第一章 JavaWeb简介

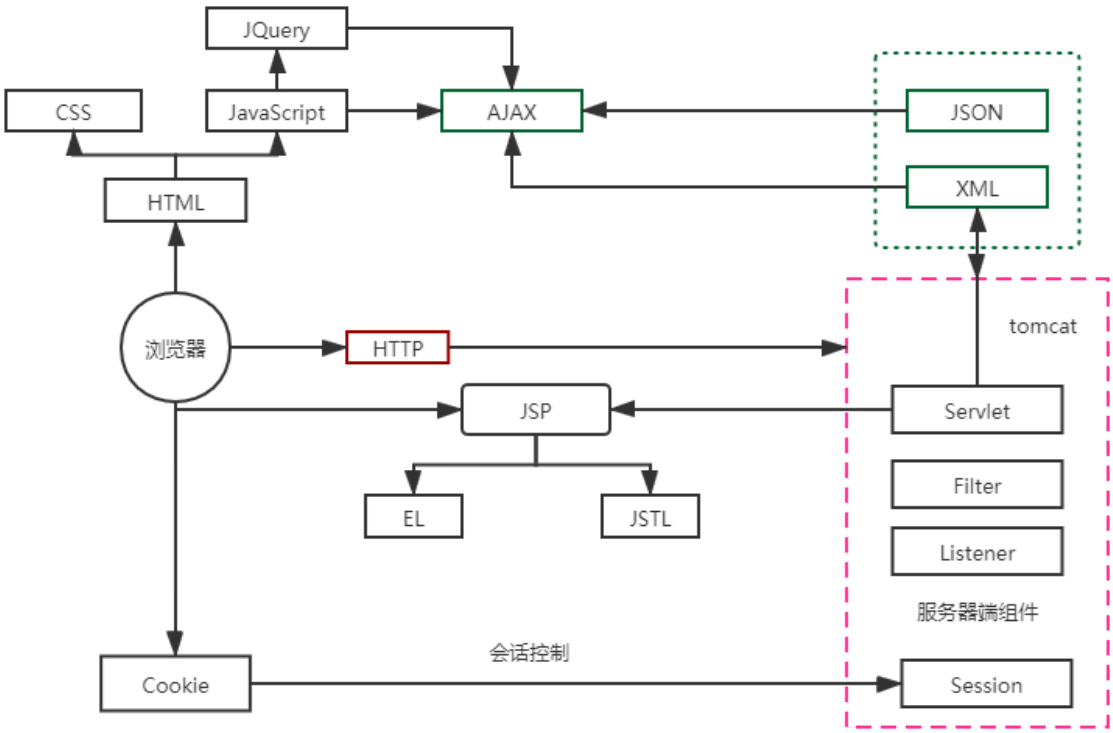
第1节 什么是web

web（World Wide Web）即全球广域网，也称为万维网，它是一种基于超文本和HTTP的、全球性的、动态交互的、跨平台的分布式图形信息系统。是建立在Internet上的一种网络服务，为浏览者在Internet上查找和浏览信息提供了图形化的、易于访问的直观界面，其中的文档及超级链接将Internet上的信息节点组织成一个互为关联的网状结构

第2节 什么是JavaWeb

使用Java技术实现上面的功能，即使用Java技术实现网络的互联互通

第二章 JavaWeb的技术体系



第三章 JavaWeb服务器

第1节 JavaWeb服务器是什么

JavaWeb服务器又被称为JavaWeb容器是为JavaWeb应用提供运行时环境，它负责管理Servlet和JSP的生命周期，以及管理它们的共享数据。

第2节 常见的JavaWeb服务器介绍

常见的JavaWeb服务器有Tomcat,jetty,Weblogic等

- Tomcat当前最流行的web容器(免费)

Tomcat是Apache 软件基金会（Apache Software Foundation）的Jakarta（[dʒə'kɑ:tə]）项目中的一个核心项目由Apache、Sun(现已被oracle公司收购)和其他一些公司及个人共同开发而成

- jetty(免费)

Jetty 是一个开源的servlet容器，它为基于Java的web容器，例如JSP和servlet提供运行环境。Jetty是使用Java语言编写的，它的API以一组JAR包的形式发布。开发人员可以将Jetty容器实例化成一个对象，可以迅速为一些独立运行（stand-alone）的Java应用提供网络和web连接

- Weblogic(收费)

WebLogic是美国Oracle公司出品的一个application server，确切的说是一个基于JAVAEE架构的中间件，WebLogic是用于开发、集成、部署和管理大型分布式Web应用、网络应用和数据库应用的Java应用服务器。将Java的动态功能和JavaEnterprise标准的安全性引入大型网络应用的开发、集成、部署和管理之中

第3节 Tomcat服务器

- tomcat服务器介绍

Tomcat 是一个免费的开放源代码的Servlet容器，它是Apache软件基金会的一个顶级项目，由Apache，Sun和其他一些公司及个人共同开发而成。由于有了Sun 的参与与支持，最新的 Servlet和JSP规范总是能在Tomcat中的到体现

- tomcat的目录结构

```
1. bin: 用来存放tomcat的命令,主要有两大类，一类是以.sh结尾的（linux命令），另一类是以.bat结尾的（windows命令）
2. conf: 主要是用来存放tomcat的一些配置文件
3. lib: 主要用来存放tomcat运行需要加载的jar包
4. logs: logs目录用来存放tomcat在运行过程中产生的日志文件
    -- 在windows环境中，控制台的输出日志在catalina.xxxx-xx-xx.log文件中
    -- 在linux环境中，控制台的输出日志在catalina.out文件中
5. temp: temp目录用户存放tomcat在运行过程中产生的临时文件
6. webapps: webapps目录用来存放应用程序，当tomcat启动时会去加载webapps目录下的应用程序。可以以文件夹、war包、jar包的形式发布应用
7. work: work目录用来存放tomcat在运行时的编译后文件
```

- tomcat的常用命令

```
1.启动 startup
2.停止 shutdown
```

- tomcat的用户管理

tomcat-users.xml用来配置管理tomcat的用户与权限

- tomcat的服务配置管理

server.xml可以设置端口号、设置域名或IP、默认加载的项目、请求编码

- 在一台服务器上配置多个tomcat的配置

需要修改三个端口号
AJP
HTTP
SHUTDOWN

第四章 JavaWeb的运行流程

第五章 第一个JavaWeb应用

第1节 创建第一个JavaWeb项目

1. 打开Eclipse
2. File--> New-->Dynamic Web Project
3. 设置Project name
4. Dynamic web module version 选择2.5
5. finish

第2节 项目结构介绍

1. src 里面放的是 java 源程序
2. JRE System Library: 指Java SE 的常用库文件集合，也就是 jar 包
3. Web App Libraries 是自己导入的项目依赖 jar 包
4. WebContent：一般我们用 Eclipse 的时候创建一个 Web Project，就会生成 WebContent 文件夹，用 MyEclipse 的时候创建一个 Web Project，就会生成 WebRoot 文件夹，这两个文件夹作用一样只是名称不同而已。WebContent 用来存放 JSP，JS，CSS，图片等文件，是项目访问的默认路径，也是工程的发布文件夹，发布时会把该文件夹发布到 tomcat 的 webapps 里.(用户可以直接访问到该目录下的资源)
5. META-INF：存放一些 meta information相关的文件的这么一个文件夹,一般来说尽量不要自己手工放置文件到这个文件夹
6. WEB-INF：WEB-INF目录是一个专用区域，容器不能把此目录中的内容提供给用户。这个目录下的文件只供容器使用，里面包含不应该由客户直接下载的资源。Web 容器要求在你的应用程序中必须有 WEB-INF 目录。WEB-INF 中包含着发布描述符（也就是web.xml文件）,一个classes目录和一个lib目录, 以及其它内容。注意：如果你的Web应用程序中没有包含这个目录,它可能将无法工作.（这是一个安全目录，此目录下的资源不能被用户直接访问）
7. WEB-INF/lib 目录，该目录中的jar包是运行时环境下使用的jar包,所谓运行时环境下使用的jar包,就是说你在运行你的项目的时候所需要使用的jar包的集合

第3节 第一个web页面

在WebContent下创建一个index.html文件

添加 <h1>Hello JavaWeb</h1>

第六章 HTTP协议

第1节 HTTP 基本概念

HTTP 是超文本传输协议，也就是HyperText Transfer Protocol，我们可以将HTTP的名字「超文本协议传输」，拆成三个部分
1. 超文本
2. 传输
3. 协议

- 超文本

HTTP 传输的内容是「超文本」

我们先来理解「文本」，在互联网早期的时候只是简单的字符文字，既现在「文本」

再来理解「超文本」，它就是超越了普通文本的文本，它是文字、图片、视频等的混合体最关键有超链接，能从一个超文本跳转到另外一个超文本

HTML 就是最常见的超文本了，它本身只是纯文字文件，但内部用很多标签定义了图片、视频等的链接，在经过浏览器的解释，呈现给我们的就是一个文字、有画面的网页了

- 传输

所谓的「传输」，很好理解，就是把一堆东西从 A 点搬到 B 点，或者从 B 点 搬到 A 点

- 协议

行为约定和规范，HTTP 协议是一个双向协议

- HTTP 概念总结

HTTP 是一个在计算机世界里专门在「两点」之间「传输」文字、图片、音频、视频等「超文本」数据的「约定和规范」

第2节 HTTP 常见的状态码

状态码	描述	常见状态码
2xx	成功,报文已经收到并被正确处理	200,204,206
3xx	重定向,资源位置发生变动,需要客户端重新发送请求	301,302,304
4xx	客户端错误,请求报文有误,服务器无法处理	400,403,404
5xx	服务器端错误,服务器在处理请求时,内部发生错误	500,501,502,503

「200 OK」是最常见的成功状态码，表示一切正常。如果是非 HEAD 请求，服务器返回的响应头都会有 body 数据。
「204 No Content」也是常见的成功状态码，与 200 OK 基本相同，但响应头没有 body 数据。
「206 Partial Content」是应用于 HTTP 分块下载或断电续传，表示响应返回的 body 数据并不是资源的全部，而是其中的一部分，也是服务器处理成功的状态

「301 Moved Permanently」表示永久重定向，说明请求的资源已经不存在了，需改用新的 URL 再次访问。
「302 Found」表示临时重定向，说明请求的资源还在，但暂时需要用另一个 URL 来访问

「304 Not Modified」不具有跳转的含义，表示资源未修改，重定向已存在的缓冲文件，也称缓存重定向，用于缓存控制

「400 Bad Request」表示客户端请求的报文有错误，但只是个笼统的错误。
「403 Forbidden」表示服务器禁止访问资源，并不是客户端的请求出错。
「404 Not Found」表示请求的资源在服务器上不存在或未找到，所以无法提供给客户端

「500 Internal Server Error」与 400 类型，是个笼统通用的错误码，服务器发生了什么错误，我们并不知道。
「501 Not Implemented」表示客户端请求的功能还不支持
「502 Bad Gateway」通常是服务器作为网关或代理时返回的错误码，表示服务器自身工作正常，访问后端服务器发生了错误。
「503 Service Unavailable」表示服务器当前很忙，暂时无法响应服务器，类似“网络服务正忙，请稍后重试”的意思

第3节 HTTP 常见属性字段

- Host

客户端发送请求时，用来指定服务器的域名。

```
Host: www.baidu.com
```

- Content-Length

服务器在返回数据时，会有 Content-Length 字段，表明本次回应的数据长度

```
Content-Length: 23552
```

- Connection

Connection 字段最常用于客户端要求服务器使用 TCP 持久连接，以便其他请求复用

```
Connection: keep-alive
```

但是，这不是标准字段，为了兼容老版本的 HTTP

- Content-Type

Content-Type 字段用于服务器回应时，告诉客户端，本次数据是什么格式

```
Content-Type: text/html; charset=utf-8
```

上面的类型表明，发送的是网页，而且编码是UTF-8

- Content-Disposition

如果服务器希望浏览器不是直接处理响应的实体内容而是让用户将响应的实体内容保存到一个文件中，这个时候需要设置Content-Disposition字段

方式:
1.inline　　: 直接在浏览器中打开
2.attachment　: 以附件的形式下载

attachment的后面还可以指定filename参数

例子: Content-Disposition: attachment;filename=xyz.zip

- Accept

客户端请求的时候，可以使用 Accept 字段声明自己可以接受哪些数据格式

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

q:相对品质因子,默认是1,表示偏好哪一个配置

Accept: */*　客户端声明自己可以接受任何格式的数据

- Content-Encoding

Content-Encoding 字段说明数据的压缩方法。表示服务器返回的数据使用了什么压缩格式

Accept-Encoding: gzip

上面表示服务器返回的数据采用了 gzip 方式压缩，告知客户端需要用此方式解压

- Accept-Encoding

Accept-Encoding: gzip, deflate

客户端在请求时，用 Accept-Encoding 字段说明自己可以接受哪些压缩方法

- User-Agent

用户代理，主要是告诉服务器我是用什么操作系统,什么浏览器发的请求,可以防止一部分机器人扫描网站数据.

- Referer

一般被网站管理人员用来追踪访问者是如何导航进入网站的,因为进入一个网站不止可以从浏览器地址栏输入网址,还可以从一个超链接进入,通过这个属性来记录用户的访问来源,还可以防盗链.

什么叫防盗链?
比如一个网站,他自己的服务器上本身没有图片资源，但是他的网站上面却展示了很多图片资源,这些图片资源一般直接来源于别的网站,别的网站会根据此网站发送的图片请求,通过此属性判断是否为自己网站的请求,来禁止盗链

- 请求头

- 响应头

第4节 GET和POST请求

客户端向服务器端发送请求的方式有两种
- GET请求
- POST请求

- GET请求

Get 方法的含义是请求从服务器获取资源，这个资源可以是静态的文本、页面、图片视频等

- POST请求

而POST 方法则是相反操作，它向 URI 指定的资源提交数据，数据就放在报文的 body 里

第5节 HTTP协议特性(优势)

HTTP 最凸出的优点是「简单、灵活和易于扩展、应用广泛和跨平台」

- 简单

HTTP 基本的报文格式就是 header + body，头部信息也是 key-value 简单文本的形式，易于理解。

- 灵活和易于扩展

HTTP协议里的各类请求方法、URI/URL、状态码、头字段等每个组成要求都没有被固定死，都允许开发人员自定义和扩充

同时 HTTP 由于是工作在应用层(OSI 第七层)，则它下层可以随意变化
HTTPS 也就是在 HTTP 与 TCP 层之间增加了 SSL/TLS 安全传输层

- 应用广泛和跨平台

互联网发展至今，HTTP 的应用范围非常的广泛，从台式机的浏览器到手机上的各种 APP，从看新闻、刷贴吧到购物、理财,HTTP 的应用片地开花，同时天然具有跨平台的优越性

- HTTP缺点

HTTP 协议里有优缺点一体的双刃剑，分别是「无状态、明文传输」，同时还有一大缺点「不安全」

1. 无状态
无状态的好处，因为服务器不会去记忆 HTTP 的状态，所以不需要额外的资源来记录状态信息，这能减轻服务器的负担，能够把更多的 CPU 和内存用来对外提供服务。

无状态的坏处，既然服务器没有记忆能力，它在完成有关联性的操作时会非常麻烦。

例如登录->添加购物车->下单->结算->支付，这系列操作都要知道用户的身份才行。但服务器不知道这些请求是有关联的，每次都要问一遍身份信息

对于无状态的问题，解法方案有很多种，其中比较简单的方式用 Cookie 技术

2. 明文
明文意味着在传输过程中的信息，是可方便阅读的，通过浏览器的 F12 控制台或 Wireshark 抓包都可以直接肉眼查看，为我们调试工作带了极大的便利性

但是这正是这样，HTTP 的所有信息都暴露在了光天化日下，相当于信息裸奔

3. 不安全
通信使用明文（不加密），内容可能会被窃听。比如，账号信息容易泄漏
不验证通信方的身份，因此有可能遭遇伪装。比如，访问假的淘宝、拼多多
无法证明报文的完整性，所以有可能已遭篡改。比如，网页上植入垃圾广告，视觉污染

HTTP 的安全问题，可以用 HTTPS 的方式解决，也就是通过引入 SSL/TLS 层，使得在安全上达到了极致

第6节 HTTP 与 HTTPS

1. HTTP 是超文本传输协议，信息是明文传输，存在安全风险的问题。HTTPS 则解决 HTTP 不安全的缺陷，在 TCP 和 HTTP 网络层之间加入了 SSL/TLS 安全协议，使得报文能够加密传输。

2. HTTP 连接建立相对简单， TCP 三次握手之后便可进行 HTTP 的报文传输。而 HTTPS 在 TCP 三次握手之后，还需进行 SSL/TLS 的握手过程，才可进入加密报文传输。

3. HTTP 的端口号是 80，HTTPS 的端口号是 443。

4. HTTPS 协议需要向 CA（证书权威机构）申请数字证书，来保证服务器的身份是可信的

第七章 Servlet组件

第1节 Servlet是什么

1. 从广义上来讲，Servlet规范是Sun公司制定的一套技术标准，包含与Web应用相关的一系列接口，是Web应用实现方式的宏观解决方案。而具体的Servlet容器负责提供标准的实现

2. 从狭义上来讲，Servlet指的是javax.servlet.Servlet接口及其子接口，也可以指实现了Servlet接口的实现类

3. Servlet作为服务器端的一个组件，它的本意是“服务器端的小程序”。Servlet的实例对象由Servlet容器负责创建;Servlet的方法由容器在特定情况下调用;Servlet容器会在Web应用卸载时销毁Servlet对象的实例

第2节 Servlet的技术体系

1. Servlet接口:是sun公司针对于web技术提供的一套标准(接口)
2. GenericServlet:对servlet功能进行了封装和完善，将service方法保留为抽象方法,让使用者仅关心业务实现。
3. HttpServlet抽象类是对GenericServlet进一步封装和扩展更贴近HTTP协议下的应用程序编写，在service方法中根据不同HTTP请求类型调用专门的方法进行处理
4. 今后在实际使用中我们只需要继承HttpServlet并且实现doget和dopost方法即可。
5. ServletConfig接口：封装了Servlet配置信息

第3节 ServletConfig配置获取

```
// 在web.xml中配置

<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>com.qianfeng.servlet.HelloServlet</servlet-class>
    <!-- 我们可以在servlet中添加配置信息 -->
    <init-param>
        <param-name>userName</param-name>
        <param-value>admin123</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/HelloServlet</url-pattern>
</servlet-mapping>
```

```
在servlet组件中获取配置信息

//获取上下文ServletConfig
ServletConfig servletConfig = getServletConfig();
String userName = servletConfig.getInitParameter("userName");
System.out.println(userName);
```

第4节 Servlet使用

- 使用Servlet接口实现Web的方式一

```
1. 使用eclipse创建JavaWeb动态工程 (2.5版本的web项目)
2. 添加jar包 javaee-api-7.0.jar
3. 创建MyServlet类，extend Servlet接口
4. 在web.xml中注册MyServlet类
5. 创建请求页面,编写业务代码
```

- 使用GenericServlet类实现Web的方式二

```
GenericServlet类是对Servlet接口的进步一封装，并且将service方法抽象化，让用户将重心放在具体的业务上
```

- 使用HttpServlet类实现Web的方式三

```
HttpServlet类是对GenericServlet类的更进步一封装，并且抽象出doGet、doPost方法，更方便用户处理前端发送过来的GET/POST请求


request对象的常用方法?
1. String getParamter(String name) : 返回指定name参数的参数值
2. String[] getParameterValues(String name) : 返回包含name参数的所有值
3. void setAttribute(String, object) : 存储此请求域中的数据
4. Object getAttribute(String) : 从域中取出对象
5. String getProtocol() : 返回请求中的协议类型及版本号
```

```
6. String getServerName() : 返回接受请求的服务器主机名
7. int getServerPort() : 获取服务器对应的端口号
8. String getCharacterEncoding() : 返回字符集编码
9. void setCharacterEncoding() : 设置请求的字符集编码
10. int getContentLength() : 返回请求体的长度【字节数】
11. String getRemoteAddr() : 返回发送此请求的客户端IP地址
```

第5节 转发和重定向

- 请求转发

```
1. Servlet接受浏览器发送过来的请求之后进行初步处理而不是直接响应给前端页面，而是在服务器内部转发给其他的servlet程序继续处理，这种情况下浏览器只发出了一次请求，浏览器地址栏不会发生变化，用户不会感知到地址栏被转发
2. 转发请求的Servlet和目标Servlet共享同一个request对象
3. 转发可以访问WEB-INF目录下的资源

转发语法:
request.getRequestDispatcher("/ok.html").forward(request, response);
```

- 请求重定向

```
1. Servlet接收到浏览器端请求并处理完成后，给浏览器端一个特殊的响应，这个特殊的响应要求浏览器去请求一个新的资源，整个过程中浏览器端会发出两次请求，且浏览器地址栏会改变为新资源的地址
2. 重定向的情况下，因为发送两次请求,所以原Servlet和目标资源之间就不能共享请求数据了
3. 重定向不能访问WEB-INF下的资源(因为重定向是客户端发出的动作)

重定向语法:
resp.sendRedirect("ok.html");
```

- 转发和重定向的区别

转发和重定向		
	转发	重定向
浏览器地址栏	不改变	改变
发送请求次数	1	2
能否共享request对象数据	是	否
目标资源：WEB-INF下的资源	能访问	不能访问

第6节 登录例子练习

第八章 JSP技术

第1节 动态网页技术

- 常见的动态网页技术

```
1. ASP  Microsoft
2. PHP
3. JSP
```

- 动态网页技术解决的问题

```
1. Html：用来显示页面，缺点：不能使用变量和java代码
2. Servlet:可以写页面，也可以在页面中使用变量,写代码方便，但是写页面麻烦
3. JSP：html+servlet强强联合，在页面中可以使用html标签也可以使用java代码
```

第2节 JSP简介

- JSP是什么

```
JSP全名Java Server Pages，根本就是一个简化的Servlet,主要实现了在Java当中使用HTML标签。
Jsp是一种动态网页的技术,标准也是JavaEE的标准,Jsp与Servlet一样,是在服务端运行的
```

- JSP特点

```
1. 其本身是一个动态网页技术标准，它的主要构成有HTML网页代码、Java代码片段、JSP标签几部分组成，后缀是.jsp
2. JSP相比HTML页面来说，最直观的功能就是可以在页面中使用变量，这些变量一般都是从域对象中获取。有了变量的好处就是我们的页面可以动态的显示信息
3. 相比于Servlet，JSP更加善于处理显示页面，而Servlet跟擅长处理业务逻辑，两种技术各有专长，所以一般我们会将Servlet和JSP结合使用，Servlet负责业务，JSP负责显示
```

第3节 JSP页面构成

```
1. 指令
2. 注释
3. 小脚本(在JSP页面中执行的Java代码)
4. 声明(在JSP页面中定义一些变量或者方法)
5. 表达式(可以在JSP页面中执行表达式)
6. 静态内容(HTML/CSS/JavaScript等)
```

- 指令

```
1. page
    -> 位于JSP页面的顶端,同一个页面可以有多个page
    -> language:java
    -> import : 导包
    -> contentType: 内容和编码
2. include
    -> 将一个外部文件嵌入到当前的JSP文件中，同时解析这个页面中的JSP语句
3. taglib指令
    -> 使用标签库定义新的自定义标签，在JSP页面中启动定制行为
```

- 注释

```
1. HTML注释    <!--注释-->
2. JSP注释     <%-- 注释--%>  因为JSP中可以写Java语言,所以也可以使用Java语言注释
```

- 小脚本

```
<% Java代码 %>
<% out.println(“大家好，我是一个JSP脚本"); %>
```

- 声明

```
<%! Java代码; %>
-> 声明变量
-> 声明方法

例子:

<%!
    int i=10;
    public int add(int a,int b){return a+b;}
%>
<%=i %>
<%= add(100,300) %>
```

- 表达式

```
<%= 表达式 %>    表达式不以分号结束
```

- 静态内容

```
HTML、CSS、JavaScript等
```

第4节 JSP的生命周期

```
JSP生命周期就是从创建到销毁的整个过程，类似于servlet生命周期，区别在于JSP生命周期还包括将JSP文件编译成servlet
```

- JSP生命周期的几个阶段
- ◦ 1. 编译阶段

```
Servlet容器编译JSP源文件，生成servlet类
```

- ◦ 2. 初始化阶段

```
加载与JSP对应的Servlet类，创建其实例，并调用它的初始化方法
```

- ◦ 3. 执行阶段

```
调用与JSP对应的Servlet实例的服务方法
```

- ◦ 4. 销毁阶段

```
调用与JSP对应的Servlet实例的销毁方法，然后销毁Servlet实例
```

第5节 JSP九大隐含对象

JSP内置对象是web容器创建的一组对象，不使用new关键字就可以使用的内置对象

- out : JspWriter实例，相当于response.getWriter()获取的对象，用于在页面中显示信息

```
9*9乘法表

1*1=1
2*1=2    2*2=4
3*1=3    3*2=6    3*3=9
4*1=4    4*2=8    4*3=12    4*4=16
5*1=5    5*2=10    5*3=15    5*4=20    5*5=25
6*1=6    6*2=12    6*3=18    6*4=24    6*5=30    6*6=36
7*1=7    7*2=14    7*3=21    7*4=28    7*5=35    7*6=42    7*7=49
8*1=8    8*2=16    8*3=24    8*4=32    8*5=40    8*6=48    8*7=56    8*8=64
9*1=9    9*2=18    9*3=27    9*4=36    9*5=45    9*6=54    9*7=63    9*8=72
9*9=81


<%
    for(int i=1;i<=9;i++){
        for(int j=1;j<=i;j++){
            out.print(i+"*"+j+"="+i*j+"    ");
        }
        out.print("<br/>");
    }

%>
```

- request : HttpServletRequest对象

```
表示一次的客户端请求
客户端的请求信息被封装进request对象中
是HttpServletRequest的实例对象
request具有请求域,在一个请求没有完成之前，该对象一直有效

request对象的常用方法?
1. String getParamter(String name) : 返回指定name参数的参数值
2. String[] getParameterValues(String name) : 返回包含name参数的所有值
3. void setAttribute(String, object) : 存储此请求域中的数据
4. Object getAttribute(String) : 从域中取出对象
5. String getProtocol() : 返回请求中的协议类型及版本号
6. String getServerName() : 返回接受请求的服务器主机名
7. int getServerPort() : 获取服务器对应的端口号
8. String getCharacterEncoding() : 返回字符集编码
9. void setCharacterEncoding() : 设置请求的字符集编码
10. int getContentLength() : 返回请求体的长度【字节数】
11. String getRemoteAddr() : 返回发送此请求的客户端IP地址
13. getRequestDispatcher("/转发的地址").forward(request, response) : 请求转发
```

- response : HttpServletResponse对象

```
1. 向前端打印消息 response.getWriter()
2. 用于重定向 response.sendRedirect(目标地址)
3. 设置字符编码
    - response.setCharacterEncoding("UTF-8");
    - response.setContentType("text/html;charset=UTF-8");
```

- session : HttpSession对象

```
表示客户端与服务端的一次会话
web中session指的是用户浏览某个网站，从进入网站到浏览结束关闭浏览器所经过的这段时间
session应该是一个特定的时间概念
在第一个jsp页面被装载时自动创建，到客户关闭浏览器离开服务器结束
session是HttpSession的实例

session的常用方法
1. long getCreateTime() : 返回session创建时间
2. public String getId() : 返回session创建时jsp引擎为他创建的唯一id
3. public setAttribute(String,Object) : 将一个对象设置到session会话中
4. Object getAttribute(String name) : 取出session中的对象
5. String[] getValueNames() ： 返回此session中多所有可用属性的数组
6. int getMaxInactiveInterval() ： 返回两次请求间隔多长时间此session被取消(单位秒)
7. void setMaxInactiveInterval(10) ： 设置超时时间


session销毁的方式
1.调用session.invalidate()
2. session过期
    ->tomcat默认超时时间是30分钟
    ->在web.xml中配置session过期时间
3. 服务器重启

web.xml配置session过期时间（分钟）
<session-config>
  <session-timeout>10</session-timeout>
</session-config>
```

- application : ServletContext对象

```
实现用户间数据共享，可以存放全局变量
开始于服务器启动，结束于服务器终止
在相同用户或者不同用户之间的连接中，可以对application进行同一个属性值的操作
在任何地方对此对象操作都会影响其他用户对此访问
服务器的启动和关闭决定了此对象的生命周期

常用方法:
1. void setAttribute(Srting,object) : 向域中设置数据
2. Object getAttribute(String name) : 从域中拿出数据
3. String getServerInfo() : 返回JSP引擎名和版本号

统计当前页面的pv(页面访问量)

<%
    if(application.getAttribute("count")!=null){
        int count = (Integer)application.getAttribute("count");
        application.setAttribute("count", count+1);
    }else{
```

```
        application.setAttribute("count", 0);
    }
%>

<%= application.getAttribute("count") %>
```

- page : object对象,对应当前Servlet对象，实际上就是this
- pageContext : 当前页面的上下文

```
获取绝对路径
pageContext.request.contextPath
```

- config : 对应Servlet中的ServletConfig对象
- exception : 错误页面中异常对象

第九章 EL/JSTL表达式

第1节 EL表达式简介

```
1. EL是JSP内置的表达式语言，用以访问页面的上下文以及不同作用域中的对象，取得对象属性的值，或执行简单的运算或判断操作。EL在得到某个数据时，会自动进行数据类型的转换
2. EL表达式用于代替JSP表达式(<%= %>)在页面中做输出操作
3. EL表达式仅仅用来读取数据，而不能对数据进行修改
4. 使用EL表达式输出数据时，如果有则输出数据，如果为null则什么也不输出
```

第2节 EL表达式的使用

- EL表达式的语法格式

```
${表达式}
```

- EL表达式获取域中数据的方式

```
1. 获取域中对象 ${user}
2. 获取域中对象的属性${user.name}（注意:此name不是类中的成员变量的属性名,而是set/get方法的getName名字）
```

- EL表达式的常见隐含对象

```
1. requestScope
2. sessionScope
3. applicationScope
```

- EL表达式获取指定域中数据的方式

在EL表达式中如果我们直接使用属性名如：${user}，它将会在四个域中由小到大依次查找。顺序：requestScope、sessionScope、applicationScope

还可以从指定的域中直接回去数据

1. ${requestScope.user}：当前请求
2. ${sessionScope.user}：当前会话
3. ${applicationScope.user}：当前应用

第3节 EL表达式运算符

第4节 JSTL标签库

JSTL（JavaServer Pages Standard Tag Library，JSP标准标签库）

1. JSP虽然为我们提供了EL表达式用来替代JSP表达式，但是由于EL表达式仅仅具有输出功能，而不替代页面中的JSP脚本片段
2. 为了解决这个问题，JSP为我们提供了可以自定义标签库(Tag Library)的功能
3. 所谓自定义标签库就是指可以在JSP页面中以类似于HTML标签的形式调用Java中的方法。使用方法和我们JSP动作标签类似
4. 为了方便开发使用Sun公司又定义了一套通用的标签库名为JSTL(JSP Standard Tag Library)，里面定义很多我们开发中常用的方法，方便我们使用
5. JSTL由5个不同功能的标签库组成

- JSTL使用

1. 导入jar包或者依赖
2. 在JSP页面通过taglib标签引入标签库

- JSTL标签库
- JSTL常用标签

第5节 JSTL时间格式化

导入fmt核心库

```
<fmt:formatDate value="${date.date }" pattern="yyyy-MM-dd HH:mm:ss"/>
```

第十章 Filter过滤器

1. Filter中文意思为过滤器。顾名思义，过滤器可在浏览器以及目标资源之间起到一个过滤的作用
2. 对于WEB应用来说，过滤器是一个驻留在服务器中的WEB组件，他可以截取客户端和WEB资源之间的请求和响应信息
3. WEB资源可能包括Servlet、JSP、HTML页面等

Filter的使用:
1. 自定义类，并且实现Filter接口
2. 在web.xml中进行注册(注册方式和servlet类似 (在拦截时指定拦截的url地址,可以设置成 /* 拦截全部))
3. dofilter方法 里面有一个chain.doFilter(request, response);方法是调用下一个拦截器的，如果不放行不能进行下一步的动作

Filter的生命周期:
1. 构造器：创建Filter实例是调用，Filter实例服务器一旦启动就会被创建
2. init()：实例创建后马上被调用，用来对Filter做一些初始化的操作
3. doFilter()：Filter的主要方法，用来完成过滤器主要功能的方法，每次访问目标资源时都会调用
4. destroy()：服务器停止时调用，用来释放资源

第十一章 Listener监听器(了解)

第1节 Listener简介

1. 用于监听JavaWeb程序中的事件
2. 比如ServletContext、HttpSession、ServletRequest的创建、修改和删除
3. 当监听的某些事件发生变化时被调用

第2节 观察者模式

1. Listener的原理是基于观察者模式的，所谓观察者模式简单来说，就是当被观察者的特定事件被触发（一般这某些方法被调用）后，会通知观察者（调用观察者的方法），观察者可以在自己的方法中来对事件做一些处理
2. 在我们的JavaWeb中，观察者就是Listener，而被观察者可能有三个ServletContext、HttpSession、ServletRequest。而事件指的就是这些对象的创建、修改和删除等

第3节 监听器分类

- 监听对象

1. ServletContextListener
2. HttpSessionListener
3. ServletRequestListener

- 监听对象中属性的变化

1. ServletContextAttributeListener
2. HttpSessionAttributeListener
3. ServletRequestAttributeListener

第4节 监听器举例

以监听请求域中的属性变化为例ServletRequestAttributeListener

1. 创建MyListener类实现ServletRequestAttributeListener接口

```
public class MyListener implements ServletRequestAttributeListener {
    public MyListener() {
        System.out.println("ServletRequestAttributeListener监听被创建了");
    }
    public void attributeRemoved(ServletRequestAttributeEvent srae)  {
        System.out.println("属性被移除了");
    }
    public void attributeAdded(ServletRequestAttributeEvent srae)  {
        System.out.println("属性被添加了");
    }
    public void attributeReplaced(ServletRequestAttributeEvent srae)  {
        System.out.println("属性被改变了");
    }
}

2. Servlet中测试
public void doGet(ServletRequest req, ServletResponse res) throws
ServletException, IOException {
    req.setAttribute("name", "lilei");
    //修改属性值
    //req.setAttribute("name", "lilei11111");
}
```

第十二章 文件上传和下载

```
1. 文件的上传在web应用中是很常见的操作,比如我们注册用户时上传头像,或者我们发送邮件时，有的需要发送附件,那个附件需要从我们本机上传到服务器.
2. 文件的下载,一般情况下只要资源存放在用户可访问的目录中,用户就可以直接通过地址下载
3. 上传/下载需要添加common-io/common-fileupload两个jar包
```

第1节 文件上传

- 文件上传的步骤

```
1. 用户在页面中选择要上传的文件，然后将请求提交到Servlet
2. Servlet收到请求，解析用户上传的文件，然后将文件存储到服务器
3. jar
   - commons-fileupload-1.4.jar
   - commons-io-2.6.jar
```

- 文件上传代码操作
- - 创建上传文件的表单

```
<form action="#" method="post" enctype="multipart/form-data">
    <input type="file" name="file"/><br><br>
    <input type="submit" value="上传"/>
</form>

1. 表单的method属性必须为post
2. 表单的enctype属性必须为multipart/form-data
3. 上传文件的控件是intput，type属性为file
```

- - 编写文件上传代码

```
DiskFileItemFactory factory = new DiskFileItemFactory();
ServletFileUpload fileUpload = new ServletFileUpload(factory);
//设置单个文件为3M
fileUpload.setFileSizeMax(1024*1024*3);
//总文件大小为30M
fileUpload.setSizeMax(1024*1024*3*10);
try {
    //获取所有的表单项(普通表单项/文件表单项)
    List<FileItem> list = fileUpload.parseRequest(request);
    for (FileItem fileItem : list) {
        //普通表单提交
        if(fileItem.isFormField()) {
            //获取表单的name名
            String fieldName = fileItem.getFieldName();//获取元素名称
            //获取表单的value值
            String value = fileItem.getString("UTF-8"); //获取元素值
            System.out.println(fieldName+" : "+value);
        }else {
            //文件上传表单
            //获取文件名
            String name = fileItem.getName();
            //防止重名,使用时间戳
            String id = new Date().getTime()+"";
            //重新生成文件名称
            name = id + name;
            //上传到指定位置
            String realPath = getServletContext().getRealPath("/upload");
            System.out.println(realPath);
            //获取路径
            File f = new File(realPath);
            //判断是否存在
            if(!f.exists()){
                //如果不存在就创建
                f.mkdir();
            };
            //将文件写到服务器当中
            fileItem.write(new File(realPath+"/"+name));
            //清空缓存
            fileItem.delete();
        }

}catch (Exception e) {
    e.printStackTrace();
}
```

第2节 文件下载

```
//获取文件真实路径
String realPath = getServletContext().getRealPath("/123.pdf");
//获取文件的抽象对象
File file = new File(realPath);
//获取文件输入流
FileInputStream inputStream = new FileInputStream(file);
//获取文件类型
String mimeType = getServletContext().getMimeType(realPath);
//设置内容格式
```

```
response.setContentType(mimeType);
/**
 * 设置Content-Disposition
 *  设置fileName
 */
String fileName="123.pdf";
String disposition = "attachment;filename="+fileName;
response.setHeader("Content-Disposition", disposition);
//设置内容大小
long size = file.length();
response.setContentLength((int)size);
//发送文件
ServletOutputStream outputStream = response.getOutputStream();
byte[] c = new byte[1024];
int len=0;
while((len=inputStream.read(c))!=-1) {
    outputStream.write(c, 0, len);
}
inputStream.close();
```

- 解决各浏览器中文名字乱码问题

```
String fileName="中国.pdf";
//获取浏览器代理
String ua = request.getHeader("User-Agent");
if(ua.contains("Firefox")) {
    //火狐浏览器
    fileName = "=?utf-8?B?"+new BASE64Encoder().encode(fileName.getBytes("utf-
8"))+"?=";
}else {
    //谷歌及其他
    fileName = java.net.URLEncoder.encode(fileName,"utf-8");
}
```

第十三章 json

第1节 json的介绍

JSON(JavaScript Object Notation, JS 对象简谱) 是一种轻量级的数据交换格式，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率.

第2节 json的数据格式

- 对象

```
{"name": "John Doe", "age": 18}
```

- 数组

```
[3, 1, 4, 1, 5, 9, 2, 6]
```

- 对象数据混合

```
{"a": 1, "b": [1, 2, 3]}
[1, 2, "3", {"a": 4}]
```

第3节 常用的json工具

- fastjson

```
官网: https://github.com/alibaba/fastjson/wiki/Quick-Start-CN

使用方式:

1. 添加jar包 fastjson-1.2.68.jar
2. 调用API方法

POJO--> JSON
String text = JSON.toJSONString(obj); //序列化

JSON--> POJO
VO vo = JSON.parseObject("{...}", VO.class); //反序列化


fastjson时间格式化:

Person person = new Person(100, "user01", new Date());
String format = JSON.toJSONStringWithDateFormat(person, "yyyy-MM-dd HH:mm:ss",
SerializerFeature.WriteDateUseDateFormat);
或者:
String format = JSON.toJSONString(person,
SerializerFeature.WriteDateUseDateFormat);

System.out.println(format);
{"createTime":"2019-09-18 19:49:35","pId":100,"pName":"user01"}
```

- gson

```
使用方式:

1. 添加jar包  gson-2.8.6.jar
2. 调用API方法

POJO--> JSON
Gson gson = new Gson();
User user = new User("张三",24);
String jsonObject = gson.toJson(user);

JSON--> POJO
Gson gson = new Gson();
String jsonString = "{\"name\":\"张三\",\"age\":24}";
User user = gson.fromJson(jsonString, User.class);


日期格式化:

List<Person> pList = new ArrayList<>();
```

```
Person p1 = new Person(1001,"李雷1",new Date());
Person p2 = new Person(1002,"李雷2",new Date());
Person p3 = new Person(1003,"李雷3",new Date());
pList.add(p1);
pList.add(p2);
pList.add(p3);
Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd HH:mm:ss").create();
String json = gson.toJson(pList);
System.out.println(json);
```

- Jackson

```
jackson-core-2.2.3.jar（核心jar包）
jackson-annotations-2.2.3.jar（该包提供Json注解支持）
jackson-databind-2.2.3.jar


//创建ObjectMapper
ObjectMapper om = new ObjectMapper();

//创建测试数据
List<Person> pList = new ArrayList<>();
Person p1 = new Person(1001,"李雷1",new Date());
Person p2 = new Person(1002,"李雷2",new Date());
Person p3 = new Person(1003,"李雷3",new Date());

pList.add(p1);
pList.add(p2);
pList.add(p3);

//序列化对象,将对象序列化成json字符串  writeValueAsString
String pJsons = om.writeValueAsString(pList);
System.out.println(pJsons);
//对带有时间格式的序列化(自定义时间格式)
//设置时间格式
om.setDateFormat(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"));
String pJson = om.writeValueAsString(p1);
System.out.println(pJson);

//反序列化对象
Person list = om.readValue(pJson, Person.class);
System.out.println(list);
```

- 后台返回JSON数据中文乱码解决

```
response.setCharacterEncoding("utf-8");
response.setContentType("text/html; charset=utf-8");
```

第十四章 Ajax技术

第1节 AJAX简介

```
1. AJAX 是 Asynchronous([eɪˈsɪŋkrənəs]) JavaScript And XML的简称。直译为，异步的JS和XML
2. AJAX的实际意义是，不发生页面跳转、异步载入内容并改写页面内容的技术
3. AJAX也可以简单的理解为通过JS向服务器发送请求
```

第2节 AJAX使用

- XMLHttpRequest对象简介

```
1. XMLHttpRequest对象是AJAX中非常重要的对象，所有的AJAX操作都是基于该对象的
2. XMLHttpRequest对象用来封装请求报文，我们向服务器发送的请求信息全部都需要封装到该对象中
3. 这里需要稍微注意一下，XMLHttpRequest对象并没有成为标准，但是现在的主流浏览器都支持该对象，而一些如IE6的老版本浏览器中的创建方式有一些区别
```

- XMLHttpRequest对象创建

```
1. var xhr = new XMLHttpRequest():目前主流浏览器都支持
2. var xhr = new ActiveXObject("Msxml2.XMLHTTP"):IE6支持的方式（了解）
3. var xhr = new ActiveXObject("Microsoft.XMLHTTP"):IE5.5一下支持的方式（了解）
```

- AJAX发送HTTP请求

第3节 jQuery实现AJAX技术

```
$.get(url,[data],[fn],[type])
- url:请求URL地址
- data:待发送 Key/value 参数。
- callback:载入成功时回调函数。
- type:返回内容格式，xml, html, script, json, text, _default。

**************************************

$.getJSON(url,[data],[fn])
- url:发送请求地址。
- data:待发送 Key/value 参数。
- callback:载入成功时回调函数。

**************************************

$.post(url,[data],[fn],[type])
- url:发送请求地址。
- data:待发送 Key/value 参数。
- callback:发送成功时回调函数。
- type:返回内容格式，xml, html, script, json, text, _default。
```

第十五章 Bootstrap框架

```
简洁、直观、强悍的前端开发框架，让web开发更迅速、简单
```

- 中文官网

```
https://v3.bootcss.com/
```

第1节 HTML5文档类型

```
Bootstrap 使用到的某些 HTML 元素和 CSS 属性需要将页面设置为 HTML5 文档类型。在你项目中的每个页面都要参照下面的格式进行设置
```

```
<!DOCTYPE html>
<html lang="zh-CN">
  ...
</html>
```

- 环境搭建 导入css/js/jquery
- BootStrap模板

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
    <title>Bootstrap 101 Template</title>

    <!-- Bootstrap -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.css"
rel="stylesheet">

    <!-- HTML5 shim 和 Respond.js 是为了让 IE8 支持 HTML5 元素和媒体查询（media
queries）功能 -->
    <!-- 警告：通过 file:// 协议（就是直接将 html 页面拖拽到浏览器中）访问页面时
Respond.js 不起作用 -->
    <!--[if lt IE 9]>
      <script
src="https://cdn.jsdelivr.net/npm/html5shiv@3.7.3/dist/html5shiv.min.js">
</script>
      <script
src="https://cdn.jsdelivr.net/npm/respond.js@1.4.2/dest/respond.min.js">
</script>
    <![endif]-->
  </head>
  <body>
    <h1>你好，世界！</h1>

    <!-- jQuery (Bootstrap 的所有 JavaScript 插件都依赖 jQuery，所以必须放在前边) -->
    <script src="https://cdn.jsdelivr.net/npm/jquery@1.12.4/dist/jquery.min.js">
</script>
    <!-- 加载 Bootstrap 的所有 JavaScript 插件。你也可以根据需要只加载单个插件。 -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/js/bootstrap.min.js">
</script>
  </body>
</html>
```

第2节 移动设备优先

在 Bootstrap 2 中，我们对框架中的某些关键部分增加了对移动设备友好的样式。而在 Bootstrap3 中，重写了整个框架，使其一开始就是对移动设备友好的。

为了确保适当的绘制和触屏缩放，需要在 head 之中添加 viewport 元数据标签

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

在移动设备浏览器上，通过为视口（viewport）设置 meta 属性为 user-scalable=no 可以禁用其缩放（zooming）功能。这样禁用缩放功能后，用户只能滚动屏幕，就能让你的网站看上去更像原生应用的感觉。

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
```

第3节 布局容器

Bootstrap 需要为页面内容和栅格系统包裹一个.container容器。我们提供了两个作此用处的类。注意，由于padding等属性的原因，这两种容器类不能互相嵌套.

- .container 类用于固定宽度并支持响应式布局的容器。

```
<div class="container">
  ...
</div>
```

- .container-fluid 类用于100%宽度，占据全部视口（viewport）的容器。

```
<div class="container-fluid">
  ...
</div>
```

第4节 栅格系统

Bootstrap 提供了一套响应式、移动设备优先的流式栅格系统，随着屏幕或视口（viewport）尺寸的增加，系统会自动分为最多12列。

4.1 简介

栅格系统用于通过一系列的行（row）与列（column）的组合来创建页面布局，你的内容就可以放入这些创建好的布局中。

- "行（row）"必须包含在 .container （固定宽度）或 .container-fluid （100% 宽度）中，以便为其赋予合适的排列（aligment）和内补（padding）。
- 通过"行（row）"在水平方向创建一组"列（column）"。
- 你的内容应当放置于"列（column）"内，并且，只有"列（column）"可以作为行（row）"的直接子元素。

- 栅格系统中的列是通过指定1到12的值来表示其跨越的范围。例如，三个等宽的列可以使用三个 .col-xs-4 来创建
- 如果一“行（row）”中包含了的“列（column）”大于12，多余的“列（column）”所在的元素将被作为一个整体另起一行排列。

4.2 栅格参数

	超小屏幕 手机 (<768px)	小屏幕 平板 (≥768px)	中等屏幕 桌面显示器 (≥992px)	大屏幕 大桌面显示器 (≥1200px)
栅格系统行为	总是水平排列	开始是堆叠在一起的，当大于这些阈值时将变为水平排列		
.container 最大宽度	None（自动）	750px	970px	1170px
类前缀	.col-xs-	.col-sm-	.col-md-	.col-lg-
列（column）数	12			
最大列（column）宽	自动	~62px	~81px	~97px
槽（gutter）宽	30px（每列左右均有 15px）			
可嵌套	是			
偏移（Offsets）	是			
列排序	是			

4.3 从堆叠到水平排列

使用单一的一组 `.col-md-*`栅格类，就可以创建一个基本的栅格系统，在手机和平板设备上一开始是堆叠在一起的（超小屏幕到小屏幕这一范围），在桌面（中等）屏幕设备上变为水平排列。所有“列（`column`）必须放在 ” `.row` 内

```
<div class="row">
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
```

```
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
</div>
<div class="row">
  <div class="col-md-8">.col-md-8</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-6">.col-md-6</div>
  <div class="col-md-6">.col-md-6</div>
</div>
```

4.4 流式布局容器

将最外面的布局元素.container修改为.container-fluid，就可以将固定宽度的栅格布局转换为 100% 宽度的布局

```
<div class="container-fluid">
  <div class="row">
    ...
  </div>
</div>
```

4.5 移动设备和桌面屏幕

是否不希望在小屏幕设备上所有列都堆叠在一起？那就使用针对小屏幕和大屏幕设备所定义的类吧，即.col-lg-*和.col-md-*。请看下面的实例，研究一下这些是如何工作的。

```
<div class="row">
  <div class="col-lg-6 col-md-12" style="background-color: red;">col-lg-6 col-md-12</div>
  <div class="col-lg-6 col-md-12" style="background-color: #1B6D85;">col-lg-6 col-md-12</div>
</div>
```

4.6 列偏移

使用 .col-md-offset-*类可以将列向右侧偏移。这些类实际是通过使用*选择器为当前元素增加了左侧的边距（margin）。例如，.col-md-offset-4类将 .col-md-4 元素向右侧偏移了4个列（column）的宽度。

```
<div class="row">
  <div class="col-md-4" style="background-color: red;">.col-md-4</div>
  <div class="col-md-4 col-md-offset-4" style="background-color: blue;">.col-md-
4 .col-md-offset-4</div>
</div>
<div class="row">
  <div class="col-md-3 col-md-offset-3" style="background-color: yellow;">.col-
md-3 .col-md-offset-3</div>
  <div class="col-md-3 col-md-offset-3" style="background-color:
yellowgreen;">.col-md-3 .col-md-offset-3</div>
</div>
<div class="row">
  <div class="col-md-6 col-md-offset-3" style="background-color:
blueviolet;">.col-md-6 .col-md-offset-3</div>
</div>
```

4.7 嵌套列

为了使用内置的栅格系统将内容再次嵌套，可以通过添加一个新的.row元素和一系列 .col-sm-*元素到已经存在的.col-sm-*元素内。被嵌套的行（row）所包含的列（column）的个数不能超过12（其实，没有要求你必须占满12列）。

```
<div class="row">
  <div class="col-sm-9">
    Level 1: .col-sm-9
    <div class="row">
      <div class="col-xs-8 col-sm-6">
        Level 2: .col-xs-8 .col-sm-6
      </div>
      <div class="col-xs-4 col-sm-6">
        Level 2: .col-xs-4 .col-sm-6
      </div>
    </div>
  </div>
</div>
```

第5节 排版

5.1 标题

- HTML 中的所有标题标签，h1 到 h6 均可使用。

```
<h1>h1. Bootstrap heading</h1>
<h2>h2. Bootstrap heading</h2>
<h3>h3. Bootstrap heading</h3>
<h4>h4. Bootstrap heading</h4>
<h5>h5. Bootstrap heading</h5>
<h6>h6. Bootstrap heading</h6>
```

- 在标题内还可以包含small标签或赋予.small类的元素，可以用来标记副标题

```
<h1>h1. Bootstrap heading <small>Secondary text</small></h1>
<h2>h2. Bootstrap heading <small>Secondary text</small></h2>
<h3>h3. Bootstrap heading <small>Secondary text</small></h3>
<h4>h4. Bootstrap heading <small>Secondary text</small></h4>
<h5>h5. Bootstrap heading <small>Secondary text</small></h5>
<h6>h6. Bootstrap heading <small>Secondary text</small></h6>
```

5.2 页面主体

Bootstrap 将全局 font-size设置为14px，line-height设置为1.428。这些属性直接赋予 body元素和所有段落元素。另外，p（段落）元素还被设置了等于 1/2 行高（即 10px）的底部外边距（margin）。

```
<p>...</p>
```

5.3 标记文本

- 将文本使用mark标签标注

```
You can use the mark tag to <mark>highlight</mark> text.
```

5.4 被删除的文本

```
<del>This line of text is meant to be treated as deleted text.</del>
```

5.5 无用文本

- 对于没用的文本使用 s 标签。

```
<s>This line of text is meant to be treated as no longer accurate.</s>
```

5.6 插入文本

- 额外插入的文本使用 ins 标签。

```
<ins>This line of text is meant to be treated as an addition to the document.
</ins>
```

5.7 带下划线的文本

- 为文本添加下划线，使用 u 标签

```
<u>This line of text will render as underlined</u>
```

5.8 对齐

```
<p class="text-left">Left aligned text.</p>
<p class="text-center">Center aligned text.</p>
<p class="text-right">Right aligned text.</p>
```

5.9 改变大小写

```
<p class="text-lowercase">Lowercased text.</p>
<p class="text-uppercase">Uppercased text.</p>
<p class="text-capitalize">Capitalized text.</p>
```

第6节 表格

6.1 基本实例

- 为任意 table 标签添加.table类可以为其赋予基本的样式—少量的内补（padding）和水平方向的分隔线.

```
<table class="table">
    <thead>
        <tr>
            <th>id</th>
            <th>name</th>
            <th>age</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>1001</td>
            <td>李雷</td>
            <td>10</td>
        </tr>
    </tbody>
</table>
```

6.2 条纹状表格

- 通过 .table-striped 类可以给 tbody 之内的每一行增加斑马条纹样式

```
<table class="table table-striped">
    ...
</table>
```

6.3 带边框的表格

- 添加 .table-bordered 类为表格和其中的每个单元格增加边框

```
<table class="table table-bordered">
  ...
</table>
```

6.4 鼠标悬停

- 通过添加 .table-hover 类可以让 tbody 中的每一行对鼠标悬停状态作出响应

```
<table class="table table-hover">
  ...
</table>
```

6.5 紧缩表格

```
<table class="table table-condensed">
  ...
</table>
```

6.6 状态类

- 通过这些状态类可以为行或单元格设置颜色。

Class	描述
.active	鼠标悬停在行或单元格上时所设置的颜色
.success	标识成功或积极的动作
.info	标识普通的提示信息或动作
.warning	标识警告或需要用户注意
.danger	标识危险或潜在的带来负面影响的动作

```
<!-- On rows -->
<tr class="active">...</tr>
<tr class="success">...</tr>
<tr class="warning">...</tr>
<tr class="danger">...</tr>
<tr class="info">...</tr>

<!-- On cells (`td` or `th`) -->
<tr>
  <td class="active">...</td>
  <td class="success">...</td>
  <td class="warning">...</td>
  <td class="danger">...</td>
  <td class="info">...</td>
</tr>
```

6.7 响应式表格

- 将任何 .table 元素包裹在.table-responsive元素内，即可创建响应式表格，其会在小屏幕设备上（小于768px）水平滚动。当屏幕大于768px宽度时，水平滚动条消失。

```
<div class="table-responsive">
  <table class="table">
    ...
  </table>
</div>
```

- 注意:Firefox浏览器对fieldset元素设置了一些影响width属性的样式，导致响应式表格出现问题.

第7节 按钮

7.1 可作为按钮使用的标签或元素

为 <a>、<button> 或 <input> 元素添加按钮类（button class）即可使用 Bootstrap 提供的样式

```
<a class="btn btn-default" href="#" role="button">Link</a>
<button class="btn btn-default" type="submit">Button</button>
<input class="btn btn-default" type="button" value="Input">
<input class="btn btn-default" type="submit" value="Submit">
```

7.2 预定义样式

- 使用下面列出的类可以快速创建一个带有预定义样式的按钮。

```
<!-- Standard button -->
<button type="button" class="btn btn-default">（默认样式）Default</button>

<!-- Provides extra visual weight and identifies the primary action in a set of
buttons -->
<button type="button" class="btn btn-primary">（首选项）Primary</button>

<!-- Indicates a successful or positive action -->
<button type="button" class="btn btn-success">（成功）Success</button>

<!-- Contextual button for informational alert messages -->
<button type="button" class="btn btn-info">（一般信息）Info</button>

<!-- Indicates caution should be taken with this action -->
<button type="button" class="btn btn-warning">（警告）Warning</button>

<!-- Indicates a dangerous or potentially negative action -->
<button type="button" class="btn btn-danger">（危险）Danger</button>

<!-- Deemphasize a button by making it look like a link while maintaining button
behavior -->
<button type="button" class="btn btn-link">（链接）Link</button>
```

7.3 尺寸

```
<p>
  <button type="button" class="btn btn-primary btn-lg">（大按钮）Large
button</button>
  <button type="button" class="btn btn-default btn-lg">（大按钮）Large
button</button>
</p>
<p>
  <button type="button" class="btn btn-primary">（默认尺寸）Default
button</button>
  <button type="button" class="btn btn-default">（默认尺寸）Default
button</button>
</p>
<p>
  <button type="button" class="btn btn-primary btn-sm">（小按钮）Small
button</button>
  <button type="button" class="btn btn-default btn-sm">（小按钮）Small
button</button>
</p>
```

```
<p>
  <button type="button" class="btn btn-primary btn-xs">（超小尺寸）Extra small
button</button>
  <button type="button" class="btn btn-default btn-xs">（超小尺寸）Extra small
button</button>
</p>
```

- 通过给按钮添加 .btn-block类可以将其拉伸至父元素100%的宽度，而且按钮也变为了块级（ block ）元素.

```
<button type="button" class="btn btn-primary btn-lg btn-block">（块级元素）Block
level button</button>
<button type="button" class="btn btn-default btn-lg btn-block">（块级元素）Block
level button</button>
```

7.4 图片形状

- 通过为 img 元素添加以下相应的类，可以让图片呈现不同的形状

```
<img src="..." alt="..." class="img-rounded">
<img src="..." alt="..." class="img-circle">
<img src="..." alt="..." class="img-thumbnail">
```

- 注意: IE8不支持 CSS3 圆角属性

7.5 关闭按钮

```
<button type="button" class="close" aria-label="Close"><span aria-
hidden="true">&times;</span></button>
```

7.6 三角符号

- 通过使用三角符号可以指示某个元素具有下拉菜单的功能，反方向请参考组件下的Glyphicons 字体图标

```
<span class="caret"></span>
```

7.7 显示或隐藏内容

- .show 和 .hidden 类可以强制任意元素显示或隐藏

```
<div class="show">...</div>
<div class="hidden">...</div>
```

第十六章 Bootstrap框架+练习(JSP/html/css/jquery)

- 功能介绍

- 技术介绍

```
1. JSP servlet filter mysql
2. EL JSTL
3. jQuery cookie jquery.cookie.js
4. bootstrap
5. 分页 记住我
```

课件用到的资料文件以及练习题静态页面

链接：https://pan.baidu.com/s/15LYN4O_-ahF5KNqQRaLaRQ
提取码：e9td
复制这段内容后打开百度网盘手机App，操作更方便哦