3.1 Tomcat服务器软件

3.1.1 C/S和B/S软件架构介绍

交互方式：

计算机之间的信息交流称为交互

1. B/S 交互架构：browser/server；B/S架构是特殊的C/S架构

   特点：想要访问服务器资源，不需要安装客户端，直接在浏览器上进行服务器资源的访问

   优点：开发，部署和维护简单

   缺点：有户体验稍差，资源都存在服务器端

2. C/S 交互架构： client/server

   特点：**想要访问服务器资源，必须要安装客户端软件**

   优点：有户体验好

   缺点：需要对客户端和服务器代码进行开发，部署和维护；客户端存储了大量数据。

3.1.2 Web服务器作用及资源的分类

作用：

1. 开发者可以把本地资源发布到互联网上
2. 其他用户可以通过浏览器访问这些资源

资源的分类

1. 静态资源：对于同一个页面，不同的用户看到的内容是一样的。例如，百度首页，门户网站 .html/.css/ .js
2. 动态资源：对于同一个页面，不同的用户看到的内容是不一样的。例如，购物车页面，订单页面 .servlet/.jsp

注意⚠️：浏览器不能直接解析动态资源，首先要先将动态资源转换成静态资源。**Tomcat**可以将动态资源转换成静态资源。

3.1.3 URL请求格式解析

Uniform Resouce Locator

格式：协议://域名:端口号/资源位置?参数=值

协议：http/https/ftp等等 底层协议：tcp/utp

域名：域名/ip地址

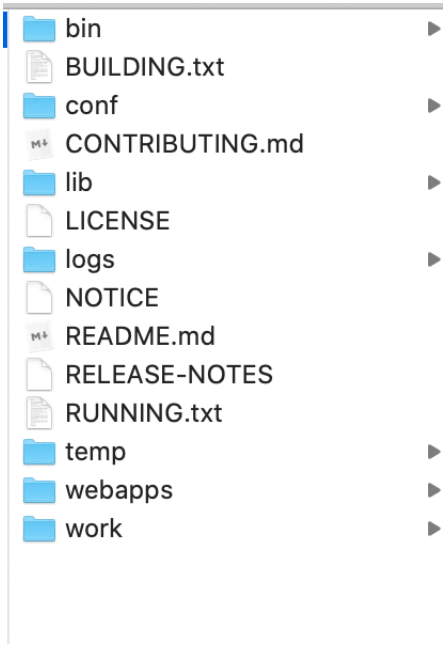
端口号：程序必须使用端口号，才能被另一台计算机访问。http的端口号：80

资源位置：用于描述资源在服务器上的位置

3.1.4 常见的Web服务器

Tomcat

3.1.6 Tomcat目录结构



1. bin: binary 二进制文件 启动和停止tomcat的相关脚本文件
2. conf: configaration 配置文件
3. lib: tomcat运行时，依赖的jar包
4. logs：运行日志
5. temp：临时文件
6. webapps：**发布自己的网站目录**
7. work：运行时生成的一些文件
8. RUNNING.txt：当前tomcat的版本信息

3.1.8 Tomcat启动报错分析

端口号：1024 - 65535

3.1.9 项目发布方式一

将资源直接放到webapps目录下

优点：简单

缺点：代码更新时，需要把更新后的代码重新复制放到webapps目录下

热部署：不需要重启tomcat服务器，页面自动更新

3.1.10 项目发布方式二

在server.xml中配置，匹配映射关系；需要重启tomcat.

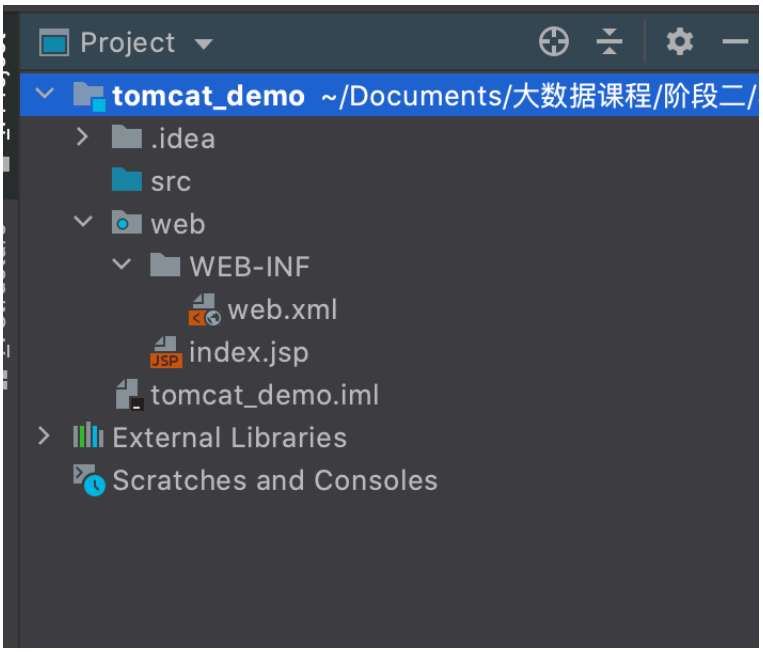
缺点：

1. 不支持热部署
2. 不应该频繁修改server.xml文件

3.1.11 项目发布方式三

独立xml部署；在tomcat/conf/Catalina/localhost创建一个xml文件，添加Context标签。**文件名代表虚拟路径，不能乱取**

3.1.13 创建web项目



src：编写java代码

WEB_INF：安全目录，没有办法通过浏览器直接访问

web.xml

index.jsp

3.2 HTTP协议

3.2.1 HTTP协议的概念

用于定义WEB浏览器与WEB服务器之间交换数据的过程。

传输协议：在客户端和服务器通信时，规范了数据传输的格式

http协议特点：

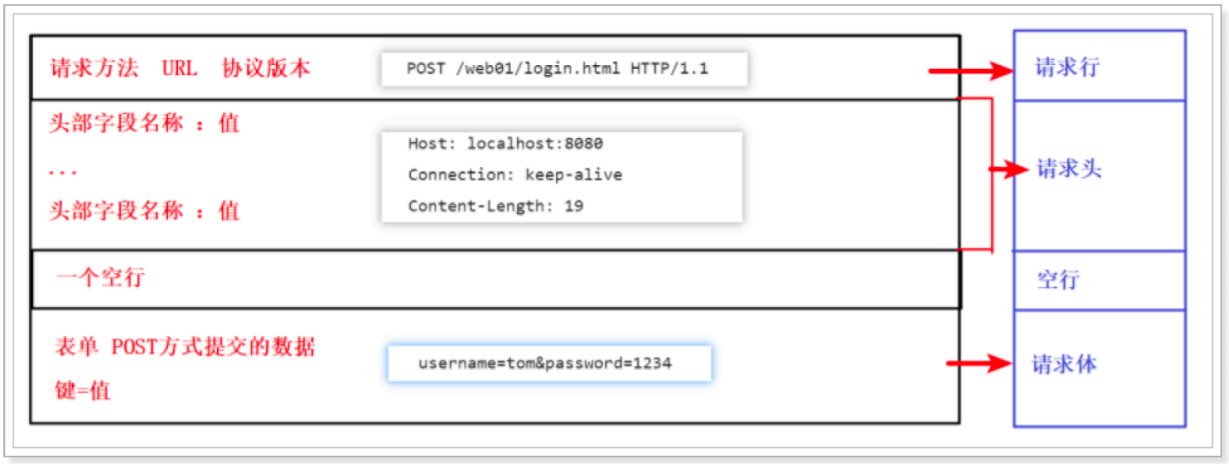
1. 基础协议：tcp协议
2. 默认端口：80端口
3. 基于请求/响应模型
4. 无状态协议：多次请求之间，各个请求是相互独立的，不能交换数据

https本质上是http协议，但对通信的数据进行了加密。默认端口号：443

http版本：

1. http/1.0：每次请求都是一个新的连接
2. http/1.1：多个请求可以复用同一个连接

3.2.2 HTTP请求报文格式



http的请求报文由请求行，请求头（key-value格式），空行，请求体（只有post提交方式有请求体）组成。

3.2.3 请求行，请求头，请求体详解

请求行

必须在请求格式的第一行

格式：请求方法 URL 协议版本

请求方式有7种，常用两种：get和post

get请求：

1. 请求参数追加到url后面，不安全
2. URL的长度限制了get请求方式的数据大小
3. 没有请求头

post请求：

1. 请求参数不会显示在url后面而是显示在请求体中，较为安全
2. 请求数据大小没有限制

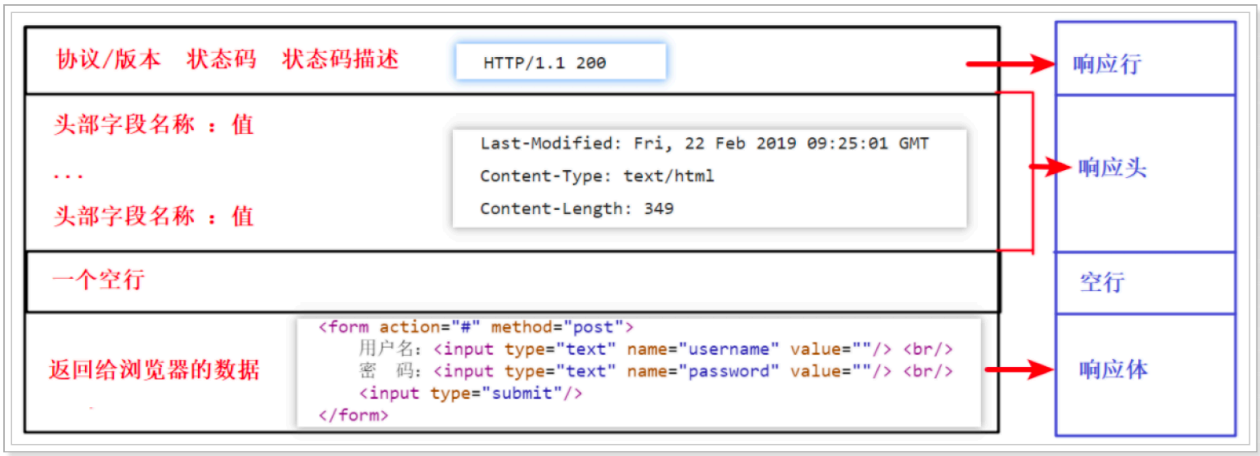
请求头

key-value格式

请求体

只有post请求方式会有请求体

3.2.5 HTTP响应报文格式



http的响应报文由响应行，响应体，空行，响应体组成

3.2.6 响应行，响应头，响应体详解

响应行

格式：协议/版本 状态码 状态码描述

状态码：

1. 200 成功
2. 302 请求重定向
3. 304 请求资源没有改变，访问本地缓存
4. 404 请求资源不存在
5. 500 服务器内部错误，后台代码编写错误

响应头

响应体

服务器发送给客户端的正文

3.3 Servlet

3.3.2 Servlet概述

servlet = server + applet：运行在服务器上的java小程序

web阶段：程序的入口是浏览器发送请求，由tomcat执行某个具体类中的某个的方法。tomcat怎么能根据请求找到具体的执行方法呢？这就要求java类实现servlet规范。

Servlet是一个接口：一个类想要通过浏览器被访问到，那么这个类就必须实现或间接实现Servlet接口。

执行的业务逻辑：

1. 接受请求
2. 处理逻辑
3. 响应结构

3.3.3 Servlet快速入门

1. 创建java类，实现Servlet接口
2. Service方法：对外提供服务的方法，tomcat会调用servlet中service方法执行具体的业务逻辑

   ServletRequest：请求对象，来获取请求参数

   ServletResponse：响应对象，向浏览器响应一些数据

   ```
   @Override
   public void service(ServletRequest servletRequest, ServletResponse
   servletResponse) throws ServletException, IOException {

   }
   ```

3. 此时不知道访问路径，需要在web.xml的url-pattern标签中配置

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <web-app ...>

       <servlet>
           //当前servlet的一个别名
           <servlet-name>QuickServlet</servlet-name>
           //servlet的一个全限定类名
           <servlet-class>com.zichen.servlet.QuickServlet</servlet-class>
   ```

```
    </servlet>

    <servlet-mapping>
        //给指定名称的servlet来配置映射地址
        <servlet-name>QuickServlet</servlet-name>
        //具体该servlet的映射地址，必须以/开头
        <url-pattern>/quickServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

3.3.4 Servlet执行原理

http://localhost:8080 /tomcat_demo_war_exploded /quickServlet

tomcat服务器　　项目　　要访问的资源路径

1. tomcat拿到要访问的资源路径和所有的对比
2. 对比成功后，根据找到
3. 找到全限定名后，通过反射机制创建该对象
4. tomcat会默认执行类中的service方法

3.3.5 Servlet生命周期

生命周期	说明
init	servlet对象创建时，调用此方法完成初始化操作
service	用户访问serlvet时，调用此方法完成业务逻辑的处理
destory	当servlet对象销毁时，会调用此方法完成销毁操作

默认生命周期中，服务器启动时，不会创建serlvet对象，只有第一次请求发送时，才会创建serlvet对象，并执行init方法。随之后面的请求都只会调用service方法。

缺点：当并发量很大时，会影响用户体验。

改进：服务器启动时，完成servlet对象的创建。

方法：配置4

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>

    <servlet>
        //当前servlet的一个别名
        <servlet-name>QuickServlet</servlet-name>
        //servlet的一个全限定类名
```

```
        <servlet-class>com.zichen.servlet.QuickServlet</servlet-class>
        //服务器启动时，完成servlet对象的创建。
        <load-on-startup>4</load-on-startup>
    </servlet>

    <servlet-mapping>
        //给指定名称的servlet来配置映射地址
        <servlet-name>QuickServlet</servlet-name>
        //具体该servlet的映射地址，必须以/开头
        <url-pattern>/quickServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

笔试题：描述一下servlet的生命周期

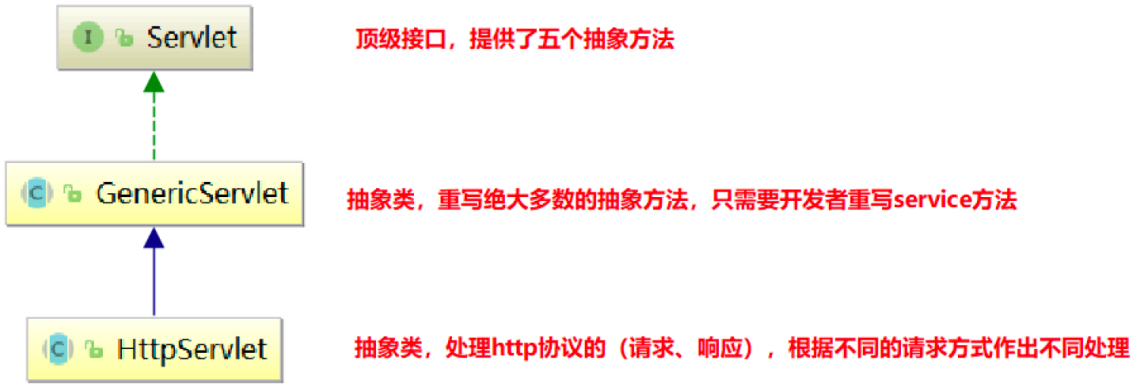
Servlet是单实例多线程的，默认情况下，第一次请求来的时候，会创建serlvet对象，并执行init方法。随后在执行service方法，完成业务处理。当每一次请求发送过来，都会开启一个线程，来执行service方法。当服务器关闭或serlvet被移除时，会调用destory方法。

3.3.6 Servlet体系结构

Servlet接口：顶级接口，五个声明方法

GenericServlet类：抽象类，重写了绝大多数的方法，只需重写service方法即可

HttpServlet类：抽象类，处理http协议中的交互



3.3.9 url-pattern配置方式

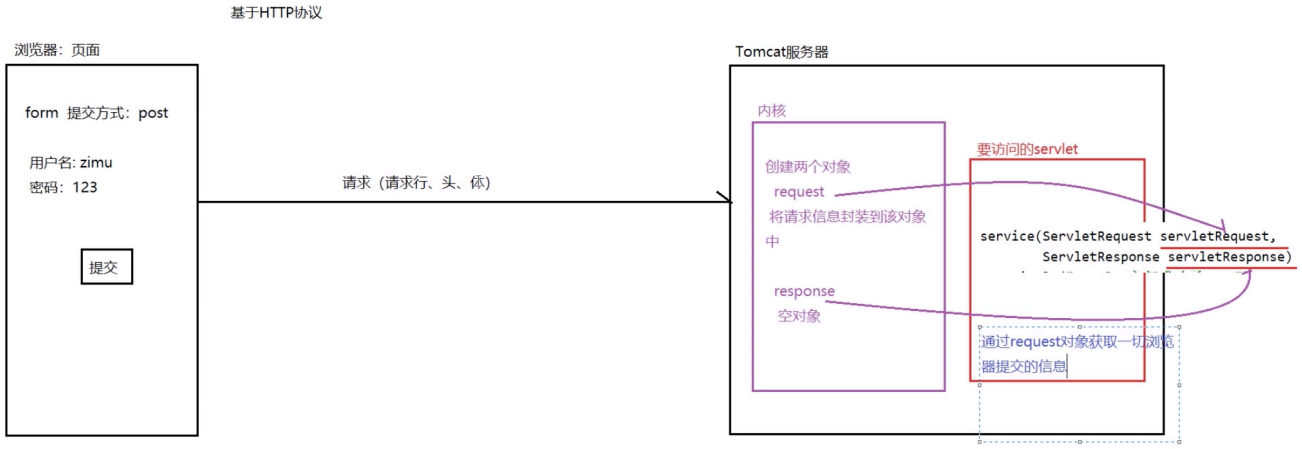
中可取的值：

1. 精准匹配 /quickServlet
2. 目录匹配 /xx/*
3. 后缀名匹配 *.xx

3.3.10 request对象概述和作用

用户通过浏览器访问服务器时，tomcat将http请求中所有的信息封装到request对象中。

作用：开发者可以通过request对象来获取浏览器发来的所有信息



3.3.11 获取请求行信息

方法	说明
String getMethod()	请求方法
String getContextPath()	虚拟路径 /项目名
StringBuffer getRequestURL()	URL
String getProtocal()	
String getRemoteAddr()	

3.3.12 获取请求头信息

方法	说明
String getHeader(String name)	
Eumeration getHeaderNames()	获取所有的请求头

3.3.13 获取请求参数

方法	说明
String getParameter(String name)	获取指定参数名的值
String[] getPatameterValues(String name)	
Map<String, String[]> getParameterMap()	获取所有参数名和对应值，封装到map集合中

post提交方式存在中文乱码问题，get提交方式没有。

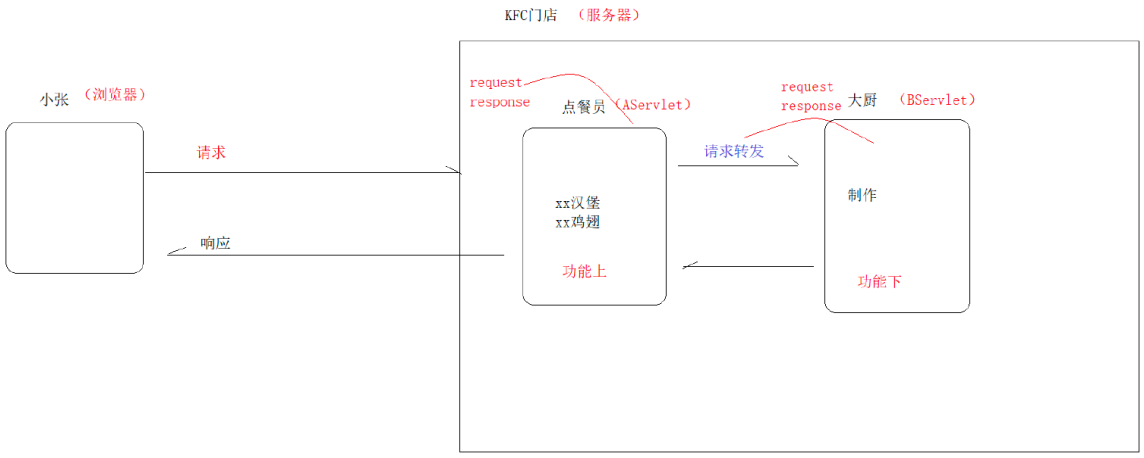
浏览器对中文采用的编码方式是utf-8，如果服务器的解码方式不是utf-8，就会出现乱码。

tomcat对url的编解码是utf-8，但对请求体采用的解码方式是ISO-8859-1。

解决方式：req.setCharacterEncoding("UTF-8")，必须放在第一行

3.3.15 请求转发

一种在服务器内部的资源跳转方式



方法	说明
RequestDispatcher getRequestDispatcher(string path)	通过request对象，获取转发器对象； path：要跳转的资源路径
void forward(forward(ServletRequest request, ServletResponse response)	通过转发器对象，实现转发功能

特点：

1. 浏览器只发生一次请求

2. 地址栏没有发生改变
3. 只能转发服务器内部的资源

3.3.16 Request作为域对象（数据共享）

域：共享区域

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，一般用于一次请求转发的多个资源中共享数据

方法	说明
void setAttribute(String name, Object obj)	设置数据
Object getAttribute(String name)	获取数据
void removeAttribute(String name)	移除数据

生命周期：

1. 当用户发送请求，创建request域
2. 当服务器发生响应是，request域销毁
3. 作用范围：一次请求，包含多次转发

3.3.17 response对象

response对象表示服务器给浏览器返回的响应信息

作用：开发者可以利用response对象设置返回给浏览器的响应信息

3.3.18 设置Http响应信息

响应行

方法	说明
void setStatus(int sc)	设置状态码

响应头

方法	说明
void setHeader(String name, String value)	

响应体

方法	说明
PrintWriter getWriter()	字符输出流
ServletOutputStream getOutputStream()	字节输出流

3.3.19 响应重定向

1. 设置响应状态码 302
2. 设置重定向地址

方法一：

```
resp.setStatus(302);
resp.setHeader("Location", "bServlet");
```

方法二：

```
resp.sendRedirect("bServlet");
```

特点：

1. 地址栏会发生改变
2. 重定向是二次请求
3. 重定向是客户端行为，可以跳转到外部资源的
4. 不能使用request域共享数据

3.3.20 请求转发和重定向的区别

1. 对象不同：请求转发是request对象的行为，重定向是response对象的行为。
2. 请求次数不同：请求转发是一次请求，浏览器地址栏不会发生改变；重定向是两次请求，浏览器地址栏会发生改变。
3. 发生的位置不同：请求转发是在服务器，重定向是在浏览器。

使用场景：如果需要传递数据，使用请求转发；如果不需要传递数据，使用重定向。

3.3.21响应中文

服务器默认的编码方式是ISO-8859-1，浏览器默认的解码方式是GBK。

统一设置utf-8编解码

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

3.3.22 ServletContext对象概述

当tomcat启动时，会为每一个web项目承建一个ServletContext对象。ServletContext也是一个域对象，可以进行数据共享。

应用上下文对象：

1. 应用：项目
2. 上文：获取tomcat的相关信息
3. 下文：获取servlet的相关信息

主要作用：

1. 共享数据
2. 获取资源在服务器中的真实地址
3. 获取全局的配置参数
4. 获取文件MIME类型

获取servletContext对象：

```
ServletContext sc = req.getServletContext();

ServletContext sc = this.getServletContext();
```

3.3.23 ServletContext作为域对象

作用范围：当前项目

方法	说明
void setAttribute(String name, Object obj)	设置数据
Object getAttribute(String name)	获取数据
void removeAttribute(String name)	移除数据

3.3.24 获取资源在服务器的真实地址

方法	说明
getRealPath(String path)	获取资源在服务器中的真实路径

3.3.25 获取全局参数

可以在web.xml中配置全局参数，使用标签，实现参数和代码解耦合。

```
<context-param>
  <param-name>encode</param-name>
  <param-value>UTF-8</param-value>
</context-param>
```

方法	说明
String getInitParameter(String name)	

3.3.16 获取文件MIME类型

在互联网通信中定义出来的一种文件数据类型格式

操作系统是根据文件的扩展名来识别文件的类型

http协议传输的所有内容都是字符串。浏览器怎么根据字符串来区别文件类型?

我们可以设置Content-Type，例如 大类型/小类型 text/html image/jpeg。浏览器根据这些MIME类型来区分文件格式。

方法	说明
String getMimeType()	

3.4 Cookie和Session

3.4.1 会话技术概述

会话：B/S架构中，从浏览器第一次给服务器发送请求开始，建立会话，直到有一方断开，会话结束。一次会话包含多次请求和响应。

http是无状态协议，同一个会话中的连续的多个请求是相互独立的，彼此互不了解。

会话技术就是存储浏览器和服务器之间多次请求之间的数据。

如果浏览器想存储请求的数据，我们可以使用cookie。

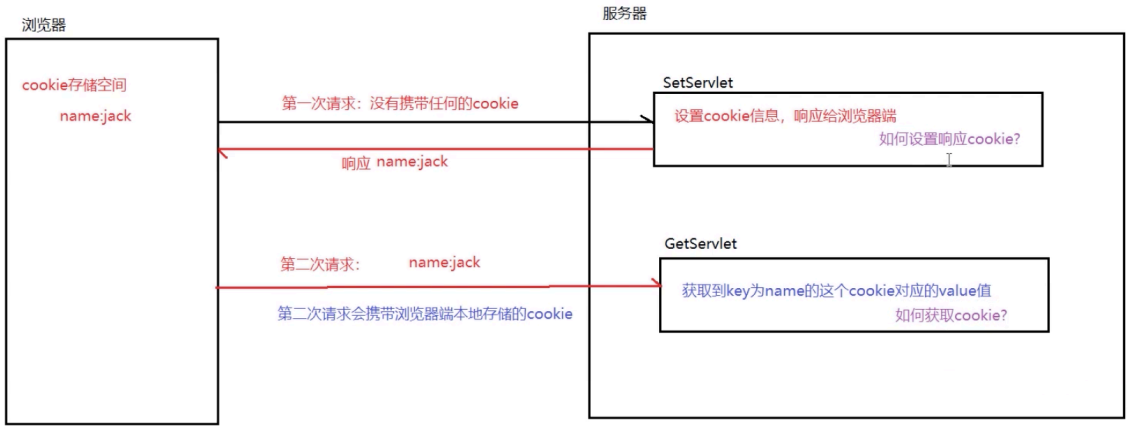
如果服务器想存储请求的数据，我们可以使用session。

3.4.2 Cookie概述

作用：客户端会话技术，在一次会话的多次请求之间共享数据，将数据保存客户端（浏览器）

举例：未登录的购物车

3.4.3 Cookie快速入门




```
1. 设置数据到cookie中
// 1.创建cookie对象，设置数据 *value只能存字符串
Cookie cookie = new Cookie(String name,String value);
// 2.通过response，响应（返回）cookie
response.addCookie(cookie);

2. 从cookie中获取数据
// 1.通过request对象，接收cookie数组
Cookie[] cookies = request.getCookies();
// 2.遍历数组
```

3.4.4 Cookie工作原理

基于http协议的：请求头cookie 和 响应头set-cookie

3.4.5 Cookie细节之服务器发送多个cookie

可以

3.4.6 Cookie细节之Cookie在浏览器保存时间

默认情况下，浏览器关闭，cookie销毁

设置cookie的存活时间：

方法	说明
cookie.setMaxAge(int second)	单位是秒；正数：指定存活时间，持久化浏览器的磁盘中，到期后自动销毁；负数：默认浏览器关闭，cookie销毁；零：立即销毁（自杀）

3.4.7 Cookie细节之Cookie是否支持中文

tomcat8之前不支持中文；tomcat8之后支持中文，Rfc6265Cookie规范，不允许使用 分号、空格等一些特殊符号...

可以让各个版本tomcat支持中文吗？可以

方法：使用URLEncoder编码，URLDecoder解码

3.4.8 Cookie特点

1. cookie存储数据都在客户端
2. cookie存储的数据只能是字符串
3. cookie的大小不能超过4kb
4. cookie存储的数据不安全

3.4.9 Session的概念

作用：在一次会话的多次请求之间共享数据，数据保存在客户端

session是基于cookie来实现的。响应中有一个session id，便于下次请求查找

3.4.10 Session快速入门案例

HttpSession 域对象

作用范围：一次会话

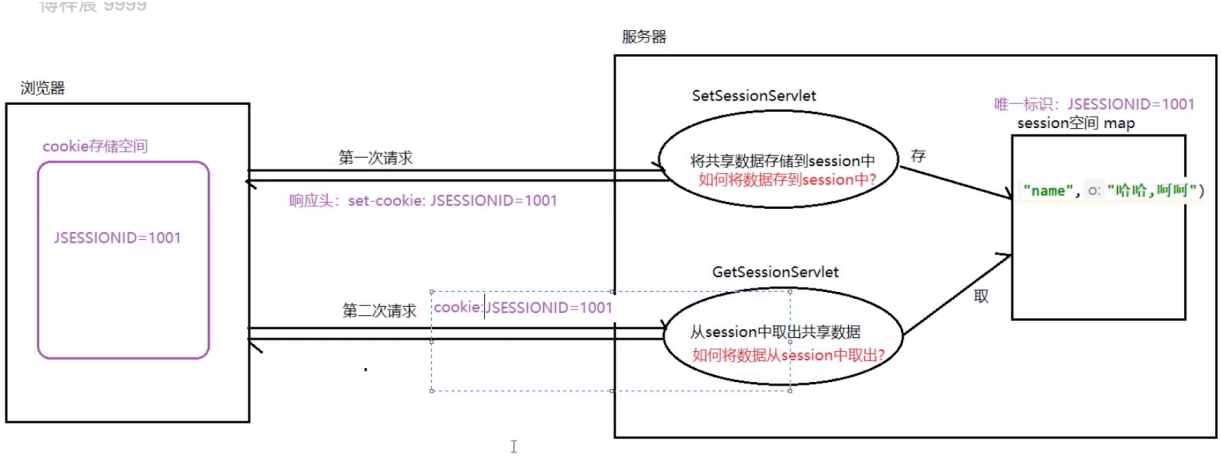
方法	说明
void setAttribute(String name, Object obj)	设置数据
Object getAttribute(String name)	获取数据
void removeAttribute(String name)	移除数据


```
1. 将数据存储到session中
// 1.通过rquest对象，获取session对象
HttpSession session = request.getSession();
// 2.操作session的API，存储数据
session.setAttribute("username","哈哈，呵呵");

2. 从session中获取数据
// 1.通过rquest对象，获取session对象
HttpSession session = request.getSession();
// 2.操作session的API，获取数据
session.getAttribute("username");
```

3.4.11 Session的工作原理

响应头：set-cookie:JSESSIONID=1001 （唯一标志）；请求头：cookie:JSESSIONID=1001



3.4.12 Session的生命周期

何时创建：用户第一次调用request.getSession()方法时，创建

何时销毁：

1. 服务器非正常关闭
2. 非活跃状态30分钟后：tomcat进行配置 /tocmat安装目录/conf/web.xml
3. session.invalidate(); 自杀

作用范围：一次会话中，多次请求之间

注意⚠️：每一个浏览器跟服务器都是独立的会话.

3.4.13 三大域对象总结

request，session，servletContext

API相同

方法	说明
void setAttribute(String name, Object obj)	设置数据
Object getAttribute(String name)	获取数据
void removeAttribute(String name)	移除数据

生命周期和作用范围不同

	何时创建	何时销毁	作用范围
request	用户发送请求时	服务器作出响应后销毁	一次请求
session	第一次调用request.getSession()时创建	服务器非正常关闭；非活跃状态30分钟后；自杀	一个会话
serlvetContext	服务器启动，项目加载时创建	服务器关闭，项目被移除	整个项目

3.5 Filter过滤器和Listener监听器

3.5.1 Filter概述

当用户访问服务器中，过滤器将请求拦截下来，完成一些通用的操作。双向拦截

应用场景：

1. 登陆校验
2. 统一网站编码
3. 非法字符过滤

3.5.2 快速入门

1. 创建一个类继承Filter接口

```
public class QuickFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    //servletRequest 请求对象
    //servletResponse 响应对象
    //filterChain 过滤器链(是否放行)
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException
{
        System.out.println("拦截...");
        filterChain.doFilter(servletRequest, servletResponse);
        System.out.println("再次拦截...");

    }

    @Override
    public void destroy() {

    }
}
```

2. 在web.xml中进行配置

```
<filter>
  <filter-name>QuickFilter</filter-name>
  <filter-class>com.zichen.filter.QuickFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>QuickFilter</filter-name>
  <url-pattern>/targetServlet</url-pattern>
</filter-mapping>
```

3.5.3 工作原理

3.5.4 生命周期

创建：服务器启动项目加载，创建filter对象，执行init方法（只执行一次）

运行（过滤拦截）：用户访问被拦截目标资源时，执行doFilter方法

销毁：服务器关闭项目卸载时，销毁filter对象，执行destroy方法（只执行一次）

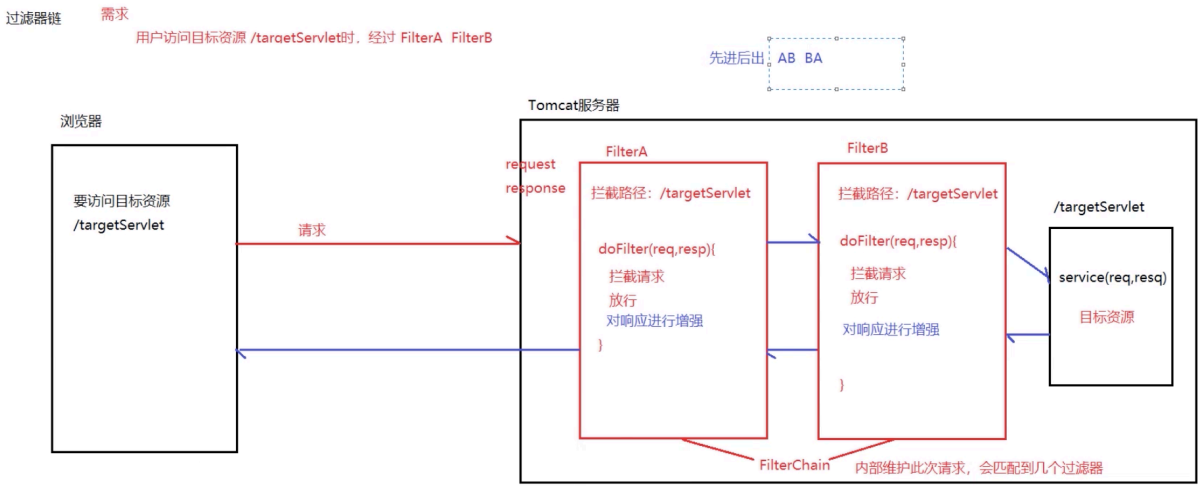
补充：过滤器一定是优先于servlet创建的

3.5.5 Filter拦截路径

中可取的值：

1. 精准匹配 /quickServlet
2. 目录匹配 /xx/*
3. 后缀名匹配 *.xx
4. 匹配所有 /*

3.5.6 Filter过滤器链



3.5.9 Listner监听器概念

监听web三大域对象创建和销毁，request，session，servletContext

3.5.10 Listner快速入门

ServletContextListener接口的API介绍--重要

```
void contextDestroyed(ServletContextEvent sce) //监听servletcontext销毁
void contextInitialized(ServletContextEvent sce) //监听servletcontext创建
```

1.创建一个类实现ServletContextListenner接口

2.实现ServletContextListenner的contextInitialized和contextDestroyed方法。

3.给这个类在xml中配置

HttpSessionListener：监听Httpsession域的创建于销毁的监听器

ServletRequestListener：监听ServletRequest域的创建于销毁的监听器

3.6 MVC模型和三层架构

3.6.1 JSP发展史

3.6.2 MVC设计模式介绍

Model-View-Controller：分离业务逻辑和页面显示

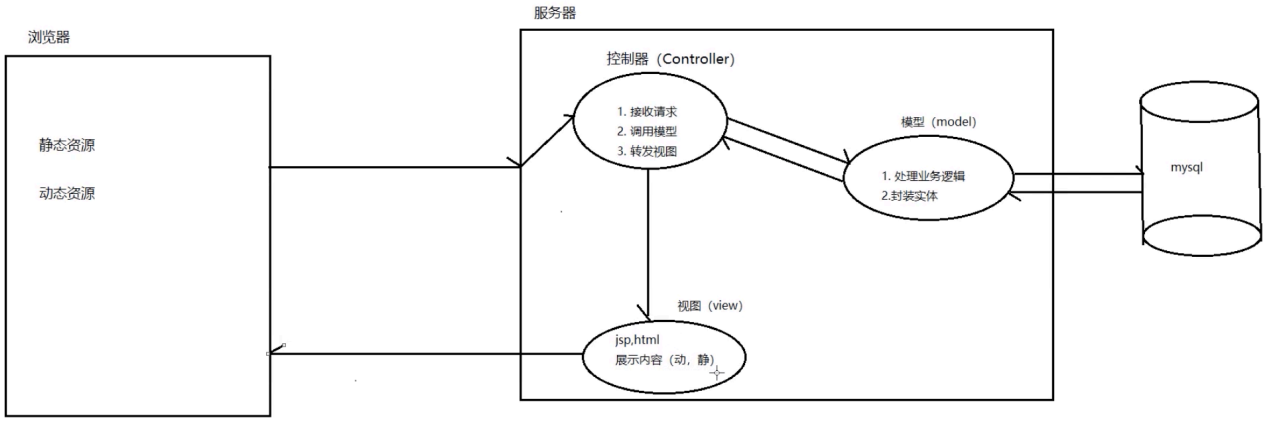
M：model（模型） JavaBean（1.处理业务逻辑、2.封装实体）

V：view（视图） Jsp（展示数据）

C：controller（控制器）Servlet（1.接收请求、2.调用模型、3.转发视图） MVC:笔试题

优点：降低耦合性，方便维护和拓展，利于分工协作

缺点：使得项目架构变得复杂，对开发人员要求高



3.6.3 三层架构介绍

将整个业务应用划分为：

1. 表示（现）层：又称为web层，与浏览器进行数据交互（控制器和视图）
2. 业务逻辑层：又称为service层，处理业务数据（if判断，for循环）
3. 数据访问（持久）层：又称为dao层，与数据库进行交互的（每一条（行）记录与javaBean实体对应）

包目录：

* com.xxx 基本包（公司域名倒写）

* com.xxx.dao 持久层

* com.xxx.service 业务层

* com.xxx.web 表示层

* com.xxx.domain 实体（JavaBean）

* com.xxx.util 工具

三层架构的体现：降低各层之间的耦合，提高代码的复用性