JavaWeb之Servlet

本节目标

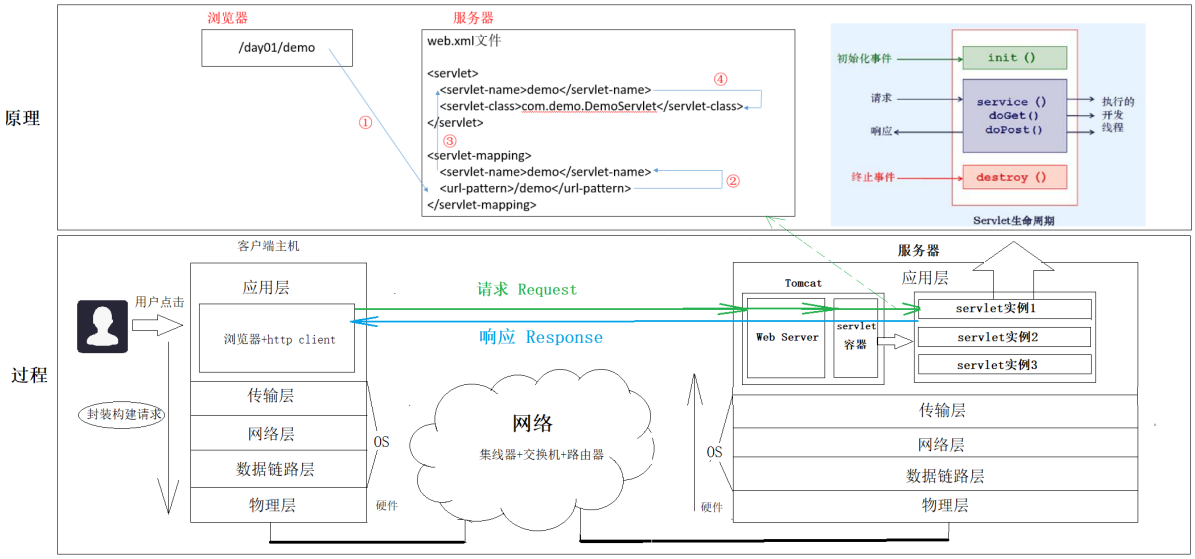
- 熟悉servlet基本概念与作用
- 熟悉servlet定位
- 熟悉servlet生命周期与一般使用过程
- 熟悉servlet request & response的处理机制
- 熟练掌握servlet所有常见方法
- 深刻理解cookie & session
- 掌握文件上传的一般过程

1. Servlet简介

1.1 什么是servlet

- 概念：Java Servlet 是运行在 Web 服务器或应用服务器上的程序，它是作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层。
- 定位：Java Servlet用Java编写的服务器端程序（web application）。
- 作用：其主要功能在于交互式地浏览和修改数据，生成动态Web内容。
- 理解：狭义的Servlet是指Java语言实现的一个接口，广义的Servlet是指任何实现了这个Servlet接口的类，一般情况下，我们将Servlet理解为后者。

1.2 Servlet定位图



1.3 Servlet核心作用

> - 参考上图

1.4 Servlet标准 API核心包（在线文档）

Servlet API有以下3个Java包：

> - javax.servlet（重点）：其中包含定义Servlet和Servlet容器之间的类和接口
> - javax.servlet.http(重点)：其中包含定义HTTP Servlet和Servlet容器之间的类和接口
> - javax.servlet.annotation：其中包含标注Servlet，Filter，Listener的注解

备注：JavaWeb中主要关注javax.servlet和javax.servlet.http的成员。

参考http://tomcat.apache.org/tomcat-7.0-doc/servletapi/index.html

2. Servlet环境设置

- 参照<<初识 Servlet>>课件

3. Servlet常见方法与生命周期

3.1 概览

Servlet 生命周期可被定义为从创建直到毁灭的整个过程。以下是 Servlet 遵循的过程：

- Servlet 通过调用 **init ()** 方法进行初始化。
- Servlet 调用 **service()** 方法来处理客户端的请求。
- Servlet 通过调用 **destroy()** 方法终止（结束）。
- 最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

3.2 HttpServlet处理Http请求

- Servlet的service()方法是请求的入口方法，HttpServlet实现service()方法在这个入口方法中根据不同的Http请求方法（如GET、POST请求）调用不同的方法。
- 大多数应用程序都是要于HTTP结合起来使用。这意味着可以利用HTTP提供的特性。`javax.servlet.http`包是Servlet API中的第二个包，其中包含了用于编写Servlet应用程序的类和接口，并且许多类型都覆写了`javax.servlet`中的类型。
- HttpServlet类覆盖了`javax.servlet.GenericServlet`类。使用HttpServlet时，需要使用代表Servlet请求和Servlet响应的`HttpServletRequest`和`HttpServletResponse`对象。

HttpServlet中的Service方法会检验用来发送请求的HTTP方法(通过调用request.getMethod()), 并调用以下方法之一：

方法名称	重要性
doGet	重要且常用
doPost	重要且常用
doHead	了解
doPut	了解
doTrace	了解
doOptions	了解
doDelete	了解

这7中方法，每一种方法都表示一个HTTP方法，**`doGet`和`doPost`是最常用的。**

而service方法在内部已经对收到的http请求的方法做了分类，从而让请求可以路由到不同的具体方法(如：doGet)

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        String method = req.getMethod();
```

```
        if (method.equals(METHOD_GET)) {
            ...
            if (lastModified == -1) {
                doGet(req, resp);
            } else {
                ...
                if (ifModifiedSince < lastModified) {
                    ...
                    doGet(req, resp);
                } else {
                    resp.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
                }
            }
        } else if (method.equals(METHOD_HEAD)) {
            ...
            doHead(req, resp);
        } else if (method.equals(METHOD_POST)) {
            doPost(req, resp);
        } else if (method.equals(METHOD_PUT)) {
            doPut(req, resp);
        } else if (method.equals(METHOD_DELETE)) {
            doDelete(req, resp);
        } else if (method.equals(METHOD_OPTIONS)) {
            doOptions(req,resp);
        } else if (method.equals(METHOD_TRACE)) {
            doTrace(req,resp);
        } else {
            ...
        }
    }
```

因此，实际在编写servlet程序的时候，不再需要覆盖Service方法了，只要覆盖doGet或者doPost即可。

3.3 细节

A. init 方法被设计成只调用一次。

它在第一次创建 Servlet 时被调用，在后续每次用户请求时不再调用。因此，它是用于一次性初始化，就像 Applet 的 init 方法一样。

Servlet 创建于用户第一次调用对应于该 Servlet 的 URL 时，但是也可以指定 Servlet 在服务器第一次启动时被加载。

当用户调用一个 Servlet 时，就会创建一个 Servlet 实例，每一个用户请求都会产生一个新的线程，适当的时候移交给 doGet 或 doPost 方法。init() 方法简单地创建或加载一些数据，这些数据将被用于 Servlet 的整个生命周期。

init 方法的定义如下：

```
public void init() throws ServletException {
  // 初始化代码...
}
```

B. service() 方法是执行实际任务的主要方法。

Servlet 容器（即 Web 服务器）调用 service() 方法来处理来自客户端（浏览器）的请求，并把格式化的响应写回给客户端。

每次服务器接收到一个 Servlet 请求时，服务器会产生一个新的线程并调用服务。service() 方法检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut，doDelete 等方法。

下面是该方法的特征：

```
public void service(ServletRequest request,
                    ServletResponse response)
      throws ServletException, IOException{
}
```

service() 方法由容器调用，service 方法在适当的时候调用 doGet、doPost、doPut、doDelete 等方法。所以，不用对 service() 方法做任何动作，只需要根据来自客户端的请求类型来重写 doGet() 或 doPost() 即可。

doGet() 和 doPost() 方法是每次服务请求中最常用的方法,后面我们会重点使用。

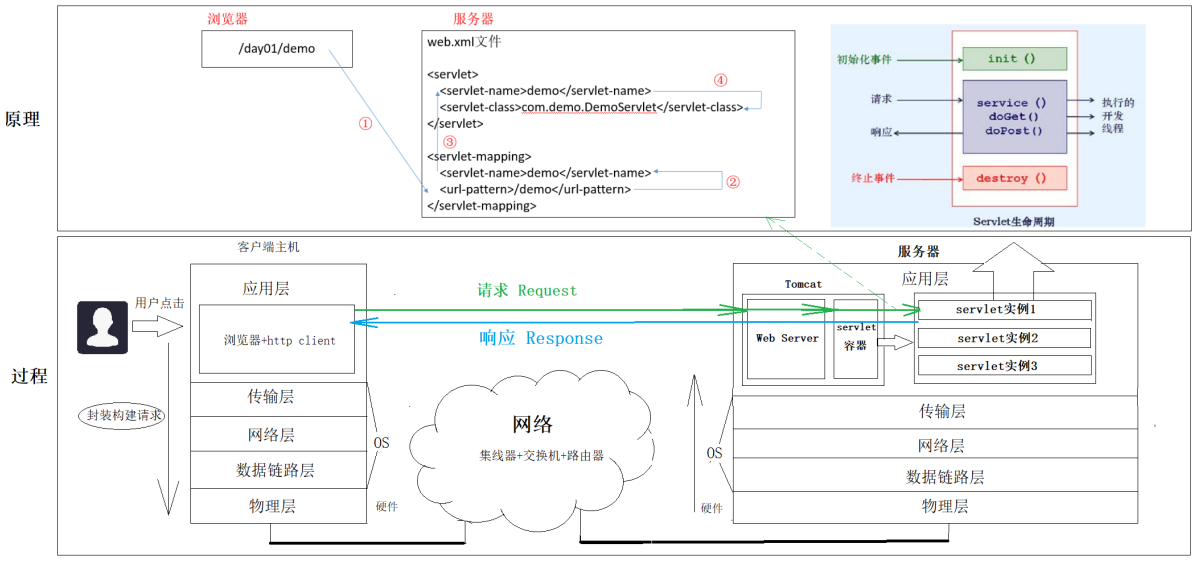
C. destroy() 方法只会被调用一次，在 Servlet 生命周期结束时被调用。

destroy() 方法可以让您的 Servlet 关闭数据库连接、停止后台线程、把 Cookie 列表或点击计数器写入到磁盘，并执行其他类似的清理活动。

在调用 destroy() 方法之后，servlet 对象被标记为垃圾回收。destroy 方法定义如下所示：

```
public void destroy() {
   // 终止化代码...
 }
```

3.4 Servlet请求处理过程图



经过之前的学习，我们已经知道url和servlet实例是要有对应的映射关系的，所以往往需要我们配置web.xml来进行映射，此处就不在给大家细讲了，大家可以回头看看《初识 Servlet》。

除了上面的配置，还有其他更多的配置项，可以具体问题具体对待。

全配置链接：

4. Servlet 客户端 HTTP 请求

4.1 客户端Request常见报头

当浏览器请求网页时，它会向 Web 服务器发送特定信息，这些信息不能被直接读取，因为这些信息是作为 HTTP 请求的头的一部分进行传输的。我们可以查看 HTTP 协议 了解更多相关信息。

以下是来自于浏览器端的重要头信息，我们可以在 Web 编程中频繁使用【后续如没有特殊说明，黑体为重要，需要掌握，其他作为了解】：

头信息	描述
Accept	这个头信息指定浏览器或其他客户端可以处理的 MIME 类型。值 **image/png** 或 **image/jpeg** 是最常见的两种可能值。
Accept-Charset	这个头信息指定浏览器可以用来显示信息的字符集。例如 ISO-8859-1。
Accept-Encoding	这个头信息指定浏览器知道如何处理的编码类型。值 **gzip** 或 **compress** 是最常见的两种可能值。
Accept-Language	这个头信息指定客户端的首选语言，在这种情况下，Servlet 会产生多种语言的结果。例如，en、en-us、ru 等。
Authorization	这个头信息用于客户端在访问受密码保护的网页时识别自己的身份。
Connection	这个头信息指示客户端是否可以处理持久 HTTP 连接。持久连接允许客户端或其他浏览器通过单个请求来检索多个文件。值 **Keep-Alive** 意味着使用了持续连接。
Content-Length	这个头信息只适用于 POST 请求，并给出 POST 数据的大小（以字节为单位）。
Cookie	这个头信息把之前发送到浏览器的 cookies 返回到服务器。
Host	这个头信息指定原始的 URL 中的主机和端口。
If-Modified-Since	这个头信息表示只有当页面在指定的日期后已更改时，客户端想要的页面。如果没有新的结果可以使用，服务器会发送一个 304 代码，表示 **Not Modified** 头信息。
If-Unmodified-Since	这个头信息是 If-Modified-Since 的对立面，它指定只有当文档早于指定日期时，操作才会成功。
Referer	这个头信息指示所指向的 Web 页的 URL。例如，如果您在网页 1，点击一个链接到网页 2，当浏览器请求网页 2 时，网页 1 的 URL 就会包含在 Referer 头信息中。
User-Agent	这个头信息识别发出请求的浏览器或其他客户端，并可以向不同类型的浏览器返回不同的内容。

4.2 操作 HTTP Request头的方法

下面的方法可用在 Servlet 程序中读取 HTTP 头。这些方法通过 *HttpServletRequest* 对象使用。

方法	描述
Cookie[] getCookies()	返回一个数组，包含客户端发送该请求的所有的 Cookie 对象。
Enumeration getHeaderNames()	返回一个枚举，包含在该请求中包含的所有的头名。
Enumeration getParameterNames()	返回一个 String 对象的枚举，包含在该请求中包含的参数的名称。
HttpSession getSession()	返回与该请求关联的当前 session 会话，或者如果请求没有 session 会话，则创建一个。
HttpSession getSession(boolean create)	返回与该请求关联的当前 HttpSession，或者如果没有当前会话，且创建是真的，则返回一个新的 session 会话。
String getCharacterEncoding()	返回请求主体中使用的字符编码的名称。
String getContentType()	返回请求主体的 MIME 类型，如果不知道类型则返回 null。
String getContextPath()	返回指示请求上下文的请求 URI 部分。
String getHeader(String name)	以字符串形式返回指定的请求头的值。
String getMethod()	返回请求的 HTTP 方法的名称，例如，GET、POST 或 PUT。
String getParameter(String name)	以字符串形式返回请求参数的值，或者如果参数不存在则返回 null。
String getProtocol()	返回请求协议的名称和版本。
String getQueryString()	返回包含在路径后的请求 URL 中的查询字符串。
String getRemoteUser()	如果用户已通过身份验证，则返回发出请求的登录用户，或者如果用户未通过身份验证，则返回 null。
String getRequestURI()	从协议名称直到 HTTP 请求的第一行的查询字符串中，返回该请求的 URL 的一部分。
String getRequestedSessionId()	返回由客户端指定的 session 会话 ID。
String[] getParameterValues(String name)	返回一个字符串对象的数组，包含所有给定的请求参数的值，如果参数不存在则返回 null。
int getContentLength()	以字节为单位返回请求主体的长度，并提供输入流，或者如果长度未知则返回 -1。

4.3 代码示例

- 安装之前的内容，建立maven web项目
- 配置web.xml
- 后面就可以直接编写代码了

web.xml配置说明

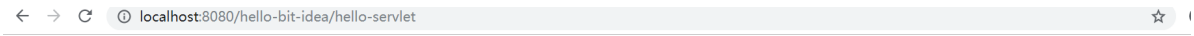
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1"
         metadata-complete="true">
    <servlet>
        <servlet-name>Hello</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Hello</servlet-name>
        <url-pattern>/hello-servlet</url-pattern>
    </servlet-mapping>
</web-app>
```

核心方法使用1-基本方法使用：

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

public class HelloServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse
response)throws ServletException,IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String method = request.getMethod();//返回请求的 HTTP 方法的名称
        String encoding = request.getCharacterEncoding();//返回请求主体中使用的字符编
码的名称
        String url = request.getContextPath();//返回指示请求上下文的请求 URI 部分
        String contentType = request.getContentType();//返回请求主体的 MIME 类型，如
果不知道类型则返回 null

        String title = "HTTP Function Test!";
        String docType = "<!doctype html>\n";
        out.println(docType + "<html>\n" +
                "<head><title>" +title+ "</title></head>\n" +
                "<body>"+
                "<h3> Method: " +method+ "</h3>" +
                "<h3> encoding: " +encoding+ "</h3>" +
```

```
                "<h3> url: " +url+ "</h3>" +
                "<h3> contentType: " +contentType+ "</h3>" +

                "</body>"+
                "</html>");
    }
    @Override
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException{
        doGet(request,response);
    }
}
```

运行结果



Method: GET

encoding: null

url: /hello-bit-idea

contentType: null

核心方法使用2-表单

先编写我们基本的index.html

```
<!DOCTYPE html>
<html lang="en">
<body>
<form action="hello-servlet" method="GET"> <!--先使用GET-->
    first_name:<input type="text" name="first_name">
    <br />
    last_name:<input type="text" name="last_name" />
    <input type="submit" value="submit" />
</form>
</body>
</html>
```

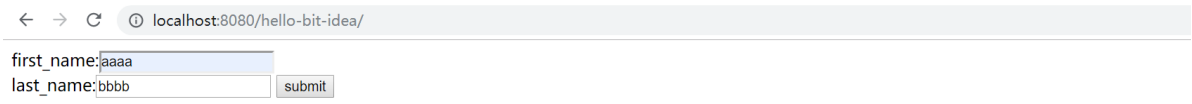
编写处理Servlet逻辑

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

public class HelloServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse
response)throws ServletException,IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String method = request.getMethod();
```

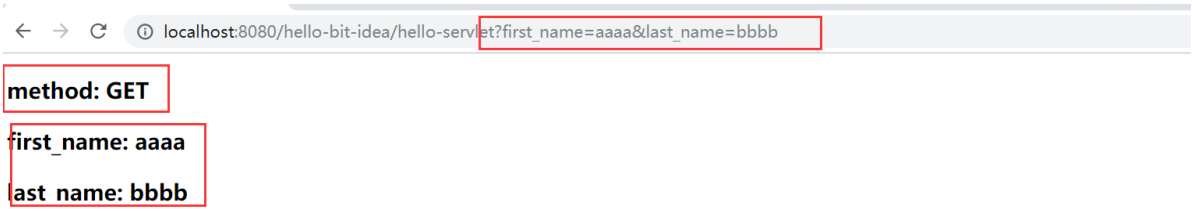
```
        String first_name = request.getParameter("first_name");
        String last_name = request.getParameter("last_name");

        String title = "HTTP Function Test!";
        String docType = "<!doctype html>\n";
        out.println(docType + "<html>\n" +
                "<head><title>" +title+ "</title></head>\n" +
                "<body>"+
                "<h3> method: " +method+ "</h3>" +
                "<h3> first_name: " +first_name+ "</h3>" +
                "<h3> last_name: " +last_name+ "</h3>" +
                "</body>"+
                "</html>");
    }
    @Override
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException{
        doGet(request,response);
    }
}
```

验证结果：

- 先获取表单，填充参数



- GET提交，参数回显到url中！！



- 改成POST请求,修改index.html

```
...
<form action="hello-servlet" method="POST">
...
```

核心方法使用3-获取HTTP Request头部信息

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```java
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;


public class HelloServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse
response)throws ServletException,IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String title = "HTTP Header Request";
        String docType = "<!doctype html\">\n";
        out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n"+
                "<body>\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                "<table width=\"100%\" border=\"2\" align=\"center\">\n" +
                "<tr>\n" +
                "<th>Header name</th><th>Header value</th>\n"+
                "</tr>\n");

        Enumeration headerNames = request.getHeaderNames();//返回一个枚举，包含在该请
求中包含的所有的头名。

        while(headerNames.hasMoreElements()) {
            String paramName = (String)headerNames.nextElement();
            out.print("<tr><td>" + paramName + "</td>\n");
            String paramValue = request.getHeader(paramName);//以字符串形式返回指定
的请求头的值
            out.println("<td> " + paramValue + "</td></tr>\n");
        }
        out.println("</table>\n</body></html>");
    }
    @Override
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException{
        doGet(request,response);
    }
}
```

运行结果：

HTTP Header Request

Header name	Header value
host	localhost:8080
connection	keep-alive
upgrade-insecure-requests	1
user-agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36
sec-fetch-user	?1
accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
purpose	prefetch
sec-fetch-site	none
sec-fetch-mode	navigate
accept-encoding	gzip, deflate, br
accept-language	zh-CN,zh;q=0.9
cookie	Webstorm-b7ff17d6=31ad0d1a-5067-454d-b73a-5befbc4debe6; Idea-cb2cf4de=7fb16337-395f-41eb-a431-3811458b9b7d

5. Servlet 服务器 HTTP 响应

5.1 服务器端Response常见报头

下表总结了从 Web 服务器端返回到浏览器的最有用的 HTTP 1.1 响应报头，您会在 Web 编程中频繁地使用它们：

头信息	描述
Allow	这个头信息指定服务器支持的请求方法（GET、POST 等）。
Cache-Control	这个头信息指定响应文档在何种情况下可以安全地缓存。可能的值有：**public、private** 或 **no-cache** 等。Public 意味着文档是可缓存，Private 意味着文档是单个用户私用文档，且只能存储在私有（非共享）缓存中，no-cache 意味着文档不应被缓存。
Connection	这个头信息指示浏览器是否使用持久 HTTP 连接。值 **close** 指示浏览器不使用持久 HTTP 连接，值 **keep-alive** 意味着使用持久连接。
Content-Disposition	这个头信息可以让您请求浏览器要求用户以给定名称的文件把响应保存到磁盘。
Content-Encoding	在传输过程中，这个头信息指定页面的编码方式。
Content-Language	这个头信息表示文档编写所使用的语言。例如，en、en-us、ru 等。
Content-Length	这个头信息指示响应中的字节数。只有当浏览器使用持久（keep-alive）HTTP 连接时才需要这些信息。
Content-Type	这个头信息提供了响应文档的 MIME（Multipurpose Internet Mail Extension）类型。
Expires	这个头信息指定内容过期的时间，在这之后内容不再被缓存。
Last-Modified	这个头信息指示文档的最后修改时间。然后，客户端可以缓存文件，并在以后的请求中通过 **If-Modified-Since** 请求头信息提供一个日期。
Location	这个头信息应被包含在所有的带有状态码的响应中。在 300s 内，这会通知浏览器文档的地址。浏览器会自动重新连接到这个位置，并获取新的文档。
Refresh	这个头信息指定浏览器应该如何尽快请求更新的页面。您可以指定页面刷新的秒数。
Retry-After	这个头信息可以与 503（Service Unavailable 服务不可用）响应配合使用，这会告诉客户端多久就可以重复它的请求。
Set-Cookie	这个头信息指定一个与页面关联的 cookie。

5.2 操作HTTP Response头的方法

下面的方法可用于在 Servlet 程序中设置 HTTP 响应报头。这些方法通过 *HttpServletResponse* 对象可用。

序号	方法 & 描述
String encodeRedirectURL(String url)	为 sendRedirect 方法中使用的指定的 URL 进行编码，或者如果编码不是必需的，则返回 URL 未改变。
String encodeURL(String url)	对包含 session 会话 ID 的指定 URL 进行编码，或者如果编码不是必需的，则返回 URL 未改变。
boolean isCommitted()	返回一个布尔值，指示响应是否已经提交。
void addCookie(Cookie cookie)	把指定的 cookie 添加到响应。
void addDateHeader(String name, long date)	添加一个带有给定的名称和日期值的响应报头。
void addHeader(String name, String value)	添加一个带有给定的名称和值的响应报头。
void addIntHeader(String name, int value)	添加一个带有给定的名称和整数值的响应报头。
void sendRedirect(String location)	使用指定的重定向位置 URL 发送临时重定向响应到客户端。
void setCharacterEncoding(String charset)	设置被发送到客户端的响应的字符编码（MIME 字符集）例如，UTF-8。
void setContentLength(int len)	设置在 HTTP Servlet 响应中的内容主体的长度，该方法设置 HTTP Content-Length 头。
void setContentType(String type)	如果响应还未被提交，设置被发送到客户端的响应的内容类型。
void setDateHeader(String name, long date)	设置一个带有给定的名称和日期值的响应报头。
void setHeader(String name, String value)	设置一个带有给定的名称和值的响应报头。
void setIntHeader(String name, int value)	设置一个带有给定的名称和整数值的响应报头。
void setStatus(int sc)	为该响应设置状态码。

5.3 代码示例

核心方法使用-自动刷新页面

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
```

```java
import java.util.Calendar;
import java.util.GregorianCalendar;

// 扩展 HttpServlet 类
public class HelloServlet extends HttpServlet {

    // 处理 GET 方法请求的方法
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
            throws ServletException, IOException
    {
        // 设置刷新自动加载时间为 1 秒
        response.setIntHeader("Refresh", 1); //设置一个带有给定的名称和整数值的响应报
头。

        // 设置响应内容类型
        response.setContentType("text/html");

        // Get current time
        Calendar calendar = new GregorianCalendar();//获取当前系统时间
        String am_pm;
        int hour = calendar.get(Calendar.HOUR);//获得时
        int minute = calendar.get(Calendar.MINUTE);//获得分
        int second = calendar.get(Calendar.SECOND);//获得秒
        if(calendar.get(Calendar.AM_PM) == 0) //判定是否是上午和下午
            am_pm = "AM";
        else
            am_pm = "PM";

        String CT = hour+":"+ minute +":"+ second +" "+ am_pm;

        PrintWriter out = response.getWriter();
        String title = "auto refresh Header set";
        String docType = "<!doctype html>\n";
        out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n"+
                "<body>\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                "<p>current time: " + CT + "</p>\n");
    }
    // 处理 POST 方法请求的方法
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```

6. Http状态码

6.1 HTTP 状态码列表

以下是可能从 Web 服务器返回的 HTTP 状态码和相关的信息列表：

代码	消息	描述
100	Continue	只有请求的一部分已经被服务器接收，但只要它没有被拒绝，客户端应继续该请求。
101	Switching Protocols	服务器切换协议。
200	OK	请求成功。
201	Created	该请求是完整的，并创建一个新的资源。
202	Accepted	该请求被接受处理，但是该处理是不完整的。
204	No Content	无内容
206	Partial Content	局部请求
301	Moved Permanently	所请求的页面已经转移到一个新的 URL，永久重定向
302	Found	所请求的页面已经临时转移到一个新的 URL。
303	See Other	所请求的页面可以在另一个不同的 URL 下被找到。
307	Temporary Redirect	所请求的页面已经临时转移到一个新的 URL，临时重定向
400	Bad Request	服务器不理解请求。
401	Unauthorized	所请求的页面需要用户名和密码。
402	Payment Required	*您还不能使用该代码。*
403	Forbidden	禁止访问所请求的页面。
404	Not Found	服务器无法找到所请求的页面。.
405	Method Not Allowed	在请求中指定的方法是不允许的。
408	Request Timeout	请求需要的时间比服务器能够等待的时间长，超时。
413	Request Entity Too Large	服务器不接受该请求，因为请求实体过大。
414	Request-url Too Long	服务器不接受该请求，因为 URL 太长。当您转换一个 "post" 请求为一个带有长的查询信息的 "get" 请求时发生。
500	Internal Server Error	未完成的请求。服务器遇到了一个意外的情况。
502	Bad Gateway	未完成的请求。服务器从上游服务器收到无效响应。
503	Service Unavailable	未完成的请求。服务器暂时超载或死机。

代码	消息	描述
504	Gateway Timeout	网关超时。
505	HTTP Version Not Supported	服务器不支持"HTTP协议"版本。

6.2 设置 HTTP 状态代码的方法

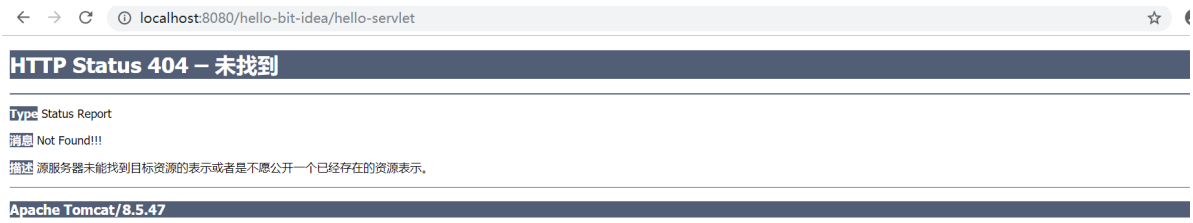
下面的方法可用于在 Servlet 程序中设置 HTTP 状态码。这些方法通过 *HttpServletResponse* 对象可用。

序号	方法 & 描述
1	**public void setStatus (int statusCode)** 该方法设置一个任意的状态码。setStatus 方法接受一个 int（状态码）作为参数。如果您的反应包含了一个特殊的状态码和文档，请确保在使用 *PrintWriter* 实际返回任何内容之前调用 setStatus。
2	**public void sendRedirect(String url)** 该方法生成一个 302 响应，连同一个带有新文档 URL 的 *Location* 头。
3	**public void sendError(int code, String message)** 该方法发送一个状态码（通常为 404），连同一个在 HTML 文档内部自动格式化并发送到客户端的短消息。

6.3 代码示例

核心方法1-返回404错误

```
// 导入必需的 java 库
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

// 扩展 HttpServlet 类
public class HelloServlet extends HttpServlet {

    // 处理 GET 方法请求的方法
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
            throws ServletException, IOException
    {
        // 设置错误代码和原因
        response.sendError(404, "Not Found!!!" );
    }
    // 处理 POST 方法请求的方法
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```



核心方法2-重定向

```
// 导入必需的 java 库
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

// 扩展 HttpServlet 类
public class HelloServlet extends HttpServlet {

    // 处理 GET 方法请求的方法
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
            throws ServletException, IOException
    {
        // 设置错误代码和原因
        response.sendRedirect("https://www.baidu.com");
    }
    // 处理 POST 方法请求的方法
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```

7. Servlet Cookies

7.1 cookie概念回顾

> - 定义: Cookies 是存储在客户端计算机上的文本文件，并保留了用户的各种跟踪信息
> - 作用: 会话保持，如完成用户的登录与状态保持

7.2 cookie工作原理

- 客户端向服务区发起登录请求
- 服务器脚本向浏览器发送一组 Cookies。例如：姓名、年龄或识别号码等。
- 浏览器将这些信息存储在本地计算机上，以备将来使用。
- 当下一次浏览器向 Web 服务器发送任何请求时，浏览器会把这些 Cookies 信息发送到服务器，服务器将使用这些信息来识别用户。

7.3 cookie构成

Cookies 通常设置在 HTTP 头信息中。设置 Cookie 的http请求，会向 Servlet 会发送如下的头信息:

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
                path=/; domain=bit.com
Connection: close
Content-Type: text/html
```

- Set-Cookie 头包含了一个名称值对、一个 GMT 日期、一个路径和一个域。名称和值会被 URL 编码。
- expires 字段是一个指令，告诉浏览器在给定的时间和日期之后过期("忘记")该 Cookie。
- 如果浏览器被配置为存储 Cookies，它将会保留此信息直到到期日期。

如果用户的浏览器指向任何匹配该 Cookie 的路径和域的页面，它会重新发送 Cookie 到服务器。浏览器的头信息可能如下所示：

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

Servlet 就能够通过请求方法 *request.getCookies()* 访问 Cookie，该方法将返回一个 *Cookie* 对象的数组。

7.4 Servlet 操作cookie方法

以下是在 Servlet 中操作 Cookies 时可使用的有用的方法列表。

序号	方法 & 描述
public void setDomain(String pattern)	该方法设置 cookie 适用的域，例如 w3cschool.cn。
public String getDomain()	该方法获取 cookie 适用的域，例如 w3cschool.cn。
public void setMaxAge(int expiry)	该方法设置 cookie 过期的时间（以秒为单位）。如果不这样设置，cookie 只会在当前 session 会话中持续有效。
public int getMaxAge()	该方法返回 cookie 的最大生存周期（以秒为单位），默认情况下，-1 表示 cookie 将持续下去，直到浏览器关闭。
public String getName()	该方法返回 cookie 的名称。名称在创建后不能改变。
public void setValue(String newValue)	该方法设置与 cookie 关联的值。
public String getValue()	该方法获取与 cookie 关联的值。
public void setPath(String uri)	该方法设置 cookie 适用的路径。如果您不指定路径，与当前页面相同目录下的（包括子目录下的）所有 URL 都会返回 cookie。
public String getPath()	该方法获取 cookie 适用的路径。

7.5 代码示例

代码示例1-提交表单，设置cookie

index.html内容：

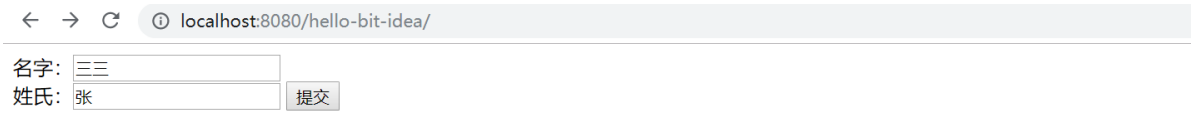
```
<!DOCTYPE html>
<html lang="en">
<head>
     <!--设置编码格式-->
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>s
    <form action="hello-servlet" method="GET">
    名字：<input type="text" name="first_name">
    <br />
    姓氏：<input type="text" name="last_name" />
    <input type="submit" value="提交" />
    </form>
</body>
</html>
```

Servlet 代码

```java
// 导入必需的 java 库
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

// 扩展 HttpServlet 类
public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
            throws ServletException, IOException
    {
        // 为名字和姓氏创建 Cookies
        Cookie firstName = new Cookie("first_name",
                request.getParameter("first_name"));
        Cookie lastName = new Cookie("last_name",
                request.getParameter("last_name"));

        // 为两个 Cookies 设置过期日期为 24 小时后
        firstName.setMaxAge(60*60*24);
        lastName.setMaxAge(60*60*24);

        // 在响应头中添加两个 Cookies
        response.addCookie( firstName );
        response.addCookie( lastName );

        // 设置响应内容类型
        response.setContentType("text/html");
        // 设置响应的编码格式
        response.setCharacterEncoding("UTF-8");

        PrintWriter out = response.getWriter();
        String title = "设置 Cookies 实例";
        String docType =
                "<!doctype html public>\n";
        out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                "<ul>\n" +
                "  <li><b>名字</b>: "
                + request.getParameter("first_name") + "\n" +
                "  <li><b>姓氏</b>: "
                + request.getParameter("last_name") + "\n" +
                "</ul>\n" +
                "</body></html>");
    }
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException
    {
        doGet(request,response);
    }
}
```

查看结果

- 获取静态网页，提交



- 设置成功

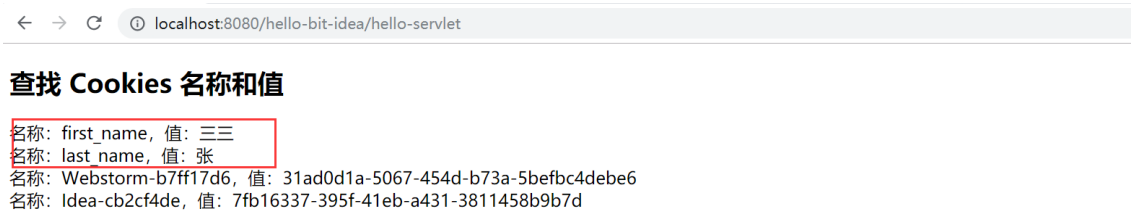
- 查看cookie

- 获取写入的cookie

  代码：

```
// 导入必需的 java 库
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

// 扩展 HttpServlet 类
public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
            throws ServletException, IOException
    {
        Cookie cookie = null;
        Cookie[] cookies = null;
        // 获取与该域相关的 Cookies 的数组
```

```
        cookies = request.getCookies();

        // 设置响应内容类型
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");

        PrintWriter out = response.getWriter();
        String title = "Reading Cookies Example";
        String docType =
                "<!doctype html>\n";
        out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body>\n" );
        if( cookies != null ){
            out.println("<h2>查找 Cookies 名称和值</h2>");
            for (int i = 0; i < cookies.length; i++){
                cookie = cookies[i];
                out.print("名称: " + cookie.getName( ) + ", ");
                out.print("值: " + cookie.getValue( )+" <br/>");
            }
        }
        out.println("</body>");
        out.println("</html>");
    }
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException
    {
        doGet(request,response);
    }
}
```



代码示例2-删除cookie，代码参考

删除 Cookies 是非常简单的。如果想删除一个 cookie，那么需要按照以下三个步骤进行：

- 读取一个现有的 cookie，并把它存储在 Cookie 对象中。
- 使用 **setMaxAge()** 方法设置 cookie 的年龄为零，来删除现有的 cookie。
- 把这个 cookie 添加到响应头。

```
        Cookie cookie = null;
        Cookie[] cookies = null;
        // 获取与该域相关的 Cookies 的数组
        cookies = request.getCookies();
        // 设置响应内容类型
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");

        PrintWriter out = response.getWriter();
        String title = "Delete Cookies Example";
```

```
        String docType =
                "<!doctype html>\n";
        out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body>\n" );
        if( cookies != null ){
            out.println("<h2>Cookies 名称和值</h2>");
            for (int i = 0; i < cookies.length; i++){
                cookie = cookies[i];
                if((cookie.getName( )).compareTo("first_name") == 0 ) {
                    cookie.setMaxAge(0); //将过期时间设置为0，删除指定cookie
                    response.addCookie(cookie);
                    out.print("已删除的 cookie: " +
                            cookie.getName( ) + "<br/>");
                }
                out.print("名称: " + cookie.getName( ) + ", ");
                out.print("值: " + cookie.getValue( )+" <br/>");
            }
        }
        out.println("</body>");
        out.println("</html>");
```

8. Servlet Session

8.1 session概念回顾

> - 定义: session 是存储在服务器上的文本文件，并保留了用户的各种跟踪信息
> - 作用: 会话保持，如完成用户的登录与状态保持，因为在服务器端，所以相对安全一些

8.2 Servlet 操作session方法

HttpSession 对象

- Servlet 还提供了 HttpSession 接口，该接口提供了一种跨多个页面请求或访问网站时识别用户以及存储有关用户信息的方式。
- Servlet 容器使用这个接口来创建一个 HTTP 客户端和 HTTP 服务器之间的 session 会话。会话持续一个指定的时间段，跨多个连接或页面请求。
- 我们可以通过调用 HttpServletRequest 的公共方法 **getSession()** 来获取 HttpSession 对象，如下所示:

  ```
  HttpSession session = request.getSession();
  ```

  需要在向客户端发送任何文档内容之前调用 *request.getSession()*

下面总结了 HttpSession 对象中可用的几个重要的方法:

方法	描述
public Object getAttribute(String name)	该方法返回在该 session 会话中具有指定名称的对象，如果没有指定名称的对象，则返回 null。
public Enumeration getAttributeNames()	该方法返回 String 对象的枚举，String 对象包含所有绑定到该 session 会话的对象的名称。
public long getCreationTime()	该方法返回该 session 会话被创建的时间，自格林尼治标准时间 1970 年 1 月 1 日午夜算起，以毫秒为单位。
public String getId()	该方法返回一个包含分配给该 session 会话的唯一标识符的字符串。
public long getLastAccessedTime()	该方法返回客户端最后一次发送与该 session 会话相关的请求的时间自格林尼治标准时间 1970 年 1 月 1 日午夜算起，以毫秒为单位。
public int getMaxInactiveInterval()	该方法返回 Servlet 容器在客户端访问时保持 session 会话打开的最大时间间隔，以秒为单位。
public void invalidate()	该方法指示该 session 会话无效，并解除绑定到它上面的任何对象。
public boolean isNew()	如果客户端还不知道该 session 会话，或者如果客户选择不参入该 session 会话，则该方法返回 true。
public void removeAttribute(String name)	该方法将从该 session 会话移除指定名称的对象。
public void setAttribute(String name, Object value)	该方法使用指定的名称绑定一个对象到该 session 会话。
public void setMaxInactiveInterval(int interval)	该方法在 Servlet 容器指示该 session 会话无效之前，指定客户端请求之间的时间，以秒为单位。

8.3 代码示例

本实例说明了如何使用 HttpSession 对象获取 session 会话创建时间和最后访问时间。如果不存在 session 会话，我们将通过请求创建一个新的 session 会话

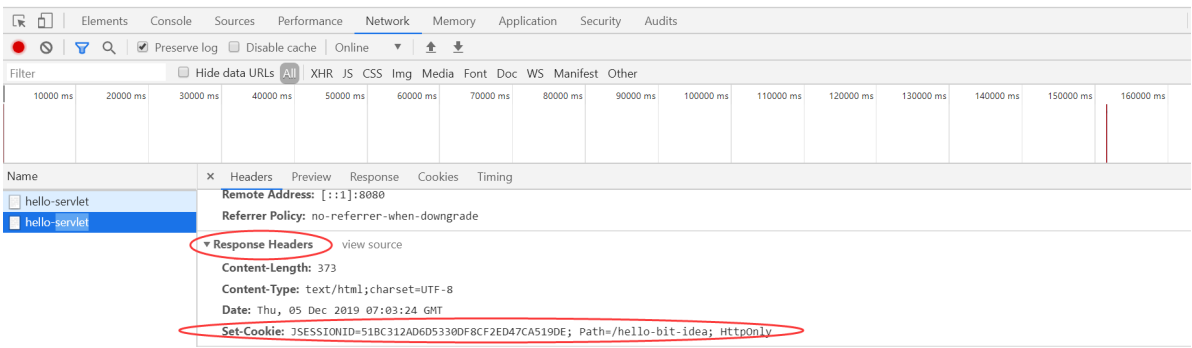
```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

public class HelloServlet extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
```

```
        // 如果不存在 session 会话，则创建一个 session 对象
        HttpSession session = request.getSession(true);
        // 获取 session 创建时间
        Date createTime = new Date(session.getCreationTime());
        // 获取该网页的最后一次访问时间
        Date lastAccessTime = new Date(session.getLastAccessedTime());

        String title = "欢迎回来";

        String visitCountKey = new String("visitCount");
        Integer visitCount = new Integer(0);

        String userIDKey = new String("userID");
        String userID = new String("abcd");

        // 检查网页上是否是新的访问者
        if(session.isNew()){
            title = "欢迎来到我的网站";
            session.setAttribute(userIDKey, userID);
        }
        else{
            visitCount = (Integer) session.getAttribute(visitCountKey);
            visitCount++;
            userID = (String)session.getAttribute(userIDKey);
        }
        session.setAttribute(visitCountKey,visitCount);

        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        PrintWriter out = response.getWriter();

        String docType = "<!doctype html>\n";
        out.println(docType + "<html>\n" +
                "<head><title>" +title+ "</title></head>\n" +
                "<body>\n" +
                "<h1>" + title + "</h1>\n"+
                "<h2> Session 信息 </h2>" +
                "<p>" +
                "ID: " + session.getId() + "<hr/><br/>"+
                "Create Time: " + createTime + "<hr/><br/>"+
                "Time of Last Access: " + lastAccessTime + "<hr/><br/>"+
                "User ID: " + userID + "<hr/><br/>"+
                "Number of visits: " + visitCount + "<hr/><br/>"+
                "</p>"+ "</body>" +
                "</html>");
    }
}
```

- **Step 1：发起请求，查看结果**

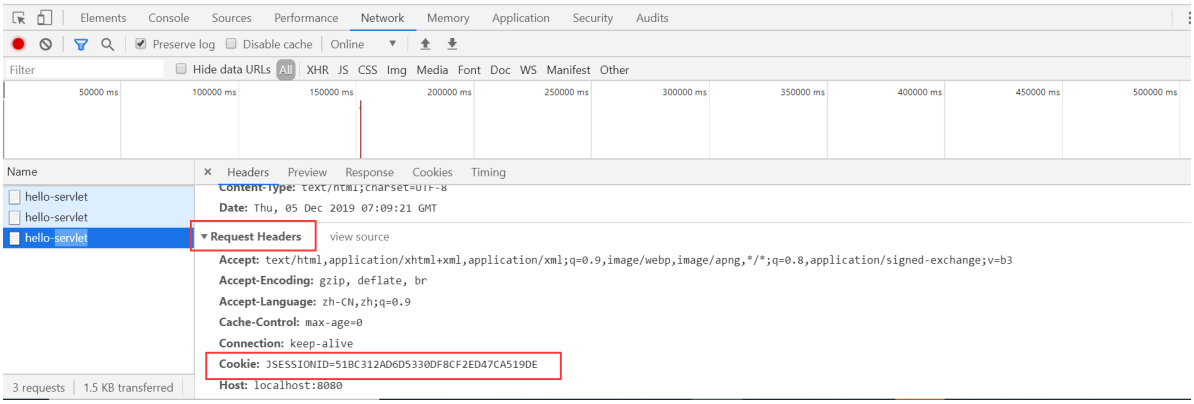
- **Step 2: 分析请求结果**

得到响应时，服务器会自动向浏览器设置cookie，将session id设置进浏览器[**这里ID在实验的时候，截图有点问题，后面具体问题，具体分析**]



- **Step 3: 多次刷新**

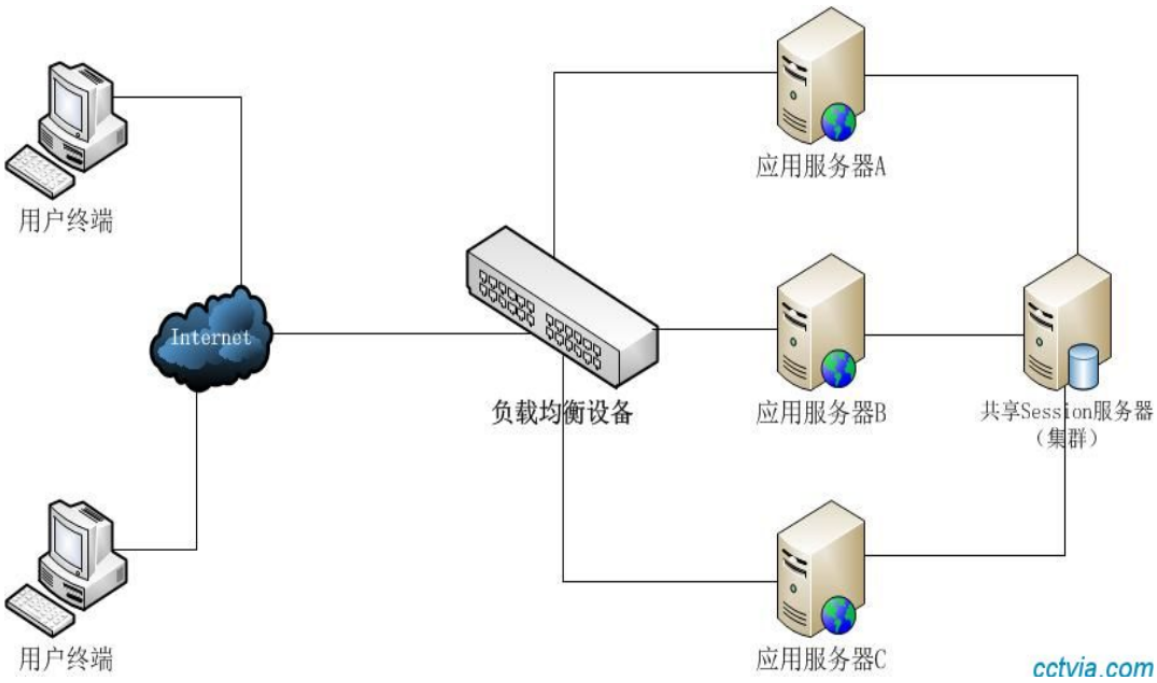
每次请求，浏览器request都会自动携带session id，以供服务器器来进行标识



8.4 session持久化

一般情况下，session是直接存储在指定服务器的内存中的，一般使用够了，但是在集群场景当中，可能需要通过session共享，来保证用户在不同的服务器上都可以得到认证。所以我们一般可以将用户的session信息统一保存在后端的数据库服务器中[比如，mysql/redis/memcache等]，从而达到不同服务器间session的共享。

我们可以简单理解：将session保存在服务器数据库或者文件中的行为称之为session持久化



9. Servlet文件上传理解

HttpServletRequest 对文件上传的支持

- 此前，Servlet 本身没有对文件上传提供直接的支持，一般需要使用第三方框架来实现，实现起来很麻烦
- 不过，Servlet 3.0 之后提供了这个功能，而且使用也非常简单。为此，HttpServletRequest 提供了两个方法用于从请求中解析出上传的文件：

  ```
  Part getPart(String name)   //获取请求中给定 name 的文件
  Collection<Part> getParts() //获取所有的文件
  ```

  其中，每一个文件用一个 `javax.servlet.http.Part` 对象来表示。该接口提供了处理文件的简易方法，比如 write()、delete() 等。至此，结合 HttpServletRequest 和 Part 来保存上传的文件变得非常简单.

  ```
  Part img = request.getPart("img");
  img.write("E:\\apache-tomcat-8.5.47\\webapps\\file_dir\\img.jpg");
  ```

  另外，可以配合前面提到的 @MultipartConfig 注解来对上传操作进行一些自定义的配置，比如限制上传文件的大小，以及保存文件的路径等。其用法非常简单，故不在此赘述了。

  需要注意的是，如果请求的 MIME 类型不是 multipart/form-data，则不能使用上面的两个方法，否则将抛异常。

前端网页

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>文件上传表单</title>
</head>
<body>
<h3>文件上传：</h3>
```

```
请选择要上传的文件：<br />
<form action="upload" method="post" enctype="multipart/form-data">
    <input type="file" name="file" size="50" />
    <br />
    <input type="submit" value="上传文件" />
</form>
</body>
</html>
```

UploadServlet

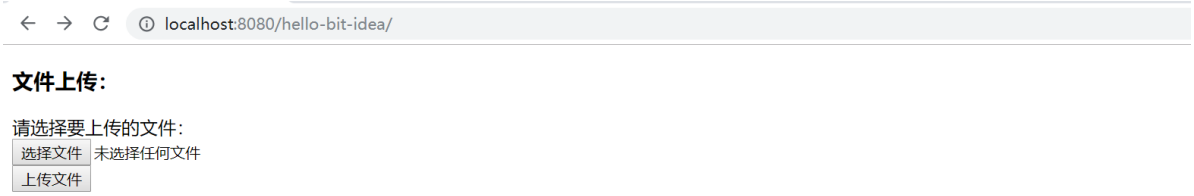
```
import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Part;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

@MultipartConfig
public class UploadServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException,IOException{
        //具体上传上来的文件放在什么地方，由自己决定
        File path = new File("E:\\apache-tomcat-8.5.47\\webapps\\file_dir");
        //获取文件，文件在html中的name是“file”
        Part img = request.getPart("file");
        //制作文件全路径
        String filePath = path.getPath()+File.separator +
img.getSubmittedFileName();
        //获取成功之后，写入指定路径
        img.write(filePath);
        //显示到标准输出
        System.out.println("file Upload: " + filePath);
        //同样的信息，显示给用户浏览器
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("file Upload: " + filePath);
    }
    public void doGet(HttpRequest request, HttpResponse response)
            throws ServletException,IOException {
        //上传文件，不能用GET方法
        System.out.println("上传文件只能用POST方法!");
    }
}
```

web.xml新增

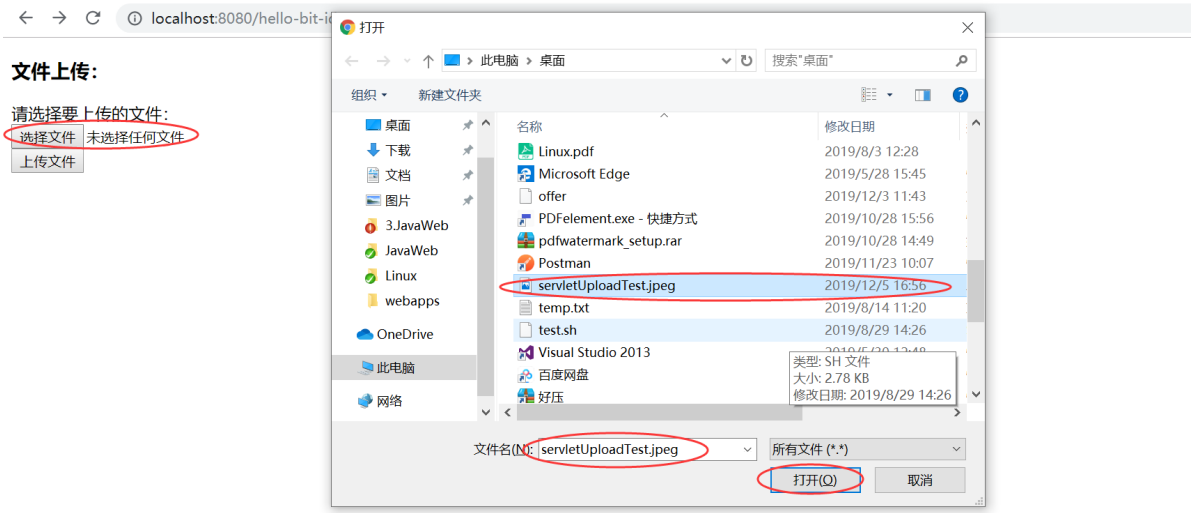
```
    <servlet>
        <servlet-name>upload</servlet-name>
        <servlet-class>UploadServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>upload</servlet-name>
        <url-pattern>/upload</url-pattern>
    </servlet-mapping>
```

结果演示：

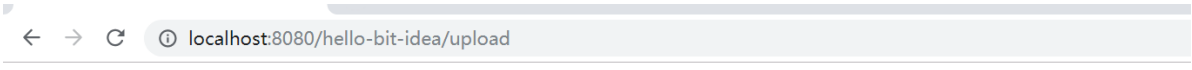
1. 获取上传页面



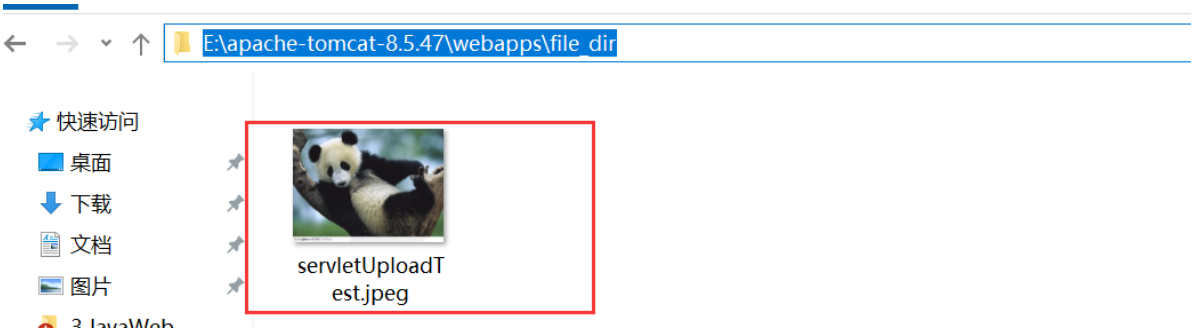
2. 添加文件到浏览器



3. 上传文件，成功



4. 查看服务器端



10.其他内容<大家酌情自行了解>:

> - Servlet 异常处理
> - Servlet过滤器
> - ServletContext