Servlet学习笔记

1、环境搭建

1.1、Maven下载配置

1. **下载地址：**https://maven.apache.org/download.cgi

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.6.3-bin.tar.gz	apache-maven-3.6.3-bin.tar.gz.sha512	apache-maven-3.6.3-bin.tar.gz.asc
Binary zip archive	apache-maven-3.6.3-bin.zip	apache-maven-3.6.3-bin.zip.sha512	apache-maven-3.6.3-bin.zip.asc
Source tar.gz archive	apache-maven-3.6.3-src.tar.gz	apache-maven-3.6.3-src.tar.gz.sha512	apache-maven-3.6.3-src.tar.gz.asc
Source zip archive	apache-maven-3.6.3-src.zip	apache-maven-3.6.3-src.zip.sha512	apache-maven-3.6.3-src.zip.asc

2. **配置环境变量**

   M2_HOME　Maven**目录下的**bin**目录，例：**C:\Program Files\Environment\apache-maven-3.6.3\bin

   MAVEN_HOME　Maven**的目录，例：**C:\Program Files\Environment\apache-maven-3.6.3

   在系统Path**中配置**　%MAVEN_HOME%\bin

3. **查看是否安装成功**

   mvn -v

```
PS C:\Users\wulele> mvn -v
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: C:\Program Files\Environment\apache-maven-3.6.3\bin\..
Java version: 15.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-15.0.1
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

4. **配置阿里云镜像**

   打开conf**文件夹下的**settings.xml**文件，找到** `<mirrors></mirror>` **，在其中插入代码**

```
<mirror>
            <id>nexus-aliyun</id>
            <mirrorOf>*,!jeecg,!jeecg-snapshots</mirrorOf>
            <name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url
>
        </mirror>
```

```xml
<mirrors>
  <!-- mirror
   | Specifies a repository mirror site to use instead of a given repository. The repository that
   | this mirror serves has an ID that matches the mirrorOf element of this mirror. IDs are used
   | for inheritance and direct lookup purposes, and must be unique across the set of mirrors.
   |
  <mirror>
    <id>mirrorId</id>
    <mirrorOf>repositoryId</mirrorOf>
    <name>Human Readable Name for this Mirror.</name>
    <url>http://my.repository.com/repo/path</url>
  </mirror>
   -->
    <mirror>
        <id>nexus-aliyun</id>
        <mirrorOf>*,!jeecg,!jeecg-snapshots</mirrorOf>
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
     </mirror>
</mirrors>
```

5. **本地仓库配置**

   找到`<localRepository></localRepository>`，默认本地仓库在user/.m2/repository**下，可以自己新建文件夹设置为本地仓库，例：**`<localRepository>C:\Program Files\Environment\apache-maven-3.6.3\maven-repo</localRepository>`

```xml
<!-- localRepository
  | The path to the local repository maven will use to store artifacts.
  |
  | Default: ${user.home}/.m2/repository
<localRepository>/path/to/local/repo</localRepository>
-->
<localRepository>C:\Program Files\Environment\apache-maven-3.6.3\maven-repo</localRepository>
```

1.2、IDEA中Maven的一些操作

1. pom.xml**文件**

   创建一个Maven**的示例**webapps**项目后**

pom.xml (maven-web)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--Maven版本和头文件-->
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.wll</groupId>
  <artifactId>maven-web</artifactId>
  <version>1.0-SNAPSHOT</version>
  <!--项目的打包方式-->
  <packaging>war</packaging>

  <name>maven-web Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <!--配置-->
  <properties>
    <!--项目默认构建编码-->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!--编码版本-->
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <!--项目依赖-->
  <dependencies...>

  <!--项目构建用的东西-->
  <build...>
</project>
```

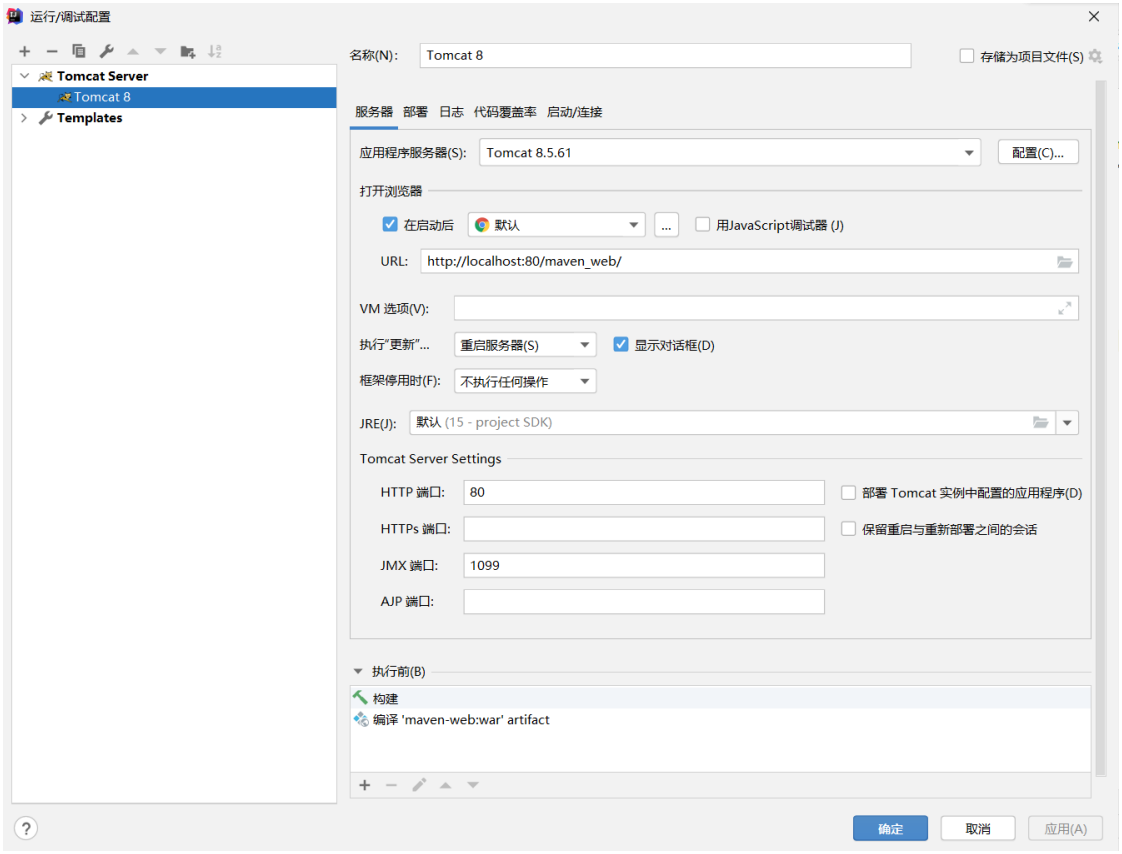
2. **导入**jar**包**

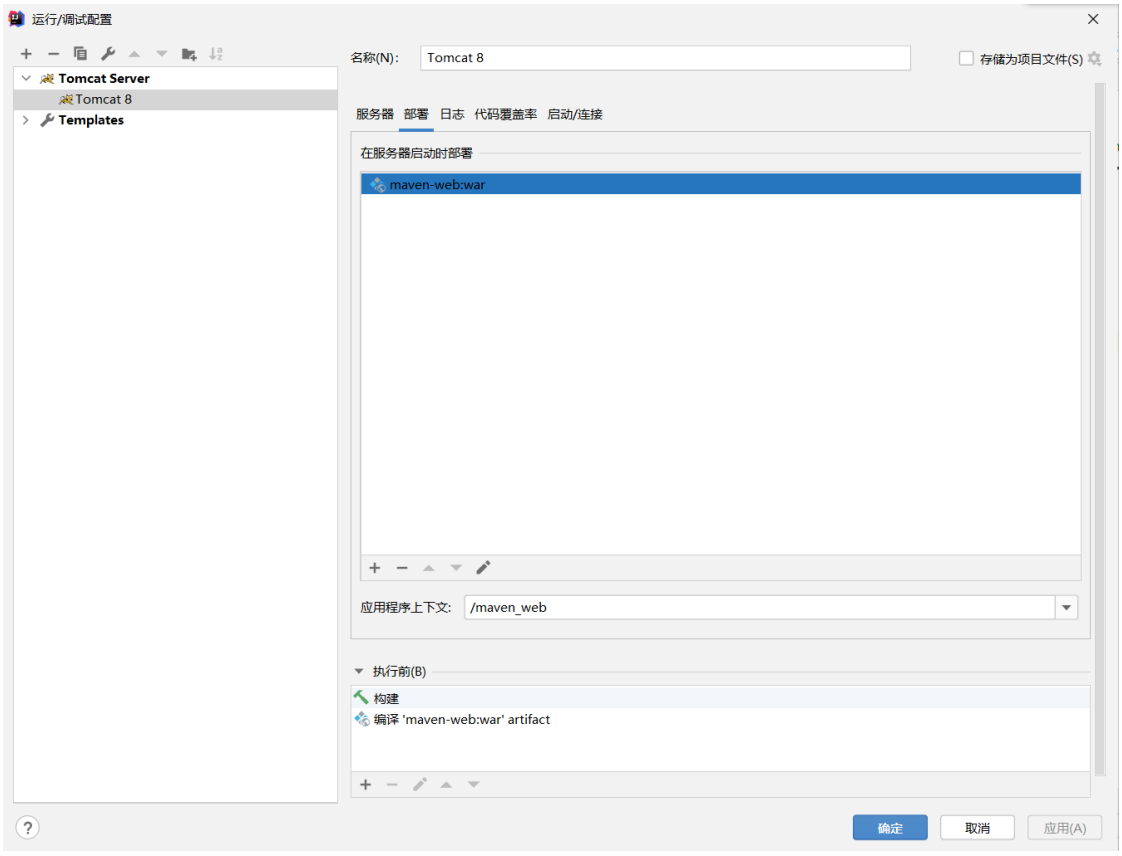
jar包地址：https://mvnrepository.com/

例：需要Servlet的jar包，搜索Servlet，选择自己所需要的jar包，复制这一段内容，在pom.xml中进行配置，Maven会自动配置所需jar包及其所需依赖jar包

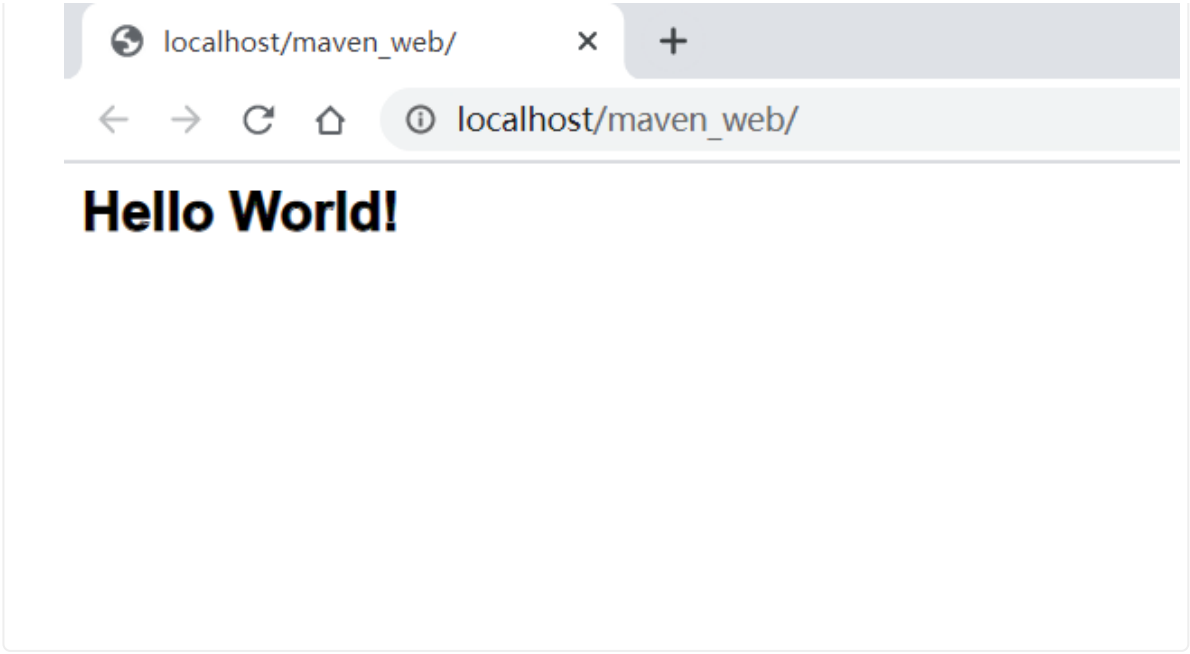

```xml
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
  </dependency>
  <!-- https://mvnrepository.com/artifact/javax.servlet.jsp/javax.servlet.jsp-api -->
  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.3</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

3. **配置**Tomcat


运行/调试配置
Tomcat Server
Tomcat 8
Templates
名称(N): Tomcat 8
存储为项目文件(S)
服务器 部署 日志 代码覆盖率 启动/连接
应用程序服务器(S): Tomcat 8.5.61
配置(C)...
打开浏览器
在启动后
默认
用JavaScript调试器 (J)
URL: http://localhost:80/maven_web/
VM 选项(V):
执行"更新"...
重启服务器(S)
显示对话框(D)
框架停用时(F):
不执行任何操作
JRE(J): 默认 (15 - project SDK)
Tomcat Server Settings
HTTP 端口: 80
部署 Tomcat 实例中配置的应用程序(D)
HTTPs 端口:
保留重启与重新部署之间的会话
JMX 端口: 1099
AJP 端口:
执行前(B)
构建
编译 'maven-web:war' artifact
确定
取消
应用(A)

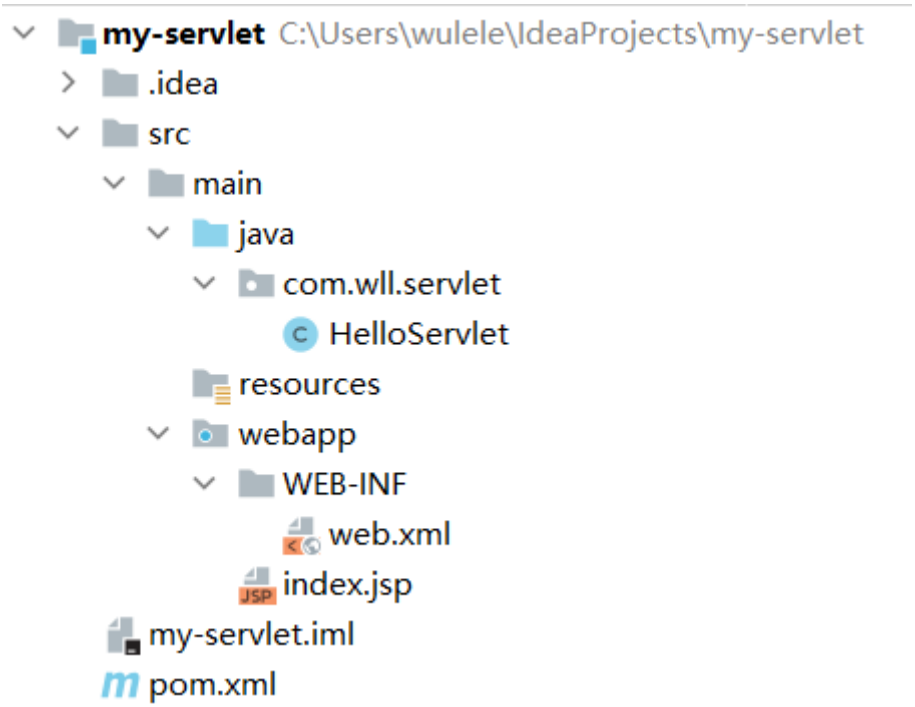

运行/调试配置
Tomcat Server
Tomcat 8
Templates
名称(N): Tomcat 8
存储为项目文件(S)
服务器 部署 日志 代码覆盖率 启动/连接
在服务器启动时部署
maven-web:war
应用程序上下文: /maven_web
执行前(B)
构建
编译 'maven-web:war' artifact
确定
取消
应用(A)

4. 运行测试



2、HelloServlet

2.1、创建项目



2.2、编写pom.xml

文件中导入Servlet的jar包即可

2.3、编写web.xml

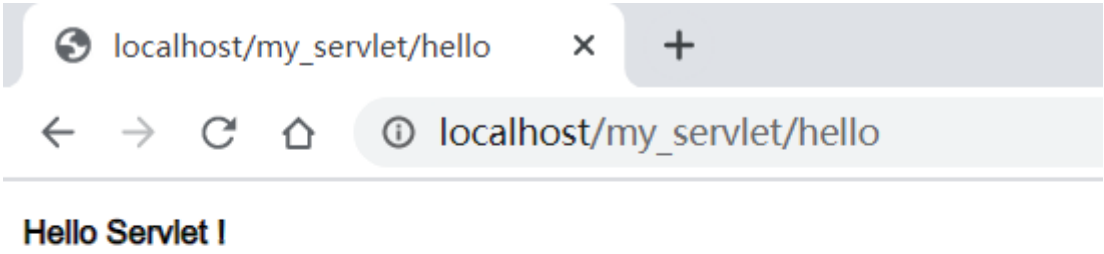
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
```

```
        version="4.0"
        metadata-complete="true">
    <!--我们写的是JAVA程序，但是要通过浏览器访问，二浏览器需要连接web服务器，
    所以我们需要在web服务中注册我们写的Servlet，还需给它一个浏览器能够访问的路径-->
    <!--注册Servlet-->
    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>com.wll.servlet.HelloServlet</servlet-class>
    </servlet>
    <!--Servlet的请求路径-->
    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

2.4、源程序

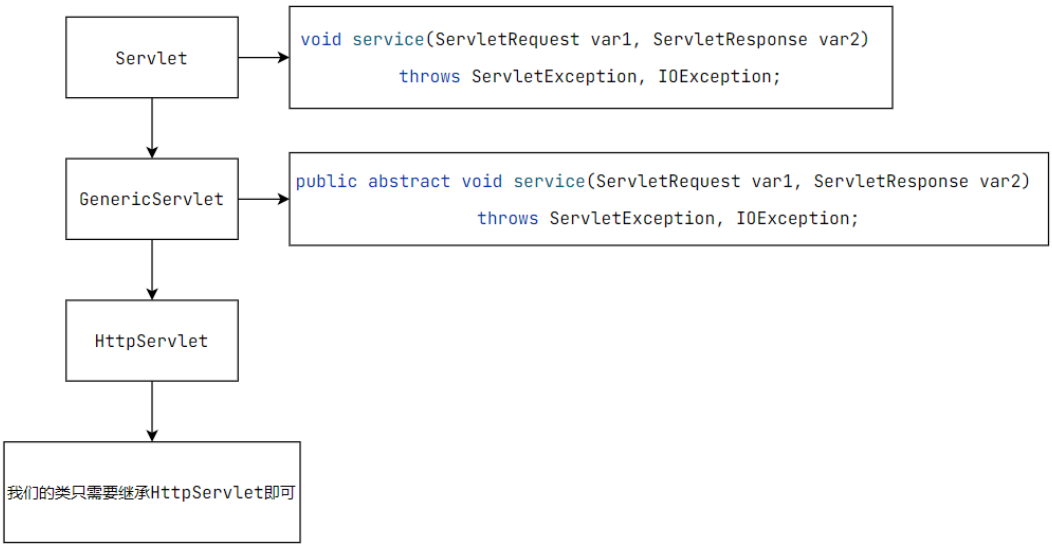
```
package com.wll.servlet;

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 {
    //get或post只是请求的不同方式，可以互相调用，业务逻辑一样
    @Override
    protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
        PrintWriter writer = resp.getWriter();
        writer.print("Hello Servlet !");
    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
        doGet(req, resp);
    }
}
```

2.5、测试运行



2.6、总结

1.



2. mapping**问题**

- **一个**Servlet**可以指定一个映射路径**

```
<servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

- **一个**Servlet**可以指定多个映射路径**

```
<servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
</servlet-mapping>
<servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello1</url-pattern>
</servlet-mapping>
<servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello2</url-pattern>
</servlet-mapping>
```

- **一个Servlet可以指定通用映射路径**

```
<servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
<!--默认请求路径-->
<servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/*</url-pattern>
</servlet-mapping>
```

- **一个Servlet可以指定一些后缀或前缀**

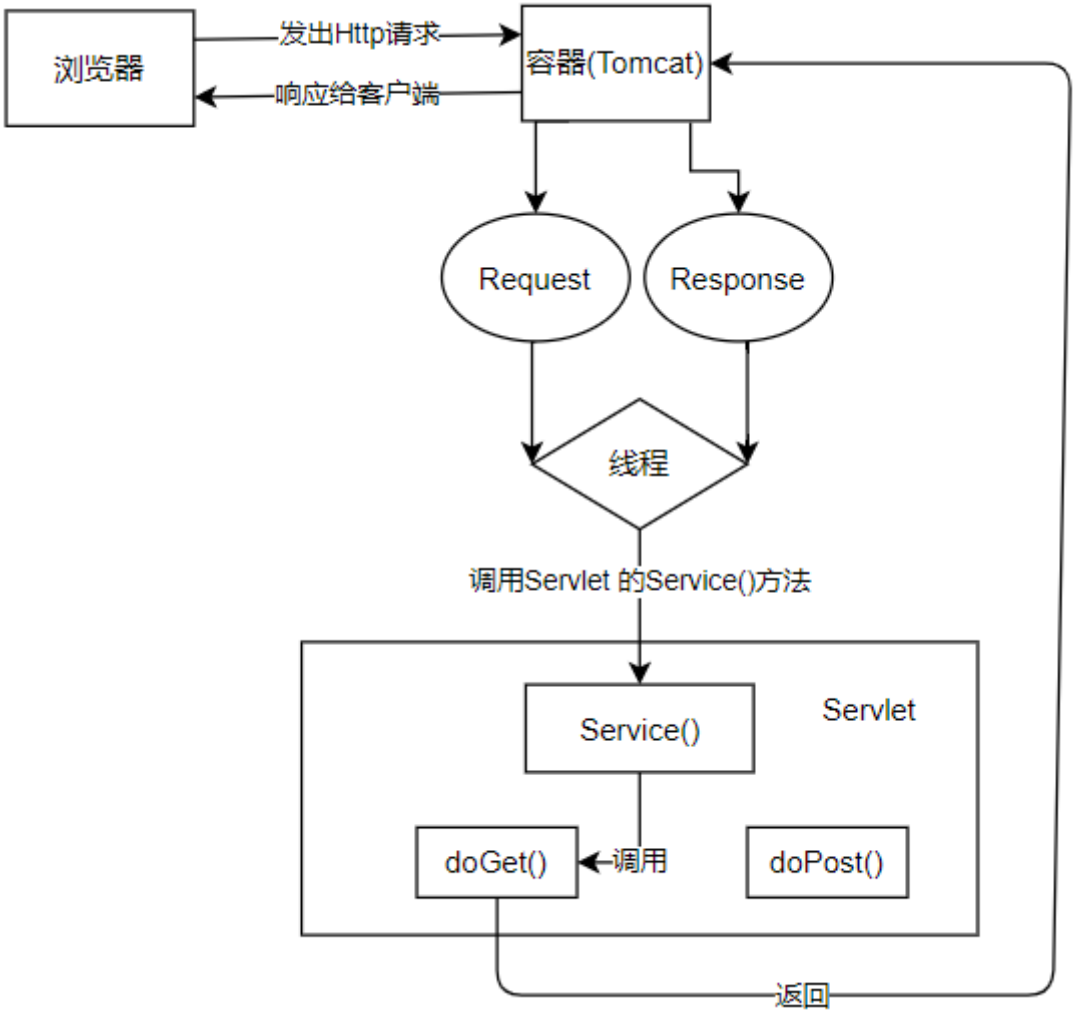
```
<!--可以自定义后缀实现请求映射，*前面不能加项目映射路径-->
<servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>*.wll</url-pattern>
</servlet-mapping>
```

- **优先级**

  指定了固有的映射路径优先级最高（/hello），如果找不到就会走默认的处理请求（/*）

3、Servlet

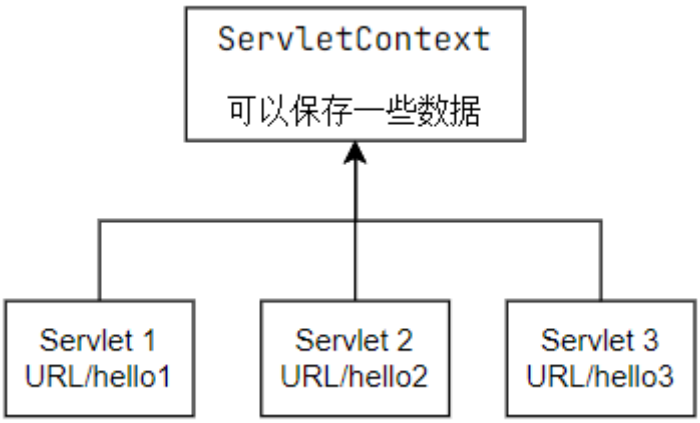
3.1、Servlet运行原理图



3.2、ServletContext

- web容器在启动的时候，会为每个web程序创建一个对应的ServletContext对象，它代表了当前的web应用
  - 共享数据

    我们在这个Servlet中保存的数据，可以在另外一个Servlet中获得，不要存太多，服务器会炸




```
public class HelloServlet extends HttpServlet {
    @Override
```

```
    protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
{
    /*
        this.getInitParameter()    初始化参数
        this.getServletConfig()    Servlet配置
        this.getServletContext()   Servlet上下文
     */
        ServletContext servletContext =
this.getServletContext();
        String name = "wll";     //数据
        servletContext.setAttribute("name",name);    //保存数据到
ServletContext 键: name 值: name

        System.out.println("hello");
    }
}

public class GetServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
{
        ServletContext servletContext =
this.getServletContext();
        String name = (String)
servletContext.getAttribute("name");
        System.out.println("名字: "+name);
    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
{
        doGet(req, resp);
    }
}
```

```
<servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>com.wll.servlet.HelloServlet</servlet-
class>
</servlet>

<servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

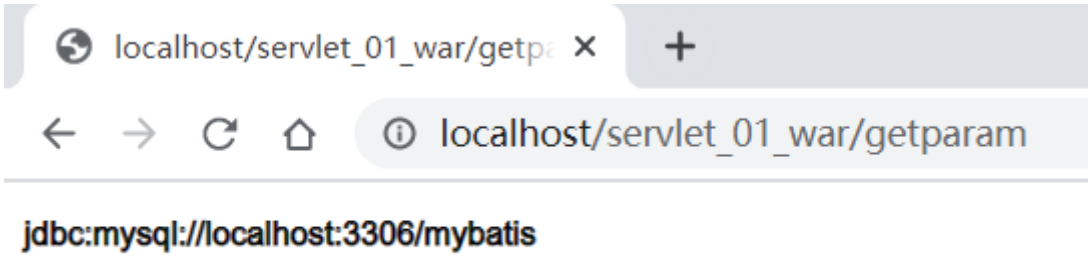
```
<servlet>
    <servlet-name>getcontext</servlet-name>
    <servlet-class>com.wll.servlet.GetServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>getcontext</servlet-name>
    <url-pattern>/getcontext</url-pattern>
</servlet-mapping>
```

hello 访问hello页面
名字：wll 访问getcontext页面

- **获取初始化参数** getInitParameter

```
    <!—配置web应用的一些初始化参数—>
    <context-param>
        <param-name>jdbc</param-name>
        <param-
value>jdbc:mysql://localhost:3306/mybatis</param-value>
    </context-param>
```

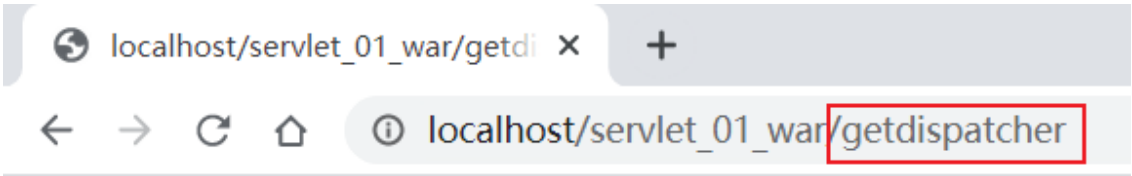
```
    protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
{
        ServletContext servletContext =
this.getServletContext();
        String jdbc = servletContext.getInitParameter("jdbc");
        resp.getWriter().print(jdbc);
    }
```



- **请求转发**

```
    protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
{
        ServletContext servletContext =
this.getServletContext();
//        RequestDispatcher getparam =
servletContext.getRequestDispatcher("/getparam");    //获取请求转
发路径
//        getparam.forward(req,resp); //调用forward实现请求转发

 servletContext.getRequestDispatcher("/getparam").forward(req,
resp);
    }
```

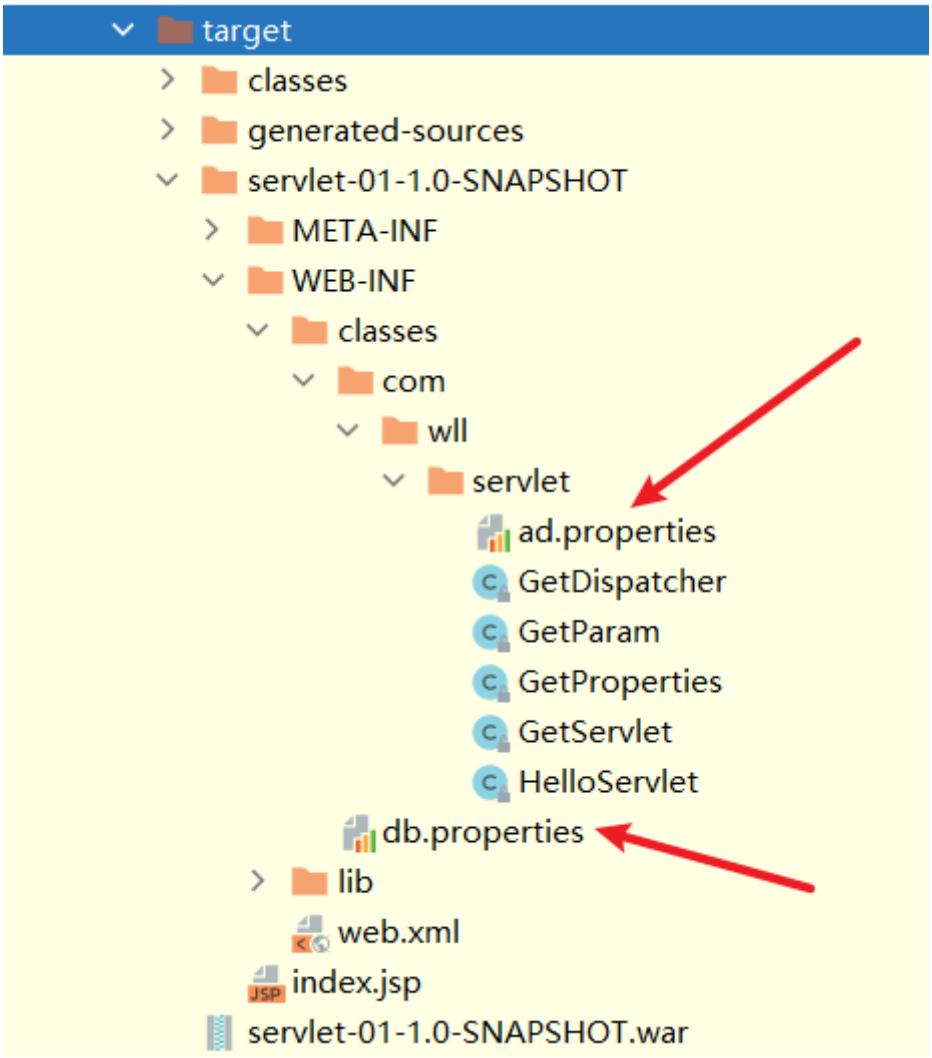
```
    <servlet>
        <servlet-name>getdispatcher</servlet-name>
        <servlet-class>com.wll.servlet.GetDispatcher</servlet-
class>
    </servlet>
    <servlet-mapping>
        <servlet-name>getdispatcher</servlet-name>
        <url-pattern>/getdispatcher</url-pattern>
    </servlet-mapping>
```



jdbc:mysql://localhost:3306/mybatis

- Properties

  在java目录下新建properties文件，在resources目录下新建properties文件，都被打包到了同一个路径下：classes（classpath）



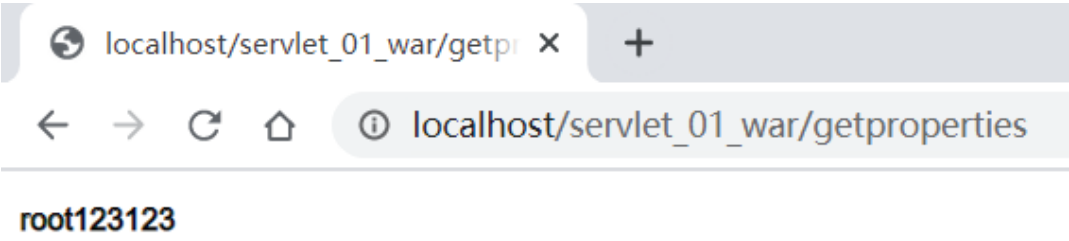

```
user=root
password=123123
```

```
   protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        InputStream resourceAsStream =
this.getServletContext().getResourceAsStream("/WEB-
INF/classes/db.properties");
       //InputStream resourceAsStream =
this.getServletContext().getResourceAsStream("/WEB-
INF/classes/com/wll/servlet/ad.properties");
        Properties properties = new Properties();
        properties.load(resourceAsStream);
        String user = properties.getProperty("user");
        String password =
properties.getProperty("password");
        resp.getWriter().print(user+password);
    }
```

```
    <servlet>
        <servlet-name>getproperties</servlet-name>
        <servlet-
class>com.wll.servlet.GetProperties</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>getproperties</servlet-name>
        <url-pattern>/getproperties</url-pattern>
    </servlet-mapping>
```

注意：Maven**约定大于配置，如果**properties**写在**java**目录下无法导出，所以配置**pom.xml**文件**

```
<build>
    <!--在build中配置resources，来防止我们资源导出失败的问题-->
        <resources>
            <resource>
                <directory>src/main/resources</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                </includes>
            </resource>
            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                </includes>
                <filtering>true</filtering>
            </resource>
        </resources>
</build>
```



3.3、HttpServletRequest

- **获取前端传递的参数&&请求转发**

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
```

```
<head>
    <title>Login</title>
</head>
<body>
    <form action="${pageContext.request.contextPath}/login"
method="post">
        账号: <input type="text" name="username"> <br>
        密码: <input type="password" name="password"> <br>
        爱好:
        <input type="checkbox" name="hobbies" value="吃饭">吃饭
        <input type="checkbox" name="hobbies" value="睡觉">睡觉
        <input type="checkbox" name="hobbies" value="打游戏">打游戏
        <input type="checkbox" name="hobbies" value="奥力给">奥力给
        <input type="submit">
    </form>
</body>
</html>
```

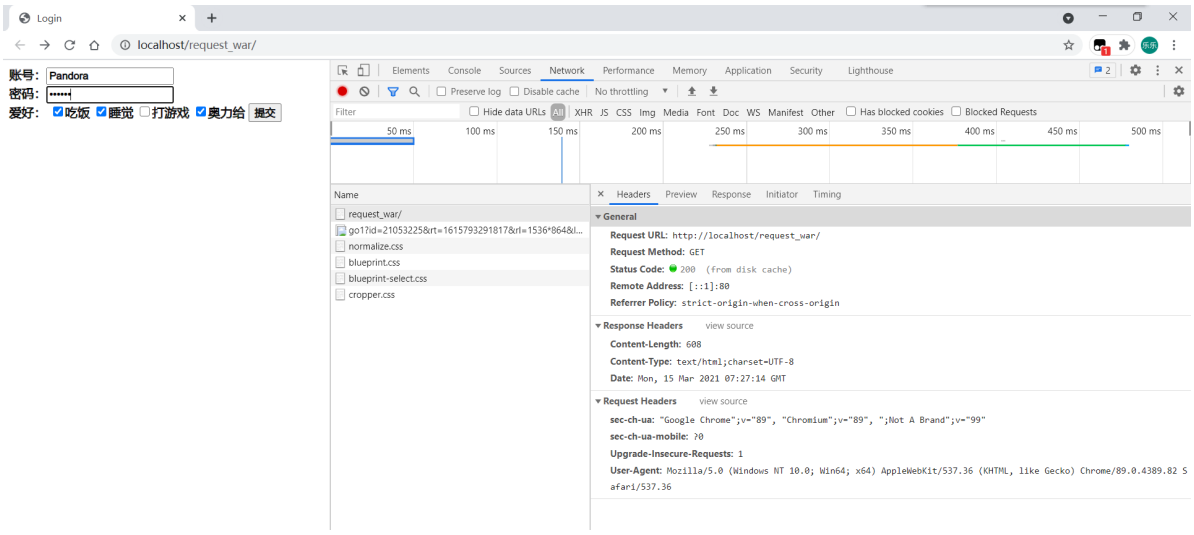
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>success</title>
</head>
<body>
    <h1>登录成功</h1>
</body>
</html>
```

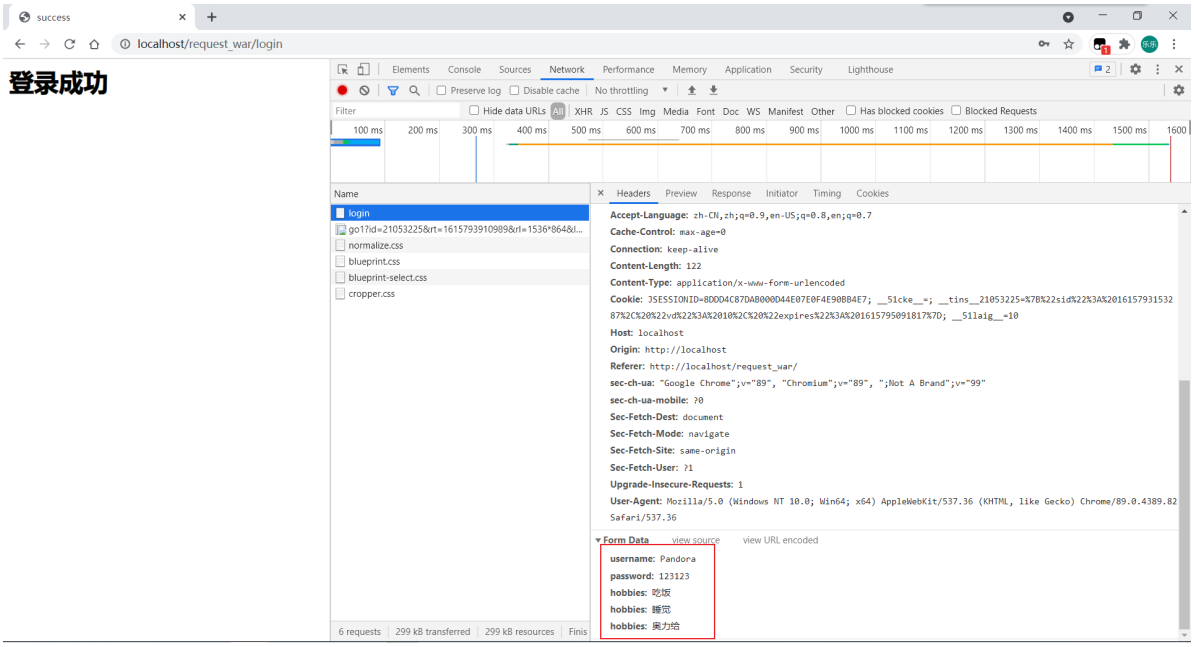
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0"
         metadata-complete="true">
    <welcome-file-list>
        <welcome-file>/login.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>request</servlet-name>
        <servlet-class>com.wll.servlet.RequestForward</servlet-
class>
    </servlet>
    <servlet-mapping>
        <servlet-name>request</servlet-name>
        <url-pattern>/login</url-pattern>
    </servlet-mapping>
```

```
</web-app>
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        req.setCharacterEncoding("utf-8");
        resp.setCharacterEncoding("utf-8");

        String username = req.getParameter("username");
        String password = req.getParameter("password");
        //获取checkbox内容
        String[] hobbies = req.getParameterValues("hobbies");
        System.out.println(username);
        System.out.println(password);
        System.out.println(Arrays.toString(hobbies));
        //请求转发  对比ServletContext请求转发

 req.getRequestDispatcher("/success.jsp").forward(req,resp);
    }
    //post调用了get，虽然submit方式是post，但是直接在get里写就行
    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
        doGet(req, resp);
    }
```

测试





Pandora
123123
[吃饭, 睡觉, 奥力给]

3.4、HttpServletResponse

1. **向浏览器输出信息**
2. **下载文件**

    1. **获取下载文件的路径**
    2. **获取下载的文件名**
    3. **设置办法让浏览器能够支持**(Content-Disposition)**下载我们需要的东西，中文文件名**URLEncoder.encode**编码，否则有可能乱码**
    4. **获取下载文件的输入流**
    5. **创建缓冲区**
    6. **获取**OutputStream**对象**
    7. **将**FileOutputStream**流写入到**buffer**缓冲区，使用**OutputStream**将缓冲区中的数据输出到客户端**
    8. **关闭流**

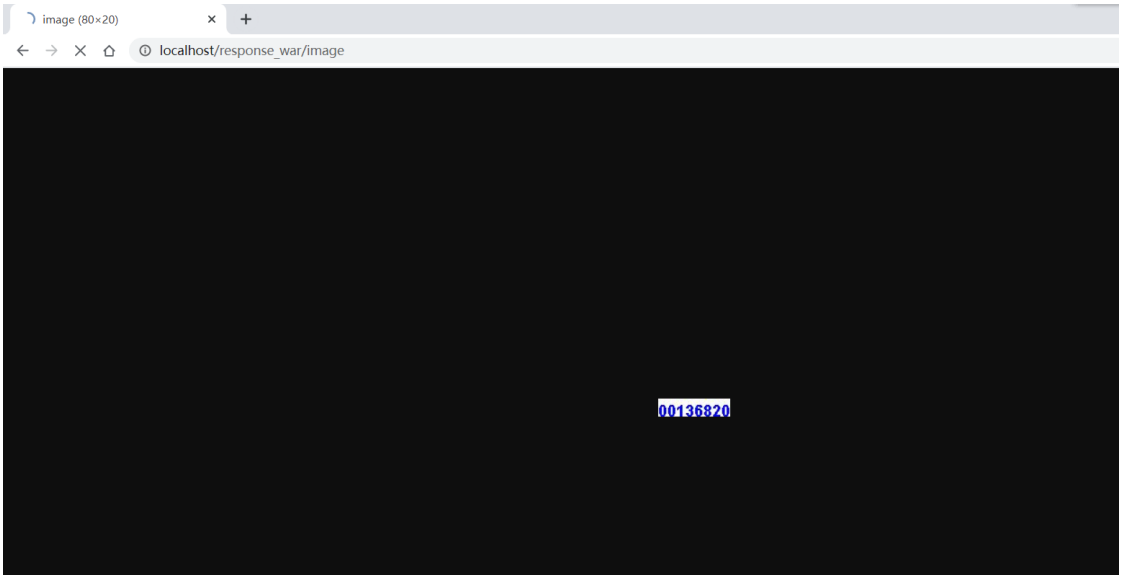
```
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws IOException {
        //1. 获取下载文件的路径
        String realPath =
this.getServletContext().getRealPath("\\WEB-
INF\\classes\\a.png");
        //2. 获取下载的文件名，subString：截取父字符串的一部分
lastIndexOf：返回此字符在字符串中最后一次出现的位置
        String fileName =
realPath.substring(realPath.lastIndexOf("\\") + 1);
```

```
        //3. 设置办法让浏览器能够支持(Content-Disposition)下载我们
需要的东西，中文文件名URLEncoder.encode编码，否则有可能乱码
        resp.setHeader("Content-
Disposition","attachment;filename="+
URLEncoder.encode(fileName,"UTF-8"));
        //4. 获取下载文件的输入流
        FileInputStream fileInputStream = new
FileInputStream(realPath);
        //5. 创建缓冲区
        int len = 0;
        byte[] buffer = new byte[1024];
        //6. 获取OutputStream对象
        ServletOutputStream outputStream =
resp.getOutputStream();
        //7. 将FileOutputStream流写入到buffer缓冲区，使用
OutputStream将缓冲区中的数据输出到客户端
        while((len=fileInputStream.read(buffer))>0){
            outputStream.write(buffer,0,len);
        }
        //8. 关闭流
        fileInputStream.close();
        outputStream.close();
    }
```

3. **验证码功能**

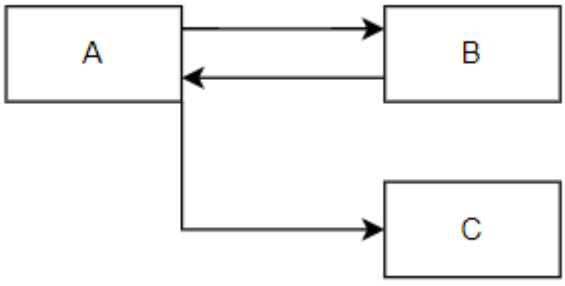
```
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        //浏览器3秒自动刷新
        resp.setHeader("refresh","3");
        //内存中创建一张图片
        BufferedImage image = new BufferedImage(80, 20,
BufferedImage.TYPE_INT_RGB);
        //得到图片
        Graphics2D graphics2D =
(Graphics2D)image.getGraphics();    //笔
        //设置图片背景颜色
        graphics2D.setColor(Color.white);
        graphics2D.fillRect(0,0,80,20);
        //给图片写数据
        graphics2D.setColor(Color.blue);
        graphics2D.setFont(new Font(null,Font.BOLD,18));
        graphics2D.drawString(fun(),0,20);

        //告诉浏览器，这个请求用图片方式打开
        resp.setContentType("Image/jpeg");
        //当网站存在缓存，不让浏览器缓存
        resp.setDateHeader("expires",-1);
```

```
            resp.setHeader("Cache-Control","no-cache");
            resp.setHeader("Pragma","no-cache");

            //把图片写给浏览器
            ImageIO.write(image,"jpg",resp.getOutputStream());
        }
        //生成随机数
        private String fun(){
            Random random = new Random();
            String s = random.nextInt(99999999) + "";
            StringBuffer buffer = new StringBuffer();
            for (int i = 0; i < 8-s.length(); i++) {
                buffer.append("0");
            }
            //保证一定是八位数
            String s1 = buffer.toString() + s;
            return s1;
        }
```



4. **实现重定向**

   一个web资源B收到客户端A请求后，B它会通知客户端去访问另外一个web资源C，这个过程叫重定向



   常见场景：

   - **用户登录**

```
public void sendRedirect(String location) throws IOException;
```

- **重定向和转发的区别**

  相同点：页面都会实现跳转 307

  不同点：跳转地址url不会变化，重定向会变化　302

```
<%@ page language="java" contentType="text/html; charset=UTF-
8" pageEncoding="UTF-8"%>
<html>
<body>
<h2>Hello World!</h2>
<%--这里提交的路径：需要寻找到的项目的路径--%>
<%--${pageContext.request.contextPath}代表当前项目--%>
<form action="${pageContext.request.contextPath}/login"
method="post">
    用户名：<input type="username" name="username"><br>
    密码：<input type="password" name="password"><br>
    <input type="submit">
</form>
</body>
</html>
```

```
<%@ page contentType="text/html;charset=UTF-8"
language="java" %>
<html>
<body>
<h2>Success</h2>
</body>
</html>
```

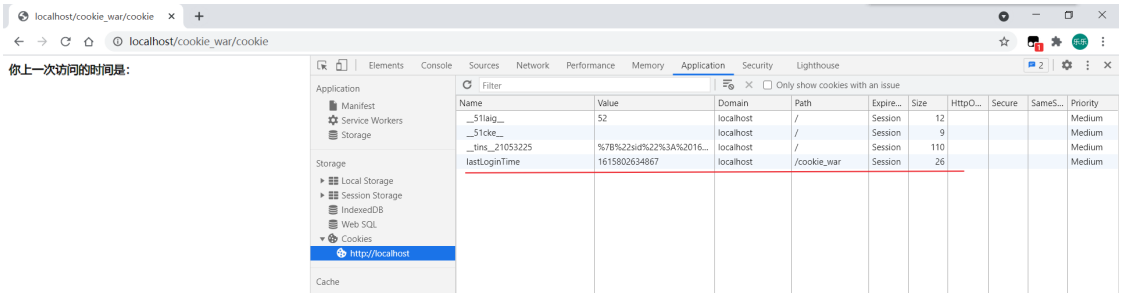
```
protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        System.out.println(username+" "+password);
        /*
            重定向

 resp.setHeader("location","/response_war/success");
            resp.setStatus(302);
         */
        resp.sendRedirect("/response_war/success.jsp");
    }
```

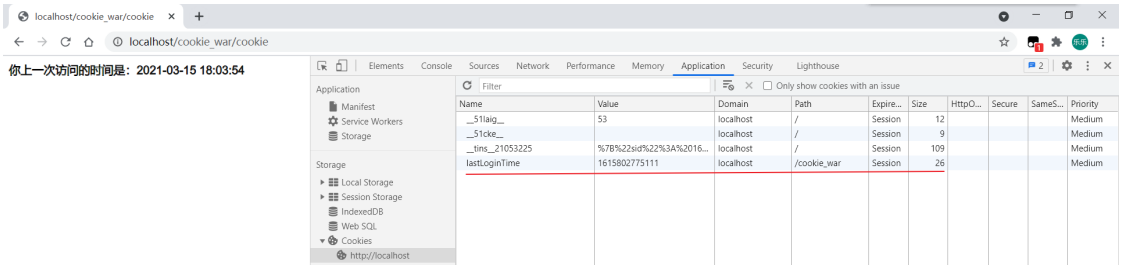
3.5、Cookie、Session

1. Cookie
    - **一个网站如何判断用户是否来过**
        - **服务端给客户端一个信件，客户端下次访问服务器带上信件就行了：**Cookie**（客户端技术）**
        - **服务器登录用户来过了，下次用户来的时侯服务器匹配用户：**Session**（服务器技术）**

```
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter writer = resp.getWriter();
        //从服务器端从客户端获取cookie
        Cookie[] cookies = req.getCookies();
        if (cookies != null) {
            writer.print("你上一次访问的时间是：");
            for (int i = 0; i < cookies.length; i++) {
                Cookie cookie = cookies[i];
```

```
                    if (cookie.getName().equals("lastLoginTime"))
{
                        //获取cookie中的值
                        Long value =
Long.parseLong(cookie.getValue());
                        SimpleDateFormat simpleDateFormat = new
SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

 writer.print(simpleDateFormat.format(value));
                    }
                }
            } else {
                writer.print("这是你第一次访问本网站");
            }

            //Cookie lastLoginTime = new Cookie("lastLoginTime",
String.valueOf(System.currentTimeMillis()));
            //设置有效期为一天
            //lastLoginTime.setMaxAge(24*60*60);
            //resp.addCookie(lastLoginTime);

            //第一次访问服务器给客户端响应一个cookie
            resp.addCookie(new Cookie("lastLoginTime",
String.valueOf(System.currentTimeMillis())));
        }
```





删除cookie：

- **不设置有效期，关闭浏览器，自动失效**
- **设置有效期时间为0**

2. Session

什么是Session：

- **服务器会给每一个用户（浏览器）创建一个Session对象**
- **一个Session独占一个浏览器，只要浏览器没关，这个Session就存在**
- **用户登陆后，整个网站都可以访问——>B站个人信息管理**


```java
public class Person {
    private String name;
    private int age;
    private String sex;

    public Person(String name, int age, String sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
    }

    public Person() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }
```

```
    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", sex='" + sex + '\'' +
                '}';
    }
}
```

SessionTest01

```
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        req.setCharacterEncoding("utf-8");
        resp.setCharacterEncoding("utf-8");
        resp.setContentType("text/html;charset=utf-8");
        //获取Session
        HttpSession session = req.getSession();
        //Session中存东西
        session.setAttribute("person", new Person("jack", 18,
"男"));
        //获取Session的id
        String sessionId = session.getId();

        //Session是不是新建
        if (session.isNew()) {
            resp.getWriter().print("Session创建成功，ID：" +
sessionId);
        } else {
            resp.getWriter().print("Session已经创建，ID：" +
sessionId);
        }
        //Session创建时候干了什么
        //Cookie cookie = new Cookie("JSESSIONID",sessionId);
        //resp.addCookie(cookie);
    }
```

SessionTest02

```
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        req.setCharacterEncoding("utf-8");
        resp.setCharacterEncoding("utf-8");
        resp.setContentType("text/html;charset=utf-8");
        //获取Session
        HttpSession session = req.getSession();
        //获取session 中的数据
        Object person = session.getAttribute("person");
        resp.getWriter().print(person);
    }
```

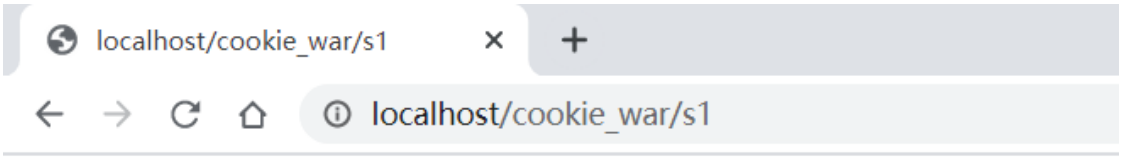
SessionTest03

```
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        //手动注销Session，常见场景：用户注销登录，自动注销Session在
web.xml文件中配置
        HttpSession session = req.getSession();
        session.removeAttribute("person");
        session.invalidate();
    }
```

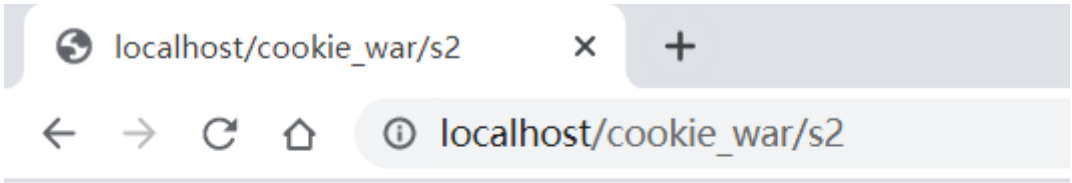
```
<servlet>
        <servlet-name>session01</servlet-name>
        <servlet-
class>com.wll.servlet.SessionTest01</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>session01</servlet-name>
        <url-pattern>/s1</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>session02</servlet-name>
        <servlet-
class>com.wll.servlet.SessionTest02</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>session02</servlet-name>
        <url-pattern>/s2</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>session03</servlet-name>
```

```
        <servlet-
class>com.wll.servlet.SessionTest03</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>session03</servlet-name>
        <url-pattern>/s3</url-pattern>
    </servlet-mapping>

    <!—Session自动注销，1分钟—>
    <session-config>
        <session-timeout>1</session-timeout>
    </session-config>
```

测试

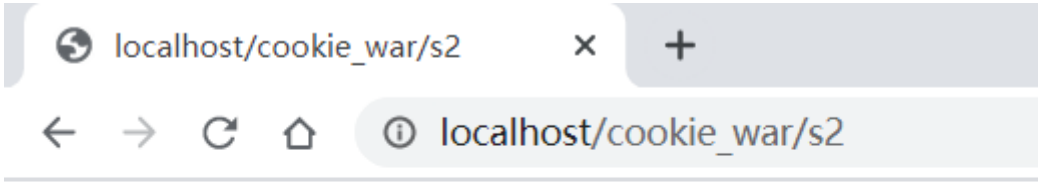


Session创建成功，ID：BEEE9EF3EA276E3F246D701F1F07DE3B

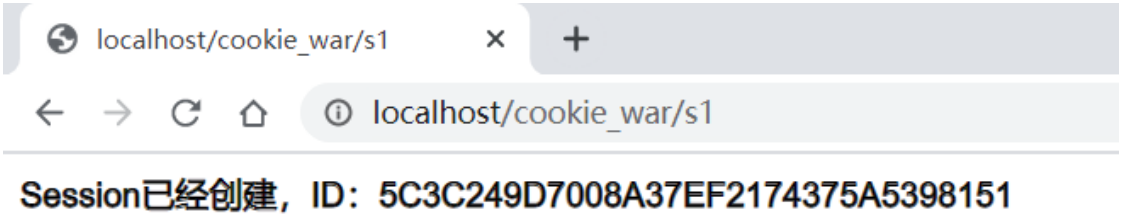


Person{name='jack', age=18, sex='男'}

调用s3后直接访问s2为null、s1中sessionid改变



null



3. **总结**
    - Session**与**Cookie**的区别：**
        - Cookie**是把用户的数据直接写给用户的浏览器，浏览器可以保存（多个）**
        - Session**把用户的数据写到用户独占的**Session**中，服务端保存（保存重要的信息，避免服务器资源浪费）**
    - Session**使用场景**
        - **保存一个用户的登录信息**
        - **购物车信息**
        - **在整个网站中经常使用的数据，保存到**Session**中**