 方式发送 URL 请求时，URL 请求参数的设定顺序是重中之重，

对 connection 对象的一切配置（那一堆 set 函数）

都必须要在 connect()函数执行之前完成。而对 outputStream 的写操作，又必须要在 inputStream 的读操作之前。

这些顺序实际上是由 http 请求的格式决定的。

如果 inputStream 读操作在 outputStream 的写操作之前，会抛出例外：

java.net.ProtocolException: Cannot write output after reading input.....

c:) http 请求实际上由两部分组成，

一个是 http 头，所有关于此次 http 请求的配置都在 http 头里面定义，

一个是正文 content。

connect()函数会根据 HttpURLConnection 对象的配置值生成 http 头部信息，因此在调用 connect 函数之前，

就必须把所有的配置准备好。

d:) 在 http 头后面紧跟着的是 http 请求的正文，正文的内容是通过 outputStream 流写入的，

实际上 outputStream 不是一个网络流，充其量是个字符串流，往里面写入的东西不会立即发送到网络，

而是存在于内存缓冲区中，待 outputStream 流关闭时，根据输入的内容生成 http 正文。

至此，http 请求的东西已经全部准备就绪。在 getInputStream()函数调用的时候，就会把准备好的 http 请求

正式发送到服务器了，然后返回一个输入流，用于读取服务器对于此次 http 请求的返回信息。由于 http

请求在 getInputStream 的时候已经发送出去了（包括 http 头和正文），因此在 getInputStream()函数

之后对 connection 对象进行设置（对 http 头的信息进行修改）或者写入 outputStream（对正文进行修改）

都是没有意义的了，执行这些操作会导致异常的发生。

6:> Servlet 端的开发注意点：

a:) 对于客户端发送的 POST 类型的 HTTP 请求，Servlet 必须实现 doPost 方法，而不能用 doGet 方法。

b:) 用 HttpServletRequest 的 getInputStream()方法取得 InputStream 的对象，比如：

InputStream inStream = httpRequest.getInputStream();

现在调用 inStream.available()（该方法用于"返回此输入流下一个方法调用可以不受阻塞地

从此输入流读取（或跳过）的估计字节数"）时，永远都反回 0。试图使用此方法的返回

值分配缓冲区，

以保存此流所有数据的做法是不正确的。那么，现在的解决办法是

Servlet 这一端用如下实现：

```
InputStream inStream = httpRequest.getInputStream();
ObjectInputStream objInStream = new ObjectInputStream(inStream);
Object obj = objInStream.readObject();
// 做后续的处理
// 。。。。。。
// 。。。 。。。
```

而客户端，无论是否发送实际数据都要写入一个对象（那怕这个对象不用），如：

```
ObjectOutputStream objOutputStrm = new ObjectOutputStream(outStrm);
objOutputStrm.writeObject(new String("")); // 这里发送一个空数据
// 甚至可以发一个 null 对象，服务端取到后再做判断处理。
objOutputStrm.writeObject(null);
objOutputStrm.flush();
objOutputStrm.close();
```

注意:上述在创建对象输出流 ObjectOutputStream 时,如果将从 HttpServletRequest 取得的输入流

(即:new ObjectOutputStream(outStrm)中的 outStrm)包装在 BufferedOutputStream 流里面,

则必须有 objOutputStrm.flush();这一句,以便将流信息刷入缓冲输出流.如下:

```
ObjectOutputStream objOutputStrm = new ObjectOutputStream(new
BufferedOutputStream(outStrm));
objOutputStrm.writeObject(null);
objOutputStrm.flush(); // <======此处必须要有.
objOutputStrm.close();
```

HttpURLConnection 是基于 HTTP 协议的，其底层通过 socket 通信实现。如果不设置超时（timeout），在网络异常的情况下，可能会导致程序僵死而不继续往下执行。可以通过以下两个语句来设置相应的超时：

```
System.setProperty("sun.net.client.defaultConnectTimeout", 超时毫秒数字符串);
System.setProperty("sun.net.client.defaultReadTimeout", 超时毫秒数字符串);
```

其中： sun.net.client.defaultConnectTimeout：连接主机的超时时间（单位：毫秒）
sun.net.client.defaultReadTimeout：从主机读取数据的超时时间（单位：毫秒）

例如：

```
System.setProperty("sun.net.client.defaultConnectTimeout", "30000");
System.setProperty("sun.net.client.defaultReadTime
```

Java 中可以使用 HttpURLConnection 来请求 WEB 资源。
HttpURLConnection 对象不能直接构造，需要通过 URL.openConnection()来获得 HttpURLConnection 对象，示例代码如下：

```
String szUrl = "http://www.ee2ee.com/";
URL url = new URL(szUrl);
HttpURLConnection urlCon = (HttpURLConnection)url.openConnection();
```

HttpURLConnection 是基于 HTTP 协议的，其底层通过 socket 通信实现。如果不设置超时（timeout），在网络异常的情况下，可能会导致程序僵死而不继续往下执行。可以通过以下两个语句来设置相应的超时：
System.setProperty("sun.net.client.defaultConnectTimeout", 超时毫秒数字符串);
System.setProperty("sun.net.client.defaultReadTimeout", 超时毫秒数字符串);

其中： sun.net.client.defaultConnectTimeout：连接主机的超时时间（单位：毫秒）
sun.net.client.defaultReadTimeout：从主机读取数据的超时时间（单位：毫秒）

例如：
System.setProperty("sun.net.client.defaultConnectTimeout", "30000");
System.setProperty("sun.net.client.defaultReadTimeout", "30000");

JDK 1.5 以前的版本，只能通过设置这两个系统属性来控制网络超时。在 1.5 中，还可以使用 HttpURLConnection 的父类 URLConnection 的以下两个方法：
setConnectTimeout：设置连接主机超时（单位：毫秒）
setReadTimeout：设置从主机读取数据超时（单位：毫秒）

例如：

```
HttpURLConnection urlCon = (HttpURLConnection)url.openConnection();
urlCon.setConnectTimeout(30000);
urlCon.setReadTimeout(30000);
```

需要注意的是，笔者在 JDK1.4.2 环境下，发现在设置了 defaultReadTimeout 的情况下，如果发生网络超时，HttpURLConnection 会自动重新提交一次请求，出现一次请求调用，请求服务器两次的问题（Trouble）。我认为这是 JDK1.4.2 的一个 bug。在 JDK1.5.0 中，此问题已得到解决，不存在自动重发现象。out", "30000");