## javaweb简介

### 概念

Java Web，是用Java技术来解决相关web互联网领域的技术总和。web包括：web服务器和web客户端两部分。Java在客户端的应用有java applet，不过[使用](https://baike.baidu.com/item/%E4%BD%BF%E7%94%A8/7741550" \t "https://baike.baidu.com/item/java%20web/_blank)得很少，Java在服务器端的应用非常的丰富，比如Servlet，JSP和第三方框架等等。Java技术对Web领域的发展注入了强大的动力。

Java的Web框架虽然各不相同，但基本也都是遵循特定的路数的：使用[Servlet](https://baike.baidu.com/item/Servlet" \t "https://baike.baidu.com/item/java%20web/_blank)或者[Filter](https://baike.baidu.com/item/Filter" \t "https://baike.baidu.com/item/java%20web/_blank)拦截请求，使用[MVC](https://baike.baidu.com/item/MVC" \t "https://baike.baidu.com/item/java%20web/_blank)的思想设计架构，使用约定，[XML](https://baike.baidu.com/item/XML" \t "https://baike.baidu.com/item/java%20web/_blank)或 Annotation实现配置，运用Java[面向对象](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1" \t "https://baike.baidu.com/item/java%20web/_blank)的特点，将各种协议属性封装成对象，如request、response、context等，面向对象实现请求和响应的流程。

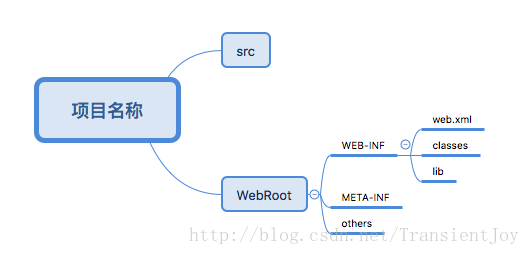
### 请求过程

一次完整的http请求过程，以java tomcat为例，涉及的技术主要有网络编程（数据传输、各种协议http、ip/tcp,socket）、servlet、容器等等

1. 浏览器输入url发起
2. 浏览器解析封装http请求报文socket发送
3. tomcat容器接受http请求报文解析封转request、response，调用app服务器程序
4. app服务器接受业务处理再返回
5. tomcat容器解析response生成响应报文socket发送给浏览器
6. 浏览器接受响应报文解析处理数据或页面（响应body里是一个页面，浏览器会解析并展示）。

### Java web工程

**结构**



src:用于存放项目的java源文件。

WebRoot(文件夹):项目的根目录。

WEB-INT（文件夹）：对客户端不可见。

        web.xml:控件web行为的配置文件。web项目启动时读取该配置。

        classes(文件夹):java文件编译后的class文件。

        lib(文件夹):项目的依赖文件jar等。

others:其他文件，包括jsp,html,css、资源文件等自定义的文件夹。

META-INF:存放MANIFEST.MF文件，该文件是jar的manifest文件，包含了jar文件的内容描述

发布的包就是webroot部分

**简单的打包命令**

进入到项目目录，通过jar -cvf name.war .  即可把当前的项目打包成war包。

## servlet-api

### servlet概念

Servlet（Server Applet）是Java Servlet的简称，称为小服务程序或服务连接器，用Java编写的服务器端程序，具有独立于平台和协议的特性，主要功能在于交互式地浏览和生成数据，生成动态Web内容。

狭义的Servlet是指Java语言实现的一个接口，广义的Servlet是指任何实现了这个Servlet接口(servlet标准规范servlet-api中)的类，一般情况下，人们将Servlet理解为后者。Servlet运行于支持Java的应用服务器中。从原理上讲，Servlet可以响应任何类型的请求，但绝大多数情况下Servlet只用来扩展基于HTTP协议的Web服务器。

ps：

**动态资源**：当用户多次访问这个资源，资源的源代码永远不会改变的资源。

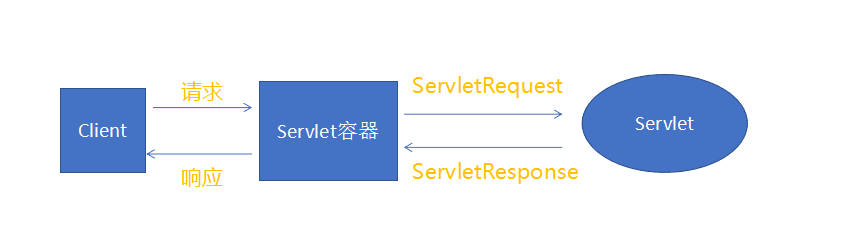
**静态资源**：当用户多次访问这个资源，资源的源代码可能会发送改变。

**动静分离**：动态资源(jsp、ftl、thymeleaf)与静态资源(js、css、img)分开部署，提高用户访问静态代码的速度，降低对后台应用访问，提高网站吞吐量！所以网站中的静态资源一般都存放在第三方存储平台上，例如：七牛云、阿里云、腾讯云等。这些第三方存储平台内置CDN，进行内容分发，采用就近访问原则。

### servlet容器

servlet不是独立的应用程序，没有 [main](https://baike.baidu.com/item/main" \t "https://baike.baidu.com/item/servlet/_blank)() 方法,不是由用户或程序员调用，而是由另外一个应用程序(容器)调用,提供了 Servlet 功能的服务器，叫做 Servlet 容器如tomcat

Servlet容器响应客户请求过程



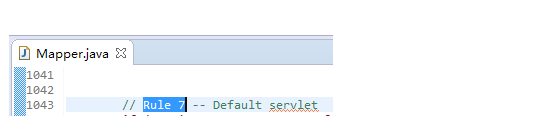
tomcat访问所有的资源，都是用Servlet来实现的。在Tomcat看来，资源分3种

1. 静态资源css,html,js,jpg,png等，如<http://localhost:8082/hello.html>静态页面请求都经过servlet处理Tomcat最后会交由一个叫做DefaultServlet的类来处理。

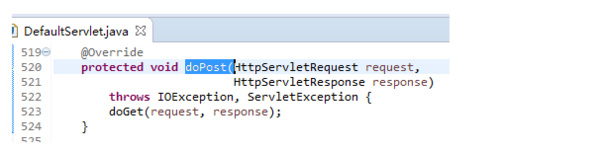
2. Servlet，普通的servlet处理，tomcat最后会交由一个叫做 InvokerServlet的类来处理，找到启动时注册的url再找到对应的servelt。

3. JSP，Tomcat最后会交由一个叫做JspServlet的类来处理

所以Tomcat又叫Servlet容器，什么都交给Servlet来处理。那么什么时候调用哪个Servlet呢？ 有一个类叫做org.apache.tomcat.util.http.mapper.Mapper，它一共进行了7个大的规则判断，第7个，就是判断是否是该用DefaultServlet。简单地说。。。先看是不是servlet,然后看是不是jsp，如果都不是，那么就是你DefaultServlet的活儿了。

[](https://gss0.baidu.com/-fo3dSag_xI4khGko9WTAnF6hhy/zhidao/pic/item/d788d43f8794a4c263267bb204f41bd5ad6e3925.jpg)

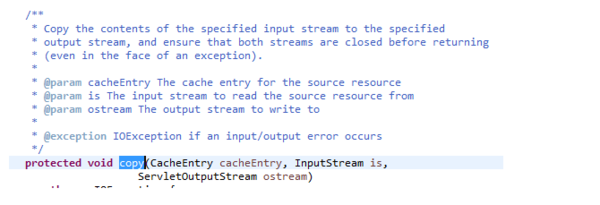
到了DefaultServlet之后，就是一个普通的HttpServlet了，doPost方法会交由doGet处理：

[](https://gss0.baidu.com/-4o3dSag_xI4khGko9WTAnF6hhy/zhidao/pic/item/3801213fb80e7bec98eb9f78252eb9389b506b72.jpg)

doGet又交由一个叫做 serveResource的方法处理

[](https://gss0.baidu.com/9fo3dSag_xI4khGko9WTAnF6hhy/zhidao/pic/item/91ef76c6a7efce1b47325453a551f3deb48f650a.jpg)

在serveResource方法里又瞎搞八搞了许多事情，最后在一个叫做copy()方法里，把静态资源对应的输入流 读取出来，扔到了输出流里，这样你的浏览器就看到数据了。

[](https://gss0.baidu.com/94o3dSag_xI4khGko9WTAnF6hhy/zhidao/pic/item/cefc1e178a82b9011a815faa798da9773912ef7f.jpg)

### servlet生命周期

public interface Servlet {  
 public void init(ServletConfig config) throws ServletException;  
 public ServletConfig getServletConfig();  
 public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;  
 public String getServletInfo();  
 public void destroy();  
}

所有servlet必须实现这个接口，可以继承GenericServlet、HttpServlet等已近实现此接口的类或接口，通过注解@WebServlet或者在部署描述符（web.xml）中用xml配置，在开启web容器时将servlet注册到容器中，一般只在启动项目时注册所以添加修改servlet的url需要重启，当客户端请求某个servlet时，根据注册的url找到相应的servlet，这个servlet会经历一下生命周期

1. 加载 Servlet 类到内存实例化，默认第一次访问servlet的时候创建servlet对象只调用1次，保证servlet对象在tomcat是单实例的。
2. 调用init(ServletConfig config)初始化该 Servlet，实例化后调用，根据config参数初始化该 Servlet ，所以也只调用1次
3. 调用service(req，res)，方法内根据请求方法不同调用doGet() 、doPost()等处理具体业务逻辑，每次发出请求时调用，调用n次。
4. 调用destroy(),销毁servlet对象的时候调用，停止服务器或者重新部署web应用时销毁servlet对象，所以只调用1次。



### Servlet三大域对象

| **对象名称** | **对象的类型** |
| --- | --- |
| request | HttpServletRequest |
| session | HttpSession |
| application | ServletContext |

**作用域**（scope）：程序设计概念，一段程序代码中所用到的名字并不总是有效/可用的，而限定这个名字的可用性的代码范围就是这个名字的作用域。

**域对象的目的**：共享数据，通过setAttrebute、getAttrebute实现保存数据、获取数据。

HttpServletRequest：一次http请求信息的封装，作用域为一次请求

HttpSession：一次会话的封装，对象实例的作用域为一次会话，一次会话里所有的requset共享一个session，会话技术原理参考http协议文档。

ServletContex：上下文对象，作用域为整个应用

**session机制记住登录状态示例**

登录操作



修改操作，需要登录后才能修改



spring security中的登录验证机制原理就是这样的。

**分布式中HttpSession对象实例session共享问题**

产生原因

项目集群部署，由于是集群，浏览器发起登录请求访问的是A服务器，A服务创建一个session对象保存用户相关信息，登录后浏览器再请求时访问的是B服务器，而B服务器没有创建session无法判断是否登录，客户端又得登录，显示是不合理的。

解决方案

|  |  |
| --- | --- |
| 方案 | 介绍 |
| Session Replication | 简介：即session复制，将一台机器上的Session数据广播复制到集群中其余机器上  使用场景：机器较少，网络流量较小  优点：实现简单、配置较少、当网络中有机器Down掉时不影响用户访问  缺点：广播式复制到其余机器有一定廷时，带来一定网络开销 |
| Session Sticky | 简介：即粘性Session，当用户访问集群中某台机器后，强制指定后续所有请求均落到此机器上  使用场景：机器数适中、对稳定性要求不是非常苛刻  优点：实现简单、配置方便、没有额外网络开销  缺点：网络中有机器Down掉时、用户Session会丢失、容易造成单点故障 |
| 缓存集中式管理 | 简介：将Session存入分布式缓存集群中的某台redis机器上，当用户访问不同节点时先从缓存中拿Session信息  使用场景：集群中机器数多、网络环境复杂  优点：可靠性好  缺点：实现复杂、稳定性依赖于缓存的稳定性、Session信息放入缓存时要有合理的策略写入 |

### 过滤器Filter

public interface Filter {  
 public void init(FilterConfig filterConfig) throws ServletException;  
 public void doFilter(ServletRequest request, ServletResponse response,FilterChain chain)throws IOException, ServletException;  
 public void destroy();

}

过滤器是一个实现了 javax.servlet.Filter 接口的 Java 类，它对资源的请求(servlet或静态内容)或资源的响应执行过滤任务，或者两者都执行，一种设计模式，主要用来封装Servlet中一些通用的代码。可用@WebFilter或xml配置声明配置相关参数

public @interface WebFilter {  
 String description() default "";  
 String displayName() default "";  
 WebInitParam[] initParams() default {};  
 String filterName() default "";   
 String smallIcon() default "";  
 String largeIcon() default ""，  
 String[] servletNames() default {};  
 String[] value() default {};

String[] urlPatterns() default {};

//Filter四种拦截方式：request(默认),forward (拦截转发）,include（拦截包含）,error (只拦截错误页面)，forward和include区别详见下面的请求分派器  
 DispatcherType[] dispatcherTypes() default {DispatcherType.REQUEST};

boolean asyncSupported() default false;  
}

**执行过程/生命周期**

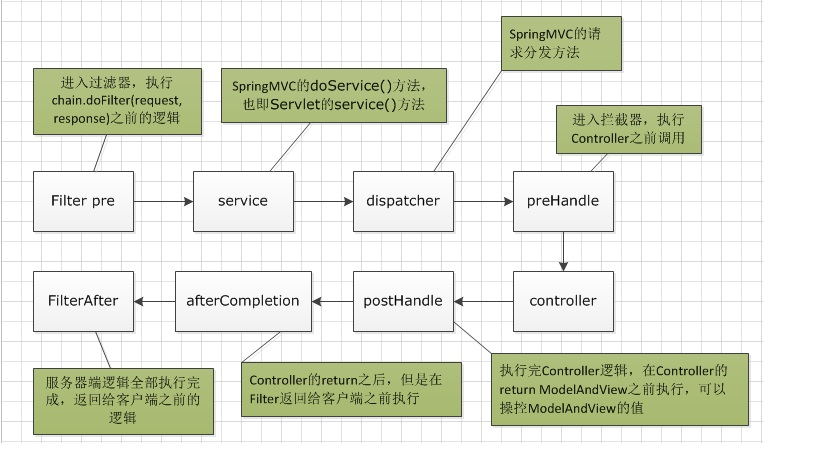


**拦截器**

java里的拦截器是动态拦截Action调用的对象。它提供了一种机制可以使开发者可以定义在一个action执行的前后执行的代码，也可以在一个action执行前阻止其执行，同时也提供了一种可以提取action中可重用部分的方式。在AOP中拦截器用于在某个方法或字段被访问之前或之后，进行拦截然后在之前或之后加入某些操作，如项目中的积分埋点，具体创建学习springmvc拦截器。

**过滤器和拦截器区别**

1. Filter是servlet规范规定的，依赖servlet容器，拦截器是第三方框架支持的如springmvc，可用于web程序、Applicaton、swing等程序中
2. 过滤器是基于函数回调的，拦截器是基于动态代理java反射机制的
3. 过滤器只在servlet前后起作用拦截所有匹配的请求，拦截器能够深入到方法前后、异常抛出前后等可以使用spring里的任何组件、对象更加灵活，在spring项目中优选使用拦截器，在拦截器里注入一个service，可以调用业务逻辑。



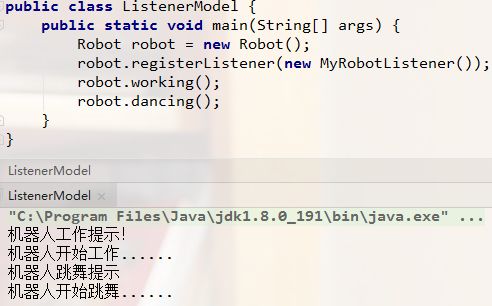
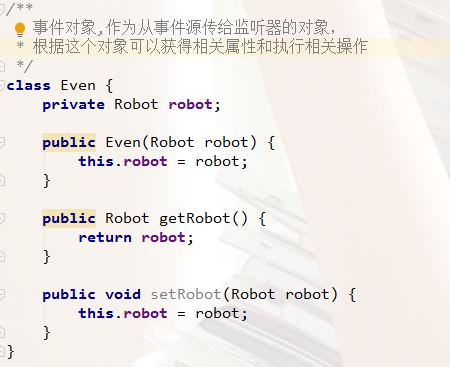
### 监听器Listener

监听观察某个事件（程序）的发生情况，当被监听的事件真的发生了的时候，事件发生者（事件源） 就会给注册该事件的监听者（监听器）发送消息，告诉监听者某些信息，同时监听者也可以获得一份事件对象，根据这个对象可以获得相关属性和执行相关操作。

监听器模型涉及以下三个对象，模型图如下：

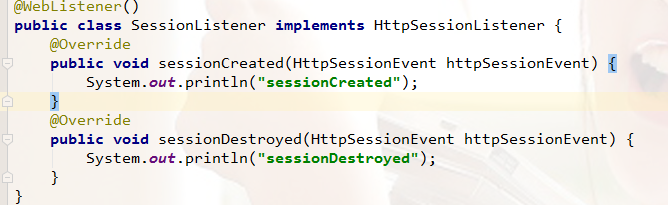
（1）事件：用户对组件的一个操作，或者说程序执行某个方法，称之为一个事件，如机器人程序执行工作。  
（2）事件源：发生事件的组件就是事件源，也就是被监听的对象，如机器人可以工作，可以跳舞，那么就可以把机器人看做是一个事件源。  
（3）事件监听器（处理器）：监听并负责处理事件的方法，如监听机器人工作情况，在机器人工作前后做出相应的动作，或者获取机器人的状态信息。

示例



监听器模式：事件源经过事件的封装传给监听器，当事件源触发事件后，监听器接收到事件对象可以执行监听器要实现的业务功能，也可以回调事件源的方法。  
观察者模式：观察者(Observer)相当于事件监听者，被观察者(Observable)相当于事件源和事件，执行逻辑时通知触发observer的通知方法，将要处理的数据做为参数传递给观察者。

servlet中的监听器接口主要是对request、response、session、context的监听，如对session的监听可以监控到创建session的次数从而知道登录的用户数。



### 请求分派器（转发）

public interface RequestDispatcher{

public void forward(ServletRequest request, ServletResponse response)throws ServletException, IOException;

public void include(ServletRequest request, ServletResponse response)throws ServletException, IOException;

}

RequestDispatcher是一个接口一种规范由servlet容器实现创建对象，其中2个方法代表转发，这就是转发的本质。

RequestDispatcher.forward(request, response)

这个方法将请求从一个 Servlet or JSP目标资源 上 转发到服务器上的另一个资源（servlet、JSP 文件或 HTML 文件，这些资源必须是当前Web上下文中的），让其它的资源去生成响应数据。

例如用户请求的是目标资源A，A接受到请求后，转发到B，真正产生响应数据是被转发的资源B，而A只是起个引导转发作用。浏览器的地址栏不会变，依然是A的URL。 这个方法可以允许被请求的目标资源做一些准备工作后，再让转发的资源去响应请求。

注意事项：

1、在目标资源中调用forward方法时，必须保证此响应没有提交。也就是不要使用 ServletResponse 对象的输出流对象，因为即便你写入了数据到响应缓冲区，最后也会被清空，如果缓冲区数据被刷新提交（out.flush），还会抛出IllegalStateException异常。

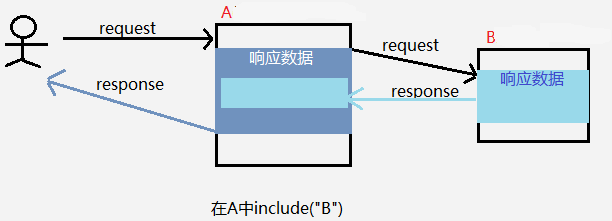
2、对于forward方法传递的request对象：虽然我们从调用上看，好像是将request对象传递给转动的资源上去了，但是我发现目标资源使用的request对象和转发的资源使用的request对象不是同一个request对象，因为分别从这2个request中获取RequestURL，发现是不一样的。但是在目标资源request提取的Paramter 和 Attribute   ，在转发后的资源的request对象中，依然都可以提取到，且是相同的。所以，二者只是在请求路径相关的属性上不同，其它API调用返回的都是一样的。

3、在forward语句的前后，都不应该有响应输出的语句，应该会被忽略。

RequestDispatcher.include(request, response)

此方法用于包含响应中某个资源（servlet、JSP 页面和 HTML 文件）的内容。调用者指定一个被包含的资源，将这个包含的资源（JSP,Servlet，HTML）的响应数据包含到自己的响应体中。即将第二个servlet响应内容包含在第一个响应内容里。

这个过程实质是用一个相同的Request再请求一次被包含的资源，将被包含的资源的响应数据包含到原本的资源中去，构成它的响应数据的一部分。



注意事项：

1、被包含者不能设置ServletResponse的响应状态和响应头（否则并不会产生效果），因为这些都是包含者做的事，被包含者只需要产生响应数据解可以了。

2、不同于 forward中的request的传递特性：在被包含的资源中从request中获取请求路径相关的信息，发现依然是原始请求的路径，也就是浏览器地址栏相关的路径，也就是说被包含的资源获得的request对象的路径属性和原始请求资源的路径一样。其它的API调用也是一样的（Attribute 和Parameter）。

ps：servlet转发和http重定向协议区别参考http协议文档重定向部分。

### servlet异常处理

当一个 Servlet 抛出一个异常时，web容器会在web.xml中查找 error-page配置，可以配置异常类型或状态码对应的servlet或页面，如果没有配置那么会直接将异常信息传给客户端。

|  |
| --- |
| <error-page>            <error-code>500</error-code>            <location>/500error.html</location>      </error-page>  <error-page>            <exception-type>java.lang.NullException</exception-type>            <location>/NullServletHandle</location>    </error-page>  统一的异常页面  <error-page>                  <location>/globalError.jsp</location>    </error-page>  页面也可以是对应的servlet，本质jsp就是一个包含页面信息的servlet |

### servlet多线程并发

servlet对象在tomcat服务器是单实例多线程的。因为servlet是多线程的，所以当多个servlet的线程同时访问了servlet的共享数据，如成员变量，可能会引发线程安全问题。

解决办法：

1. 把使用到共享数据的代码块进行同步（使用synchronized关键字进行同步）
2. 建议在servlet类中尽量不要使用成员变量。如果确实要使用成员，必须同步。而且尽量缩小同步代码块的范围。（哪里使用到了成员变量，就同步哪里！！），以避免因为同步而导致并发效率降低。

### jsp

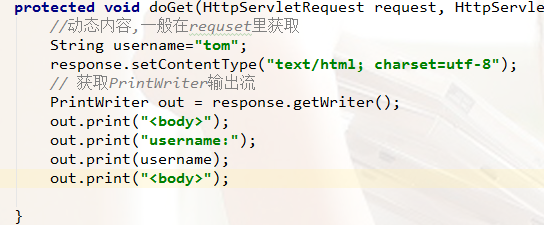
**概念**

java server page由[Sun](https://baike.baidu.com/item/Sun Microsystems" \t "https://baike.baidu.com/item/JSP/_blank) 公司主导创建的一种动态网页技术标准。java服务器页面部署于网络服务器上，可以响应客户端发送的请求，并根据请求内容动态地生成[HTML](https://baike.baidu.com/item/HTML" \t "https://baike.baidu.com/item/JSP/_blank)、[XML](https://baike.baidu.com/item/XML" \t "https://baike.baidu.com/item/JSP/_blank)或其他格式文档的[Web](https://baike.baidu.com/item/Web" \t "https://baike.baidu.com/item/JSP/_blank)网页，然后返回给请求者。其特点是在HTML代码中嵌入JAVA代码，JSP标签或用户标签来生成网页。

**产生背景**

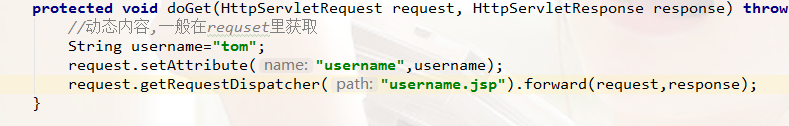
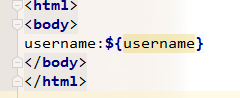
servlet主要功能在于交互式地浏览和生成数据，生成动态Web内容返回给客户端，早期的servlet技术在编写代码时经常通篇性的写一大堆HTML标签，静态文本及文本格式等表现逻辑，其开发效率非常之低下；为了解决这种情况，便随之出现了JSP，其静态部分（包括表现逻辑，如图片，文字等等）全用HTML语言来编写，只有需要动态生成的逻辑才由嵌入的JAVA代码来实现。根据username返回动态页面分别用servlet拼接和jsp方式对比

servlet方式：根据动态的username程序员手动拼接html，将生产的html返回给容器再给浏览器



jsp方式：定义好username.jsp,再在servlet转发到此jsp，本质也是转发列一个servlet，而这个servlet路径就是jsp文件名，因为jsp最终会编译成为java类servlet，

编译器根据jsp的语法拼接html返回给浏览器，不用程序员自己处理



**jsp内容组成（简单认识，详细用法还得看文档）**

* 静态数据，如[HTML](https://baike.baidu.com/item/HTML" \t "https://baike.baidu.com/item/JSP/_blank)
* JSP指令，如include指令<%@ include file="somefile.jsp" %>
* JSP脚本元素和变量

有三个基本的脚本元素，作用是使JAVA代码可以直接插入servlet.

一种是声明标签，在JAVA SERVLET的类体中放入一个变量的定义。静态的数据成员也可以如此定义。<%! int serverInstanceVariable = 1; %>

一种是脚本标签，在JAVA SERVLET的类的\_jspService()方法中放入所包含的语句。<% int localStackBasedVariable = 1; out.println(localStackBasedVariable); %>

一种是表达式标签，在JAVA SERVLET的类中放入待赋值的表达式，表达式注意不能以分号结尾。<%= "expanded inline data " + 1 %>

变量如内置变量request、page

* JSP动作，如<jsp:include page="date.jsp" flush="true" />，与JSP指令元素不同的是，JSP动作元素在请求处理阶段起作用
* 标签，JSTL是一个JSP标签集合，它封装了JSP应用的通用核心功能。如<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>用[<c:if>](https://www.runoob.com/jsp/jstl-core-if-tag.html" \t "https://www.runoob.com/jsp/_blank)表示if

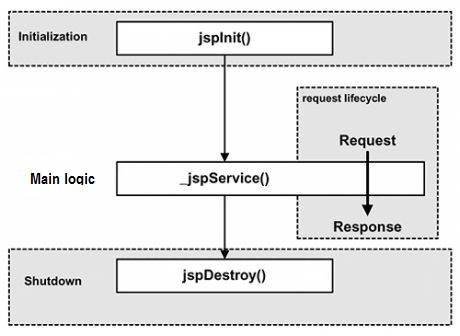
**jsp九大内置对象**

JSP隐式对象是JSP容器为每个页面提供的Java对象，开发者可以直接使用它们而不用显式声明。JSP隐式对象也被称为预定义变量。

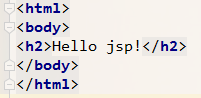
|  |  |
| --- | --- |
| 对象 | 描述 |
| request | HttpServletRequest 接口的实例,如在jsp中直接用 <% out.println(“ip adress:”+request.getRomoteAddr()) %> |
| response | HttpServletResponse 接口的实例 |
| out | JspWriter类的实例，用于把结果输出至网页上 |
| session | HttpSession类的实例 |
| application | ServletContext类的实例，与应用上下文有关 |
| config | ServletConfig类的实例 |
| pageContext | PageContext类的实例，提供对JSP页面所有对象以及命名空间的访问 |
| page | 类似于Java类中的this关键字 |
| Exception | Exception类的对象，代表发生错误的JSP页面中对应的异常对象 |

**生命周期**

Jsp本质是servlet，所以先编译成servlet对象后其生命周期和servlet一样



如hello.jsp 和编译后在work\Catalina\localhost\servlet\org\apache\jsp的hello\_jsp.java



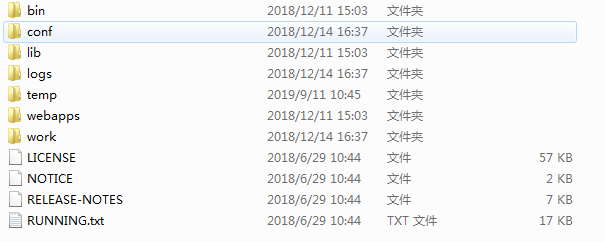


## tomcat

### 简介

Tomcat 服务器是一个免费的开放源代码的Web 应用服务器，由于有了Sun 的参与和支持，最新的Servlet 和JSP 规范总是能在Tomcat 中得到体现，技术先进、性能稳定，而且免费，因而深受Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的Web 应用服务器。

### 目录介绍



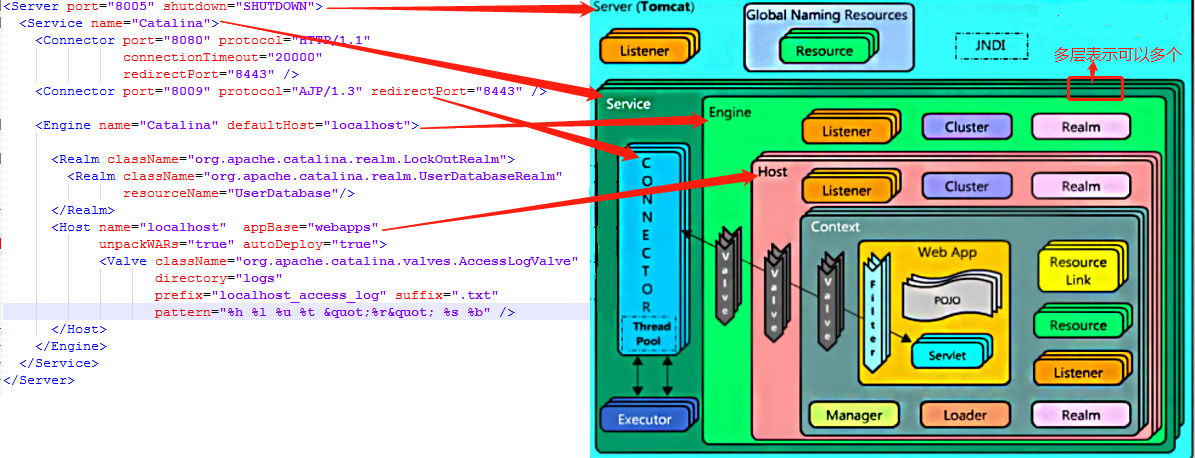
bin：存放各种平台下启动和关闭Tomcat的脚本文件  
conf：存放Tomcat服务器的各种全局配置文件，其中最重要的是server.xml和web.xml  
lib：存放Tomcat运行需要的库文件（JARS）；  
logs：存放Tomcat执行时的LOG文件；  
temp： 存放Tomcat运行时所产生的临时文件  
webapps：Tomcat的主要Web发布目录，默认情况下把Web应用文件放于此目录  
work：存放jsp编译后产生的class文件；

### 常用配置

**配置文件**

* server.xml：主要的配置文件。
* web.xml：缺省的web app配置，WEB-INF/web.xml会覆盖该配置。
* context.xml：上下文配置即一个context代表一个应用

**server.xml**



即一个由 Server->Service->Engine->Host->Context 组成的结构，从里层向外层分别是：

* Server：服务器Tomcat的顶级元素，它包含了所有东西。一个tomcat只有一个server。
* Service：一组 Engine(引擎) 的集合，包括线程池 Executor 和连接器 Connector 的定义。
* Engine(引擎)：一个 Engine代表一个完整的 Servlet 引擎，它接收来自Connector的请求，并决定传给哪个Host来处理。
* Container(容器)：Host、Context、Engine和Wraper都继承自Container接口，它们都是容器。
* Connector(连接器)：将Service和Container连接起来，注册到一个Service，把来自客户端的请求转发到Container。
* Host：即虚拟主机，所谓的”一个虚拟主机”可简单理解为”一个网站”。
* Context(上下文 )： 即 Web 应用程序，一个 Context 即对于一个 Web 应用程序。Context容器直接管理Servlet的运行，Servlet会被其给包装成一个StandardWrapper类去运行。Wrapper负责管理一个Servlet的装载、初始化、执行以及资源回收，它是最底层容器。

**server标签**

1. port：指定一个端口，这个端口负责监听关闭tomcat的请求。
2. shutdown：指定向端口发送的命令字符串。

**service标签**

1. name：指定service的名字。

**Connector标签**

1. port：指定服务器端要创建的端口号，并在这个端口监听来自客户端的请求。
2. minProcessors：服务器启动时创建的处理请求的线程数。
3. maxProcessors：最大可以创建的处理请求的线程数。
4. enableLookups：如果为true，则可以通过调用request.getRemoteHost()进行DNS查询来得到远程客户端的实际主机名，若为false则不进行DNS查询， 而是返回其ip地址。
5. redirectPort：指定服务器正在处理http请求时收到了一个SSL传输请求后重定向的端口号。
6. acceptCount：指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理
7. connectionTimeout：指定超时的时间数(以毫秒为单位)。

**Engine标签**

1. defaultHost：指定缺省的处理请求的主机名，它至少与其中的一个host元素的name属性值是一样的。

**Context标签。**

1. docBase：该web应用的文档基准目录（Document Base，也称为Context Root），或者是WAR文件的路径。可以使用绝对路径，也可以使用相对于context所属的Host的appBase 路径。
2. path：表示此web应用程序的url的前缀，这样请求的url为http://localhost:8080/path/\*\*\*\*。
3. reloadable：这个属性非常重要，如果为true，则tomcat会自动检测应用程序的/WEB-INF/lib和/WEB-INF/classes目录的变化，自动装载新的应用程序， 我们可以在不重起tomcat的情况下改变应用程序。
4. useNaming：如果希望Catalina为该web应用使能一个JNDI InitialContext对象，设为true。该InitialialContext符合J2EE平台的约定，缺省值为true。
5. Context提供的临时目录的路径，用于servlet的临时读/写。利用javax.servlet.context.tempdir属性，servlet可以访问该目录。如果没有指定，使用$CATALINA\_HOME/work下一个 合适的目录。
6. debug：与这个Engine关联的Logger记录的调试信息的详细程度。数字越大，输出越详细。如果没有指定，缺省为0。

**host标签**

1. name：指定主机名。
2. appBase：应用程序基本目录，即存放应用程序的目录。
3. unpackWARs：如果为true，则tomcat会自动将WAR文件解压，否则不解压，直接从WAR文件中运行应用程序

**Realm(标签**

1. className：指定Realm使用的类名，此类必须实现org.apache.catalina.Realm接口。  
   **Valve标签**
2. className：指定Valve使用的类名，如用org.apache.catalina.valves.AccessLogValve类可以记录应用程序的访问信息。
3. directory：指定log文件存放的位置。
4. pattern：有两个值，common方式记录远程主机名或ip地址，用户名，日期，第一行请求的字符串，HTTP响应代码，发送的字节数。combined方式 比common方式记 录的值更多。

### 优化

Tomcat作为Web服务器，它的处理性能直接关系到用户体验，下面是几种常见的优化措施：

**一、掉对web.xml的监视，把jsp提前编辑成Servlet。有富余物理内存的情况，加大tomcat使用的jvm的内存**

**二、服务器资源**服务器所能提供CPU、内存、硬盘的性能对处理能力有决定性影响。  
(1) 对于高并发情况下会有大量的运算，那么CPU的速度会直接影响到处理速度。  
(2)内存在大量数据处理的情况下，将会有较大的内存容量需求，可以用-Xmx -Xms -XX:MaxPermSize等参数对内存不同功能块进行划分。我们之前就遇到过 内存分配不足，导致虚拟机一直处于full GC，从而导致处理能力严重下降。  
(3) 硬盘主要问题就是读写性能，当大量文件进行读写时，磁盘极容易成为性能瓶颈。最好的办法还是利用下面提到的缓存。

**三、利用缓存和压缩**  
　　对于静态页面最好是能够缓存起来，这样就不必每次从磁盘上读。这里我们采用了Nginx作为缓存服务器，将图片、css、js文件都进行了缓存，有效的减少了后端tomcat的访问。另外，为了能加快网络传输速度，开启gzip压缩也是必不可少的。但考虑到tomcat已经需要处理很多东西了，所以把这个压缩的工作就交给前端的Nginx来完成。除了文本可以用gzip压缩，其实很多图片也可以用图像处理工具预先进行压缩，找到一个平衡点可以让画质损失很小而文件可以减小很多。曾经我就见过一个图片从300多kb压缩到几十kb，自己几乎看不出来区别。

**四、采用集群**  
单个服务器性能总是有限的，最好的办法自然是实现横向扩展，那么组建tomcat集群是有效提升性能的手段。我们还是采用了Nginx来作为请求分流的服务器，后端多个tomcat共享session来协同工作。

**五、 优化tomcat参数**

根据项目配置合理参数，关闭客户端dns查询。

**1、优化内存**

主要是在bin/catalina.bat/sh 配置文件中进行。linux上，在catalina.sh中添加：JAVA\_OPTS="-server -Xms1G -Xmx2G -Xss256K -Djava.awt.headless=true -Dfile.encoding=utf-8 -XX:MaxPermSize=256m -XX:PermSize=128M -XX:MaxPermSize=256M"

• -server：启用jdk的server版本。

• -Xms：虚拟机初始化时的最小堆内存。

• -Xmx：虚拟机可使用的最大堆内存。 #-Xms与-Xmx设成一样的值，避免JVM因为频繁的GC导致性能大起大落

• -XX:PermSize：设置非堆内存初始值,默认是物理内存的1/64。

• -XX:MaxNewSize：新生代占整个堆内存的最大值。

• -XX:MaxPermSize：Perm（俗称方法区）占整个堆内存的最大值，也称内存最大永久保留区域。

1）错误提示：java.lang.OutOfMemoryError:Java heap space

Tomcat默认可以使用的内存为128MB，在较大型的应用项目中，这点内存是不够的，有可能导致系统无法运行。常见的问题是报Tomcat内存溢出错误，Outof Memory(系统内存不足)的异常，从而导致客户端显示500错误，一般调整Tomcat的-Xms和-Xmx即可解决问题，通常将-Xms和-Xmx设置成一样，堆的最大值设置为物理可用内存的最大值的80%。

set JAVA\_OPTS=-Xms512m-Xmx512m

2）错误提示：java.lang.OutOfMemoryError: PermGenspace

PermGenspace的全称是Permanent Generationspace,是指内存的永久保存区域，这块内存主要是被JVM存放Class和Meta信息的,Class在被Loader时就会被放到PermGenspace中，它和存放类实例(Instance)的Heap区域不同,GC(Garbage Collection)不会在主程序运行期对PermGenspace进行清理，所以如果你的应用中有很CLASS的话,就很可能出现PermGen space错误，这种错误常见在web服务器对JSP进行precompile的时候。如果你的WEB APP下都用了大量的第三方jar, 其大小超过了jvm默认的大小(4M)那么就会产生此错误信息了。解决方法：

setJAVA\_OPTS=-XX:PermSize=128M

3）在使用-Xms和-Xmx调整tomcat的堆大小时，还需要考虑垃圾回收机制。如果系统花费很多的时间收集垃圾，请减小堆大小。一次完全的垃圾收集应该不超过3-5 秒。如果垃圾收集成为瓶颈，那么需要指定代的大小，检查垃圾收集的详细输出，研究垃圾收集参数对性能的影响。一般说来，你应该使用物理内存的 80% 作为堆大小。当增加处理器时，记得增加内存，因为分配可以并行进行，而垃圾收集不是并行的。

**2、连接数优化**

优化连接数，主要是在conf/server.xml配置文件中进行修改。

2.1、优化线程数

找到Connector port="8080" protocol="HTTP/1.1"，增加maxThreads和acceptCount属性（使acceptCount大于等于maxThreads），如下：<Connector port="8080" protocol="HTTP/1.1"connectionTimeout="20000" redirectPort="8443"acceptCount="500" maxThreads="400" />

• maxThreads：tomcat可用于请求处理的最大线程数，默认是200

• minSpareThreads：tomcat初始线程数，即最小空闲线程数

• maxSpareThreads：tomcat最大空闲线程数，超过的会被关闭

• acceptCount：当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理.默认100

2.2、使用线程池

在server.xml中增加executor节点，然后配置connector的executor属性，如下：

<Executor name="tomcatThreadPool" namePrefix="req-exec-"maxThreads="1000" minSpareThreads="50"maxIdleTime="60000"/>

<Connector port="8080" protocol="HTTP/1.1"executor="tomcatThreadPool"/>

• namePrefix：线程池中线程的命名前缀

• maxThreads：线程池的最大线程数

• minSpareThreads：线程池的最小空闲线程数

• maxIdleTime：超过最小空闲线程数时，多的线程会等待这个时间长度，然后关闭

• threadPriority：线程优先级

注：当tomcat并发用户量大的时候，单个jvm进程确实可能打开过多的文件句柄，这时会报java.net.SocketException:Too many open files错误。可使用下面步骤检查：

• ps -ef |grep tomcat 查看tomcat的进程ID，记录ID号，假设进程ID为10001

• lsof -p 10001|wc -l 查看当前进程id为10001的 文件操作数

• 使用命令：ulimit -a 查看每个用户允许打开的最大文件数

**3、Tomcat Connector三种运行模式（BIO, NIO, APR）**

3.1、三种模式比较：

1）BIO：一个线程处理一个请求。缺点：并发量高时，线程数较多，浪费资源。Tomcat7或以下在Linux系统中默认使用这种方式。

2）NIO：利用Java的异步IO处理，可以通过少量的线程处理大量的请求。Tomcat8在Linux系统中默认使用这种方式。Tomcat7必须修改Connector配置来启动（conf/server.xml配置文件）：

<Connector port="8080"protocol="org.apache.coyote.http11.Http11NioProtocol" connectionTimeout="20000"redirectPort="8443"/>

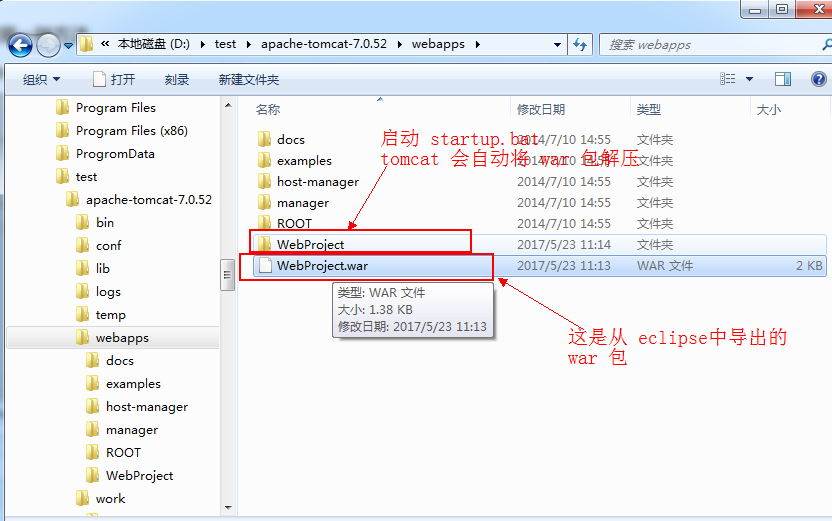
3）APR(Apache Portable Runtime)：从操作系统层面解决io阻塞问题。Linux如果安装了apr和native，Tomcat直接启动就支持apr。

**4、禁用DNS查询**  
当web应用程序想要记录客户端的信息时，它也会记录客户端的IP地址或者通过域名服务器查找机器名转换为IP地址。DNS查询需要占用网络，并且包括可能从很多很远的服务器或者不起作用的服务器上去获取对应的IP的过程，这样会消耗一定的时间。为了消除DNS查询对性能的影响我们可以关闭DNS查询，方式是修改server.xml文件中的enableLookups参数值

### 部署项目

**方式一：项目直接放入 webapps 目录中（最常用，一般部署就采用这种方式）**

将编写并编译好的web项目(注意要是编译好的，如果是 eclipse，可以将项目打成 war 包放入，放入到 webapps 中



**方式二：修改 conf/server.xml 文件**

打开tomcat下conf/server.xml，在<Host> </Host>标签之间输入项目配置信息

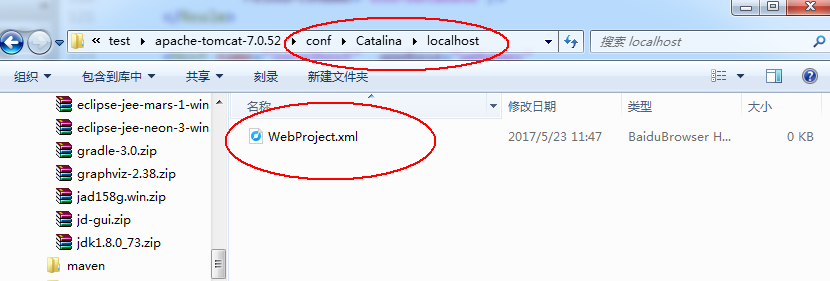
|  |
| --- |
| <Context path="/WebRoot" docBase="D:/WebRoot" /> |

path:浏览器访问时的路径名

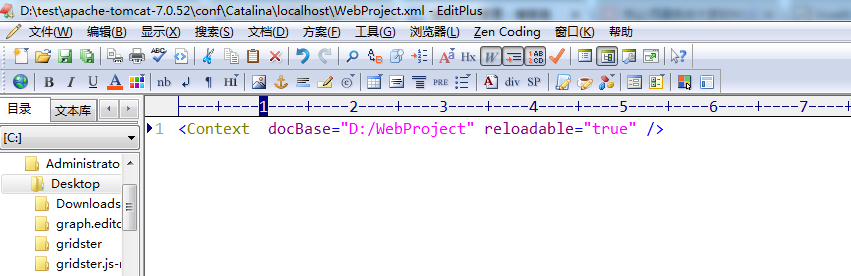
docBase:web项目的WebRoot所在的路径，注意是WebRoot的路径，不是项目的路径。其实也就是编译后的项目

**方式三：apache-tomcat-7.0.52\conf\Catalina\localhost配置**

进入到 apache-tomcat-7.0.52\conf\Catalina\localhost 目录，新建一个 项目名.xml 文件



在那个新建的 xml 文件中，增加下面配置语句（和上面的是一样的,但是不需要 path 配置，加上也没什么用）



### howtomcatworks



## web服务演变

### 演变过程

#### 前言

我们以javaweb为例，来搭建一个简单的电商系统，看看这个系统可以如何一步步演变。该系统具备的功能：

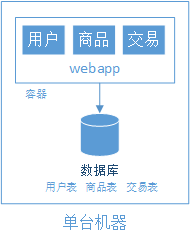
用户模块：用户注册和管理

商品模块：商品展示和管理

交易模块：创建交易和管理

#### 阶段一、单机构建网站

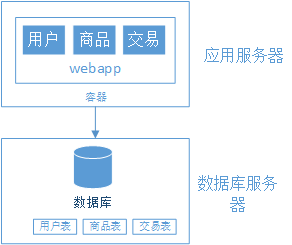
网站的初期，我们经常会在单机上跑我们所有的程序和软件。此时我们使用一个容器，如tomcat、jetty、jboos，然后直接使用JSP/servlet技术，或者使用一些开源的框架如maven+spring+struct+hibernate、maven+spring+springmvc+mybatis；最后再选择一个数据库管理系统来存储数据，如mysql、sqlserver、oracle，然后通过JDBC进行数据库的连接和操作。把以上的所有软件都装载同一台机器上，应用跑起来了，也算是一个小系统了。此时系统结果如下：



#### 阶段二、应用服务器与数据库分离

随着网站的上线，访问量逐步上升，服务器的负载慢慢提高，在服务器还没有超载的时候，我们应该就要做好准备，提升网站的负载能力。假如我们代码层面已难以优化，在不提高单台机器的性能的情况下，增加机器是一个不错的方式，不仅可以有效地提高系统的负载能力，而且性价比高。增加的机器用来做什么呢？此时我们可以把数据库，web服务器拆分开来，这样不仅提高了单台机器的负载能力，也提高了容灾能力。

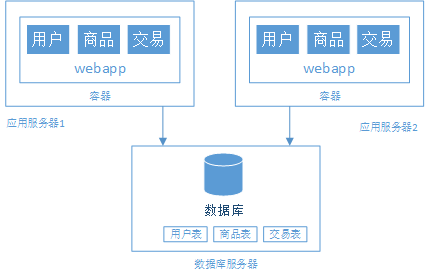
应用服务器与数据库分开后的架构如下图所示：



#### 阶段三、应用服务器集群

随着访问量继续增加，单台应用服务器已经无法满足需求了。在假设数据库服务器没有压力的情况下，我们可以把应用服务器从一台变成了两台甚至多台，把用户的请求分散到不同的服务器中，从而提高负载能力。多台应用服务器之间没有直接的交互，他们都是依赖数据库各自对外提供服务。著名的做故障切换的软件有keepalived，keepalived是一个类似于layer3、4、7交换机制的软件，他不是某个具体软件故障切换的专属品，而是可以适用于各种软件的一款产品。keepalived配合上ipvsadm又可以做负载均衡，可谓是神器。

我们以增加了一台应用服务器为例，增加后的系统结构图如下：



系统演变到这里，将会出现下面四个问题：

一、用户的请求由谁来转发到到具体的应用服务器

二、有什么转发的算法

三、应用服务器如何返回用户的请求

四、用户如果每次访问到的服务器不一样，那么如何维护session的一致性

我们来看看解决问题的方案：

一、第一个问题即是负载均衡的问题，一般有5种解决方案：

　　　　1、http重定向。HTTP重定向就是应用层的请求转发。用户的请求其实已经到了HTTP重定向负载均衡服务器，服务器根据算法要求用户重定向，用户收到重定向请求后，再次请求真正的集群

　　　　　　优点：简单。

　　　　　　缺点：性能较差。

　　　　2、DNS域名解析负载均衡。DNS域名解析负载均衡就是在用户请求DNS服务器，获取域名对应的IP地址时，DNS服务器直接给出负载均衡后的服务器IP。

　　　　　　优点：交给DNS，不用我们去维护负载均衡服务器。

　　　　　　缺点：当一个应用服务器挂了，不能及时通知DNS，而且DNS负载均衡的控制权在域名服务商那里，网站无法做更多的改善和更强大的管理。

　　　　3、反向代理服务器。在用户的请求到达反向代理服务器时（已经到达网站机房），由反向代理服务器根据算法转发到具体的服务器。常用的apache，nginx都可以充当反向代理服务器。

　　　　　　优点：部署简单。

　　　　　　缺点：代理服务器可能成为性能的瓶颈，特别是一次上传大文件。

　　　　4、IP层负载均衡。在请求到达负载均衡器后，负载均衡器通过修改请求的目的IP地址，从而实现请求的转发，做到负载均衡。

　　　　　　优点：性能更好。

　　　　　　缺点：负载均衡器的宽带成为瓶颈。

　　　　5、数据链路层负载均衡。在请求到达负载均衡器后，负载均衡器通过修改请求的mac地址，从而做到负载均衡，与IP负载均衡不一样的是，当请求访问完服务器之后，直接返回客户而无需再经过负载均衡器。

二、第二个问题即是集群调度算法问题，常见的调度算法有10种。

　　　　1、rr 轮询调度算法。顾名思义，轮询分发请求。

　　　　　　优点：实现简单

　　　　　　缺点：不考虑每台服务器的处理能力

　　　　2、wrr 加权调度算法。我们给每个服务器设置权值weight，负载均衡调度器根据权值调度服务器，服务器被调用的次数跟权值成正比。

　　　　　　优点：考虑了服务器处理能力的不同

　　　　3、sh 原地址散列：提取用户IP，根据散列函数得出一个key，再根据静态映射表，查处对应的value，即目标服务器IP。过目标机器超负荷，则返回空。

　　　　4、dh 目标地址散列：同上，只是现在提取的是目标地址的IP来做哈希。

　　　　　　优点：以上两种算法的都能实现同一个用户访问同一个服务器。

　　　　5、lc 最少连接。优先把请求转发给连接数少的服务器。

　　　　　　优点：使得集群中各个服务器的负载更加均匀。

　　　　6、wlc 加权最少连接。在lc的基础上，为每台服务器加上权值。算法为：（活动连接数\*256+非活动连接数）÷权重 ，计算出来的值小的服务器优先被选择。

　　　　　　优点：可以根据服务器的能力分配请求。

　　　　7、sed 最短期望延迟。其实sed跟wlc类似，区别是不考虑非活动连接数。算法为：（活动连接数+1)\*256÷权重，同样计算出来的值小的服务器优先被选择。

　　　　8、nq 永不排队。改进的sed算法。我们想一下什么情况下才能“永不排队”，那就是服务器的连接数为0的时候，那么假如有服务器连接数为0，均衡器直接把请求转发给它，无需经过sed的计算。

　　　　9、LBLC 基于局部性的最少连接。均衡器根据请求的目的IP地址，找出该IP地址最近被使用的服务器，把请求转发之，若该服务器超载，最采用最少连接数算法。

　　　　10、LBLCR 带复制的基于局部性的最少连接。均衡器根据请求的目的IP地址，找出该IP地址最近使用的“服务器组”，注意，并不是具体某个服务器，然后采用最少连接数从该组中挑出具体的某台服务器出来，把请求转发之。若该服务器超载，那么根据最少连接数算法，在集群的非本服务器组的服务器中，找出一台服务器出来，加入本服务器组，然后把请求转发之。

三、第三个问题是集群模式问题，一般3种解决方案：

　　　　1、NAT：负载均衡器接收用户的请求，转发给具体服务器，服务器处理完请求返回给均衡器，均衡器再重新返回给用户。

　　　　2、DR：负载均衡器接收用户的请求，转发给具体服务器，服务器出来玩请求后直接返回给用户。需要系统支持IP Tunneling协议，难以跨平台。

　　　　3、TUN：同上，但无需IP Tunneling协议，跨平台性好，大部分系统都可以支持。

四、第四个问题是session问题，一般有4种解决方案：

　　　　1、Session Sticky。session sticky就是把同一个用户在某一个会话中的请求，都分配到固定的某一台服务器中，这样我们就不需要解决跨服务器的session问题了，常见的算法有ip\_hash法，即上面提到的两种散列算法。

　　　　　　优点：实现简单。

　　　　　　缺点：应用服务器重启则session消失。

　　　　2、Session Replication。session replication就是在集群中复制session，使得每个服务器都保存有全部用户的session数据。

　　　　　　优点：减轻负载均衡服务器的压力，不需要要实现ip\_hasp算法来转发请求。

　　　　　　缺点：复制时宽带开销大，访问量大的话session占用内存大且浪费。

　　　　3、Session数据集中存储：session数据集中存储就是利用数据库来存储session数据，实现了session和应用服务器的解耦。

　　　　　　优点：相比session replication的方案，集群间对于宽带和内存的压力减少了很多。

　　　　　　缺点：需要维护存储session的数据库。

　　　　4、Cookie Base：cookie base就是把session存在cookie中，有浏览器来告诉应用服务器我的session是什么，同样实现了session和应用服务器的解耦。

　　　　　　优点：实现简单，基本免维护。

　　　　　　缺点：cookie长度限制，安全性低，宽带消耗。

　　值得一提的是：

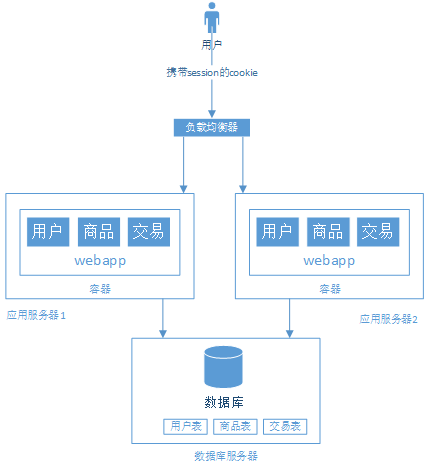
　　nginx目前支持的负载均衡算法有wrr、sh（支持一致性哈希）、fair（本人觉得可以归结为lc）。但nginx作为均衡器的话，还可以一同作为静态资源服务器。

　　keepalived+ipvsadm比较强大，目前支持的算法有：rr、wrr、lc、wlc、lblc、sh、dh

　　keepalived支持集群模式有：NAT、DR、TUN

　　nginx本身并没有提供session同步的解决方案，而apache则提供了session共享的支持。

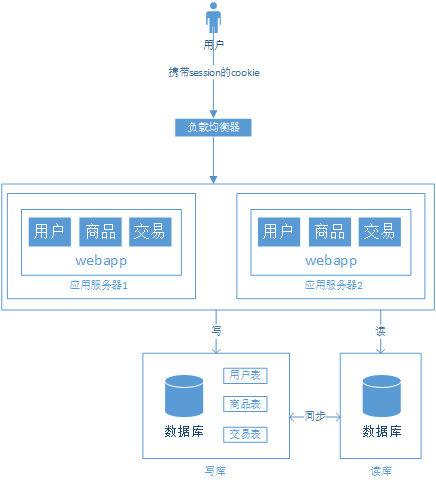
好了，解决了以上的问题之后，系统的结构如下：



#### 阶段四、数据库读写分离化

上面我们总是假设数据库负载正常，但随着访问量的的提高，数据库的负载也在慢慢增大。那么可能有人马上就想到跟应用服务器一样，把数据库一份为二再负载均衡即可。但对于数据库来说，并没有那么简单。假如我们简单的把数据库一分为二，然后对于数据库的请求，分别负载到A机器和B机器，那么显而易见会造成两台数据库数据不统一的问题。那么对于这种情况，我们可以先考虑使用读写分离的方式。

读写分离后的数据库系统结构如下：



这个结构变化后也会带来两个问题：

一、主从数据库之间数据同步问题

二、应用对于数据源的选择问题

解决问题方案：

我们可以使用MYSQL自带的master+slave的方式实现主从复制。

采用第三方数据库中间件，例如mycat。mycat是从cobar发展而来的，而cobar是阿里开源的数据库中间件，后来停止开发。mycat是国内比较好的mysql开源数据库分库分表中间件。

#### 阶段五、用搜索引擎缓解读库的压力

数据库做读库的话，常常对模糊查找力不从心，即使做了读写分离，这个问题还未能解决。以我们所举的交易网站为例，发布的商品存储在数据库中，用户最常使用的功能就是查找商品，尤其是根据商品的标题来查找对应的商品。对于这种需求，一般我们都是通过like功能来实现的，但是这种方式的代价非常大。此时我们可以使用搜索引擎的倒排索引来完成。

搜索引擎具有以下优点：

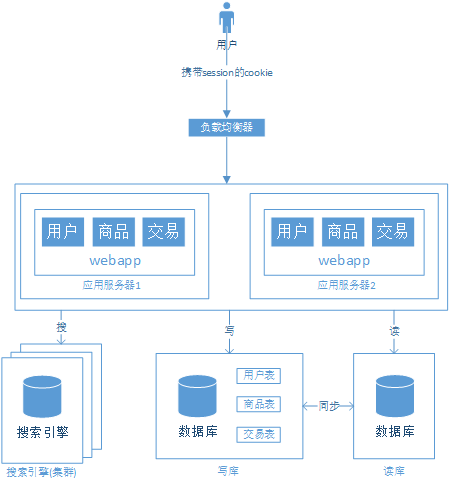
它能够大大提高查询速度。

引入搜索引擎后也会带来以下的开销：

带来大量的维护工作，我们需要自己实现索引的构建过程，设计全量/增加的构建方式来应对非实时与实时的查询需求。

需要维护搜索引擎集群

搜索引擎并不能替代数据库，他解决了某些场景下的“读”的问题，是否引入搜索引擎，需要综合考虑整个系统的需求。引入搜索引擎后的系统结构如下：



#### 阶段六、用缓存缓解读库的压力

1、后台应用层和数据库层的缓存

随着访问量的增加，逐渐出现了许多用户访问同一部分内容的情况，对于这些比较热门的内容，没必要每次都从数据库读取。我们可以使用缓存技术，例如可以使用google的开源缓存技术guava或者使用memcacahe作为应用层的缓存，也可以使用redis作为数据库层的缓存。

另外，在某些场景下，关系型数据库并不是很适合，例如我想做一个“每日输入密码错误次数限制”的功能，思路大概是在用户登录时，如果登录错误，则记录下该用户的IP和错误次数，那么这个数据要放在哪里呢？假如放在内存中，那么显然会占用太大的内容；假如放在关系型数据库中，那么既要建立数据库表，还要简历对应的java bean，还要写SQL等等。而分析一下我们要存储的数据，无非就是类似{ip:errorNumber}这样的key:value数据。对于这种数据，我们可以用NOSQL数据库来代替传统的关系型数据库。

2、页面缓存

除了数据缓存，还有页面缓存。比如使用HTML5的localstroage或者cookie。

优点：

减轻数据库的压力

大幅度提高访问速度

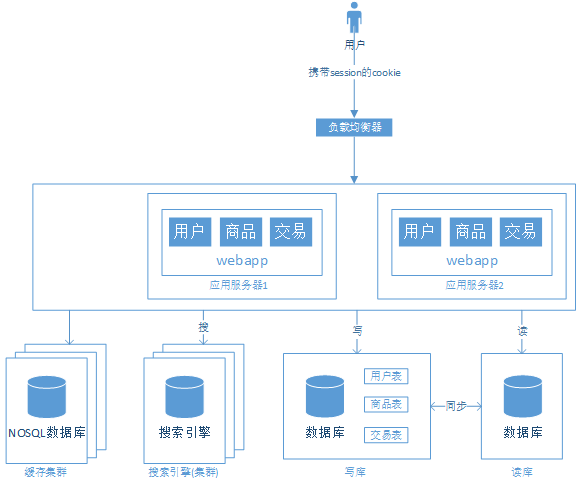
缺点：

需要维护缓存服务器

提高了编码的复杂性

值得一提的是：缓存集群的调度算法不同与上面提到的应用服务器和数据库。最好采用“一致性哈希算法”，这样才能提高命中率。这个就不展开讲了，有兴趣的可以查阅相关资料。

加入缓存后的结构：



#### 阶段七、数据库水平拆分与垂直拆分

我们的网站演进到现在，交易、商品、用户的数据都还在同一个数据库中。尽管采取了增加缓存，读写分离的方式，但随着数据库的压力继续增加，数据库的瓶颈越来越突出，此时，我们可以有数据垂直拆分和水平拆分两种选择。

1、数据垂直拆分

垂直拆分的意思是把数据库中不同的业务数据拆分道不同的数据库中，结合现在的例子，就是把交易、商品、用户的数据分开。

优点：

解决了原来把所有业务放在一个数据库中的压力问题。

可以根据业务的特点进行更多的优化

缺点：

需要维护多个数据库

问题：

需要考虑原来跨业务的事务

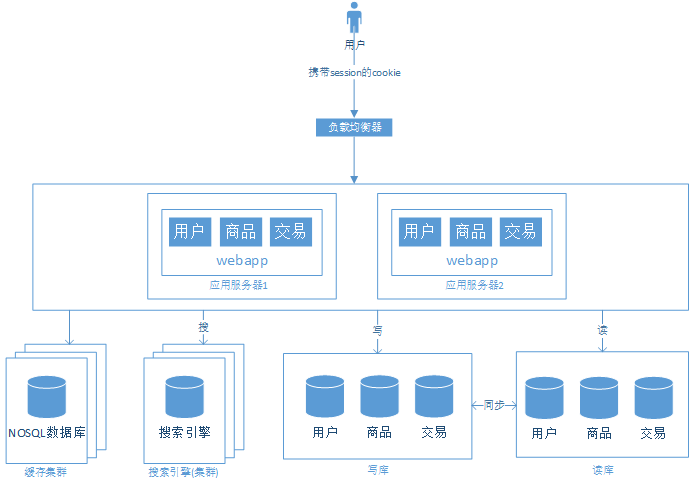
跨数据库的join

解决问题方案：

我们应该在应用层尽量避免跨数据库的事物，如果非要跨数据库，尽量在代码中控制。

我们可以通过第三方应用来解决，如上面提到的mycat，mycat提供了丰富的跨库join方案，详情可参考mycat官方文档。

垂直拆分后的结构如下：



2、数据水平拆分

数据水平拆分就是把同一个表中的数据拆分到两个甚至多个数据库中。产生数据水平拆分的原因是某个业务的数据量或者更新量到达了单个数据库的瓶颈，这时就可以把这个表拆分到两个或更多个数据库中。

优点：

如果我们能客服以上问题，那么我们将能够很好地对数据量及写入量增长的情况。

问题：

访问用户信息的应用系统需要解决SQL路由的问题，因为现在用户信息分在了两个数据库中，需要在进行数据操作时了解需要操作的数据在哪里。

主键的处理也变得不同，例如原来自增字段，现在不能简单地继续使用了。

如果需要分页，就麻烦了。

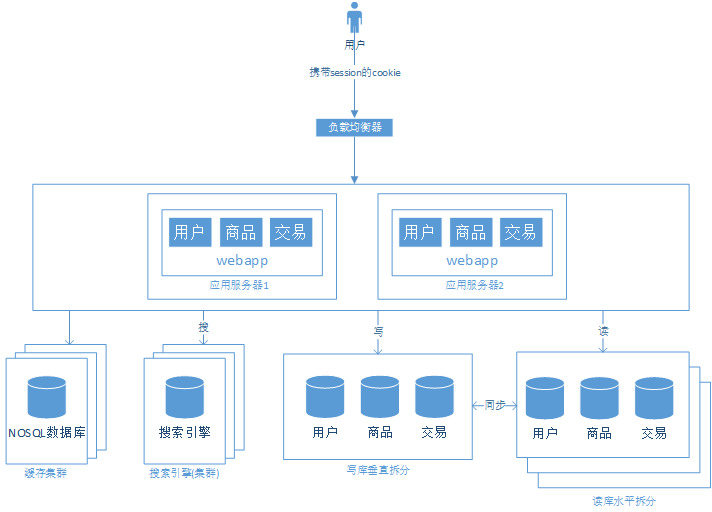
解决问题方案：

我们还是可以通过可以解决第三方中间件，如mycat。mycat可以通过SQL解析模块对我们的SQL进行解析，再根据我们的配置，把请求转发到具体的某个数据库。

我们可以通过UUID保证唯一或自定义ID方案来解决。

mycat也提供了丰富的分页查询方案，比如先从每个数据库做分页查询，再合并数据做一次分页查询等等。

数据水平拆分后的结构：

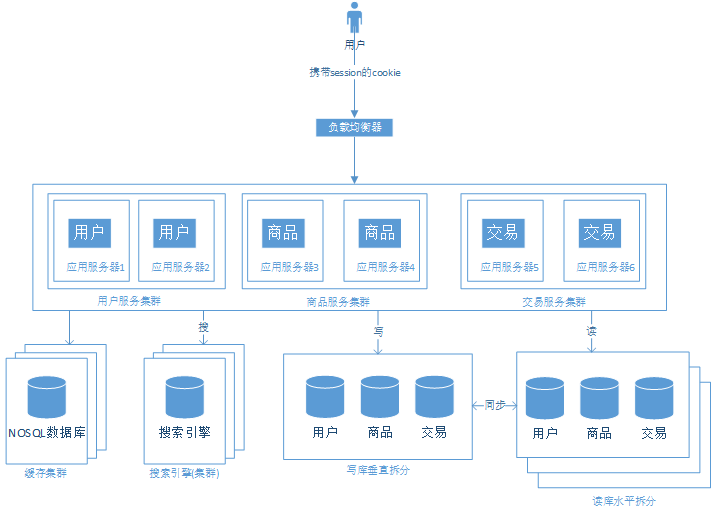


#### 阶段八、应用的拆分

1、拆分应用

随着业务的发展，业务越来越多，应用越来越大。我们需要考虑如何避免让应用越来越臃肿。这就需要把应用拆开，从一个应用变为俩个甚至更多。还是以我们上面的例子，我们可以把用户、商品、交易拆分开。变成“用户、商品”和“用户，交易”两个子系统。

拆分后的结构：



问题：

这样拆分后，可能会有一些相同的代码，如用户相关的代码，商品和交易都需要用户信息，所以在两个系统中都保留差不多的操作用户信息的代码。如何保证这些代码可以复用是一个需要解决的问题。

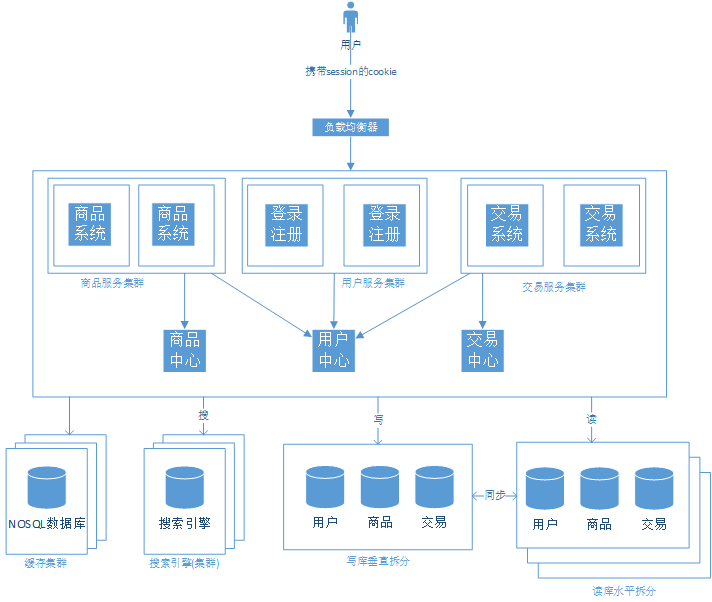
解决问题：

通过走服务化的路线来解决

2、走服务化的道路

为了解决上面拆分应用后所出现的问题，我们把公共的服务拆分出来，形成一种服务化的模式，简称SOA。

采用服务化之后的系统结构：



优点：

相同的代码不会散落在不同的应用中了，这些实现放在了各个服务中心，使代码得到更好的维护。

我们把对数据库的交互放在了各个服务中心，让”前端“的web应用更注重与浏览器交互的工作。

问题：

如何进行远程的服务调用

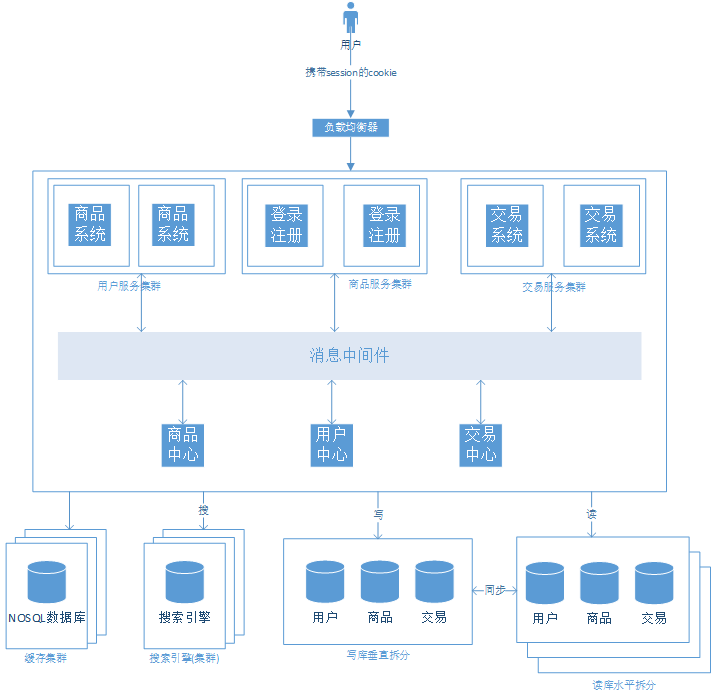
 解决方法：

我们可以通过下面的引入消息中间件来解决

#### 阶段九、引入消息中间件

随着网站的继续发展，我们的系统中可能出现不同语言开发的子模块和部署在不同平台的子系统。此时我们需要一个平台来传递可靠的，与平台和语言无关的数据，并且能够把负载均衡透明化，能在调用过程中收集调用数据并分析之，推测出网站的访问增长率等等一系列需求，对于网站应该如何成长做出预测。开源消息中间件有阿里的dubbo，可以搭配Google开源的分布式程序协调服务zookeeper实现服务器的注册与发现。

引入消息中间件后的结构：



#### 总结

以上的演变过程只是一个例子，并不适合所有的网站，实际中网站演进过程与自身业务和不同遇到的问题有密切的关系，没有固定的模式。只有认真的分析和不断地探究，才能发现适合自己网站的架构。

### 微服务架构

#### 什么是分布式

不同模块部署在不同服务器上

作用：分布式解决网站高并发带来问题

#### 什么是集群

多台服务器部署相同应用构成一个集群

作用：通过负载均衡设备共同对外提供服务，提高高可用。

#### 什么是高可用

高可用HA（High Availability）是分布式系统架构设计中必须考虑的因素之一，它通常是指，通过设计减少系统不能提供服务的时间。假设系统一直能够提供服务，我们说系统的可用性是100%。很多公司的高可用目标是4个9，也就是99.99%，这就意味着，系统的年停机时间为8.76个小时。

#### 什么是RPC

RPC 的全称是 Remote Procedure Call 是一种进程间通信方式。  
它允许程序调用另一个地址空间（通常是共享网络的另一台机器上）的过程或函数，而不用程序员显式编码这个远程调用的细节。即无论是调用本地接口/服务的还是远程的接口/服务，本质上编写的调用代码基本相同。  
比如两台服务器A，B，一个应用部署在A服务器上，想要调用B服务器上应用提供的函数或者方法，由于不在一个内存空间，不能直接调用，这时候需要通过就可以应用RPC框架的实现来解决

**restful、soap、rpc**

（1）RESTful是一种架构设计风格，提供了设计原则和约束条件，而不是架构。而满足这些约束条件和原则的应用程序或设计就是 RESTful架构或服务。  
（2）SOAP，简单对象访问协议是一种数据交换协议规范，是一种轻量的、简单的、基于XML的协议的规范。SOAP协议和HTTP协议一样，都是底层的通信协议，只是请求包的格式不同而已，SOAP包是XML格式的。SOAP的消息是基于xml并封装成了符合http协议，因此，它符合任何路由器、 防火墙或代理服务器的要求。soap可以使用任何语言来完成，只要发送正确的soap请求即可，基于soap的服务可以在任何平台无需修改即可正常使用。  
（3）RPC就是从一台机器（客户端）上通过参数传递的方式调用另一台机器（服务器）上的一个函数或方法（可以统称为服务）并得到返回的结果。RPC 会隐藏底层的通讯细节（不需要直接处理Socket通讯或Http通讯）RPC 是一个请求响应模型。客户端发起请求，服务器返回响应（类似于Http的工作方式），RPC 在使用形式上像调用本地函数（或方法）一样去调用远程的函数（或方法）。



**rpc远程调用框架**

几种比较典型的RPC的实现和调用框架。   
（1）RMI实现，利用java.rmi包实现，基于Java远程方法协议(Java Remote Method Protocol) 和java的原生序列化。   
（2）Hessian，是一个轻量级的remoting onhttp工具，使用简单的方法提供了RMI的功能。 基于HTTP协议，采用二进制编解码。   
（3）thrift是一种可伸缩的跨语言服务的软件框架。thrift允许你定义一个描述文件，描述数据类型和服务接口。依据该文件，编译器方便地生成RPC客户端和服务器通信代码。

（4）SpringCloud 为开发人员提供了快速构建分布式系统的一些工具，包括配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等。

#### 什么是SOA

面向服务的架构（SOA）是一个组件模型，它将应用程序的不同功能单元（称为服务）进行拆分，并通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种各样的系统中的服务可以以一种统一和通用的方式进行交互。

作用：简化维护,降低整体风险,伸缩灵活

#### 什么是微服务

架构设计概念,各服务间隔离（分布式也是隔离）,自治（分布式依赖整体组合）其它特性(单一职责,边界,异步通信,独立部署)是分布式概念的更严格执行，SOA到微服务架构的演进过程

作用：各服务可独立应用，组合服务也可系统应用(巨石应用[monolith]的简化实现策略-平台思想)

 1.SOA（Service Oriented Architecture）“面向服务的架构”:他是一种设计方法，其中包含多个服务， 服务之间通过相互依赖最终提供一系列的功能。一个服务 通常以独立的形式存在与操作系统进程中。各个服务之间 通过网络调用。

 2.微服务架构:其实和 SOA 架构类似,微服务是在 SOA 上做的升华，微服务架构强调的一个重点是“业务需要彻底的组件化和服务化”，原有的单个业务系统会拆分为多个可以独立开发、设计、运行的小应用。这些小应用之间通过服务完成交互和集成。

**微服务架构 = 80%的SOA服务架构思想 + 100%的组件化架构思想 + 80%的领域建模思想**

#### 微服务框架

阿里的dubble、spring的springcloud,用于简便开发微服务项目的jar

#### 微服务项目结构

web-api 只有接口的项目，提供接口给其他系统调用

..api.service api接口的实现

..manage 内部具体的业务逻辑实现