### 概念模型：类似于Windows的GUI系统，浏览器就是Windows。

DOM对象

HTML+CSS

JavaScript代码

协议栈

JavaScript引擎

HTML+CSS引擎

事件驱动：对象是啥，有啥事件，事件的响应代码是啥。

1）Window系统的窗口，比如窗口的初始化，显示，刷新，关闭等。

2）组件编程中的组件，控件。比如控件的初始化，显示，双击，关闭等等。

3）HTML文件个的各种事件，比如加载，解析html头部，渲染等等。

4）DOM对象的各种事件，比如某个节点变化了。

传统思路：从入口点开始执行（main函数），执行后输出结果。更简单的是从第1行开始执行，一直到最后一行。

事件驱动方式：搭建一个服务点，提供多种服务，编制服务响应代码。启动服务后，等待用户的事件，根据不同事件，执行对应的代码。对于框架的使用者来说，别去关心框架是怎么启动的，重点是对于每个事件该怎么响应。

常见模式：生成框架的一个对象，设置事件及事件响应的代码。

数据变了，前端怎么响应。比如怎么感知，怎么获取，怎么表现，怎么更新等等。

事件驱动的MVC模式，升级为MVVM模式。是现在前端三大框架Angular、React、Vue 的发展主线。

JS代码实现了各种响应功能。

HTML文件：只是一个容器，里面包括HTML标签，CSS代码，JavaScript代码，模板引擎可识别的代码等等。

浏览器处理HTML文件时，根据HTML头部先启动前端框架，然后再加载HTMl文件后面的内容，比如分离出来JavaScript代码与前端框架绑定，加载执行过程则是事件发生过程，事件响应代码会被执行，可能是从后台取数据，可能是修改HTMl标签（DOM对象）。完成后形成HTMl标签，让HTML引擎执行，显示等。用户点击鼠标，进入事件响应过程。

问题：后台数据变化了，怎么更新给用户。

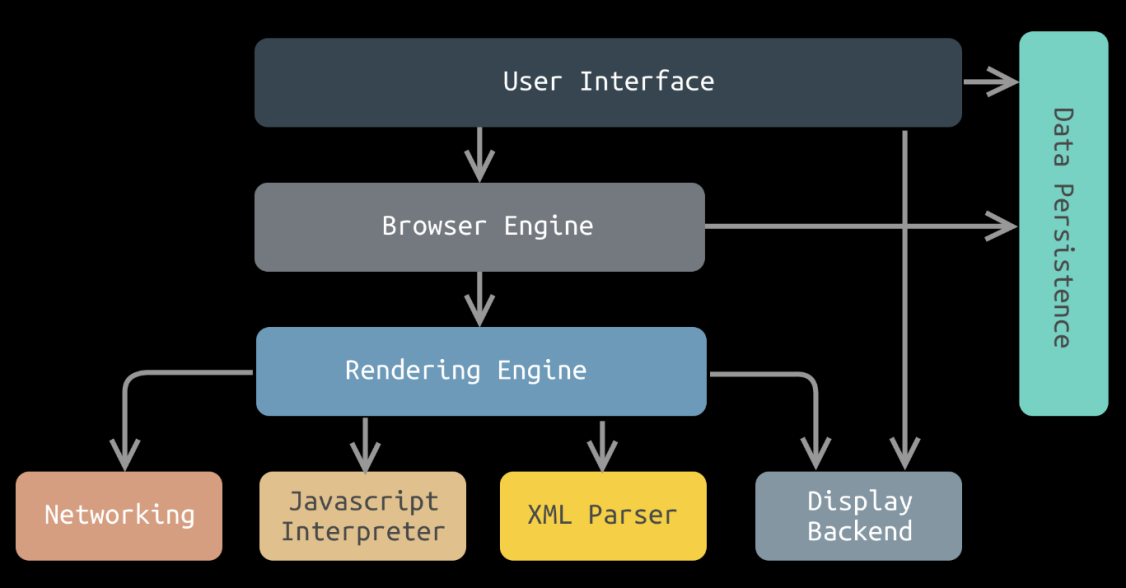
问题2：用户选择某个位置，更新了数据，怎么更新到后台服务器？

问题3：用户界面的事件影响另外一个对象的属性。

比如用户输入文本，有个控件统计输入了多少个字符，同时把数据保存到数据库中。

### 浏览器：类似于带GUI的操作系统，类比Windows。比JVM更强大。

浏览器由shell（用户界面、网络、UI后端、JavaScript解释器、XML解析器、数据存储）和内核（浏览器引擎、渲染引擎）组成，内核是浏览器的核心。不同的浏览器有不同的内核，ie 浏览器是 Trident，Firefox 浏览器是 Geoko（Mozilla自主研发的渲染引擎），Safari 和 Chrome 使用的是 webkit （后来 Chrome 推出了 Blink）。



浏览器是一个综合体：JavaScript引擎，HTML渲染引擎，CSS引擎，通信协议栈等等。

基本思路：通信协议栈拿到服务器端给的数据，先运行其中的JavaScript代码（React框架代码），生成html文件和css文件，然后由html及css引擎进行渲染，显示给客户。

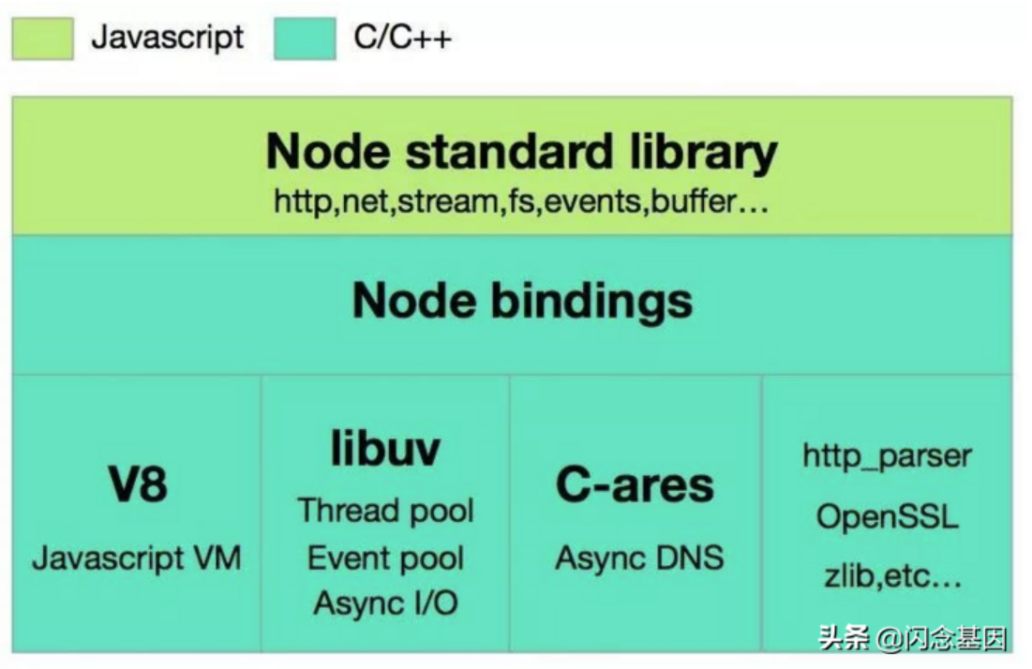
Windows操作系统将Window系统放到了内核，而Linux将Window系统作为系统一个组件。浏览器有些像Windows，Window系统在内核。

Windows APP和Windows 通过WindowsAPI交互（编译型语言，常称为运行时），而Web APP和浏览器通过JavaScript引擎和HTML引擎交互。二者是解释性语言，所以需要解释器。

事件驱动：当用户在浏览器中做了某个操作，比如选了某个下拉列表，与该事件关联的JavaScript代码就会被执行，可能是在浏览器的JavaScript引擎上算结果，可能是从服务器端取结果，然后返回给前端。

### NodeJS：JavaScript的VM，类似于Java的VM --- 类似于MS-DOS，Linux Kernel

Node.js是一个基于ChromeV8引擎（JavaScript引擎）的JavaScript运行环境，类似于JDK。在服务器端建立的一个平台。NodeJs提供了很多在服务器端的函数，也就是可以用JavaScript编写一个Apache httpd。



Node.js 标准库，这部分是由 Javascript编写的，即我们使用过程中直接能调用的 API。在源码中的 lib 目录下可以看到。Node bindings，这一层是 Javascript 与底层 C/C++ 能够沟通的关键，前者通过 bindings 调用后者，相互交换数据。第三层是支撑 Node.js 运行的关键，由 C/C++ 实现。V8：Google 推出的 Javascript VM，也是 Node.js 为什么使用的是 JavaScript 的关键，它为 JavaScript 提供了在非浏览器端运行的环境，它的高效是 Node.js 之所以高效的原因之一。Libuv：它为 Node.js 提供了跨平台，线程池，事件池，异步 I/O 等能力，是 Node.js 如此强大的关键。C-ares：提供了异步处理 DNS 相关的能力。http\_parser、OpenSSL、zlib 等：提供包括 http 解析、SSL、数据压缩等其他的能力。单线程：所有任务需要排队，前一个任务结束，才会执行后一个任务。如果前一个任务耗时很长，后一个任务就不得不一直等着。Node 单线程指的是 Node 在执行程序代码时， 主线程是单线程 。异步：主线程之外，还维护了一个"事件队列"（Event queue）。当用户的网络请求或者其它的异步操作到来时，Node 都会把它放到 Event Queue 之中，此时并不会立即执行它，代码也不会被阻塞，继续往下走，直到主线程代码执行完毕。

console.log("Hello World"); 类似于Java的System.out.println("Hello World")

NPM是随同NodeJS一起安装的包管理工具，能解决NodeJS代码部署上的很多问题

Webpackage：JavaScript的打包部署工具。这里的JS可能运行在浏览器也可能运行在NodeJS。

JavaScript 是单线程的，Node 本身其实是多线程的，只是 I/O 线程使用的 CPU 比较少；还有个重要的观点是，除了用户的代码无法并行执行外，所有的 I/O (磁盘 I/O 和网络 I/O) 则是可以并行起来的。libuv 线程池默认打开 4 个，最多打开 128 个 线程。Nodejs 所谓的单线程，只是主线程是单线程。

nodejs与vuejs的区别：1、nodejs是一个js运行于服务端的环境，是一个服务端语言；而vue是前端渲染的库，是一个前端框架。2、Node用于方便地搭建响应速度快、易于扩展的网络应用；而vue用于实现响应的数据绑定和组合的视图组件。

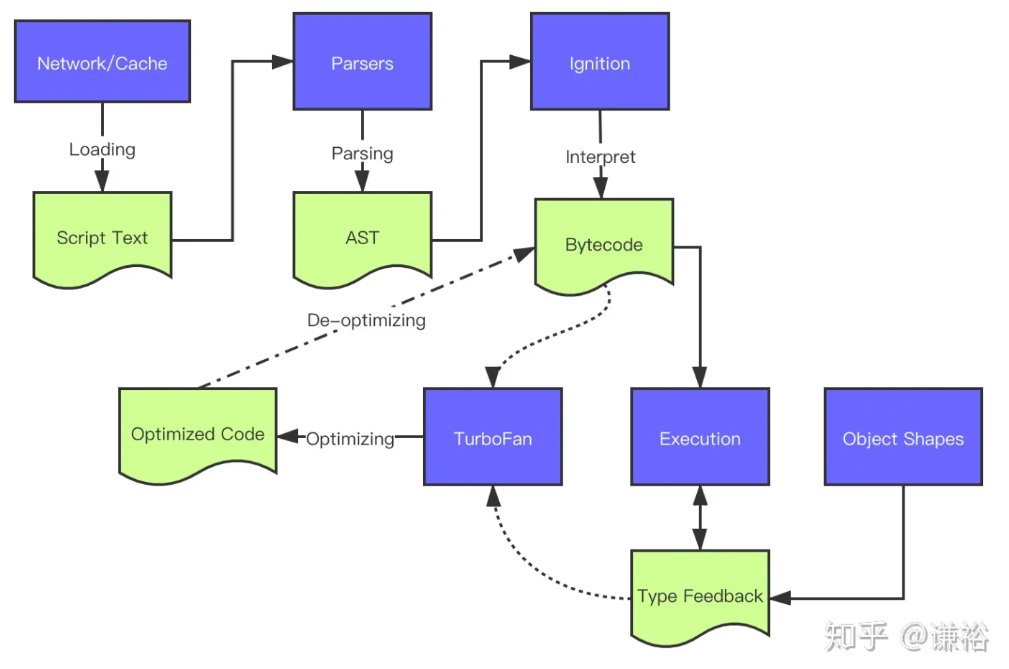
### JavaScript是商标名，可运行在浏览器，NodeJS中。JavaScript < ES6 <TypeScript

JavaScript相对于浏览器，类似于C语言相对于Linux操作系统。大名是ECMAScript。

通常说的JavaScript指ECMAScript 3.0（1999），面向过程，类似于C语言。

ES6， 全称 ECMAScript 6.0，增加了类，是基于对象的设计（不能继承），类似于C++，但是C++面向对象的混合语言。

JavaScript的解释器：浏览器的JS引擎，NodeJS等等。



JavaScript Promise：Promise 是一个 ECMAScript 6 提供的类，目的是更加优雅地书写复杂的异步任务。

TypeScript 是 JavaScript 的一个超集，面向对象。支持 ECMAScript 6 标准（ES6 教程）。TypeScript 由微软开发的自由和开源的编程语言。TypeScript 设计目标是开发大型应用，它可以编译成纯 JavaScript，编译出来的 JavaScript 可以运行在任何浏览器上。

### 输入输出（IO）：控制台，GUI，WebUI

控制台：起点是光标所在位置，内容是print语句的输出。

GUI：起点是窗口中默认的某个区域，内容可以是文本，图片等，需要调用不同函数。

WebUI：起点是某个地址（http://127.0.0.1），内容是html文件。

### Web服务器和应用服务器：

Web服务器：提供静态HTML的访问，本质是实现了HTTP协议，并支持高并发。比如Apache httpd，Nginx，Microsoft IIS。

类比FTP Server。

Web服务器+扩展模块：

案例：PHP扩展模块

案例：CGI，可以访问C，C++，等各种语言的进程。

应用服务器：往往和某种语言关联，尤其和后端Web框架关联。默认实现HTTP协议（可以当做Web服务器，只是并发较弱），动态解析包含这种语言的html文件。

案例：Tomcat是（java）应用服务器，它只是一个servlet容器，是Apache的扩展。 Apache和Tomcat都可以做为独立的web服务器来运行，但是Apache不能解释java程序（jsp,servlet）。

案例：基于Python的应用服务器：Flask（同步框架，使用uWSGI），FastAPI（异步框架，使用Uvicorn服务器）

案例：基于JavaScript/NodeJS的应用服务器：Express，支持异步接口

## 基于框架的编程；基于组件的编程：框架=毛坯房；组件=沙发，电视机

别人已经把基本布局设定好了，你进去后只需要装修一下（装饰一下）。你要大改就需要砸墙，分为承重墙和非承重墙。

不同的框架内涵差异很大。类似于农村的毛坯房和城市的毛坯房差别很大。开发商的精装房和老百姓理解的精装房差别也很大。

### 组件编程：WebUI组件编程与OS控件编程 ---类似于给家里买彩电，冰箱。

OS控件编程：典型案例Delphi VCL

1）每个组件有大量的属性，分为几大类。

2）每个组件有大量的行为，分为几大类。

3）可视化组件有很多相关属性。

JSON：类似Python的字典

{

"sites": [

{ "name":"菜鸟教程" , "url":"www.runoob.com" },

{ "name":"google" , "url":"www.google.com" },

{ "name":"微博" , "url":"www.weibo.com" }

]

}

tinydict = {'name': 'runoob', 'likes': 123, 'url': 'www.runoob.com'}

### 基于框架的编程：生成框架实例（买毛坯房），编写事件响应代码（自己装修），绑定对象的事件与响应代码

生成框架实例：开发商盖房子。使用毛坯房前需要先找物业拿钥匙，领门禁卡。

案例：对于React，需要执行npx create-react-app创建初始化目录。

事件驱动：小区有多个门，至于从那个门进入，看你的需求。针对不同事件，编写不同响应代码

很多默认：很多VM机或框架执行时先找特定对象（继承自某个类），然后执行特定函数。

1）对于Java来说，每个Java文件只能有一个public的类，如果执行该文件的，默认从public的main开始。

2）早期Web系统默认执行index.html文件。

3）nmp start 启动一个httpd，加载public目录下的manifest.json，解析找到index.html, index.js等。

需要辅助模块，类似于买了房子，住了房子，还需要物业服务。

MFC：封装Windows操作系统的API及Windows事件驱动的编程规范。组件数量很少，UI无法控制等。

Delphi的VCL：在MFC的基础上，还封装了各种UI，数据库连接等东西。

进入互联网时代后，浏览器是在OS之上的一层，可以把浏览器类比成OS，那么针对浏览器的组件有哪些呢？

Web开发更复杂：分为3部分，前端由浏览器决定，后端由各类业务决定，前后端的联系由Http等协议决定。

如果非要类比，对于操作系统来说，后端类似于操作系统内核功能，靠的是操作系统API来操作，前端则是操系统提供的各种基础组件。前后端的联系则是操作系统的GUI事件驱动模型。

J2EE体系：以后端为主，基于Java VM。使用Java语言。

Express框架：以后端为主，基于NodeJS。使用JavaScript语言。

VueJS框架：以前端为主，基于浏览器的JS引擎。使用JavaScript语言。

除此之外，还需考虑规模因素，比如基于MFC的程序，本质上是一个客户端（对应一个服务端，也就是OS内核）。如果要多个客户，那么就需要有多个安装实例。而基于Web的程序可以是一个客户端，也可以是成千上万客户端。需要根据具体问题来确定，不要是在单客户端应用中使用重量级框架，那是给自己找事。

前后端如何交互：同步方式，异步方式。

如何简单理解HTTP：

1）用户发送get命令，服务器返回一个html文本，浏览器解析html文件，显示给用户。服务器也可以返回一个jason文件，浏览器当做文本文件显示出来就行。

2）用户发送post命令，其中参数就是用户携带的数据。

POST：创建数据。

GET：读取数据。

PUT：更新数据。

DELETE：删除数据。

### 前端Web框架：UI对象和数据库怎么同步数据，UI对象之间如何交互（不需要后端介入）

早期前端框架实际上是功能模块，比如Ajax用于和后端数据交互，jQuery用于操作HTML DOM等。

1、前端框架：React、Vue、AngularJS，在浏览器JavaScript引擎和html页面间的一层，用于数据控制（更新）。

2、UI框架：Bootstrap、EasyUI，重点在布局和排版方面

3、组件库：AntDesign、Element是针对于React、Vue、AngularJS等开发的组件库。减少自己写控件的工作量。

核心：

功能1：2个WebUI之间的数据如何交互。如何绑定HTML元素

功能2：多个WebUI之间的数据如何交互。需要使用页面路由。

功能3：UI对象和后端怎么同步数据。事件驱动。怎么感知数据变化，异步传输/同步传输，CURE（获取，推送，更改等等）

功能4：多个WebUI的布局和视觉效果。

### 后端Web框架：访问某个地址（url），应该调用那个html文件，默认带应用服务器

不同语言往往有不同的框架，基于JavaScript的Express（运行在NodeJS上）。基于Java的Spring（运行在JVM上）。基于Python的Flask（同步框架）和FastAPI（异步框架）等。

核心：

功能1：Web应用服务器与Web应用怎么连接。比如CGI，RPC

功能2：怎么对业务逻辑分层。事件怎么响应（地址和响应代码的关联）。

功能3：怎么连接数据库，存取数据。

功能4：Web应用与前端怎么进行同步数据。比如使用Json，OpenAPI标准

### Bootstrap：WebUI控件，避免自己编写CSS。有没有类似于Delphi的拖拉式IDE？

不同的Bootstrap版本，类似于控件库的版本，控件类型更多，控件界面更炫，控件更多行为支持，对浏览器要求更高。

控件类型更多：比如椭圆形按钮。

控件界面更炫：方形按钮带3D阴影效果等。

控件更多鼠标行为：点击按钮后，按钮抖动。

Bootstrap5 和 Bootstrap 3 & 4 的主要区别在于 Bootstrap5 不再依赖 jQuery，使用了原生的 JavaScript，当然我们如果要想用也可以引入 jQuery。支持所有主要的最新稳定版本浏览器，但不支持 Internet Explorer 11 及以下版本。

基本结构：Bootstrap 提供了一个带有网格系统、链接样式、背景的基本结构。这将在 Bootstrap 基本结构 部分详细讲解。

CSS：Bootstrap 自带以下特性：全局的 CSS 设置、定义基本的 HTML 元素样式、可扩展的 class，以及一个先进的网格系统。这将在 Bootstrap CSS 部分详细讲解。

组件：Bootstrap 包含了十几个可重用的组件，用于创建图像、下拉菜单、导航、警告框、弹出框等等。这将在 布局组件 部分详细讲解。

JavaScript 插件：Bootstrap 包含了十几个自定义的 jQuery 插件。您可以直接包含所有的插件，也可以逐个包含这些插件。这将在 Bootstrap 插件 部分详细讲解。

定制：您可以定制 Bootstrap 的组件、LESS 变量和 jQuery 插件来得到您自己的版本。

### AngularJS：在浏览器内完成WebUI控件之间的数据交互

AngularJS 通过新的属性和表达式扩展了 HTML。可以构建一个单一页面应用程序（SPAs：Single Page Applications）。

1、ng-app=" " 定义angularJS的使用范围；

2、ng-init="变量=值;变量='值'" 初始化变量的值，有多个变量时，中间用分号隔开；

3、ng-model="变量" 定义变量名；

4、ng-bind="变量" 绑定变量名，获取该变量的数据。这里的变量就是第3条的变量名。但是一般都用双重花括号来获取变量的值，比如：{{变量}}。

第1类：2个WebUI控件之间的数据交互

<div ng-app=""> 表示AngularJS接管这个对象

<p>名字 : <input type="text" ng-model="name"></p> input的输入为name

<h1>Hello {{name}}</h1> 引用name的值。

<p ng-bind="name"></p> p的内容为name的值。

</div>

AngularJS路由：多个WebUI之间数据交互

第2类：AngularJS与后端数据交互。AngularJS XMLHttpRequest

通过$http对象的get方法从后端获取数据。后端可能是PHP，J2EE等。

### Angular2

Angular2.x与Angular1.x 的区别

Angular2.x与Angular1.x 的区别类似 Java 和 JavaScript 或者说是雷锋与雷峰塔的区别，所以在学习Angular2.x时大家需要做好重新学习一门语言的心里准备。

### Vue：起步低，不需要学习新范式

Vue.js（读音 /vjuː/, 类似于 view） 是一套构建用户界面的渐进式框架。Vue 只关注视图层， 采用自底向上增量开发的设计。Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

刚开始学习 Vue，我们不推荐使用 vue-cli 命令行工具来创建项目，更简单的方式是直接在页面引入 vue.global.js 文件来测试学习。

案例：

<body>

<div id="hello-vue" class="demo"> 定义一个vue接管的区域

{{ message }} 启动事件

</div>

<script>

const HelloVueApp = { 定义响应函数

data() {

return {

message: 'Hello Vue!!' 事件的响应代码

}

}

}

Vue.createApp(HelloVueApp).mount('#hello-vue') 定义vue app对象，并绑定到vue区域。

</script>

</body>

案例思路：

1）先定义一个div区域。设置好事件。

2）定义一个事件响应函数。

3）基于响应函数创建一个vue对象，并将该对象与区域绑定。

Vue路由：页面内路由，多个WebUI之间数据交互

<body>

<div id="app">

<h1>Hello App!</h1>

<p>

<!--使用 router-link 组件进行导航 -->

<!--通过传递 `to` 来指定链接 -->

<!--`<router-link>` 将呈现一个带有正确 `href` 属性的 `<a>` 标签-->

<router-link to="/">Go to Home</router-link> 点击超链接文本，触发/事件。

<br>

<router-link to="/about">Go to About</router-link>

</p>

<!-- 路由出口 -->

<!-- 路由匹配到的组件将渲染在这里 -->

<router-view></router-view>

</div>

<script>

// 1. 定义路由组件.

// 也可以从其他文件导入

const Home = { template: '<div>Home</div>' }

const About = { template: '<div>About</div>' }

// 2. 定义一些路由

// 每个路由都需要映射到一个组件。

// 我们后面再讨论嵌套路由。

const routes = [

{ path: '/', component: Home },

{ path: '/about', component: About },

]

// 3. 创建路由实例并传递 `routes` 配置

// 你可以在这里输入更多的配置，但我们在这里

// 暂时保持简单

const router = VueRouter.createRouter({

// 4. 内部提供了 history 模式的实现。为了简单起见，我们在这里使用 hash 模式。

history: VueRouter.createWebHashHistory(),

routes, // `routes: routes` 的缩写

})

// 5. 创建并挂载根实例

const app = Vue.createApp({})

//确保 \_use\_ 路由实例使

//整个应用支持路由。

app.use(router)

app.mount('#app')

// 现在，应用已经启动了！

</script>

</body>

疑问：Vue为什么需要NodeJS。

vue.js 是通过 webpack来打包，而webpack 又基于 npm, npm需要nodejs环境。这就是为什么vue.js 还需要安装nodejs环境。

也就是说本质上Vue不是必须要NodeJS的。仅仅是因为部署需要NodeJS而已。

第2类：Vue与后端数据交互。

1）同步方式：Vue.js Ajax(axios)。Axios 是一个基于 Promise 的 HTTP 库，可以用在浏览器和 node.js 中。

2）异步方式：Vue.js Ajax(vue-resource)

随着前端开发的网页元素不断丰富和复杂化，催生出webpack 来进一步规划js应用的打包部署。前端目标页面资源，通过webpack来打包压缩出来。可以看出vue.js 就是遵循的webpack 的方式来部署的，我们使用npm run build之后，会生成一个目标dist文件。这即是目标静态web资源，放在nginx下面即可通过网页访问。

将目标dist文件夹拷贝到一台未安装nodejs的 nginx服务器上，访问页面可以正常响应逻辑。这时跟nodejs没有任何关系，服务器又不是nodejs在担当，而是nginx。如果你用nodejs来部署服务器，则需要在目标机上安装nodejs.

### React：新范式

案例：代码如何与HTML元素绑定

<body>

<div id="example"></div> 先定义一个div区域。

<script type="text/babel">

ReactDOM.render(

<h1>Hello, world!</h1>, 定义一个事件响应函数。

document.getElementById('example') 将响应函数与区域绑定。

);

</script>

</body>

案例思路：

1）先定义一个div区域。设置好事件。

2）定义一个事件响应函数。

3）基于响应函数创建一个vue对象，并将该对象与区域绑定。

React AJAX

React和React Native的原理是相同的，都是由js实现的虚拟dom来驱动界面view层渲染。只不过React是驱动html dom渲染； React Native是驱动android/ios原生组件渲染。其实在React Native推出之前，就已经存在这种使用js驱动app原生组件的技术了，比如Native Script。

### Python Web后端框架（简称，Web框架）：Django，Flask，Pyramid，Tornado，Bottle，Streamlit，Diesel，Pecan，Falcon

Flask是一个微框架，主要面向需求简单的小应用。Django和Pyramid都是面向大的应用，比如内嵌重量级组件连接数据库。Flask 中有两大核心：Jinja2 和Werkzeug。

Pyramid 是一个小型，快速，接地气的 Python web framework. 它是 Pylons Project 的一部分。采用的授权协议是 BSD-like license.

FastAPI 是一个用于构建 API 的现代、快速（高性能）的 web 框架，使用 Python 3.6+ 并基于标准的 Python 类型提示。支持OpenAPI规范，可以使用Swagger管理和调试接口。

OpenAPI 和 JSON Schema（类似于XML Schema）

OpenAPI 为你的 API 定义 API 模式。该模式中包含了你的 API 发送和接收的数据的定义（或称为「模式」），这些定义通过 JSON 数据模式标准 JSON Schema 所生成。

### 基于Python的Web应用开发：Web应用与Python算法集成

总是需要Web服务器与Python进程之间通信，WebUI和Python算法怎么集成？

wei注：对于个人或者针对个人的小程序来说第1种方法更好。

第1类：传统的方法是进程间通信/调用，比如使用标准控制台，经典的RPC等。

1）经验：Java代码调用MATLAB算法。

2）经验：Java代码调用R算法。

案例：Express/NodeJS 调用 Python算法。进程间通信。

第2类：CGI的扩展，总是需要一定标准才好。比如Apache httpd和Php代码之间。后来的Tomcat和Java EE之间。

案例：JSP，在后端代码里面嵌入变量，JSP引擎对这些变量赋值，生成HTML，发给浏览器。当嵌入的数据发生变化时，需要重新生成HTML，也就是浏览器需要重新加载。这种方式称为同步方式。

后来AJXA出现了，不再更新整个页面，在浏览器从服务器端取数据就行。这种方式称为异步方式。

接着浏览器的JavaScript引擎变得很强大，出现JQuery负责复杂JavaScript功能，Bootstrap负责复杂的UI。前端开始独立与后端发展，JSP技术退出历史。

对于Python而言，

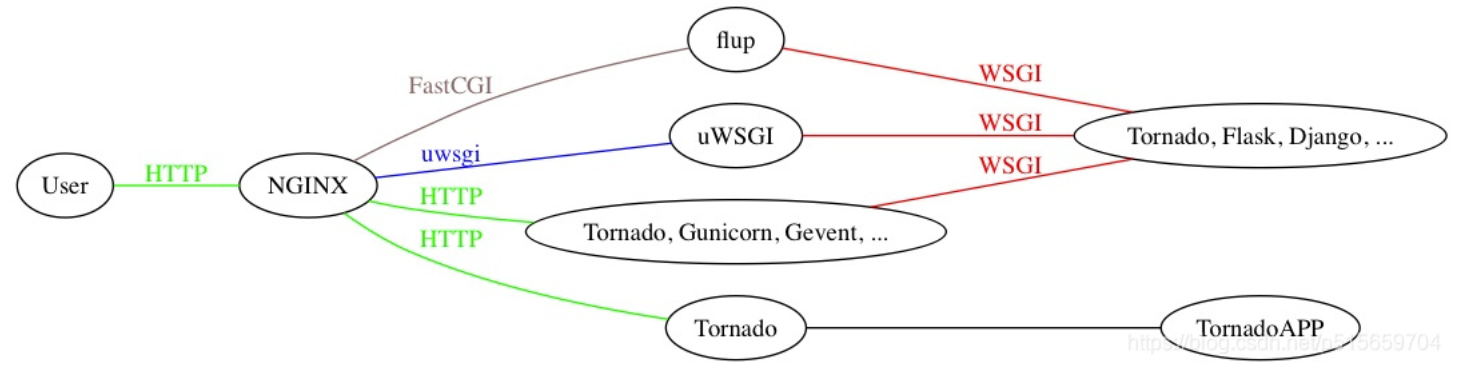
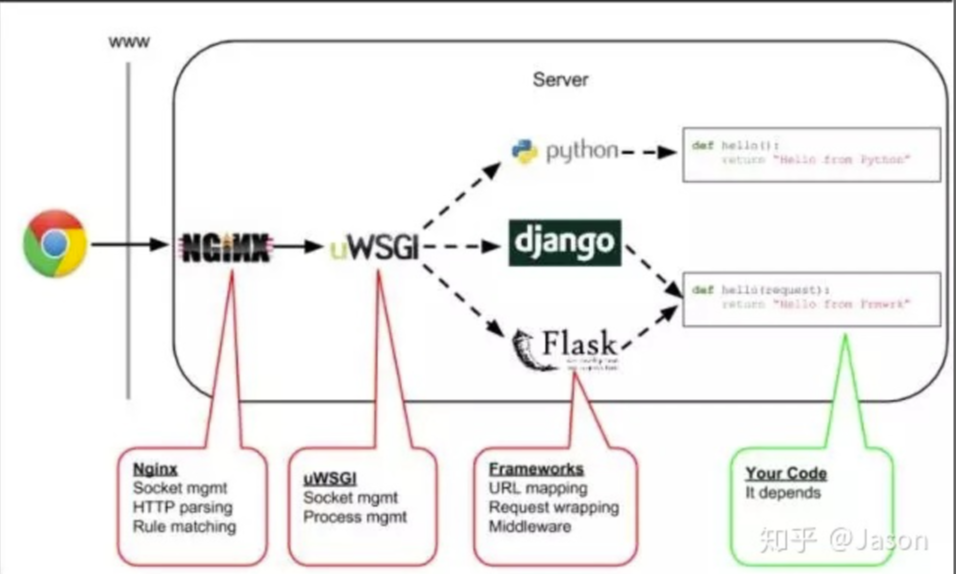
1）Web 服务器网关接口（WSGI，Web Server Gateway Interface）。它是一种规范，它是web服务器和web应用程序之间的接口。它的作用就像是桥梁，连接在web服务器和web应用框架之间。WSGI 应用是一个单调用、同步接口，即输入一个请求，返回一个响应；这个模式无法支持长连接或者 WebSocket 这样的连接。

uwsgi：是一种传输协议，用于定义传输信息的类型。

Werkzeug是一个WSGI工具包，他可以作为一个Web框架的底层库。这里稍微说一下， werkzeug 不是一个web服务器，也不是一个web框架，而是一个工具包，官方的介绍说是一个 WSGI 工具包，它可以作为一个 Web 框架的底层库，因为它封装好了很多 Web 框架的东西，例如 Request，Response 等等。Falsk默认使用这个工具包。性能较差。

uWSGI：是实现了uwsgi协议WSGI的web服务器。

Gunicorn（绿色独角兽）是一个Python WSGI的HTTP服务器。从Ruby的独角兽（Unicorn ）项目移植。该Gunicorn服务器与各种Web框架兼容，实现非常简单，轻量级的资源消耗。Gunicorn直接用命令启动，不需要编写配置文件，相对uWSGI要容易很多。



案例：Flash/Python和Python算法集成。

2）ASGI 协议规范的出现填补了这一空白，这意味着现在能够开始构建可在所有异步框架中使用的通用工具集。ASGI 帮助 Python 在 Web 框架上和 Node.JS 及 Golang 相竟争，目标是获得高性能的 IO 密集型任务，ASGI 支持 HTTP2 和 WebSockets，WSGI 是不支持的。

asgi服务器包括：Uvicorn 、Daphne 、Hypercorn。

Uvicorn 目前支持 HTTP1.1 和 WebSocket，计划支持 HTTP2。----类似于Java的Tomcat。

FastAPI是异步Web框架。

案例：FastAPI/Python和Python算法集成。

第3类：前后端分离。满足OpenAPI规范的RestfulAPI。数据驱动方式。

案例：Vue.js前端，Java SpringBoot提供WebAPI，连接Python算法。

思路4：Express/NodeJS 调用Flask的API，通过url jason传送数据。

https://github.com/matzewolf/NodeJS-Python-Example

①/响应用户，用户输入数据。

②client.js对/api发起Post请求

③/api函数获得用户数据，对/py\_api发送请求。

④/py\_api响应，获得用户数据，在Flash内。

⑤用Python处理数据，返回到/api

⑥/api函数获得Python处理结果，返回到客户端。

疑问：client.js为什么不直接对/py\_api发送post呢？

### Jinjia2：模板的预处理器 --- 类似于C语言的预处理器，字符串替换，编译前计算

处理嵌在HTML文件中的预处理字符串。 类似于C文件中的#define语句。

提供字符串操作函数。比如替换，拼接等。简单的数值计算。

三大语法块

{# #}----- 注释块

{% for %}或者 {%if %} -----循环

{{ }} -------变量取值，函数计算

### ejs：模板的预处理器 --- 类似于C语言的预处理器，字符串替换，编译前计算

处理嵌在HTML文件中的预处理字符串。 类似于C文件中的#define语句。

比如从JSON数据中生成HTML字符串。

<%= 输出数据到模板

<% 脚本标签，用于流程控制，无输出

<%\_ 删除其前面的空格符

<%- 输出非转义的数据到模板

<%# 注释标签，不执行，不输出内容

<%% 输出字符串

%> 一般结束标签

\_%> 将结束标签后面的空格符删除

ejs是JavaScript的一个类库，可以在服务器端运行，也可以在浏览器中运行（消耗浏览器资源）。

### Pug(Jade)，Mustache，Dust，Vash，Twig，Hogan，Handlabs：模板的预处理器

## CSS：从HTML标记中分离出WebUI

对HTML对象的视觉效果进行说明。一种语言，和HTML混在一起，被HTML+CSS引擎解释执行。

CSS的视觉效果附着在HTML对象或者HTML对象的行为上。也就是为节点和节点行为提供视觉属性。

CSS出现前，HTML的标记既描述数据的结构，也描述数据的视觉效果。

CSS出现后，HTML的标记只是用来描述数据的结构。当没有CSS时，每个标记有默认的视觉效果。

数据的结构：数据的类型和数据之间的关系，比如是文本，图像，超链接等；比如层次关系，表格关系，图关系等。

视觉效果：随着不同浏览器不同而不同，甚至随着同一浏览器版本的不同而不同。

HTML对象的选择：第1种直接定义HTML原有标签的属性（使用通配符）。第2种是定义div标签，然后用div选择对象。

视觉效果的说明：通过对象的属性来说明视觉效果。不同浏览器及版本支持的属性不同。类似于Jason文件。

形式化：选择器{属性：属性值；属性2：属性值；}

第1种案例：

body{  
font-size:75%;  
font-family:verdana,arial,'sans serif';  
background-color:#FFFFF0;  
color:#000080;  
margin:10px;  
}

### 选择器：id 和 class 选择器

id选择器：每个对象只能有一个id。每个html文件，这个对象用了，别的对象就不能用。

class 选择器：可以为对象同时设置多个属性。.表示任何对象。前面可以设置成具体的对象。

.color { color:#ff0000; } 设置color属性。

.center { text-align:center; } 设置文本对齐属性。

后代选择器(以空格 分隔)

子元素选择器(以大于 > 号分隔）

相邻兄弟选择器（以加号 + 分隔）

普通兄弟选择器（以波浪号 ～ 分隔）

div p

{

background-color:yellow;

}

body, h2, p, table, th, td, pre, strong, em {color:gray;}

\* {color:red;}

CSS 多类选择器

案例：将一个特定的元素同时标记为重要（important）和警告（warning）

<p class="important warning">

This paragraph is a very important warning.

</p>

important 的所有元素都是粗体，而 class 为 warning 的所有元素为斜体，class 中同时包含 important 和 warning 的所有元素还有一个银色的背景 。

.important {font-weight:bold;}

.warning {font-style:italic;}

.important.warning {background:silver;}

### 优先级

可能存在多个地方设置某个对象的属性，当有冲突时，优先顺序如下：

（内联样式）Inline style > （内部样式）Internal style sheet >（外部样式）External style sheet > 浏览器默认样式

内联样式 > id 选择器 > 类选择器 = 伪类选择器 = 属性选择器 > 标签选择器 = 伪元素选择器

层叠：

HTML: <div class="content-class" id="content-id"></div>

CSS：

#content-id {

color: red;

}

.content-class {

color: blue;

}

div {

color: grey;

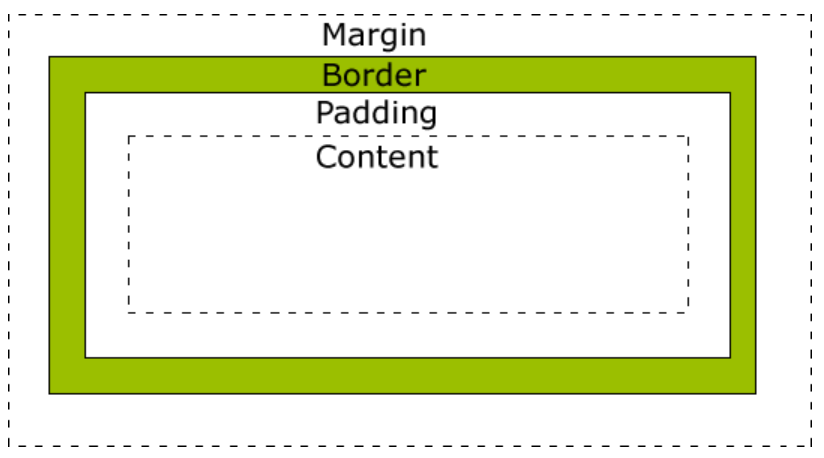
}

### CSS 盒子模型(Box Model)

所有HTML元素可以看作盒子，在CSS中，"box model"这一术语是用来设计和布局时使用。

CSS盒模型本质上是一个盒子，封装周围的HTML元素，它包括：边距，边框，填充，和实际内容。

盒模型允许我们在其它元素和周围元素边框之间的空间放置元素。



CSS定义行为：比如下拉菜单

### CSS3：增加更多的UI效果，比如动画，分页等。

### LESS：CSS预处理器 --- 类似于C语言的预处理器，字符串替换，编译前计算

案例：变量，将重复的字符串抽象成变量。

@fontSize: 10px;

.myclass {

font-size: @fontSize \* 2;

color:green;

}

案例：函数，将重复的操作抽象成函数。

@color: #FF8000;

@width:1.0;

.mycolor{

color: @color;

width: percentage(@width);

}

LESS预处理器本质上一个JavaScript程序，可以在服务器端运行，也可以在浏览器中运行（消耗浏览器资源）。

服务器端：lessc style.less style.css 调用NodeJS执行less.js将less文件转换为css文件。

浏览器端：

<script src="less.js"></script> 让浏览器下载less

<link href="styles.less" rel="stylesheet/less" type="text/css"/> 用less预处理器将less变成css

### 其他预处理器：Compass，SASS，Stylus

## 浏览器端-动态页面技术：过时的技术

由于浏览器只内置了JavaScript解释器和CSS渲染器，只能从这些方面下手。

### AJAX：通过预定义对象，从后端取数据存数据

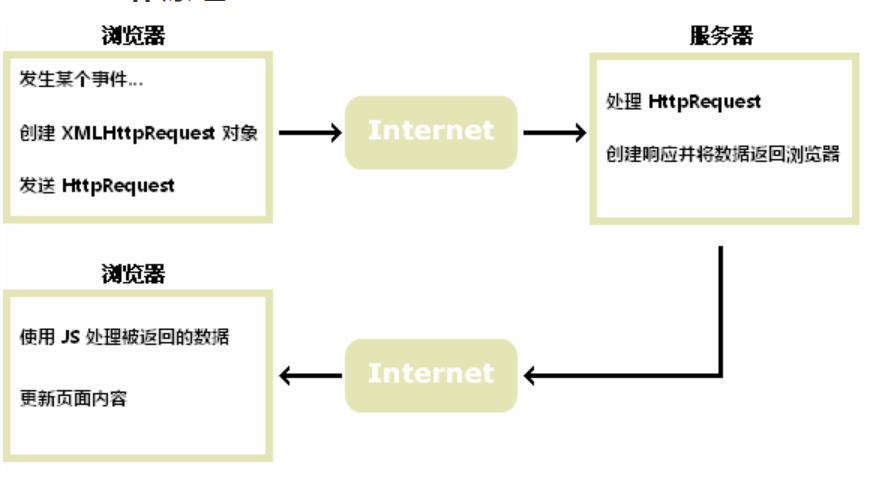
运用 XHTML+CSS 来表达资讯；

运用 JavaScript 操作 DOM（Document Object Model）来执行动态效果；

运用 XML 和 XSLT 操作资料;

运用 XMLHttpRequest 或新的 Fetch API 与网页服务器进行异步资料交换；

注意：AJAX 与 Flash、Silverlight 和 Java Applet 等 RIA 技术是有区分的。



案例：

<!DOCTYPE html>

<html>

<body>

<div id="demo">

<h1>XMLHttpRequest 对象</h1>

<button type="button" onclick="loadDoc()">修改内容</button>

</div>

<script>

function loadDoc() {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() { /\*定义响应函数\*/

if (this.readyState == 4 && this.status == 200) { /\*得到内容\*/

document.getElementById("demo").innerHTML = /\*定位到div

this.responseText;

}

};

xhttp.open("GET", "/demo/js/ajax\_info.txt", true); /\* 发送请求 \*/

xhttp.send();

}

</script>

</body>

</html>

### jQuery：在浏览器中操作HTML DOM

jQuery 是一个 JavaScript 库。jQuery 极大地简化了 JavaScript 编程。

JavaScript本身也将HTML抽象成DOM，提供document对象，只是支持的方法较少而已，类似于C++本身提供String类，但是MFC提供更强大的CString一样。

案例：$表示引用对象，read表示事件，function表示响应函数。

将HTML文件抽象成DOM，$引用的对象包括document，各种标签，比如p等等。

<html><head>

<script type="text/javascript" src="jquery.js"></script>

<script type="text/javascript">

$(document).ready(function(){

$("p").click(function(){

$(this).hide();

});

});

</script></head>

<body><p>If you click on me, I will disappear.</p></body>

</html>

$表示jQuery名称空间。由于JavaScript早期没有名称空间，调用jQuery全局对象，需要写成IQuery.xxx，所以简写为$.xxx。

$()是一个function，返回jQuery自定义的对象。$(document)返回一个jQuery的document对象。

### Restful API：一种设计API的风格，不是标准。 ---类比 COM 风格

2000年，Roy Thomas Fielding博士在他的博士论文《Architectural Styles and the Design of Network-based Software Architectures》中提出了几种软件应用的架构风格，REST作为其中的一种架构风格在这篇论文中进行了概括性的介绍。

REST:Representational State Transfer的缩写，翻译：“具象状态传输”。一般解释为“表现层状态转换”。

REST是设计风格而不是标准。是指客户端和服务器的交互形式。我们需要关注的重点是如何设计REST风格的网络接口。

REST的特点：

具象的。一般指表现层，要表现的对象就是资源。比如，客户端访问服务器，获取的数据就是资源。比如文字、图片、音视频等。

表现：资源的表现形式。txt格式、html格式、json格式、jpg格式等。浏览器通过URL确定资源的位置，但是需要在HTTP请求头中，用Accept和Content-Type字段指定，这两个字段是对资源表现的描述。

状态转换：客户端和服务器交互的过程。在这个过程中，一定会有数据和状态的转化，这种转化叫做状态转换。其中，GET表示获取资源，POST表示新建资源，PUT表示更新资源，DELETE表示删除资源。HTTP协议中最常用的就是这四种操作方式。

RESTful架构：

每个URL代表一种资源；

客户端和服务器之间，传递这种资源的某种表现层；

客户端通过四个http动词，对服务器资源进行操作，实现表现层状态转换。

## 服务器端-动态页面技术：PHP，ASP，JSP ---过时的技术

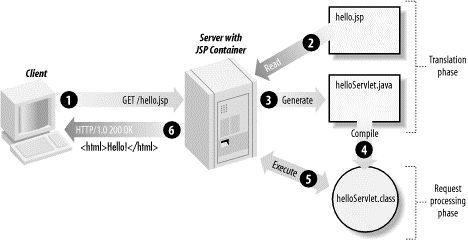
服务器可以运行各种语言，也就有各种方案：语言+框架

### JSP：

<body>

<% out.println("Hello World！"); %>

</body>



JSP工作原理

①一个JSP文件第一次被请求时，JSP引擎把该JSP文件转换成为一个servlet。只是简单地将所有模板文本改用 println() 语句，并且将所有的 JSP 元素转化成 Java 代码。

②JSP引擎使用javac把转换成的servlet的源文件编译成相应的class文件。

③对每一个请求，JSP引擎创建一个新的线程来处理请求。

JSP和Servlet本质

JSP是嵌套Java代码的HTML代码

JSP转换成对应的Servlet类运行

Servlet是嵌套HTML代码的Java代码

### PHP：HTML预处理器 ---功能更强的Jinjia2？

强在哪里？

①完备的语言，类似于Python，提供各种循环及基本数据类型（整形，浮点，多维数组等）。

②提供大量类库，比如数据库操作的库等，类似于Java类库。

③编译型的吗？

④高性能环境，比如PHP应用服务器等。

案例：

<body>

<?php echo "Hello World!"; ?>

</body>