# #tomcat-servlet

1. 软件架构

1. C/S：客户端/服务器端

2. B/S：浏览器/服务器端

2. 资源分类

1. 静态资源：所有用户访问后，得到的结果都是一样的，称为静态资源.静态资源可以直接被浏览器解析

\* 如： html,css,

2. 动态资源:每个用户访问相同资源后，得到的结果可能不一样。称为动态资源。动态资源被访问后，需要先转换为静态资源，在返回给浏览器

\* 如：servlet/jsp,php,asp.... JavaScript

\* web服务器软件：接收用户的请求，处理请求，做出响应。

\* Tomcat：Apache基金组织，中小型的JavaEE服务器，仅仅支持少量的JavaEE规范servlet/jsp。开源的，免费的。

6. 配置:

\* 部署项目的方式：

1. 直接将项目放到webapps目录下即可。

\* /hello：项目的访问路径-->虚拟目录

\* 简化部署：将项目打成一个war包，再将war包放置到webapps目录下。

\* war包会自动解压缩

2. 配置conf/server.xml文件

在<Host>标签体中配置

<Context docBase="D:\hello" path="/hehe" />

\* docBase:项目存放的路径

\* path：虚拟目录

3. 在conf\Catalina\localhost创建任意名称的xml文件。在文件中编写

<Context docBase="D:\hello" />

\* 虚拟目录：xml文件的名称

-- WEB-INF目录：

-- web.xml：web项目的核心配置文件

-- classes目录：放置字节码文件的目录

-- lib目录：放置依赖的jar包

\* 将Tomcat集成到IDEA中，并且创建JavaEE的项目，部署项目。

## ## Servlet： server applet

\* 概念：运行在服务器端的小程序

\* Servlet就是一个接口，定义了Java类被浏览器访问到(tomcat识别)的规则。

\* 快速入门：

1. 创建JavaEE项目

2. 定义一个类，实现Servlet接口

3. 实现接口中的抽象方法

4. 配置Servlet

在web.xml中配置：

\* 执行原理：

1. 当服务器接受到客户端浏览器的请求后，会解析请求URL路径，获取访问的Servlet的资源路径

2. 查找web.xml文件，是否有对应的<url-pattern>标签体内容。

3. 如果有，则在找到对应的<servlet-class>全类名

4. tomcat会将字节码文件加载进内存，并且创建其对象

5. 调用其方法

\* Servlet中的生命周期方法：

1. 被创建：执行init方法，只执行一次

\* Servlet什么时候被创建？

\* 默认情况下，第一次被访问时，Servlet被创建

\* 可以配置执行Servlet的创建时机。

\* 在<servlet>标签下配置

1. 第一次被访问时，创建

\* <load-on-startup>的值为负数

2. 在服务器启动时，创建

\* <load-on-startup>的值为0或正整数

\* Servlet的init方法，只执行一次，说明一个Servlet在内存中只存在一个对象，Servlet是单例的

\* 多个用户同时访问时，可能存在线程安全问题。

\* 解决：尽量不要在Servlet中定义成员变量。即使定义了成员变量，也不要对修改值

2. 提供服务：执行service方法，执行多次

\* 每次访问Servlet时，Service方法都会被调用一次。

3. 被销毁：执行destroy方法，只执行一次

\* Servlet被销毁时执行。服务器关闭时，Servlet被销毁

\* 只有服务器正常关闭时，才会执行destroy方法。

\* destroy方法在Servlet被销毁之前执行，一般用于释放资源

\* Servlet3.0：

\* 好处：

\* 支持注解配置。可以不需要web.xml了。

6. Servlet的体系结构

Servlet -- 接口

|

GenericServlet -- 抽象类

|

HttpServlet -- 抽象类

\* GenericServlet：将Servlet接口中其他的方法做了默认空实现，只将service()方法作为抽象

\* 将来定义Servlet类时，可以继承GenericServlet，实现service()方法即可

\* HttpServlet：对http协议的一种封装，简化操作

1. 定义类继承HttpServlet

2. 复写doGet/doPost方法

7. Servlet相关配置

1. urlpartten:Servlet访问路径

1. 一个Servlet可以定义多个访问路径 ： @WebServlet({"/d4","/dd4","/ddd4"})

2. 路径定义规则：

1. /xxx：路径匹配

2. /xxx/xxx:多层路径，目录结构

3. \*.do：扩展名匹配

# ## HTTP请求消息：

\* 概念：Hyper Text Transfer Protocol 超文本传输协议

\* 传输协议：定义了，客户端和服务器端通信时，发送数据的格式

\* 特点：

1. 基于TCP/IP的高级协议

2. 默认端口号:80

3. 基于请求/响应模型的:一次请求对应一次响应

4. 无状态的：每次请求之间相互独立，不能交互数据

\* 历史版本：

\* 1.0：每一次请求响应都会建立新的连接

\* 1.1：复用连接

\* 请求消息数据格式

1. 请求行

请求方式 请求url 请求协议/版本

GET /login.html HTTP/1.1

\* 请求方式：

\* HTTP协议有7中请求方式，常用的有2种

\* GET：

1. 请求参数在请求行中，在url后。

2. 请求的url长度有限制的

3. 不太安全

\* POST：

1. 请求参数在请求体中

2. 请求的url长度没有限制的

3. 相对安全

2. 请求头：客户端浏览器告诉服务器一些信息

请求头名称: 请求头值

\* 常见的请求头：

1. User-Agent：浏览器告诉服务器，我访问你使用的浏览器版本信息

\* 可以在服务器端获取该头的信息，解决浏览器的兼容性问题

2. Referer：<http://localhost/login.html>

Referer: http://localhost/login.html

\* 告诉服务器，我(当前请求)从哪里来？

\* 作用：

1. 防盗链：

2. 统计工作：

3. 请求空行

空行，就是用于分割POST请求的请求头，和请求体的。

4. 请求体(正文)：

\* 封装POST请求消息的请求参数的

# ## Request：

1. request对象和response对象的原理

1. request和response对象是由服务器创建的。我们来使用它们

2. request对象是来获取请求消息，response对象是来设置响应消息

2. request对象继承体系结构：

ServletRequest -- 接口

| 继承

HttpServletRequest -- 接口

| 实现

org.apache.catalina.connector.RequestFacade 类(tomcat)

3. request功能：

1. 获取请求消息数据

1. 获取请求行数据

\* GET /day14/demo1?name=zhangsan HTTP/1.1

\* 方法：

1. 获取请求方式 ：GET

\* String getMethod()

2. (\*)获取虚拟目录：/day14

\* String getContextPath()

3. 获取Servlet路径: /demo1

\* String getServletPath()

4. 获取get方式请求参数：name=zhangsan

\* String getQueryString()

5. (\*)获取请求URI：/day14/demo1

\* String getRequestURI(): /day14/demo1

\* StringBuffer getRequestURL() :http://localhost/day14/demo1

\* URL:统一资源定位符 ： http://localhost/day14/demo1 中华人民共和国

\* URI：统一资源标识符 : /day14/demo1 共和国

6. 获取协议及版本：HTTP/1.1

\* String getProtocol()

7. 获取客户机的IP地址：

\* String getRemoteAddr()

2. 获取请求头数据

\* 方法：

\* (\*)String getHeader(String name):通过请求头的名称获取请求头的值

\* Enumeration<String> getHeaderNames():获取所有的请求头名称

3. 获取请求体数据:

\* 请求体：只有POST请求方式，才有请求体，在请求体中封装了POST请求的请求参数

\* 步骤：

1. 获取流对象

\* BufferedReader getReader()：获取字符输入流，只能操作字符数据

\* ServletInputStream getInputStream()：获取字节输入流，可以操作所有类型数据

\* 在文件上传知识点后讲解

2. 再从流对象中拿数据

### 2. 其他功能：

1. 获取请求参数通用方式：不论get还是post请求方式都可以使用下列方法来获取请求参数

1. String getParameter(String name):根据参数名称获取参数值 username=zs&password=123

2. String[] getParameterValues(String name):根据参数名称获取参数值的数组 hobby=xx&hobby=game

3. Enumeration<String> getParameterNames():获取所有请求的参数名称

4. Map<String,String[]> getParameterMap():获取所有参数的map集合

\* 中文乱码问题：

\* get方式：tomcat 8 已经将get方式乱码问题解决了

\* post方式：会乱码

\* 解决：在获取参数前，设置request的编码request.setCharacterEncoding("utf-8");

2. 请求转发：一种在服务器内部的资源跳转方式

1. 步骤：

1. 通过request对象获取请求转发器对象：RequestDispatcher getRequestDispatcher(String path)

2. 使用RequestDispatcher对象来进行转发：forward(ServletRequest request, ServletResponse response)

2. 特点：

1. 浏览器地址栏路径不发生变化

2. 只能转发到当前服务器内部资源中。

3. 转发是一次请求

3. 共享数据：

\* 域对象：一个有作用范围的对象，可以在范围内共享数据

\* request域：代表一次请求的范围，一般用于请求转发的多个资源中共享数据

\* 方法：

1. void setAttribute(String name,Object obj):存储数据

2. Object getAttitude(String name):通过键获取值

3. void removeAttribute(String name):通过键移除键值对

4. 获取ServletContext：

\* ServletContext getServletContext()

## 9. BeanUtils工具类，简化数据封装

\* 用于封装JavaBean的

1. JavaBean：标准的Java类

1. 要求：

1. 类必须被public修饰

2. 必须提供空参的构造器

3. 成员变量必须使用private修饰

4. 提供公共setter和getter方法

2. 功能：封装数据

2. 概念：

成员变量：

属性：setter和getter方法截取后的产物 成员和属性不一样

例如：getUsername() --> Username--> username

3. 方法：

1. setProperty()

2. getProperty()

3. populate(Object obj , Map map):将map集合的键值对信息，封装到对应的JavaBean对象中

# HTTP协议：响应消息

2. 响应消息：服务器端发送给客户端的数据

\* 数据格式：

1. 响应行

1. 组成：协议/版本 响应状态码 状态码描述

2. 响应状态码：服务器告诉客户端浏览器本次请求和响应的一个状态。

1. 状态码都是3位数字

2. 分类：

1. 1xx：服务器就收客户端消息，但没有接受完成，等待一段时间后，发送1xx多状态码

2. 2xx：成功。代表：200

3. 3xx：重定向。代表：302(重定向)，304(访问缓存)

4. 4xx：客户端错误。

\* 代表：

\* 404（请求路径没有对应的资源）

\* 405：请求方式没有对应的doXxx方法

5. 5xx：服务器端错误。代表：500(服务器内部出现异常)

2. 响应头：

1. 格式：头名称： 值

2. 常见的响应头：

1. Content-Type：服务器告诉客户端本次响应体数据格式以及编码格式

2. Content-disposition：服务器告诉客户端以什么格式打开响应体数据

\* 值：

\* in-line:默认值,在当前页面内打开

\* attachment;filename=xxx：以附件形式打开响应体。文件下载

3. 响应空行

4. 响应体:传输的数据

\* 响应字符串格式

HTTP/1.1 200 OK

Content-Type: text/html;charset=UTF-8

Content-Length: 101

Date: Wed, 06 Jun 2018 07:08:42 GMT

<html>

<head>

<title>$Title$</title>

</head>

<body>

hello , response

</body>

</html>

# ## Response对象

\* 功能：设置响应消息

1. 设置响应行

1. 格式：HTTP/1.1 200 ok

2. 设置状态码：setStatus(int sc)

2. 设置响应头：setHeader(String name, String value)

3. 设置响应体：

\* 使用步骤：

1. 获取输出流

\* 字符输出流：PrintWriter getWriter()

\* 字节输出流：ServletOutputStream getOutputStream()

2. 使用输出流，将数据输出到客户端浏览器

\* 案例：

1. 完成重定向

\* 重定向：资源跳转的方式

\* 代码实现：

//1. 设置状态码为302

response.setStatus(302);

//2.设置响应头location

response.setHeader("location","/day15/responseDemo2");

//简单的重定向方法

response.sendRedirect("/day15/responseDemo2");

\* 重定向的特点:redirect

1. 地址栏发生变化

2. 重定向可以访问其他站点(服务器)的资源

3. 重定向是两次请求。不能使用request对象来共享数据

\* 转发的特点：forward

1. 转发地址栏路径不变

2. 转发只能访问当前服务器下的资源

3. 转发是一次请求，可以使用request对象来共享数据

\* forward 和 redirect 区别

\* 路径写法：

1. 路径分类

1. 相对路径：通过相对路径不可以确定唯一资源

\* 如：./index.html

\* 不以/开头，以.开头路径

\* 规则：找到当前资源和目标资源之间的相对位置关系

\* ./：当前目录

\* ../:后退一级目录

2. 绝对路径：通过绝对路径可以确定唯一资源

\* 如：http://localhost/day15/responseDemo2 /day15/responseDemo2（默认的绝对路径）

\* 以/开头的路径

\* 规则：判断定义的路径是给谁用的？判断请求将来从哪儿发出

\* 给客户端浏览器使用：需要加虚拟目录(项目的访问路径)

\* 建议虚拟目录动态获取：request.getContextPath()

\* <a> , <form> 重定向...

\* 给服务器使用：不需要加虚拟目录

\* 转发路径

2. 服务器输出字符数据到浏览器

\* 步骤：

1. 获取字符输出流

2. 输出数据

\* 注意：

\* 乱码问题：

1. PrintWriter pw = response.getWriter();获取的流的默认编码是ISO-8859-1

2. 设置该流的默认编码

3. 告诉浏览器响应体使用的编码

//简单的形式，设置编码，是在获取流之前设置

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

## ## ServletContext对象：

1. 概念：代表整个web应用，可以和程序的容器(服务器)来通信

2. 获取：

1. 通过request对象获取

request.getServletContext();

2. 通过HttpServlet获取

this.getServletContext();

3. 功能：

1. 获取MIME类型：

\* MIME类型:在互联网通信过程中定义的一种文件数据类型

\* 格式： 大类型/小类型 text/html image/jpeg

\* 获取：String getMimeType(String file)

2. 域对象：共享数据

1. setAttribute(String name,Object value)

2. getAttribute(String name)

3. removeAttribute(String name)

\* ServletContext对象范围：所有用户所有请求的数据

3. 获取文件的真实(服务器)路径

1. 方法：String getRealPath(String path)

1.getRealPath("/")方法返回的是项目在服务器的绝对路径，而getRealPath("WEB-INF/myfile")返回的是包含一个给定虚拟路径的绝对路径,其中/myfile是虚拟的。

2.getContextPath()方法返回虚拟目录

# ## 会话技术

1. 会话：一次会话中包含多次请求和响应。

2. 功能：在一次会话的范围内的多次请求间，共享数据

3. 方式：

1. 客户端会话技术：Cookie

2. 服务器端会话技术：Session

## Cookie：

1. 概念：客户端会话技术，将数据保存到客户端

2. 快速入门：

\* 使用步骤：

1. 创建Cookie对象，绑定数据

\* new Cookie(String name, String value)

2. 发送Cookie对象

\* response.addCookie(Cookie cookie)

3. 获取Cookie，拿到数据

\* Cookie[] request.getCookies()

3. 实现原理

\* 基于响应头set-cookie和请求头cookie实现

4. cookie的细节

1. 一次可不可以发送多个cookie?

\* 可以

\* 可以创建多个Cookie对象，使用response调用多次addCookie方法发送cookie即可。

2. cookie在浏览器中保存多长时间？

1. 默认情况下，当浏览器关闭后，Cookie数据被销毁

2. 持久化存储：

\* setMaxAge(int seconds)

1. 正数：将Cookie数据写到硬盘的文件中。持久化存储。并指定cookie存活时间，时间到后，cookie文件自动失效

2. 负数：默认值

3. 零：删除cookie信息

3. cookie能不能存中文？

\* 在tomcat 8 之前 cookie中不能直接存储中文数据。

\* 需要将中文数据转码---一般采用URL编码(%E3)

\* 在tomcat 8 之后，cookie支持中文数据。特殊字符还是不支持，建议使用URL编码存储，URL解码解析

4. cookie共享问题？

1. 假设在一个tomcat服务器中，部署了多个web项目，那么在这些web项目中cookie能不能共享？

\* 默认情况下cookie不能共享

\* setPath(String path):设置cookie的获取范围。默认情况下，设置当前的虚拟目录

\* 如果要共享，则可以将path设置为"/"

2. 不同的tomcat服务器间cookie共享问题？

\* setDomain(String path):如果设置一级域名相同，那么多个服务器之间cookie可以共享

\* setDomain(".baidu.com"),那么tieba.baidu.com和news.baidu.com中cookie可以共享

5. Cookie的特点和作用

1. cookie存储数据在客户端浏览器

2. 浏览器对于单个cookie 的大小有限制(4kb) 以及 对同一个域名下的总cookie数量也有限制(20个)

\* 作用：

1. cookie一般用于存出少量的不太敏感的数据

2. 在不登录的情况下，完成服务器对客户端的身份识别

## ## Session：主菜

1. 概念：服务器端会话技术，在一次会话的多次请求间共享数据，将数据保存在服务器端的对象中。HttpSession

2. 快速入门：

1. 获取HttpSession对象：

HttpSession session = request.getSession();

2. 使用HttpSession对象：

Object getAttribute(String name)

void setAttribute(String name, Object value)

void removeAttribute(String name)

3. 原理

\* Session的实现是依赖于Cookie的。

4. 细节：

1. 当客户端关闭后，服务器不关闭，两次获取session是否为同一个？

\* 默认情况下。不是。

\* 如果需要相同，则可以创建Cookie,键为JSESSIONID，设置最大存活时间，让cookie持久化保存。

Cookie c = new Cookie("JSESSIONID",session.getId());

c.setMaxAge(60\*60);

response.addCookie(c);

2. 客户端不关闭，服务器关闭后，两次获取的session是同一个吗？

\* 不是同一个，但是要确保数据不丢失。tomcat自动完成以下工作

\* session的钝化：

\* 在服务器正常关闭之前，将session对象系列化到硬盘上

\* session的活化：

\* 在服务器启动后，将session文件转化为内存中的session对象即可。

3. session什么时候被销毁？

1. 服务器关闭

2. session对象调用invalidate() 。

3. session默认失效时间 30分钟

选择性配置修改

<session-config>

<session-timeout>30</session-timeout>

</session-config>

5. session的特点

1. session用于存储一次会话的多次请求的数据，存在服务器端

2. session可以存储任意类型，任意大小的数据

\* session与Cookie的区别：

1. session存储数据在服务器端，Cookie在客户端

2. session没有数据大小限制，Cookie有

3. session数据安全，Cookie相对于不安全

# Jsp -EL-JSTL

1. JSP:

1. 指令

2. 注释

3. 内置对象

2. MVC开发模式

3. EL表达式

4. JSTL标签

5. 三层架构

## ## JSP:

1. 指令

\*

：

<%@ 指令名称 属性名1=属性值1 属性名2=属性值2 ... %>

1. page ： 配置JSP页面的

\* contentType：等同于response.setContentType()

1. 设置响应体的mime类型以及字符集

2. 设置当前jsp页面的编码（只能是高级的IDE才能生效，如果使用低级工具，则需要设置pageEncoding属性设置当前页面的字符集）

\* import：导包

\* errorPage：当前页面发生异常后，会自动跳转到指定的错误页面

\* isErrorPage：标识当前也是是否是错误页面。

\* true：是，可以使用内置对象exception

\* false：否。默认值。不可以使用内置对象exception

2. include ： 页面包含的。导入页面的资源文件

3. taglib ： 导入资源

3. 内置对象

\* 在jsp页面中不需要创建，直接使用的对象

\* 一共有9个：

变量名 真实类型 作用

\* pageContext PageContext 当前页面共享数据，还可以获取其他八个内置对象

\* request HttpServletRequest 一次请求访问的多个资源(转发)

\* session HttpSession 一次会话的多个请求间

\* application ServletContext 所有用户间共享数据

\* response HttpServletResponse 响应对象

\* page Object 当前页面(Servlet)的对象 this

\* out JspWriter 输出对象，数据输出到页面上

\* config ServletConfig Servlet的配置对象

\* exception Throwable 异常对象

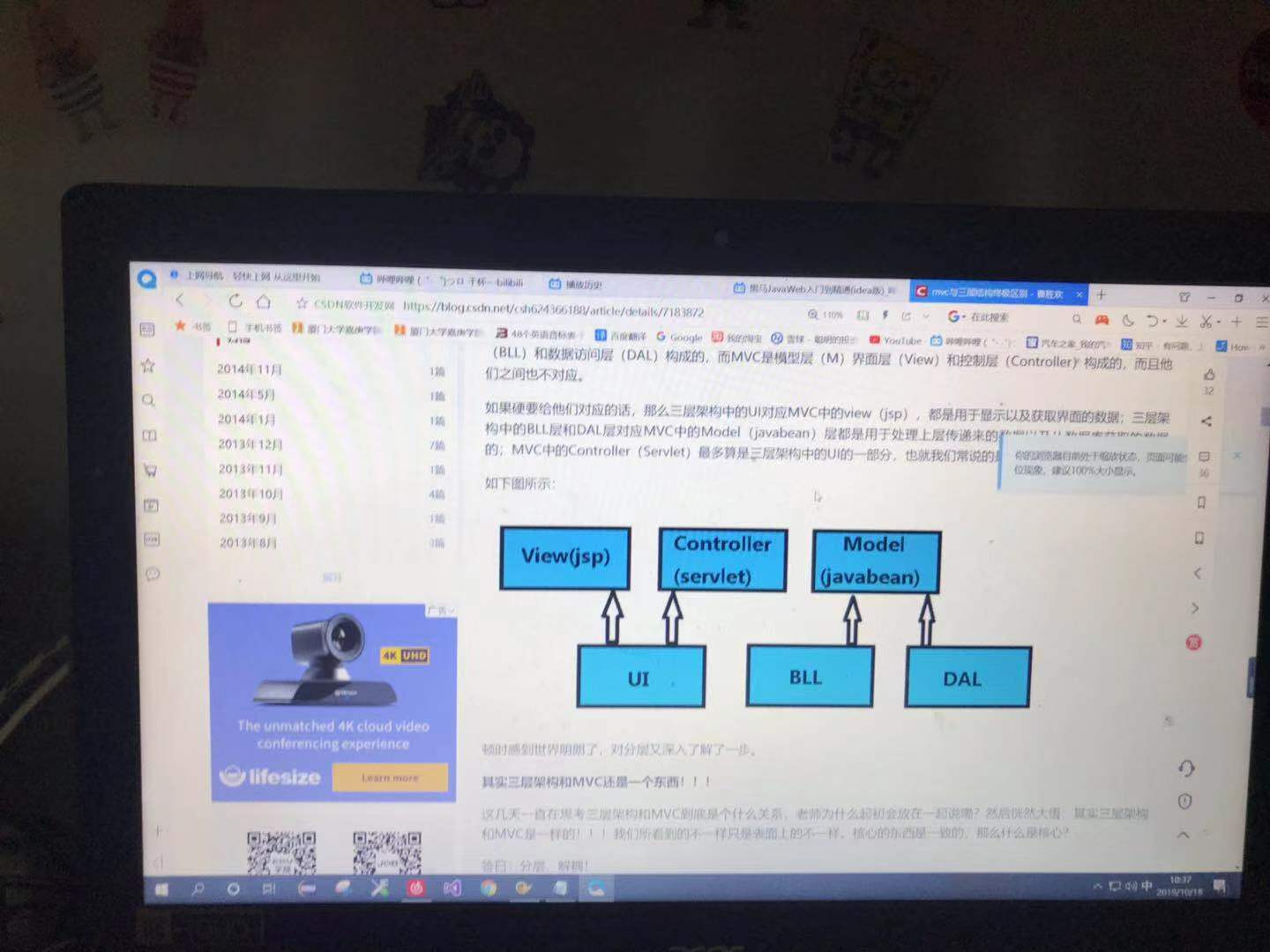
## ## MVC：开发模式

1. jsp演变历史

1. 早期只有servlet，只能使用response输出标签数据，非常麻烦

2. 后来又jsp，简化了Servlet的开发，如果过度使用jsp，在jsp中即写大量的java代码，有写html表，造成难于维护，难于分工协作

3. 再后来，java的web开发，借鉴mvc开发模式，使得程序的设计更加合理性

2. 

## ## EL表达式

1. 概念：Expression Language 表达式语言

2. 作用：替换和简化jsp页面中java代码的编写

3. 语法：${表达式}

4. 注意：

\* jsp默认支持el表达式的。如果要忽略el表达式

1. 设置jsp中page指令中：isELIgnored="true" 忽略当前jsp页面中所有的el表达式

2. \${表达式} ：忽略当前这个el表达式

5. 使用：

1. 运算：

\* 运算符：

1. 算数运算符： + - \* /(div) %(mod)

2. 比较运算符： > < >= <= == !=

3. 逻辑运算符： &&(and) ||(or) !(not)

4. 空运算符： empty

\* 功能：用于判断字符串、集合、数组对象是否为null或者长度是否为0

\* ${empty list}:判断字符串、集合、数组对象是否为null或者长度为0

\* ${not empty str}:表示判断字符串、集合、数组对象是否不为null 并且 长度>0

2. 获取值

1. el表达式只能从域对象中获取值

2. 语法：

1. ${域名称.键名}：从指定域中获取指定键的值

如果没值不会空指针异常也不会显示null 只显示空字符串 因为el内部做了优化

\* 域名称：

1. pageScope --> pageContext

2. requestScope --> request

3. sessionScope --> session

4. applicationScope --> application（ServletContext）

\* 举例：在request域中存储了name=张三

\* 获取：${requestScope.name}

2. ${键名}：表示依次从最小的域中查找是否有该键对应的值，直到找到为止。

3. 获取对象、List集合、Map集合的值

1. 对象：${域名称.键名.属性名}

\* 本质上会去调用对象的getter方法

2. List集合：${域名称.键名[索引]}

3. Map集合：

\* ${域名称.键名.key名称}

\* ${域名称.键名["key名称"]}

3. 隐式对象：

\* el表达式中有11个隐式对象

\* pageContext：

\* 获取jsp其他八个内置对象

\* ${pageContext.request.contextPath}：动态获取虚拟目录

## ## JSTL

OGNL（Object-Graph Navigation Language），大概可以理解为：对象图形化导航语言。一种可以方便地操作对象属性的开源表达式语言。用于mybatis

Spring 3引入了Spring表达式语言（ Spring Expression Language，SpEL），它能够以一种强大和简洁的方式将值装配到bean属性和构造器参数中，在这个过程中所使用的表达式会在运行时计算得到值

1. 概念：JavaServer Pages Tag Library JSP标准标签库

\* 是由Apache组织提供的开源的免费的jsp标签 <标签>

2. 作用：用于简化和替换jsp页面上的java代码

3. 使用步骤：

1. 导入jstl相关jar包

2. 引入标签库：taglib指令： <%@ taglib %>

3. 使用标签

4. 常用的JSTL标签

1. if:相当于java代码的if语句

1. 属性：

\* test 必须属性，接受boolean表达式

\* 如果表达式为true，则显示if标签体内容，如果为false，则不显示标签体内容

\* 一般情况下，test属性值会结合el表达式一起使用

2. 注意：

\* c:if标签没有else情况，想要else情况，则可以在定义一个c:if标签

2. choose:相当于java代码的switch语句

1. 使用choose标签声明 相当于switch声明

2. 使用when标签做判断 相当于case

3. 使用otherwise标签做其他情况的声明 相当于default

3. foreach:相当于java代码的for语句

Items

Varstatus