从容器到 HttpServlet

web 容器做了什么

当浏览器请求来到HTTP服务器, HTTP服务器转交给容器, 容器会创建一个代表本次请求的`HttpServlet`对象, 并传给请求的相关信息, 同时会创建一个`HttpServletResponse`对象, 作为稍后要对客户端进行响应的Java对象.

然后, 容器会根据@WebServlet标注或web.xml的设置, 找到处理该请求的Servlet, 调用它的`service()`方法, 传入容器在之前所创建的两个对象, 在service()方法中会根据HTTP请求的方式来调用对应的`doXXX()`方法.

最后容器将`HttpServletRequest`对象和`HttpServletResponse`对象销毁回收, 结束本次响应.

doXXX() 方法

Servlet接口的`service()`方法中, 实际接受的参数是`ServletRequest`, `ServletResponse`, 在定义Servlet时, 期待Servlet不止用于HTTP, 与HTTP相关的行为由两者的子接口`HttpServletRequest`, `HttpServletResponse` 定义.

关于 HttpServletRequest

处理请求参数与标头

HttpServletRequest中定义了取得通用请求信息的方法

- `getParameter()`: 指定请求参数名称来取得对应的值

  ```
  1 String username = request.getParameter("name");
  ```

  返回String对象, 如果请求中没有指定的参数名称, 返回null
- `getParameterValues()`: 若同一个请求参数名称有多个值, 可以使用这个方法

  ```
  1 String[] values = request.getParameterValues("param");
  ```

返回一个存储了参数的值的String数组

- `getParameterNames()`: 获取请求中所有的请求参数名称, 返回`Enumeration`对象

```
Enumeration<String> e = req.getParameterNames();
while (e.hasMoreElements()) {
    String name = e.nextElement();
    // ...
}
```

- `getParameterMap()`: 将请求参数以`Map`对象返回, 键为请求参数名(String), 值为请求参数值(String[])

获取HTTP标头信息

- `getHeader()`: 与`getParameter()`类似, 指定标头名称返回标头信息
- `getHeaders()`: 与`getParameterValues()`类似, 指定标头名称返回`Enumeration`对象, 元素为字符串
- `getHeaderName()`: 与`getParameterNames()`类似, 取得所有标头名称, 返回`Enumeration`

~~请求参数编码处理~~

- POST

  如果客户端没有在 Content-Type 标头中设置字符编码, 使用`HttpSerletRequest`的`getCharacterEncoding()`返回值是`null`

  可以使用`HttpServletRequest`的`setCharacterEncoding()`方法指定取得POST请求参数时使用的编码, 只有在取得参数前执行才有用.

- GET

  如果是GET, 上述的方法设置编码无用.

  另一种处理编码的方式: 通过`String` 的 `getBytes()`指定编码来取得字符串的字节数组, 然后再重新构造为正确编码的字符串.

```java
String name = req.getParameter("name");
// 假设浏览器使用UTF-8, web容器使用ISO-8859-1
name = new String(name.getBytes("ISO-8859-1"), "UTF-8");
// public String(byte[] bytes, String charsetName);
```

Tomcat8URI默认编码为“UTF-8”， 而Tomcat7URI默认编码为“ISO-8859-1”

原书为繁体, 如果要使结果正常需要使用Tomcat7环境, form-get.html, form-post.html两个网页用繁体(编码为Big5), 或者简体(编码为GBK)

参考资料: (转)Java 正确的做字符串编码转换

> 那么， 如何利用getBytes 和 new String() 来进行编码转换呢？ 网上流传着一种错误的方法:
> GBK--> UTF-8: new String(s.getBytes("GBK") , "UTF-8"); ,这种方式是完全错误的， 因为getBytes 的编码与 UTF-8 不一致， 肯定是乱码。
> 但是为什么在tomcat 下， 使用 new String(s.getBytes("iso-8859-1") ,"GBK") 却可以用呢？ 答案是：
> tomcat 默认使用iso-8859-1编码， 也就是说， 如果原本字符串是GBK的， tomcat传输过程中， 将GBK转成iso-8859-1了，
> 默认情况下， 使用iso-8859-1读取中文肯定是有问题的， 那么我们需要将iso-8859-1 再转成GBK， 而iso-8859-1 是单字节编码的，
> 即他认为一个字节是一个字符， 那么这种转换不会对原来的字节数组做任何改变，
> 因为字节数组本来就是由单个字节组成的，
> 如果之前用GBK编码， 那么转成iso-8859-1后编码内容完全没变， 则 s.getBytes("iso-8859-1") 实际上还是原来GBK的编码内容
> 则 new String(s.getBytes("iso-8859-1") ,"GBK") 就可以正确解码了。 所以说这是一种巧合。

getReader(), getInputStream() 读取 Body 内容

`HttpServletRequest`上有`getReader()`方法, 可以取得`BufferedReader`对象, 通过这个对象可以读取请求的 Body 数据.

```java
private String readBody(HttpServletRequest request) throws IOException {
    BufferedReader reader = request.getReader();
    // 取得 BufferedReader 对象, 通过该对象可以读取请求的 Body 数据
    String input = null;
    String requestBody = "";
    while ((input = reader.readLine()) != null) {
        requestBody = requestBody + input + "<br>";
    }
    return requestBody;
}
```

输出:

```
user=%E5%BC%A0%E4%B8%89&passed=1234567&login=%E9%80%81%E5%87%BA
```

如果要上传文件, <form> 标签需要设置`enctype`属性为: `multipart/form-data`

getPart() , getParts() 取得上传文件

在Servlet3.0之后, 新增了`Part`接口, 可以方便的进行文件上传处理, 可以通过`HttpServletRequest`的`getPart()`方法并指定名称才能取得`Part`实现对象.

Servlet中要设置`@MultipartConfig`标注才能取得`Part`对象, 否则会得到`null`, 仅仅标注`@MultipartConfig`表示相关属性使用默认值:

- `fileSizeThreshold`: 0

  整数值, 若上传文件大小超过设置的值, 会先写入缓存文件

- `loaction`: 空字符串 ""

  设置写入文件时的目录

- `maxFileSize`: -1L 不限制大小

  限制上传文件的大小

- `maxRequestSize`: -1L 不限制请求个数

  限制 `multipart/form-data` 请求个数

也可以使用`getParts()`来上传多个文件，该方法返回一个`Collection<Part>`

```
// ...
        for (Part part : req.getParts()) {
            if(part.getName().startsWith("file")) {
                // 使用getName()获取名称, startsWith()判断名称是否以file开头
                String filename= getFilename(part);
                part.write(filename);
            }
        }
// ...
```

在 web.xml 中也可以设置`@MultipartConfig`属性:

```
...
<servlet>
    <servlet-name>UploadServlet</servlet-name>
    <servlet-class>UploadServlet</servlet-class>
    <multipart-config>
        <location>/tmp/</location>
    </multipart-config>
</servlet>
...
```

使用 RequestDispatcher 调派请求

在 Web 应用程序中, 经常需要多个 Servlet 来完成请求, 这时可以使用`HttpServletRequest`的`getRequestDispatcher()`方法取得`RequestDispatcher`接口的实例, 调用时只需指定转发或包含相对的URL网址.

1. 使用 include() 方法

`RequestDispatcher` 的 include() 方法可以将另一个 Servlet 的操作流程包括至目前 Servlet 操作流程之中.

```
RequestDispatcher dispatcher =
req.getRequestDispatcher("other.view");
dispatcher.include(req, resp);
```

在取得`RequestDispatcher`时也可以包括查询字符串, `...("other.view?data=123456");`

2. 请求范围属性

在`include()`或`forward()`时如果包括请求参数的做法只适用于传递字符串给另一个Servlet, 在调派请求中, 如果有必须共享的对象, 可以设置请求范围属性

`HttpServletRequest`上与请求范围属性有关的方法:

- `setAttribute()`: 指定名称与对象设置属性
- `getAttribute()`: 指定名称取得属性
- `getAttributeNames()`: 取得所有属性名称
- `remoteAttribute()`: 指定名称移除属性

以`java.`, `javax.`开头的名称通常保留, 用于表示一些特定的意义:

- `javax.servlet.include.request_uri`
- `javax.servlet.include.context_path`
- `javax.servlet.include.servlet_path`
- `javax.servlet.include.path_info`
- `javax.servlet.include.query_string`

在被包含的 Servlet 中分别表示上一个 Servlet 的 Request URI, Context path, Servlet path, Path info 和取得`RequestDispatcher`时给定的参数

3. 使用 forward() 方法

调用时同样也要传入请求和响应对象, 表示要将请求处理转发给别的 Servlet , 对客户端的响应同时转发给另一个 Servlet

如果要调用`forward()`方法, 当前的Servlet中不能有任何响应确认, 如果通过响应对象设置了响应但未确认, 响应设置会全部被忽略, 如果有响应确认了仍调用这个方法, 会抛出`IllegalStateException`