最常用的Http请求无非是get和post，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，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。

在Java中可以使用HttpURLConnection发起这两种请求，了解此类，对于了解soap，和编写servlet的自动测试代码都有很大的帮助。

下面的代码简单描述了如何使用HttpURLConnection发起这两种请求，以及传递参数的方法：

```
public class HttpInvoker {

    public static final String GET_URL = "http://localhost:8080/welcome1";

    public static final String POST_URL = "http://localhost:8080/welcome1";

    public static void readContentFromGet() throws IOException {
        // 拼凑get请求的URL字串，使用URLEncoder.encode对特殊和不可见字符进行编码
        String getURL = GET_URL + "?username="
                + URLEncoder.encode("fat man", "utf-8");
        URL getUrl = new URL(getURL);
        // 根据拼凑的URL，打开连接，URL.openConnection函数会根据URL的类型，
        // 返回不同的URLConnection子类的对象，这里URL是一个http，因此实际返回的是
HttpURLConnection
        HttpURLConnection connection = (HttpURLConnection) getUrl
                .openConnection();
        // 进行连接，但是实际上get request要在下一句的connection.getInputStream()函数中才会真正发到
        // 服务器
        connection.connect();
        // 取得输入流，并使用Reader读取
        BufferedReader reader = new BufferedReader(new InputStreamReader(
                connection.getInputStream()));
        System.out.println("=============================");
        System.out.println("Contents of get request");
        System.out.println("=============================");
        String lines;
        while ((lines = reader.readLine()) != null) {
            System.out.println(lines);
        }
        reader.close();
        // 断开连接
        connection.disconnect();
        System.out.println("=============================");
```

```
        System.out.println("Contents of get request ends");
        System.out.println("==============================");
    }

    public static void readContentFromPost() throws IOException {
        // Post请求的url，与get不同的是不需要带参数
        URL postUrl = new URL(POST_URL);
        // 打开连接
        HttpURLConnection connection = (HttpURLConnection) postUrl
                .openConnection();
        // Output to the connection. Default is
        // false, set to true because post
        // method must write something to the
        // connection
        // 设置是否向connection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
        connection.setDoOutput(true);
        // Read from the connection. Default is true.
        connection.setDoInput(true);
        // Set the post method. Default is GET
        connection.setRequestMethod("POST");
        // Post cannot use caches
        // Post 请求不能使用缓存
        connection.setUseCaches(false);
        // This method takes effects to
        // every instances of this class.
        // URLConnection.setFollowRedirects是static函数，作用于所有的URLConnection对象。
        // connection.setFollowRedirects(true);

        // This methods only
        // takes effacts to this
        // instance.
        // URLConnection.setInstanceFollowRedirects是成员函数，仅作用于当前函数
        connection.setInstanceFollowRedirects(true);
        // Set the content type to urlencoded,
        // because we will write
        // some URL-encoded content to the
        // connection. Settings above must be set before connect!
```

```
        // 配置本次连接的Content-type，配置为application/x-www-form-urlencoded的
        // 意思是正文是urlencoded编码过的form参数，下面我们可以看到我们对正文内容使用
URLEncoder.encode
        // 进行编码
        connection.setRequestProperty("Content-Type",
                "application/x-www-form-urlencoded");
        // 连接，从postUrl.openConnection()至此的配置必须要在connect之前完成，
        // 要注意的是connection.getOutputStream会隐含的进行connect。
        connection.connect();
        DataOutputStream out = new DataOutputStream(connection
                .getOutputStream());
        // The URL-encoded contend
        // 正文，正文内容其实跟get的URL中'?'后的参数字符串一致
        String content = "firstname=" + URLEncoder.encode("一个大肥人", "utf-8");
        // DataOutputStream.writeBytes将字符串中的16位的unicode字符以8位的字符形式写道流里面
        out.writeBytes(content);

        out.flush();
        out.close(); // flush and close
        BufferedReader reader = new BufferedReader(new InputStreamReader(
                connection.getInputStream()));
        String line;
        System.out.println("=============================");
        System.out.println("Contents of post request");
        System.out.println("=============================");
        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }
        System.out.println("=============================");
        System.out.println("Contents of post request ends");
        System.out.println("=============================");
        reader.close();
        connection.disconnect();
    }

    /**
     * @param args
     */
```

```
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        try {
            readContentFromGet();
            readContentFromPost();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

上面的readContentFromGet()函数产生了一个get请求，传给servlet一个username参数，值为"fat man"。readContentFromPost()函数产生了一个post请求，传给servlet一个firstname参数，值为"一个大肥人"。HttpURLConnection.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.getInputStream()这个函数里面才正式发送出去。

在readContentFromPost()中，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。

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，在connect()函数里面，会根据HttpURLConnection对象的配置值生成http头，因此在调用connect函数之前，就必须把所有的配置准备好。

紧接着http头的是http请求的正文，正文的内容通过outputStream写入，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，而是在流关闭后，根据输入的内容生成http正文。

至此，http请求的东西已经准备就绪。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。