JavaWeb

1 基本概念

1.1 前言

> web开发

web，网页

静态web

html，css

提供给所有人看的数据始终不会发生变化

动态web

淘宝

提供给所有人看的数据始终会发生变化

技术栈：Servlet　asp php

在Java中，动态web资源开发的技术统称为JavaWeb；

1.2 web应用程序

web应用程序：可以提供浏览器访问的程序

a.html b.tml

能访问到的任何一个页面或者资源，都存在某一个角落的计算机上

url

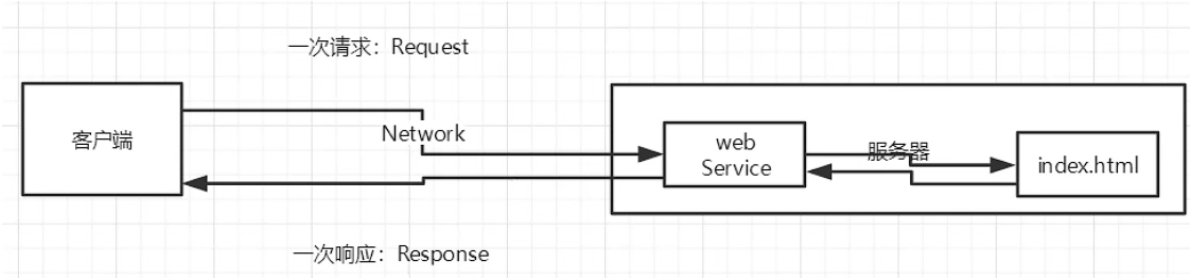
统一的web资源会被放在同一个文件夹下，web应用程序-->Tomcat：服务器

> 一个web应用由多部分组成（静态web,动态web）
>
> html css js
>
> jsp servlet
>
> java程序
>
> jar包
>
> 配置文件

web应用程序编写完毕后,若想提供给外界访问:需要一个服务器来统一管理;

1.3 静态web

`*.html,*.htm` 都是网页的后缀 ,服务器上存在这些东西,就可以直接进行读取



静态web缺点:

web页面无法动态更新,所有用户看到的都是同一个页面

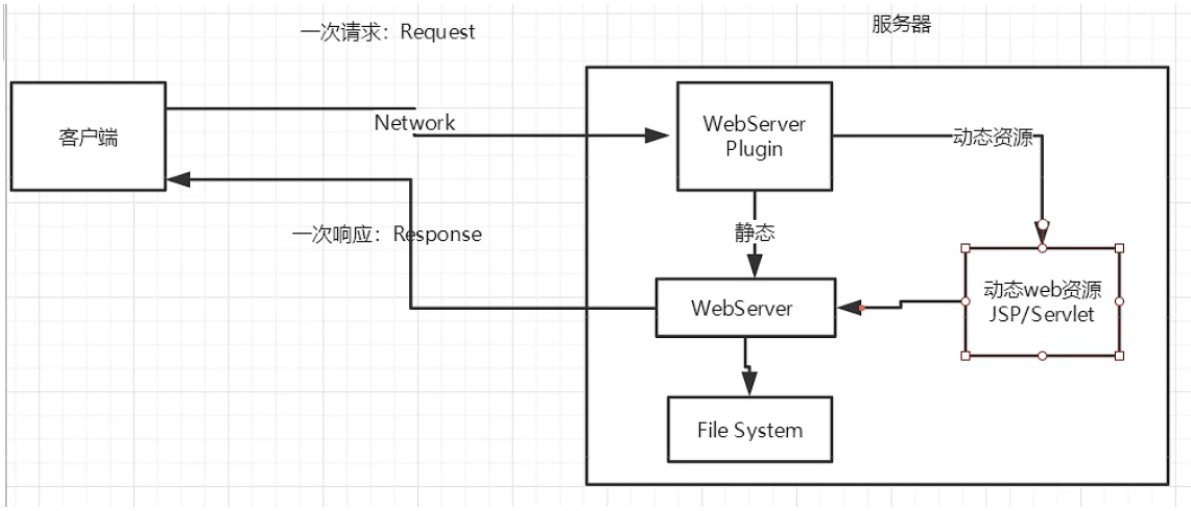
轮播图:点击特效:伪动态

JavaScript实际开发中,用的最多

无法和数据库交互(数据无法持久化,用户无法交互)

1.4 动态web

页面可以动态展示:web页面的展示效果,因人而异.



缺点:

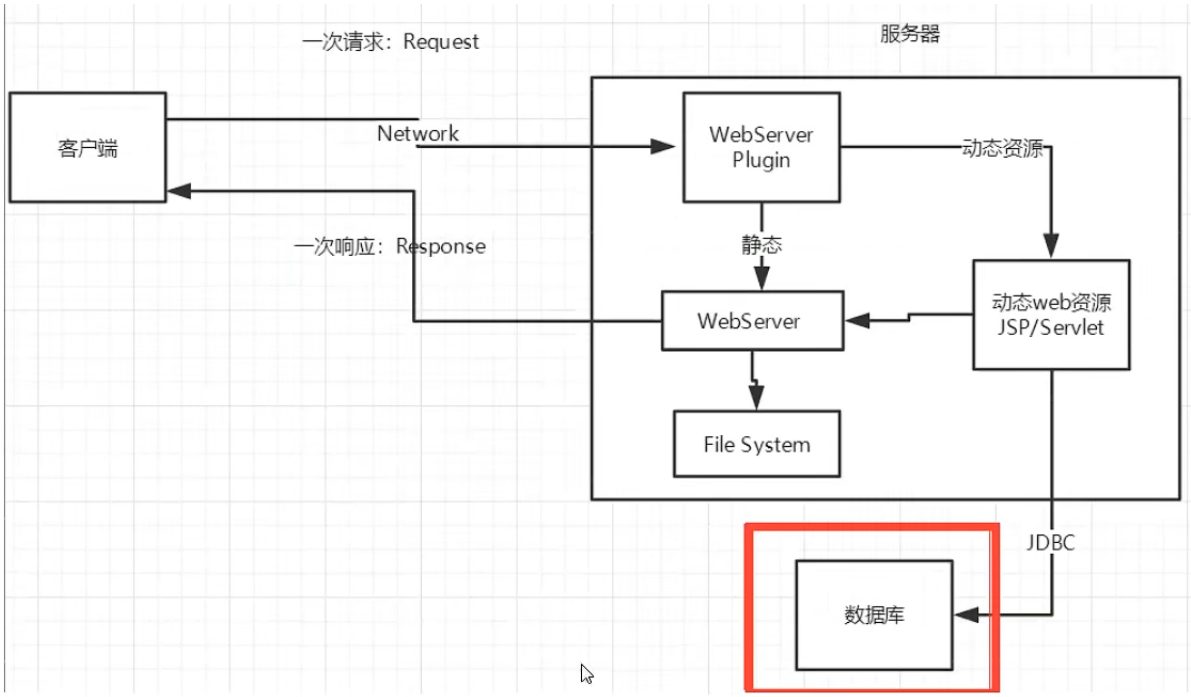
加入服务器的动态web资源出现错误,需要重新编写后台程序后重新发布

停机维护

优点:

可以动态更新,所有用户看到的不是同一个页面

可以与数据库交互(数据持久化:注册)



2 web服务器

2.1 技术讲解

ASP JSP PHP

> PHP开发速度块,功能强大,跨平台,代码简单
>
> 无法承载大访问量的情况(局限性)

> jsp/servlet
>
> BS架构
>
> CS架构
>
> 基于Java语言
>
> 可以承载高并发\高可用\高性能带来的影响

2.2 web服务器

服务器是一种被动的操作,用来接受用户请求和给用户响应信息.

下载Tomcat

1.安装

2.了解配置文件

3.作用

3 Tomcat

3.1 安装tomcat

3.2 Tomcat启动和配置

> bin 启动,关闭的脚本文件
>
> conf 配置
>
> lib依赖的jar包
>
> logs 日志
>
> webapps存放网站

启动/关闭Tomcat

3.3配置

可以配置启动的端口号:

Tomcat的默认端口号为8080

mysql:3306

http:80

https:443

server.xml服务器核心配置文件

> 可以配置启动的端口号
>
> 可以配置主机

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

```
<!-- 更改localhost后需要去Windows下system32下的etc文件下host配置更改127.0.0.1的主机映射-
->
<Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">
```

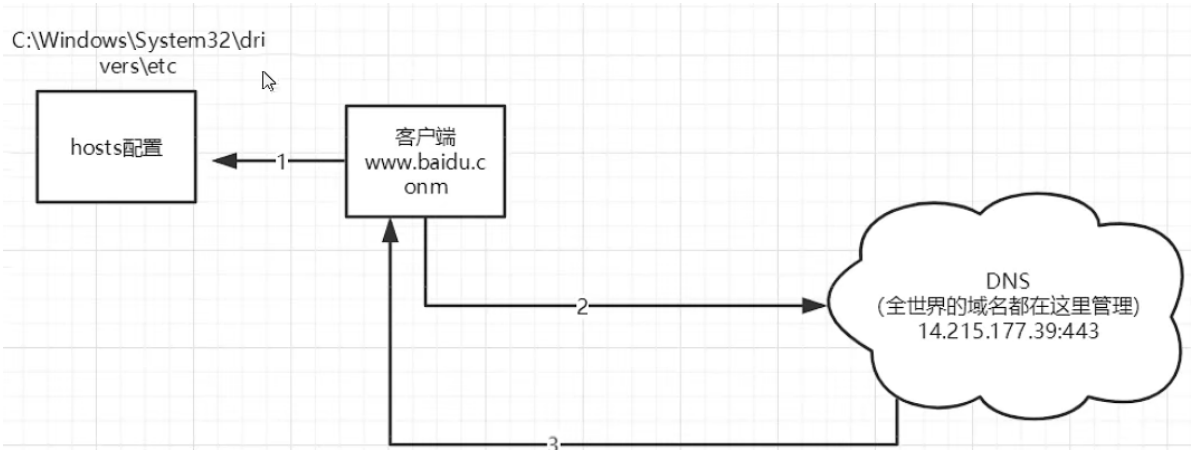
面试题

谈谈网站如何访问

1.输入域名回车

2.检查本机C:\Windows\System32\drivers\etc\hosts配置文件下有没有这个域名的映射

> 1.有:直接返回对应的ip地址,这个地址中,有需要访问的web程序可以直接访问
>
> 2.没有,去DNS服务器上,找到的话就返回,找不到就返回找不到



3.配置一下环境变量

3.4 发布一个web网站

将自己写的网站,放到服务器中指定的web应用的文件夹下,即可访问.

网站应有的结构

```
--webapps:Tomcat服务器的web目录
    -root
    -chenstudy:网站的目录名
        -WEB-INF
            -classes:java程序
            -lib:web应用以来的jar包
            -web.xml:网站配置文件
        -index.html默认首页
        -static
            -css
                -style.css
            -js
            -img
         -...
```

4 Http

请求响应协议,通常运行在TCP之上.

4.1 http

超文本传输协议

端口:80

https默认端口443

4.2 两个时代

> http1.0

http/1.0:客户端可以与web服务器连接后,只能获得一个web资源,断开连接

> http2.0

http/1.1:客户端与服务器连接后,可以获得多个web资源

4.3 http请求

客户端----发请求----服务器

百度

```
Request URL: https://www.baidu.com/                  //请求地址
Request Method: GET                                  //get方法
Status Code: 200 OK                                  //
Remote Address: 127.0.0.1:11223
Referrer Policy: strict-origin-when-cross-origin
```

> 1请求行

请求行中的请求方式:GET

请求方式:Get,Post,HEAD,DELETE,PUT,TRACT....

get:请求能够携带的参数比较少,大小有限制,会在浏览器的url地址栏显示数据内容,不安全,但高效

post:请求能够携带的参数没有限制,大小没有限制,不会在浏览器的url地址栏显示数据内容,安全,但不高效

> 2消息头

accept:告诉浏览器,它所支持的数据类型

accept-encoding:支持哪种编码格式

accept-language:告诉浏览器的语言环境

cache-control:缓存控制

connection:告诉浏览器,请求完成是断开还是保持连接

host:主机

4.4 http响应

服务器----响应----客户端

```
Cache-Control:private   //缓存控制
Connection: keep-alive
Content-Length: 161
Content-Type: text/html
Date: Sun, 06 Mar 2022 11:55:38 GMT
Location: https://www.baidu.com/
Server: bfe/1.0.8.18
```

> 1.响应体

accept:告诉浏览器,它所支持的数据类型

accept-encoding:支持哪种编码格式

accept-language:告诉浏览器的语言环境

cache-control:缓存控制

connection:告诉浏览器,请求完成是断开还是保持连接

host:主机

Refresh:告诉客户端,多久刷新一次

location:让网页重新定位;

> **2.响应状态码**

200:请求响应成功

3XX:请求重定向

4XX:找不到资源 404

5XX:服务器代码错误 500 502(网关错误)

常见面试题

浏览器中地址栏输入地址回车到展示页面,经历了什么?

5 Maven

> JavaWeb开发中需要使用大量的jar包
>
> 自动导入和配置jar包

5.1 Maven项目架构管理工具

目前用来就是方便导入jar包的!

Maven的核心思想:**约定大于配置**

-有约束,不要违反

Maven会规定好该如何编写Java代码,必须按照这个规范

5.2 下载安装Maven

5.3 配置环境变量

```
M2_HOME maven目录下的bin目录
MAVEN_HOME  maven的目录
在系统的path中配置：%MAVEN_HOME%\bin
cmd下mvn -version测试环境是否配置成功
```

5.4 修改配置文件

镜像mirrors

建议使用阿里云镜像

```
<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>
```

5.5 本地仓库

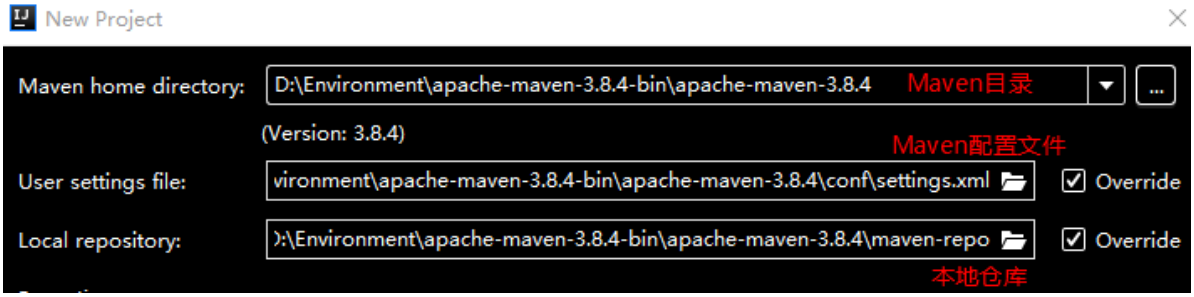
本地仓库，远程仓库

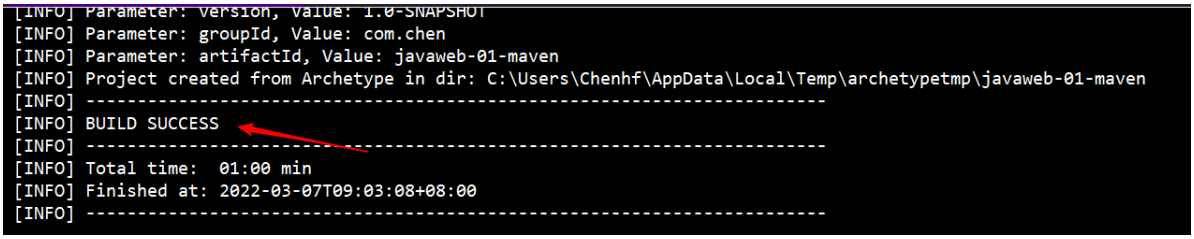
建立一个本地仓库：

```
  <localRepository>D:\Environment\apache-maven-3.8.4-bin\apache-maven-3.8.4\maven-repo</localRepository>
```

5.6 在IDA中使用Maven

1.创建一个Maven Web项目





2.项目构建成功

3.IDEA中的Maven设置

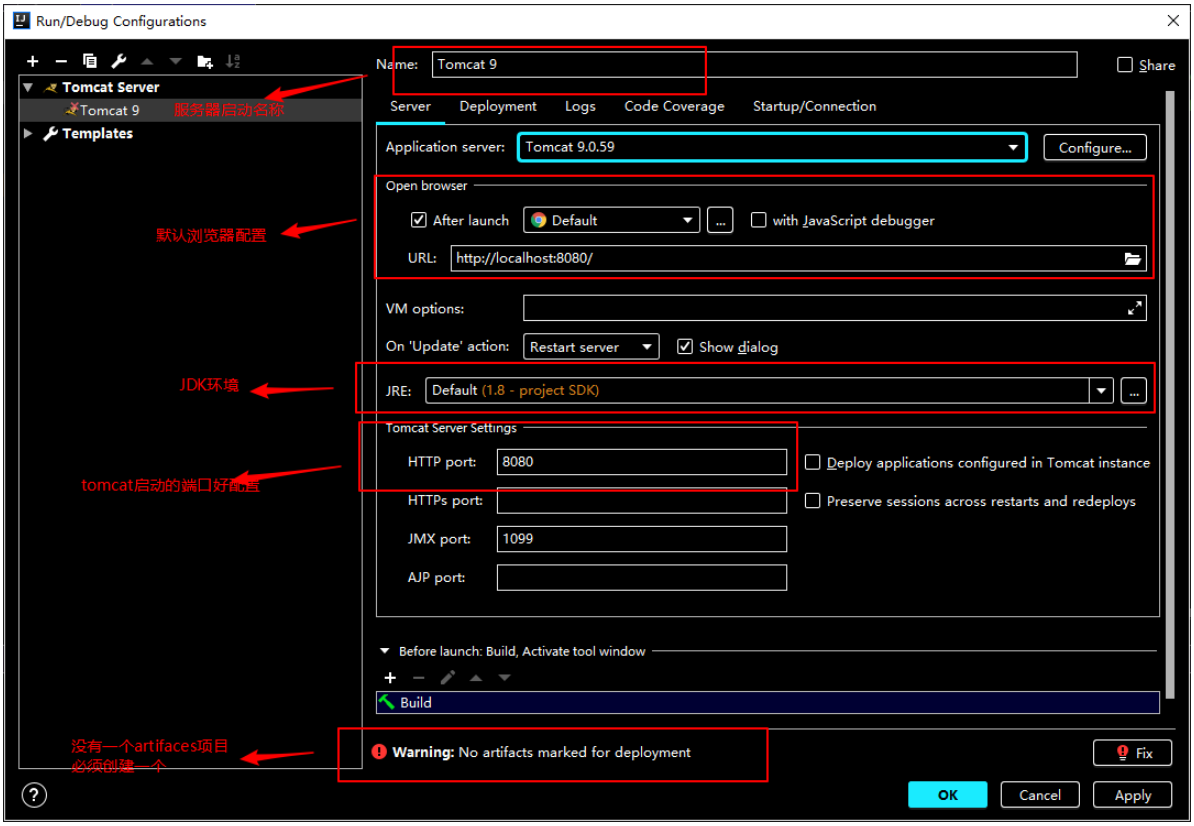
注意：项目创建成功后，看一眼Maven的配置

4.到这里Maven在IDEA中的配置就完成了

5.7配置一个普通的Maven项目

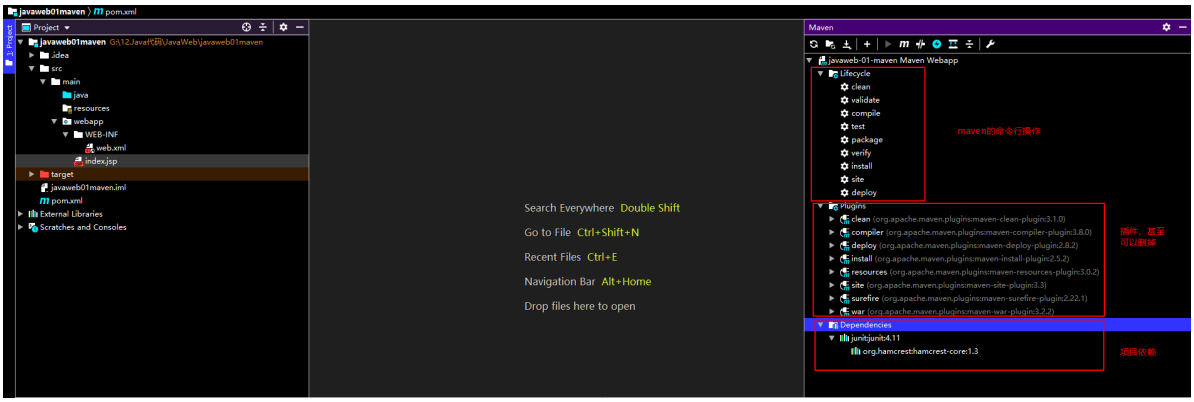
5.8在IDEA中标记文件夹功能

5.9 在IDEA中配置Tomcat

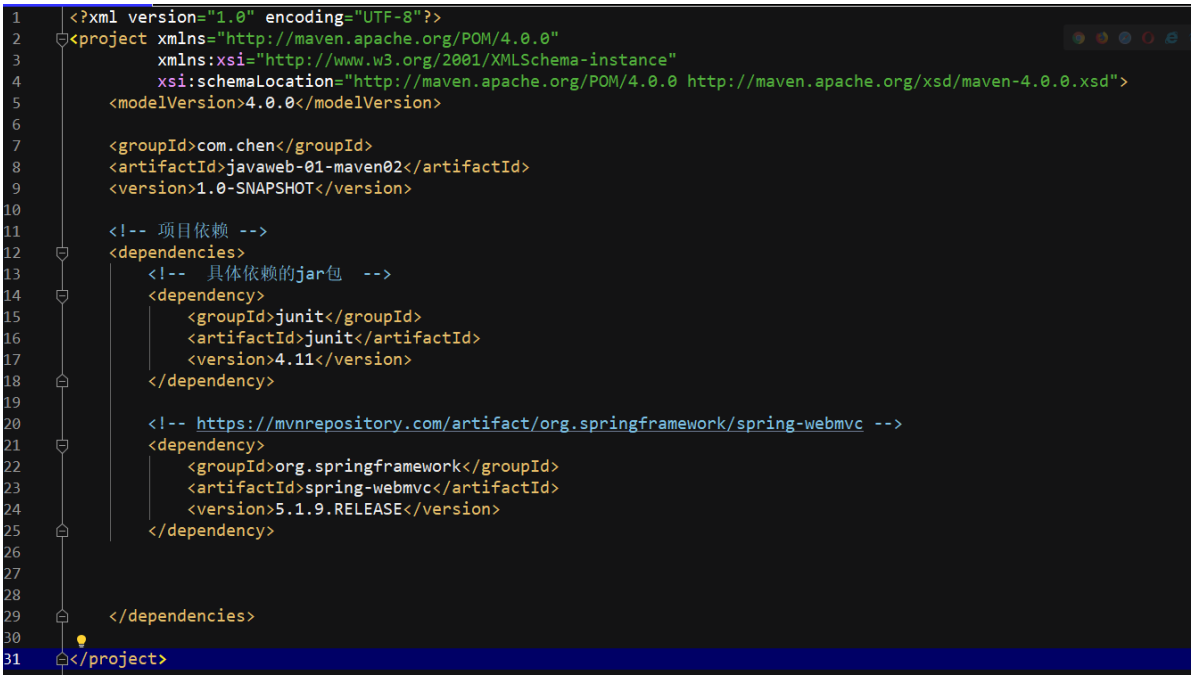


5.10 pom文件

pom.xml是maven的核心功能



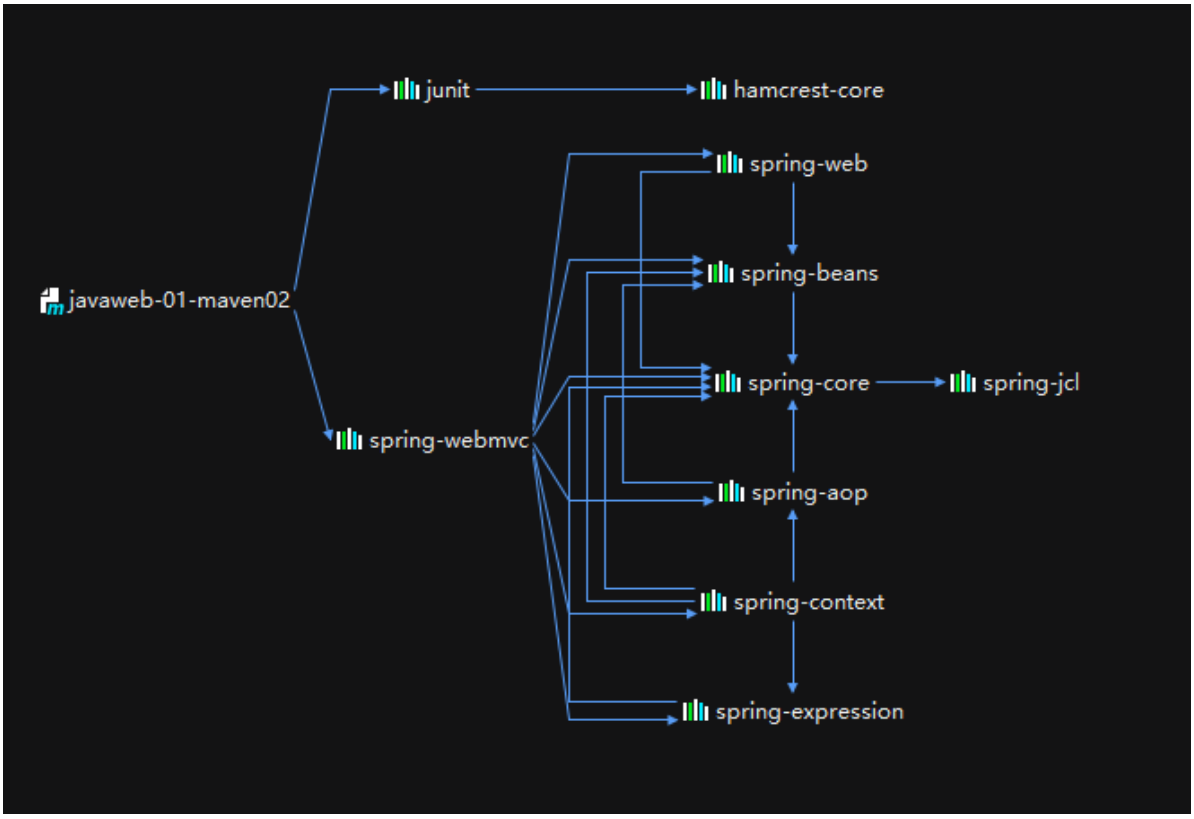
5.11 空白Maven工程的创建



```xml
<?xml version="1.0" encoding="UTF-8"?>
<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.chen</groupId>
    <artifactId>javaweb-01-maven02</artifactId>
    <version>1.0-SNAPSHOT</version>

    <!-- 项目依赖 -->
    <dependencies>
        <!--  具体依赖的jar包  -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.1.9.RELEASE</version>
        </dependency>



    </dependencies>

</project>
```

maven由于他的约定大于配置,之后可能遇到写的配置文件,无法导出或者生效的问题,解决方案:

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



替换web.xml为Tomcat4.0版本

```
<?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">


</web-app>
```

5.12 maven仓库的使用

6 servlet

6.1 Servlet

servlet,动态web的一门技术

> ### Servlet 主要功能
>
> ### 接收用户请求
>
> ### 响应给客户端内容
>
> ### 重定向或者转发

```
//请求转发,一次请求一次响应
req.getRequestDispatcher("url").forward(req,resp);

//重定向,两次请求,两次响应
resp.sendRedirect("");
```

sun在这些API中提供了一个接口叫做：servlet，若想开发一个servlet程序，只需要完成两个小步骤

编写一个类，实现servlet接口

把开发好的Java类部署到web服务器中

把实现了servlet接口的Java程序叫做servlet。

6.2 HelloServlet

servlet在sun公司有两个默认的实现类HttpServlet

1.构建一个maven项目，删掉里面的src目录，以后学习就在这个项目里面建立Moudel；这个空的工程就是Maven的主工程；

2.关于maven父子工程的理解；

父项目中会有

```
    <modules>
        <module>servlet-01</module>
    </modules>
```

子项目中会有

```
    <parent>
        <artifactId>javaweb-02-servlet</artifactId>
        <groupId>com.chen</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
```

父项目中的jar包子项目可以直接使用

```
son extends father
```

3.Maven环境优化

1.修改web.xml为最新的

2.将maven的结构搭建完整

4.编写一个servlet程序

1.编写一个普通类

2.实现servlet接口,这里直接继承HttpServerlet

```
public class HelloServlet extends HttpServlet {
```

```
    //由于get或者post请求只是请求实现的不同的方式，可以相互调用，业务逻辑一样
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //ServletOutputStream outputStream = resp.getOutputStream();
        PrintWriter writer = resp.getWriter();  //响应流

        writer.print("Hello,Servlet");

    }

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

5.编写Servlet的映射

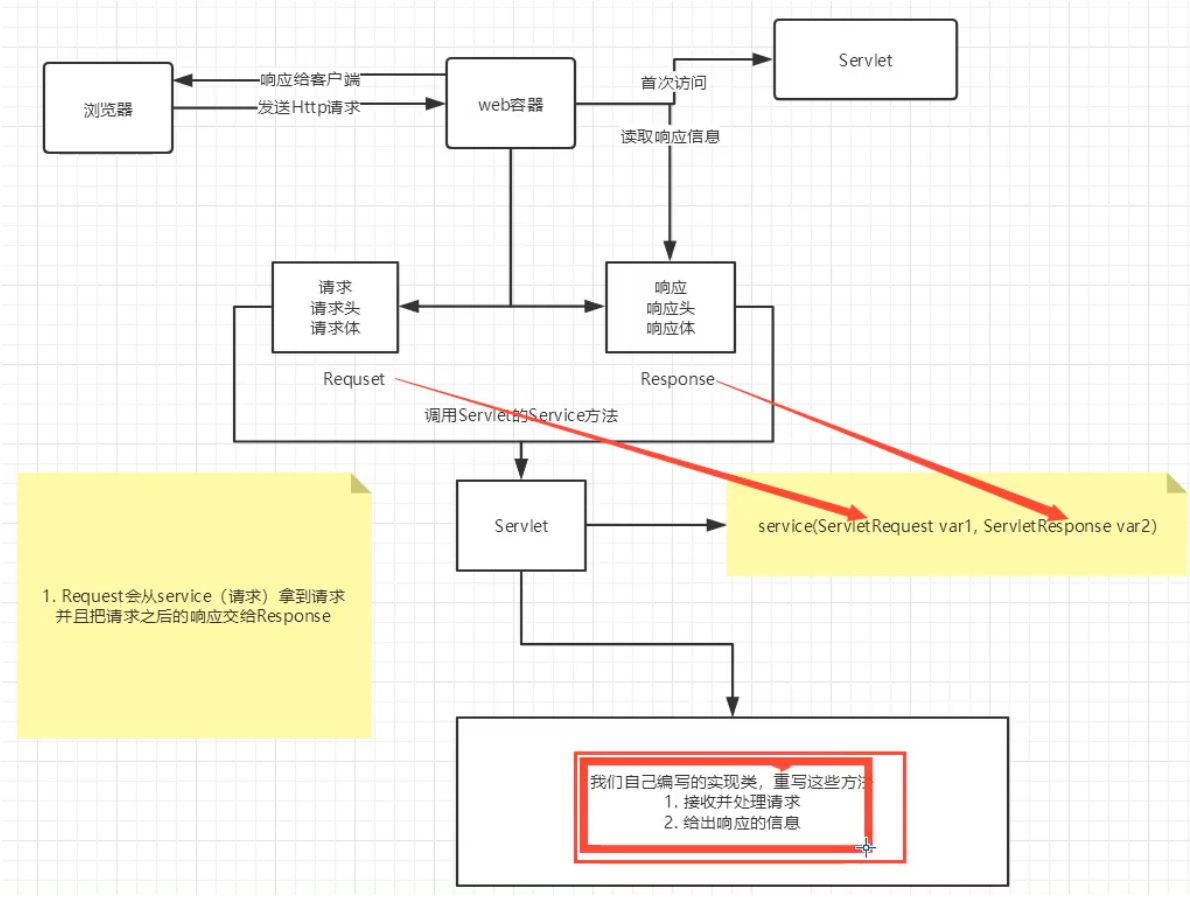
为什么要映射:写的是Java程序,但是要通过浏览器访问,而浏览需要连接web服务器,web.xml注册写servlet,还需要给他一个浏览器能够访问的路径

```
    <!-- 注册Servlet -->
    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>com.chen.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>
```

6.配置Tomcat

配置项目发布路径就可以

6.3 Servlet原理



6.4 Mapping问题

1.一个Servlet请求可以指定一个映射路径

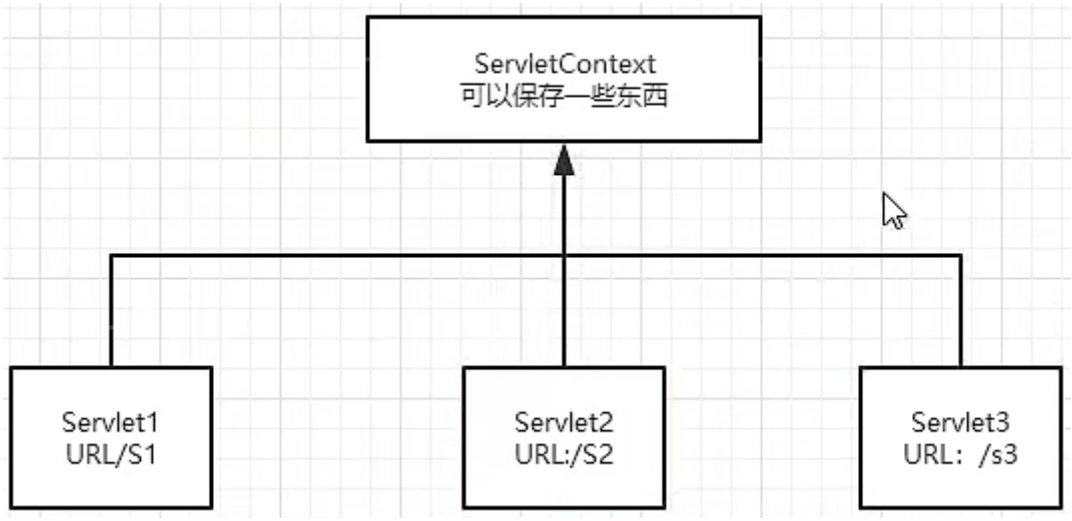
2.一个servlet可以指定多个映射路径

3.一个servlet可以指定通用路径

6.5 servletContext

web容器启动时,会为每个web程序创建一个对应的ServletContext对象,代表当前的web应用;

> ### 1 共享数据



在HelloServlet中保存的数据,在GetServlet中可以拿到

```
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 context = this.getServletContext();
        String username = "陈恒飞";
        //将一个数据保存在了ServletContext中，名字为username。值username
        context.setAttribute("username",username);
        System.out.println("Hello");
    }
}
```

```
public class GetServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        ServletContext context = this.getServletContext();
        String username = (String) context.getAttribute("username");
        resp.setContentType("text/html");
        resp.setCharacterEncoding("utf-8");
        resp.getWriter().print("名字"+username);

    }

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

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

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

    <servlet>
        <servlet-name>getc</servlet-name>
        <servlet-class>com.kuang.servlet.GetServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>getc</servlet-name>
        <url-pattern>/getc</url-pattern>
    </servlet-mapping>
```

2获取初始化参数

```
<context-param>
        <param-name>url</param-name>
        <param-value>jdbc:mysql://localhost:3306/mybatis</param-value>
    </context-param>
```

```
public class ServletDemo03 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        ServletContext context = this.getServletContext();
        String url = context.getInitParameter("url");

        resp.getWriter().print(url);
    }

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

3.请求转发

```
public class ServletDemo04 extends HttpServlet {
        @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
            ServletContext context = this.getServletContext();
            System.out.println("进入Demo04");

            //RequestDispatcher requestDispatcher =
context.getRequestDispatcher("/gp");  //转发的请求路径
            context.getRequestDispatcher("/gp").forward(req,resp);    //调用
forward实现请求转发
        }
```

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

4.读取资源文件Properties

java和resources下面新建的properties都打包到了一路径下classes,这个路径为classpath

```
@Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //利用this.getServletContext()返回流
        InputStream is = this.getServletContext().getResourceAsStream("/WEB-
INF/classes/db.properties");

        Properties prop = new Properties();
        prop.load(is);
        String user = prop.getProperty("username");
        String pwd = prop.getProperty("password");
        resp.getWriter().print(user+":"+pwd);
    }
```

6.6 HttpServletResponse

web服务器接收到客户端的http请求,针对这个请求,分别创建一个代表请求的HttpServletRequest对象,代表响应的一个HttpServletResponse对象:

> 若想获取客户端请求过来的参数:找HttpServletRequest
>
> 若想给客户端响应一些信息:找HttpServletResponse

1.简单分类

负责向浏览器发送数据的方法

```
ServletOutputStream getOutputStream() throws IOException;

PrintWriter getWriter() throws IOException;
```

负责向浏览器发送响应头的方法

```
void setCharacterEncoding(String var1);

void setContentLength(int var1);

void setContentLengthLong(long var1);

void setContentType(String var1);
```

```
void setDateHeader(String var1, long var2);

void addDateHeader(String var1, long var2);

void setHeader(String var1, String var2);

void addHeader(String var1, String var2);

void setIntHeader(String var1, int var2);

void addIntHeader(String var1, int var2);
```

2.常见应用

1向浏览器输出消息

2下载文件

> 1.要获取下载文件的路径
>
> 2.下载的文件名是什么
>
> 3.设置想办法让浏览器能够支持下载需要的东西
>
> 4.获取下载的输入流
>
> 5.创建缓冲区
>
> 6.获取OutputStream对象
>
> 7.将FileOutputStream流写入到buffer缓冲区
>
> 8.使用OutputStream将缓冲区中的数据输出到客户端

```
public class FileServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
//        1.要获取下载文件的路径
        String realPath = "G:\\12.Java代码\\JavaWeb\\javaweb-02-
servlet\\response\\target\\classes\\1.png";
        System.out.println("下载文件的路径："+realPath);
//
//        2.下载的文件名是什么
        String fileName = realPath.substring(realPath.lastIndexOf("\\") + 1);
//
//        3.设置想办法让浏览器能够支持下载需要的东西
        resp.setHeader("Content-Disposition","attachment; filename="+
URLEncoder.encode(fileName,"UTF-8"));
//
//        4.获取下载的输入流
        FileInputStream in = new FileInputStream(realPath);
//
//        5.创建缓冲区
        int len = 0;
        byte[] buffer = new byte[1024];
//
//        6.获取OutputStream对象
```

```
        ServletOutputStream out = 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=in.read(buffer))>0){
            out.write(buffer,0,len);
        }
        in.close();
        out.close();
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        doGet(req, resp);
    }
}
```

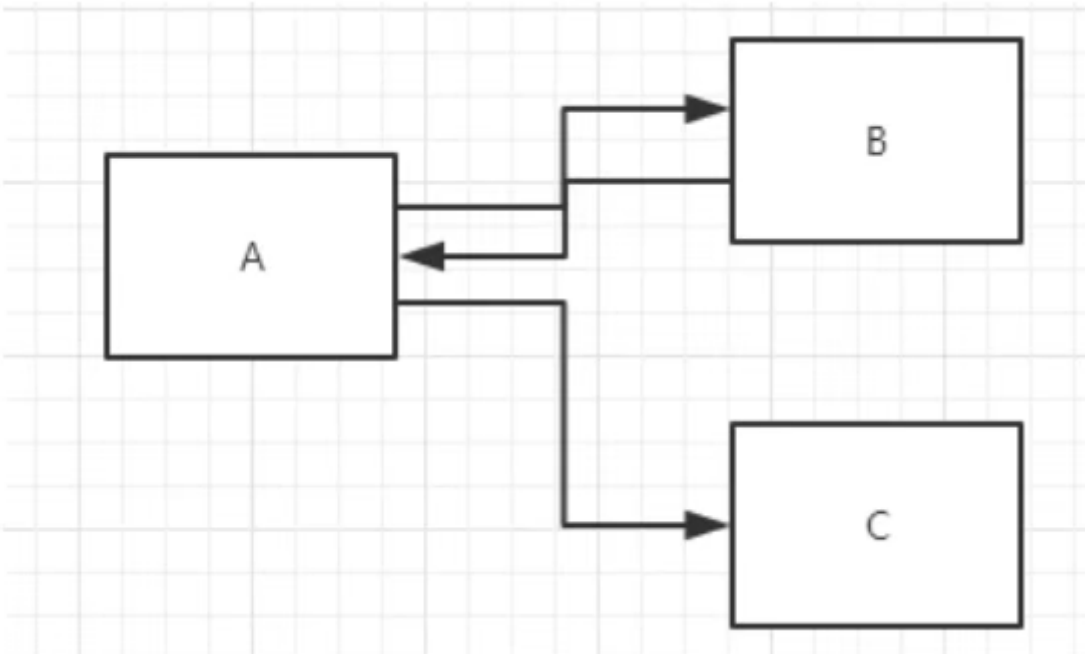
3.验证码功能

> 验证如何来?
>
> 前端实现
>
> 后端实现,需要用到ava的图片类,生产一个图片

```
public class ImageServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        //浏览器每5秒刷新一次
        resp.setHeader("refresh","3");
        //在内存中创建一个图片
        BufferedImage image = new BufferedImage(80,20,BufferedImage.TYPE_INT_RGB);
        //得到图片
        Graphics2D g = (Graphics2D)image.getGraphics();//笔
        //设置图片的背景颜色
        g.setColor(Color.WHITE);
        g.fillRect(0,0,80,20);
        //给图片写数据
        g.setColor(Color.BLUE);
        g.setFont(new Font(null, Font.BOLD,20));
        g.drawString(makeNum(),0,20);

        //告诉浏览器,这个请求用图片的方式打开
        resp.setContentType("image/jpeg");
        //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());
    }

    //生产随机数
```

```
    private String makeNum(){
        Random random = new Random();
        String num = random.nextInt(99999999) + "";
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < 7-num.length(); i++) {
            sb.append("0");
        }
        num = sb.toString() + num;
        return num;
    }

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

4.实现重定向



B的一个Web资源收到客户端A的请求后，B会通知A客户端去访问另外一个Web资源C，此过程称为重定向。

常见场景：用户登录

```
void sendRedirect(String var1) throws IOException;
```

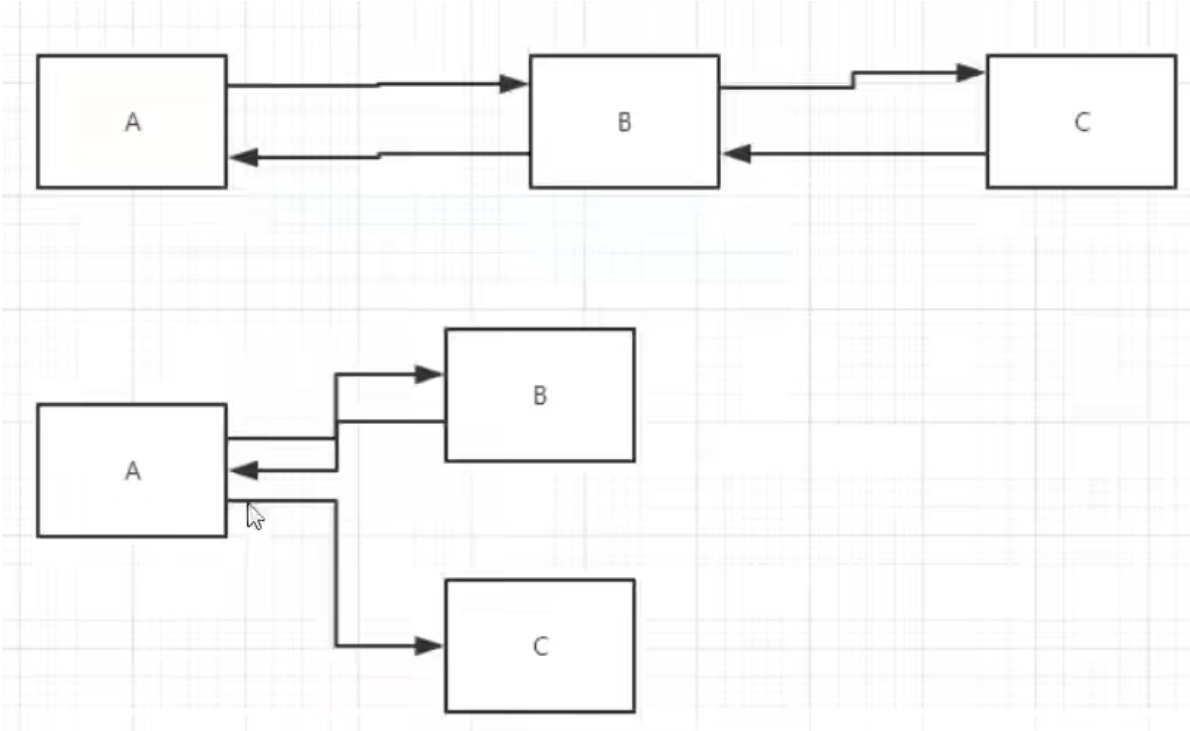
```
public class RedirectServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        resp.setHeader("Location","/img");
        resp.setStatus(302);

        resp.sendRedirect("/img");//需要加上项目相对路径
```

```
    }

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

5.重定向和转发的区别?

> 相同点:
>
> 页面都会实现跳转

> 不同点:
>
> 请求转发时,url不会发生变化　307
>
> 重定向时候,url会发生变化　　302



6.7 HttpServletRequest

HttpServletRequest代表客户端的请求,用户通过http访问服务器,Http请求中的所有信息会被封装到HttpServletRequest,通过Http中的所有信息会被封装到HttpServletRequest,通过HttpServletRequest方法,获得客户端的所有信息

```
public class RequestTest extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        System.out.println("进入这个请求");
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        System.out.println(username+":"+password);
```

```
        resp.sendRedirect("/success.jsp");
    }

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

1.获取前端传递的参数

> req.getParameter
>
> req.getParameterValues

2.请求转发

```
public class LoginServlet extends HttpServlet {
 @Override
 protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

     req.setCharacterEncoding("utf-8");
     String username = req.getParameter("username");
     String password = req.getParameter("password");
     String[] hobbys = req.getParameterValues("hobbys");
     System.out.println("==========================");
     //后台接收中文乱码问题

     System.out.println(username);
     System.out.println(password);
     System.out.println(Arrays.toString(hobbys));
     System.out.println("==========================");
     //通过重定向
     //resp.sendRedirect("/success.jsp");
     //resp.setCharacterEncoding("utf-8");
     //通过请求转发
     req.getRequestDispatcher("/success.jsp").forward(req,resp);
 }

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

7 Cookie&Session

7.1 会话Session

会话:用户打开了一个浏览器,点击了很多超链接,访问多个web资源,关闭浏览器,这个过程称之为会话.

有状态会话:客户端访问某个网站,下次再来会知道曾经访问过

> **1.服务端给客户端一个cookie,客户端下次访问服务端带上cookie就可以.cookie**
>
> **2.服务器登记来过了,下次来的时候我匹配你.session**

7.2 保存会话的两种技术

cookie

> 客户端技术(服务器响应,客户端请求)

session

> 服务器技术,利用这个技术,可以保存用户的会话信息.可以把信息或数据放在session中.

7.3 cookie

> 1.从请求中拿cookie信息
>
> 2.服务器响应客户端cookie

```
//cookie,服务器从客户端获取,
Cookie[] cookies = req.getCookies();     //返回数组,cookie可能存在多个
//获取cookie的key
cookie.getName();
//获取cookie的值
cookie.getValue()
//新建一个cookie
new Cookie("lastLoginTime", System.currentTimeMillis()+"");
//设置cookie的有效期
cookie.setMaxAge(20*60*60);
//响应给客户端一个cookie
resp.addCookie(cookie);
```

cookie:一般保存在用户目录下的appdata;

> 一个cookie只能保存一个信息
>
> 一个web站点可以给浏览器发送多个cookie,最多存放20个cookie
>
> cookie大小限制4kb
>
> 300个cookie浏览器上限

删除cookie

> 不设置有效期,关闭有效期,自动失效
>
> 设置有效期时间为0

```
//中文数据传递
public class CookieDemo03 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
```

```
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();

        //cookie,服务器从客户端获取,
        Cookie[] cookies = req.getCookies();    //返回数组,cookie可能存在多个
        //判断cookie是否存在
        if (cookies!=null){
            //如果存在怎么办
            out.write("上次访问的时间是:");
            for (int i = 0; i < cookies.length; i++) {
                Cookie cookie = cookies[i];
                //获取cookie的名字
                if (cookie.getName().equals("name")){
                    System.out.println(cookie.getValue());
                    out.write(cookie.getValue());
                }
            }
        }else{
            out.write("第一次访问.");
        }
        //URLEncoder.dncode()解码
        Cookie cookie = new Cookie("name", URLEncoder.encode("陈恒飞","utf-8"));
        resp.addCookie(cookie);
    }

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

```
public class CookieDemo02 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //创建一个cookie,名字要和删除的名字一致
        Cookie cookie = new Cookie("lastLoginTime",
System.currentTimeMillis()+"");
        //cookie有效期设置为0,立马过期
        cookie.setMaxAge(0);
        resp.addCookie(cookie);
    }

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

```
//保存用户上一次访问的时间
public class CookieDemo01 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
```

```
        //服务器,告诉你,你来的时间,把这个时间封装称为一个信件,下次带来就知道你来了.

        req.setCharacterEncoding("utf-8");
        resp.setCharacterEncoding("utf-8");
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();

        //cookie,服务器从客户端获取,
        Cookie[] cookies = req.getCookies();     //返回数组,cookie可能存在多个
        //判断cookie是否存在
        if (cookies!=null){
            //如果存在怎么办
            out.write("上次访问的时间是:");
            for (int i = 0; i < cookies.length; i++) {
                Cookie cookie = cookies[i];
                //获取cookie的名字
                if (cookie.getName().equals("lastLoginTime")){
                    //获取cookie中的值,把对象转换为时间戳,然后再用date转换为日期
                    long lastLoginTime = Long.parseLong(cookie.getValue());
                    Date date = new Date(lastLoginTime);
                    out.write(date.toLocaleString());
                }
            }
        }else{
            out.write("第一次访问.");
        }
        //服务器给客户端响应一个cookie
        Cookie cookie = new Cookie("lastLoginTime",
System.currentTimeMillis()+"");
        cookie.setMaxAge(20*60*60);
        resp.addCookie(cookie);
    }

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

7.4 Session **

session:会话.

> 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,只要浏览器没有关闭,这个session就存在
>
> 用户登录之后,整个网站都可以访问----->保存用户的信息;保存购物车的信息

session和cookie的区别:

> cookie是把用户的数据写给用户的浏览器,浏览器保存
>
> 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,减少服务器资源的浪费)
>
> session由服务器创建

session使用场景:保存一个登录用户的信息

在网站中经常使用的数据,将他保存在session中

```
public class SessionDemo01 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //解决乱码问题
        resp.setCharacterEncoding("utf-8");
        req.setCharacterEncoding("utf-8");
        //设置浏览器响应的格式
        resp.setContentType("text/html;charset=UTF-8");
        //得到session
        HttpSession session = req.getSession();

        //给session中存东西
        session.setAttribute("name",new Person("陈恒飞",1));
        //获取sessionID
        String sessionID = session.getId();

        //判断ession是不是新创建的
        if (session.isNew()){
            resp.getWriter().write("session创建成功,ID"+sessionID);
        }else{
            resp.getWriter().write("session已经在服务器存在,ID"+sessionID);
        }

        //Session创建的时候做了什么事情
//        Cookie cookie = new Cookie("JSESSIONID",sessionID);
//        resp.addCookie(cookie);
    }

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

```
public class SessionDemo02 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //解决乱码问题
        resp.setCharacterEncoding("utf-8");
        req.setCharacterEncoding("utf-8");
        //设置浏览器响应的格式
        resp.setContentType("text/html;charset=UTF-8");
        //得到session
        HttpSession session = req.getSession();

        Person person = (Person) session.getAttribute("name");
        System.out.println(person.toString());

    }
```

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

```
public class SessionDemo03 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        HttpSession session = req.getSession();
        session.removeAttribute("name");
        //手动注销session
        session.invalidate();
    }

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

8 JSP

8.1 什么是JSP

Java Server Pages:Java服务端页面,也和servlet一样,用于动态web技术

最大的特点:

> 写JSP就像在写html
>
> 区别
>
> > html只给用户提供静态的数据
> >
> > jsp页面中可以嵌入Java代码为用户提供动态数据

8.2 JSP原理

思路:JSP到底怎么执行的.

> 代码层面没有任何问题
>
> 服务器内部工作
>
> tomcat中有一个work目录;
>
> IDEA中使用Tomcat的会在IDEA的tomcat中生产一个work目录

浏览器向服务器发送请求,不管访问什么资源,其实都是在访问Servlet

JSP最终也会被转换成为一个Java类

JSP本质上就是一个servlet

> JSP页面---->转换将JSP页面转换为Java文件---->xxx_jsp.java---->编译---->xxx_jsp.clss
>
> 用户真正拿到的,就是服务器处理完毕的class对象,就是servlet

在jsp页面中,Java代码会原封不动的输出;

html代码会被转换为`out.write()`这样的格式输出到前端

8.3 JSP语法

> JSP表达式

```
<%-- JSP表达式 作用:将程序的输出,输出到客户端 --%>
<%= new java.util.Date()%>
```

> JSP脚本片段

```
<%-- jsp脚本片段 --%>
<%
  int sum = 0;
  for (int i = 0; i <= 100; i++) {
    sum+=i;
  }
  out.println("<h1>Sum="+sum+"</h1>");
%>
```

> JSP声明:会被编译到JSP生成Java的类中!其他的,就会被生成到_jspService方法中

8.5 9大内置对象

pageContext存东西

request存东西

response

session存东西

application【ServletContext】存东西

config【ServletConfig】

out

page，几乎不用

exception

```
pageContext.setAttribute("name1","1");//保存的数据只在一个页面中有效
request.setAttribute("name2","2");//保存的数据只在一次请求中有效
session.setAttribute("name3","3");//保存的数据只在一次会话中有效，从打开到关闭浏览器
application.setAttribute("name4","4");//保存的数据只在服务器中有效，从打开到关闭服务器
```

request：客户端向服务器发送请求，产生的数据，用户看完就没用了，比如：新闻

session：客户端向服务器发送请求，产生的数据，用户用户一会还要用，比如：购物车

application：客户端向服务器发送请求，产生的数据，一个用户用完，其他用户还能使用，比如聊天数据

8.6 JSP标签、JSTL标签、EL表达式

EL表达式：

获取数据

执行运算

获取web开发的常用对象

调用Java方法

9 JavaBean

实体类：

JavaBean有特定的写法：

必须要有无参构造

属性必须私有化

必须有对应的get/set方法

一般用来和数据库的字段做映射ORM；

ORM：对象关系映射

表---->类

字段---->属性

行记录--->对象

id	name	age	address
1	小陈1	3	洛阳
2	小陈2	35	洛阳
3	小陈3	66	洛阳

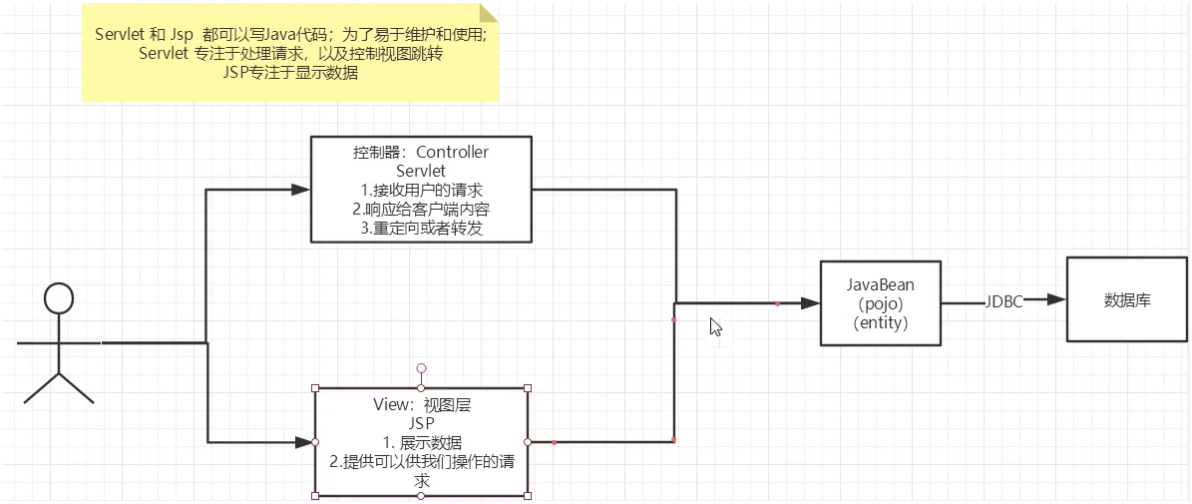
```
class People{
    private int id;
    private String name;
    private int age;
    private String address;

}

class A{
    new People(1,"小陈1",3,"洛阳");
    new People(2,"小陈2",35,"洛阳");
    new People(3,"小陈3",66,"洛阳");
}
```

10 MVC三层架构

MVC:Model　View　Controller 模型　视图　控制器

> ## Servlet 主要功能
>
> ## 接收用户请求
>
> ## 响应给客户端内容
>
> ## 重定向或者转发

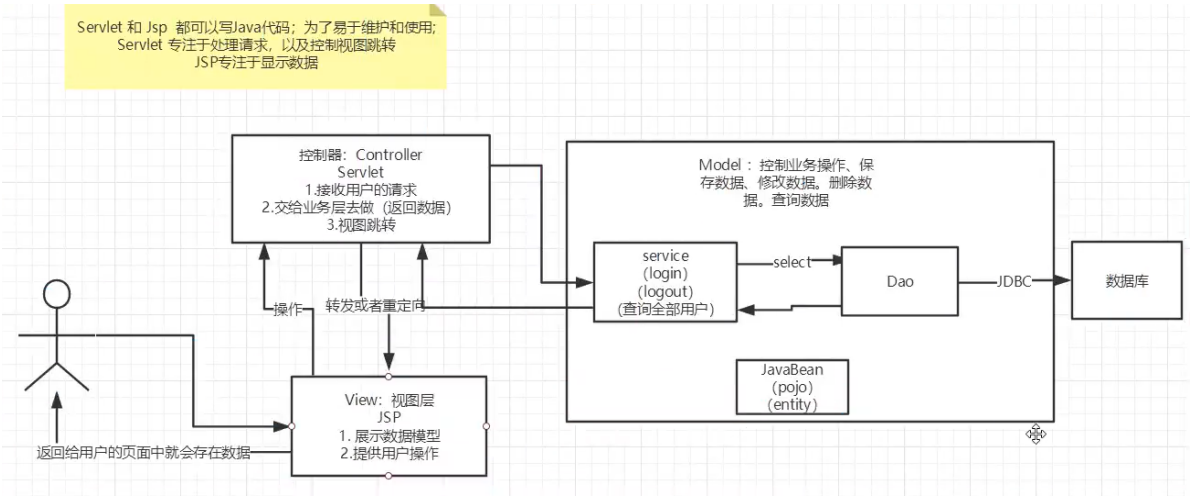
10.1 早些年开发



用户直接访问控制层就可以直接操作数据库;

```
servlet---->CRUD---->数据库
弊端:程序臃肿,不利于维护

架构:没有什么是加一层解决不了的
```

10.2 三层架构



Model

> 业务处理:业务逻辑(Service)

> 数据持久层:CRUD(Dao)

View

> 展示数据

> 提供连接发起Servlet请求(a form img)

Controller(Servlet)

> jie收用户的请求:(req:请求参数,Session信息....)

> 接收业务层处理对应的代码

> 控制视图跳转

登录---->接收用户请求---->处理用户的请求(获取用户登录参数,username,password)---->交给业务层处理登录业务（事务）---->Dao层查询用户名和密码是否正确---->数据库

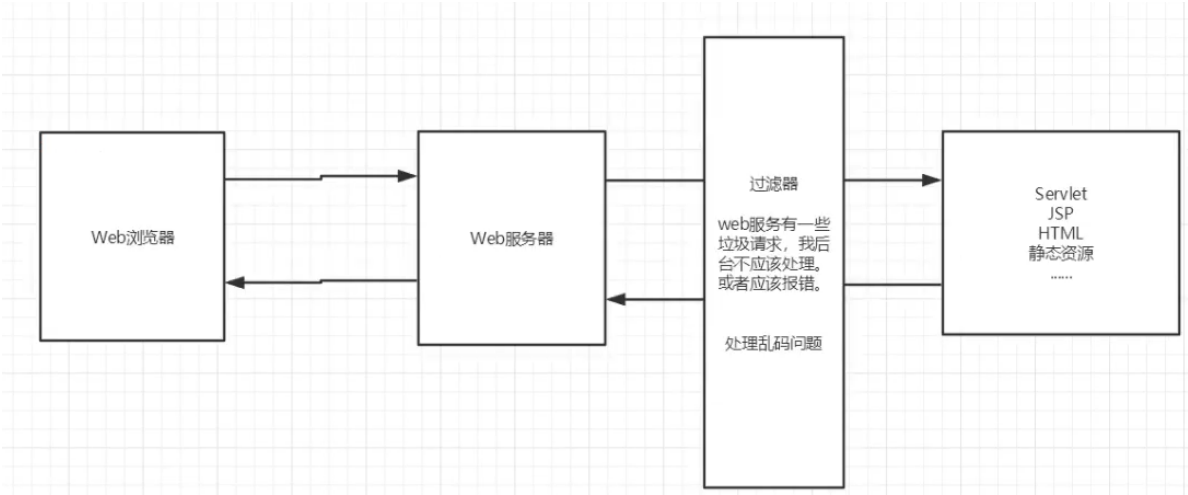
---

11 Filter

Filter:过滤器，用来过滤网站的数据

> web服务的垃圾请求，后台不应该处理，或者应该报错

> 处理乱码问题



Filter开发步骤:

1.导包，javax-servlet，不要导错

2.编写过滤器

3.在web.xml中配置Filter

```
package com.kuang.filter;

import javax.servlet.*;
import java.io.IOException;

public class CharacterEncodingFilter implements Filter {
```

```
    //初始化
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("CharacterEncodingFilter初始化");
    }

    //链
    /*
    * 1.过滤中的所有代码，在过滤特定请求的时候都会执行
    * 2.必须要让所有过了不起继续同行
    * chain.doFilter(request,response);
    *
    * */
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
        request.setCharacterEncoding("utf-8");
        response.setCharacterEncoding("utf-8");
        //response.setContentType("text.html;charset=UTF-8");
        response.setContentType("text/html;charset=GBK");
        System.out.println("CharacterEncodingFilter执行前。。。");
        chain.doFilter(request,response);//让请求继续走，如果不写，程序到这里拦截停止
        System.out.println("CharacterEncodingFilter执行后。。。");
    }

    //web服务器关闭时候filter会销毁
    //销毁
    public void destroy() {
        System.out.println("CharacterEncodingFilter销毁");
    }
}
```

```
<filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-class>com.kuang.filter.CharacterEncodingFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/servlet/*</url-pattern>
    </filter-mapping>
```

12 监听器

实现一个监听器的接口;(有N种)

> 1.编写一个监听器(实现监听器的接口)

```
//统计网站在线人数：统计session
public class OnlineCountListener implements HttpSessionListener {
    //创建session的监听：看你的一举一动
    //一旦创建一个session就会触发一次事件
    public void sessionCreated(HttpSessionEvent se) {
        //1.获取session   2.获取上下文
        ServletContext ctx = se.getSession().getServletContext();
        System.out.println(se.getSession().getId());
        Integer onlineCount = (Integer)ctx.getAttribute("OnlineCount");
```

```
        if (onlineCount ==null){
            onlineCount = new Integer(1);
        }else{
            int count = onlineCount.intValue();
            onlineCount = new Integer(count+1);
        }

        ctx.setAttribute("OnlineCount",onlineCount);

    }


    //销毁session的监听
    //一旦销毁一个session就会触发一次事件
    public void sessionDestroyed(HttpSessionEvent se) {
        ServletContext ctx = se.getSession().getServletContext();

        Integer onlineCount = (Integer)ctx.getAttribute("OnlineCount");

        if (onlineCount ==null){
            onlineCount = new Integer(0);
        }else{
            int count = onlineCount.intValue();
            onlineCount = new Integer(count-1);
        }

        ctx.setAttribute("OnlineCount",onlineCount);
    }
}
```

> 2.配置监听器web.xml 注册监听器

```
<listener>
    <listener-class>com.kuang.listener.OnlineCountListener</listener-class>
</listener>

<session-config>
    <session-timeout>1</session-timeout>
</session-config>
```

13 Filter Listener常见应用

用户登录之后才能进入主页,注销之后不能进入主页

14 JDBC

jar包的支持:java.sql Javax.sql mysql.-conneter-java...连接驱动(必须要导入)

实验环境搭建:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.47</version>
</dependency>
```

```
package com.kuang.test;

import java.sql.*;

public class TestJdbc {
    public static void main(String[] args) throws ClassNotFoundException,
SQLException {
        //配置信息
        //useUnicode=true&characterEncoding=utf-8 解决中文乱码
        String url="jdbc:mysql://localhost:3306/jdbc?
userUnicode=true&characterEncoding=utf-8";
        String username = "root";
        String password = "123456";
        //1.加载驱动
        Class.forName("com.mysql.jdbc.Driver");
        //2.连接数据库,代表数据库
        Connection connection = DriverManager.getConnection(url, username,
password);
        //3.向数据库发送SQL的对象Statement:CRUD
        Statement statement = connection.createStatement();

        //4.编写SQL
        String sql = "select * from users";

        //5.执行查询sql
        ResultSet rs = statement.executeQuery(sql);
        while (rs.next()){
            System.out.println("id="+rs.getObject("id"));
            System.out.println("name="+rs.getObject("name"));
            System.out.println("password="+rs.getObject("password"));
            System.out.println("email="+rs.getObject("email"));
            System.out.println("birthday="+rs.getObject("birthday"));
        }

        //关闭连接释放资源
        rs.close();
        statement.close();
        connection.close();
    }
}
```

1 事务

要么都成功，要么都失败

ACID原则：保证数据的安全

```
-- 事务提交
commit();
-- 事务回滚
rollback;
```

2 Junit单元测试