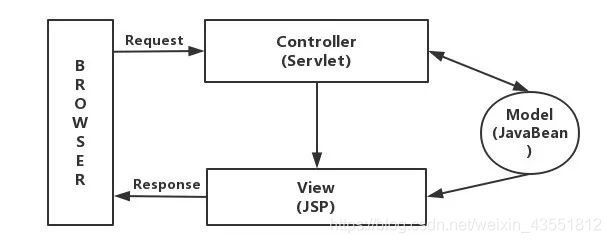
# 历史

浏览器：浏览网页，最开始都是为了分享，网页展示的都是纯静态的内容（文本、图片）。为了使web更具有活力，动态页面技术相继诞生。

## 前后端未分离时代



在前后端耦合架构中，所有的静态资源和业务代码统一部署在同一台服务器上。服务器接收到浏览器的请求后，进行处理得到数据，然后将数据填充到静态页面中，最终返回给浏览器。

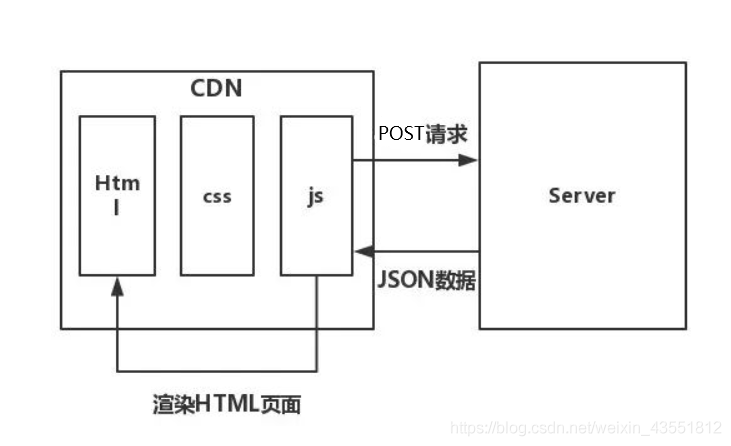
在这个前提下，前端能做的工作就只是简单地将 UI 设计师提供的原型图实现成静态的 HTML 页面，后端开发再“套用”前端开发的模板进行页面拼接。而后端过得也不轻松，因为页面交互逻辑开发的工作量都集中在后端，分配严重不均。并且开发时，前后端相互依赖很严重，需要改动时，要让前端修改静态页面，后端再修改逻辑，开发效率慢，代码难以维护。

开发技术有：JSP、Freemarker、Thymeleaf

这种方式已经逐渐淘汰。主要原因有两点：

1. 前端在开发过程中严重依赖后端，在后端没有完成的情况下，前端无法干活。
2. 会JSP、Velocity或Freemarker等模板引擎的前端开发人员越来越少。

## 前后端半分离



前后端半分离，前端负责开发页面，通过接口（Ajax）获取数据，采用Dom操作对页面进行数据绑定，最终是由前端把页面渲染出来。这也就是Ajax与SPA应用（单页应用）结合的方式。 步骤如下：

1. 浏览器请求，CDN返回HTML页面；
2. HTML中的JS代码以Ajax方式请求后台的Restful接口；
3. 接口返回Json数据，页面解析Json数据，通过Dom操作渲染页面；

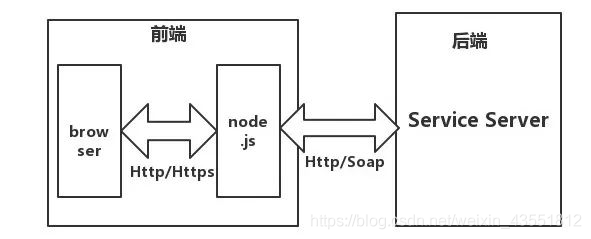
为什么说是半分离的？因为不是所有页面都是单页面应用，在多页面应用的情况下，前端因为没有掌握controller层，前端需要跟后端讨论，我们这个页面是要同步输出呢，还是异步Json渲染呢？而且，即使在这一时期，通常也是一个工程师搞定前后端所有工作。因此，在这一阶段，只能算半分离。

首先，这种方式的优点是很明显的。前端不会嵌入任何后台代码，前端专注于HTML、CSS、JS的开发，不依赖于后端。自己还能够模拟Json数据来渲染页面。发现Bug，也能迅速定位出是谁的问题。

然而，在这种架构下，还是存在明显的弊端的。最明显的有如下几点：

* JS存在大量冗余，在业务复杂的情况下，页面的渲染部分的代码，非常复杂；
* 在Json返回的数据量比较大的情况下，渲染的十分缓慢，会出现页面卡顿的情况；
* SEO（ Search Engine Optimization，即搜索引擎优化）非常不方便，由于搜索引擎的爬虫无法爬下JS异步渲染的数据，导致这样的页面，SEO会存在一定的问题；
* 资源消耗严重，在业务复杂的情况下，一个页面可能要发起多次HTTP请求才能将页面渲染完毕。（移动端建立HTTP请求非常消耗资源）

## 前后端分离



前后端分离已成为互联网项目开发的业界标准使用方式，通过nginx+tomcat的方式（也可以中间加一个nodejs）有效的进行解耦。核心思想是前端HTML页面通过AJAX调用后端的RESTFUL API接口并使用JSON数据进行交互。

* Web服务器：一般指像Nginx，Apache这类的服务器，他们一般只能解析静态资源；
* 应用服务器：一般指像Tomcat，Jetty，Resin这类的服务器可以解析动态资源也可以解析静态资源，但解析静态资源的能力没有web服务器好；
* 一般都是只有web服务器才能被外网访问，应用服务器只能内网访问。

在前后端彻底分离这一时期，前端的范围被扩展，controller层也被认为属于前端的一部分。在这一时期：

* 前端：负责View和Controller层。
* 后端：只负责Model层，业务/数据处理等。

# 前端

## URL

Uniform Resource Locator：统一资源定位符

语法规则：

比如网址 <http://segmentfault.com/html/>...，必须遵守以下的语法规则：

scheme://host.domain:port/path/filename

说明：

1. scheme - 定义因特网服务的类型。最常见的类型是 http
2. host - 定义域主机（http 的默认主机是 www）
3. domain - 定义因特网域名，比如 w3school.com.cn
4. :port - 定义主机上的端口号（http 的默认端口号是 80）
5. path - 定义服务器上的路径（如果省略，则文档必须位于网站的根目录中）。
6. filename - 定义文档/资源的名称

## HTML、CSS、JS

HTML是用来标记内容的（重在内容组织上）

HTML是超文本标记语言的简称，它是一种不严谨的、简单的标识性语言。它用各种标签将页面中的元素组织起来，告诉浏览器该如何显示其中的内容。

CSS是用来修饰内容样式的（重在内容样式美化展示上）

CSS是层叠样式表的简称，它用来表现HTML文件样式的，简单说就是负责HTML页面中元素的展现及排版。

JavaScript是用来做交互的

JavaScript是一种脚本语言，即可以运行在客户端也能运行在服务器端。JavaScript的解释器就是JS引擎，JS引擎是浏览器的一部分。而JavaScript主要是用来扩展文档交互能力的，使静态的HTML具有一定的交互行为（比如表单提交、动画特效、弹窗等）。

## es6

### 闭包

### 普通函数和箭头函数

### 防抖和节流

# 后端

## Session和Cookies

### cookie机制

在程序中，会话跟踪是很重要的事情。理论上，**一个用户的所有请求操作都应该属于同一个会话**，而另一个用户的所有请求操作则应该属于另一个会话，二者不能混淆。例如，用户A在超市购买的任何商品都应该放在A的购物车内，不论是用户A什么时间购买的，这都是属于同一个会话的，不能放入用户B或用户C的购物车内，这不属于同一个会话。

而Web应用程序是使用HTTP协议传输数据的。**HTTP协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上跟踪会话**。即用户A购买了一件商品放入购物车内，当再次购买商品时服务器已经无法判断该购买行为是属于用户A的会话还是用户B的会话了。要跟踪该会话，必须引入一种机制。

Cookie就是这样的一种机制。它可以弥补HTTP协议无状态的不足。在Session出现之前，基本上所有的网站都采用Cookie来跟踪会话。

### cookie

Cookie意为“甜饼”，是**由W3C组织提出**，最早由Netscape社区发展的一种机制。目前Cookie已经成为标准，所有的主流浏览器如IE、Netscape、Firefox、Opera等都支持Cookie。

由于HTTP是一种无状态的协议，服务器单从网络连接上无从知道客户身份。怎么办呢？就**给客户端们颁发一个通行证吧，每人一个，无论谁访问都必须携带自己通行证。这样服务器就能从通行证上确认客户身份了。这就是Cookie的工作原理**。

Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。

* Cookie的不可跨域名性

### session机制

除了使用Cookie，Web应用程序中还经常使用Session来记录客户端状态。**Session是服务器端使用的一种记录客户端状态的机制**，使用上比Cookie简单一些，相应的也**增加了服务器的存储压力**。

### session

Session是另一种记录客户状态的机制，不同的是Cookie保存在客户端浏览器中，而Session保存在服务器上。客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录在服务器上。这就是Session。客户端浏览器再次访问时只需要从该Session中查找该客户的状态就可以了。

如果说**Cookie机制是通过检查客户身上的“通行证”来确定客户身份的话，那么Session机制就是通过检查服务器上的“客户明细表”来确认客户身份。Session相当于程序在服务器上建立的一份客户档案，客户来访的时候只需要查询客户档案表就可以了。**

## tomcat

Tomcat 服务器是一个免费的开放源代码的Web 应用服务器，属于轻量级应用[服务器](https://baike.baidu.com/item/服务器)，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试JSP 程序的首选。对于一个初学者来说，可以这样认为，当在一台机器上配置好Apache 服务器，可利用它响应[HTML](https://baike.baidu.com/item/HTML)（[标准通用标记语言](https://baike.baidu.com/item/标准通用标记语言/6805073)下的一个应用）页面的访问请求。实际上Tomcat是Apache 服务器的扩展，但运行时它是独立运行的，所以当你运行tomcat 时，它实际上作为一个与Apache 独立的进程单独运行的。

诀窍是，当配置正确时，Apache 为HTML页面服务，而Tomcat 实际上运行JSP 页面和Servlet。另外，Tomcat和[IIS](https://baike.baidu.com/item/IIS)等Web服务器一样，具有处理HTML页面的功能，另外它还是一个Servlet和JSP容器，独立的Servlet容器是Tomcat的默认模式。不过，Tomcat处理静态[HTML](https://baike.baidu.com/item/HTML)的能力不如Apache服务器。Tomcat最新版本为10.0.14**。**

## servlet

Java Servlet 是运行在 Web 服务器或应用服务器上的程序，它是作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层。

JSP全称Java Server Pages，其本质也是一个servlet。

<servlet>
  
 <description>This is the description of my J2EE component</description>
  
 <display-name>This is the display name of my J2EE component</display-name>
  
 <servlet-name>BillServlet</servlet-name>
  
 <servlet-class>cn.smbms.servlet.bill.BillServlet</servlet-class>
  
</servlet>
  
  
 <servlet-mapping>
  
 <servlet-name>BillServlet</servlet-name>
  
 <url-pattern>/jsp/bill.do</url-pattern>
  
 </servlet-mapping>

## dispatcherServlet

在整个 Spring MVC 框架中，DispatcherServlet 处于核心位置，它负责协调和组织不同组件完成请求处理并返回响应工作。DispatcherServlet 是 SpringMVC统一的入口，所有的请求都通过它。DispatcherServlet 是前端控制器，配置在web.xml文件中，Servlet依自已定义的具体规则拦截匹配的请求，分发到目标Controller来处理。 初始化 DispatcherServlet时，该框架在web应用程序WEB-INF目录中寻找一个名为[servlet-名称]-servlet.xml的文件，并在那里定义相关的Beans，重写在全局中定义的任何Beans。在看DispatcherServlet 类之前，我们先来看一下请求处理的大致流程：

1. Tomcat 启动，对 DispatcherServlet 进行实例化，然后调用它的 init() 进行初始化，在这个初始化过程中完成了：对 web.xml 中初始化参数的加载；建立 WebApplicationContext(SpringMVC的IOC容器)；进行组件的初始化；
2. 客户端发出请求，由 Tomcat 接收到这个请求，如果匹配 DispatcherServlet 在 web.xml中配置的映射路径，Tomcat 就将请求转交给 DispatcherServlet 处理；
3. DispatcherServlet 从容器中取出所有 HandlerMapping 实例（每个实例对应一个 HandlerMapping接口的实现类）并遍历，每个 HandlerMapping 会根据请求信息，通过自己实现类中的方式去找到处理该请求的 Handler(执行程序，如 Controller 中的方法)，并且将这个 Handler 与一堆 HandlerInterceptor (拦截器)封装成一个 HandlerExecutionChain 对象，一旦有一个 HandlerMapping 可以找到 Handler则退出循环；
4. DispatcherServlet 取出 HandlerAdapter 组件，根据已经找到的 Handler，再从所有HandlerAdapter 中找到可以处理该 Handler 的 HandlerAdapter 对象；
5. 执行 HandlerExecutionChain 中所有拦截器的 preHandler()，然后再利用 HandlerAdapter 执行 Handler ，执行完成得到 ModelAndView，再依次调用拦截器的 postHandler()；
6. 利用 ViewResolver 将 ModelAndView 或是 Exception (可解析成 ModelAndView)解析成View，然后 View 会调用 render() 再根据 ModelAndView 中的数据渲染出页面；
7. 最后再依次调用拦截器的 afterCompletion()，这一次请求就结束了。

<!--dispatcherServlet-->
  
<servlet>
  
 <servlet-name>dispatcherServlet</servlet-name>
  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  
 <init-param>
  
 <param-name>contextConfigLocation</param-name>
  
 <param-value>classpath:applicationContext.xml</param-value>
  
 </init-param>
  
 <load-on-startup>1</load-on-startup>
  
</servlet>
  
<servlet-mapping>
  
 <servlet-name>dispatcherServlet</servlet-name>
  
 <url-pattern>/</url-pattern>
  
</servlet-mapping>

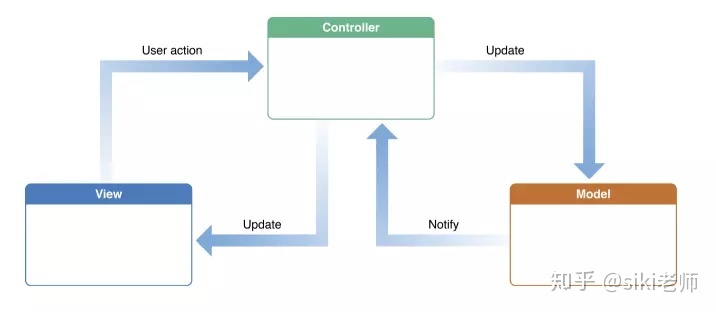
## SSH

Spring Structs Hibernate

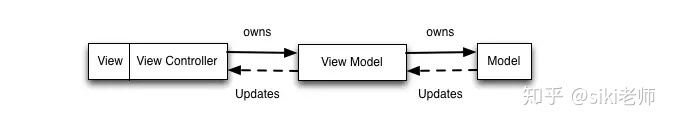
## SSM

Spring SpringMVC Mybatis

## MVC



## MVVM



# 前后端交互

## RESTful

REST 是 Representational State Transfer的缩写

### 什么是REST?

**REST**(英文：Representational State Transfer，简称**REST**，意思：表述性状态转换，描述了一个**架构**样式的网络系统，比如web应用)。

它是一种软件架构风格、设计风格，而不是标准，只是提供了一组设计原则和约束条件，它主要用于**客户端和服务端**交互类的软件。基于这个风格设计的软件可以更简介，更有层次，更易于实现缓存等机制。

它本身并没有什么使用性，其核心价值在于如何设计出符合REST风格的网络接口。

### 什么是RESTful？

**REST**：指的是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是**RESTful**。

### RESTful的特性

**资源(Resources)**：网络上的一个实体，或者说是网络上的一个具体信息。它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的存在。可以用一个URI（统一资源定位符）指向它，每种资源对应一个特性的URI。要获取这个资源，访问它的URI就可以，因此URI即为每一个资源的独一无二的识别符。

**表现层(Representation)**：把资源具体呈现出来的形式，叫做它的表现层(Representation)。比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式。

**状态转换(State Transfer)**：每发出一个请求，就代表了客户端和服务器的一次交互过程。HTTP协议，是一个无状态协议，即所有的状态都保存在服务器端。因此，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生“状态转换”(State Transfer)。而这种转换是建立在表现层之上的，所以就是“表现层状态转换”。具体说，就是HTTP协议里面，四个表示操作方式的动词：**GET、POST、PUT、DELETE**。他们分别对应四种基本操作：GET用来获取资源，POST用来新建资源，PUT用来更新资源，DELETE用来删除资源。

## 请求

请求头

请求体

## 响应

响应头

响应体

* [传统开发模式到前后端分离发展史](https://www.jianshu.com/p/5defa1675160)
* [Web前后端发展史整理](https://blog.csdn.net/weixin_43551812/article/details/104180093)
* [cookie和session的详解与区别](https://www.cnblogs.com/l199616j/p/11195667.html)
* [DispatcherServlet详解](https://www.jianshu.com/p/9b7883c6a1a0)
* [什么是MVVM框架](https://zhuanlan.zhihu.com/p/59467370)