

Web和前端技术的发展

陈俊达

2019年1月5日

目录

- Web应用架构的变化
- 前端技术的变化
- 现在的前端技术
- 未来值得关注的前端技术

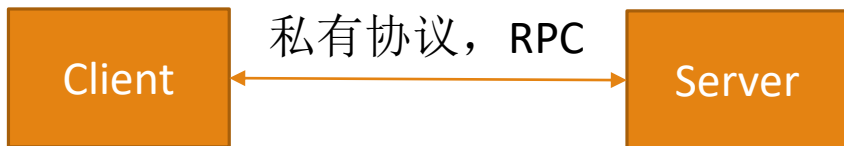
Web技术架构的变化

Web应用架构的变化

- -1998（混沌期）
 - CS架构，私有协议，RPC，CGI，浏览器，HTTP
- 1999-2005（企业级开发）
 - J2EE
 - MVC
- 2005-（快速发展期）
 - Ruby on Rails，Python Django/Flask
 - Node.js
 - 前后端分离

CS模式统治的时期

- Client-Server模式（客户端、服务器）
- 私有协议，RPC
- SOAP (XML-RPC)，RPC（RMI）
- 软工1，软工2



请求 [\[编辑 \]](#)

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <req:echo xmlns:req="http://localhost:8080/wxyz/login.do">
      <req:category>classifieds</req:category>
    </req:echo>
  </soapenv:Body>
</soapenv:Envelope>
```

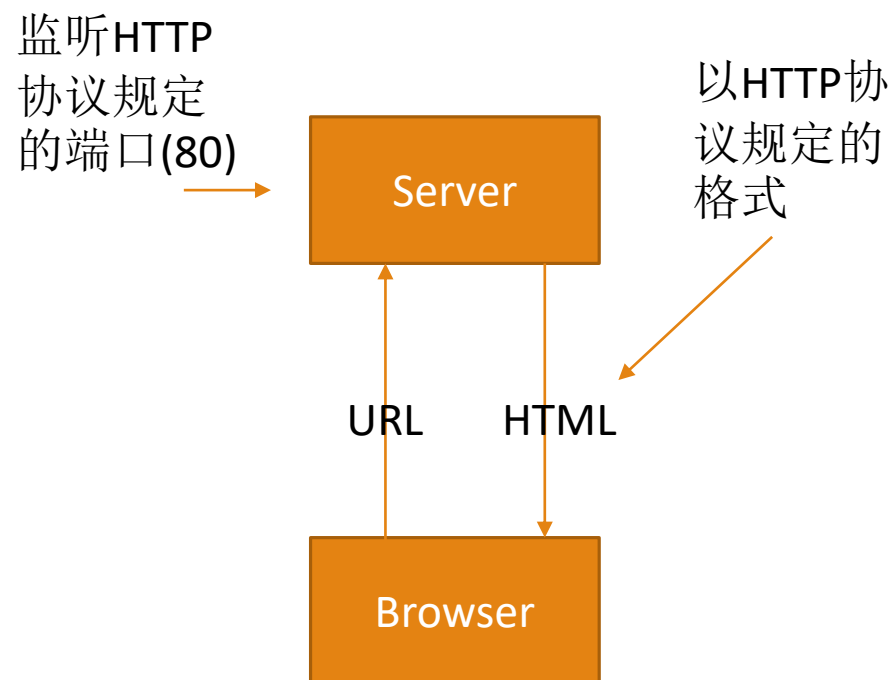
SOAP传输
格式

回应 [\[编辑 \]](#)

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <soapenv:Header>
    <wsa:ReplyTo>
      <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:From>
      <wsa:Address>http://localhost:8080/axis2/services/MyService</wsa:Address>
    </wsa:From>
    <wsa:MessageID>ECE5B3F187F29D28BC11433905662036</wsa:MessageID>
  </soapenv:Header>
  <soapenv:Body>
    <req:echo xmlns:req="http://localhost:8080/axis2/services/MyService/">
      <req:category>classifieds</req:category>
    </req:echo>
  </soapenv:Body>
</soapenv:Envelope>
```

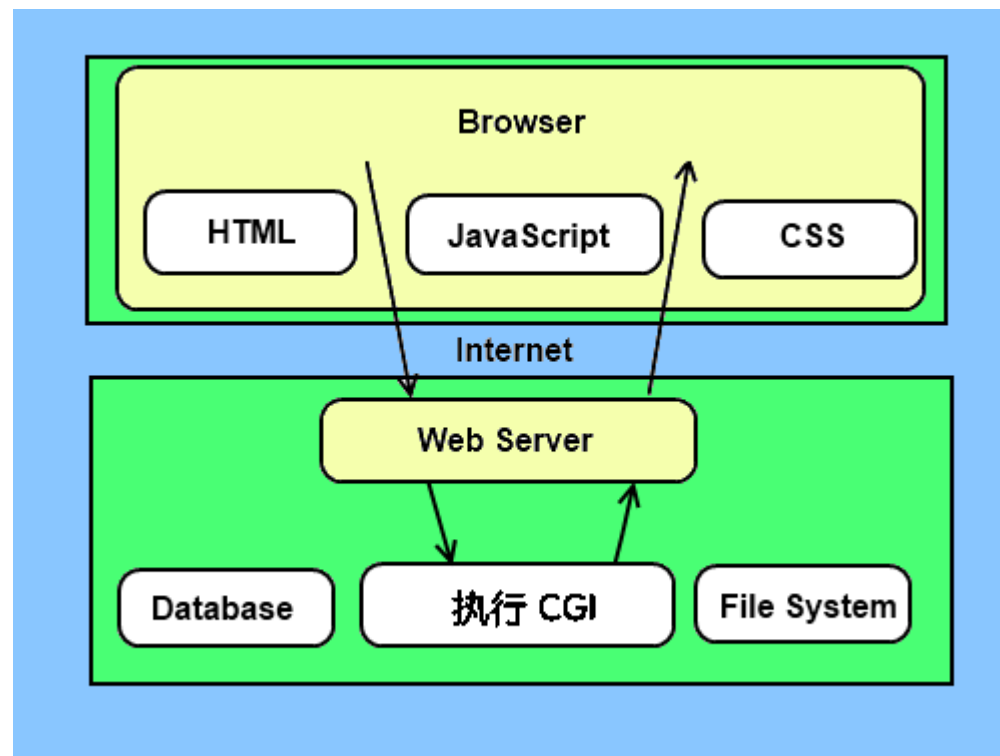
Web开发和浏览器的开端

- 1991年HTTP 0.9版本，1996年HTTP 1.0版本
- 1993年HTML
- 1994年，网景浏览器（Netscape）
- “将整个世界装入浏览器中”
- B-S模式（Browser-Server模式），使用Browser作为客户端
 - B-S和C-S模式没有本质区别
- 以显示静态内容为主



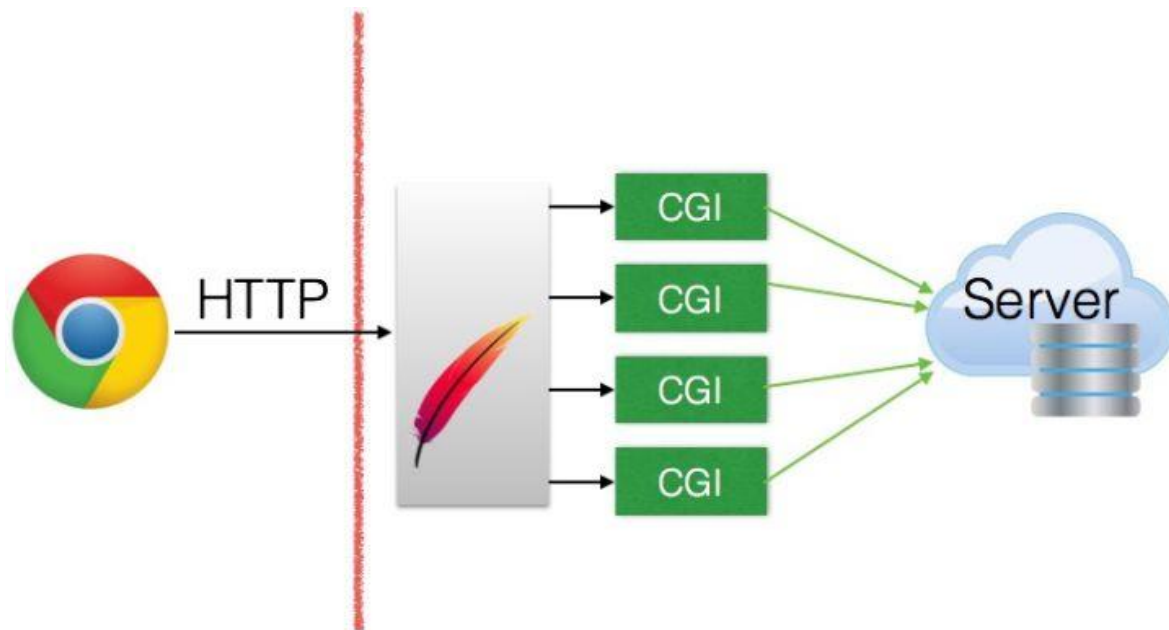
CGI

- Common Gateway Interface, 通用网关接口
- Web Server与后端脚本交互的**标准**
- 处理Web Server处理过的请求信息，并将其用HTML格式返回
 - Web Server，监听HTTP请求并处理的程序
 - Apache, nginx, IIS (Internet Information Service)
- 语言无关，只是个标准
- 实现：Python WSGI, Java Servlet, Perl, PHP



一个CGI网站的架构

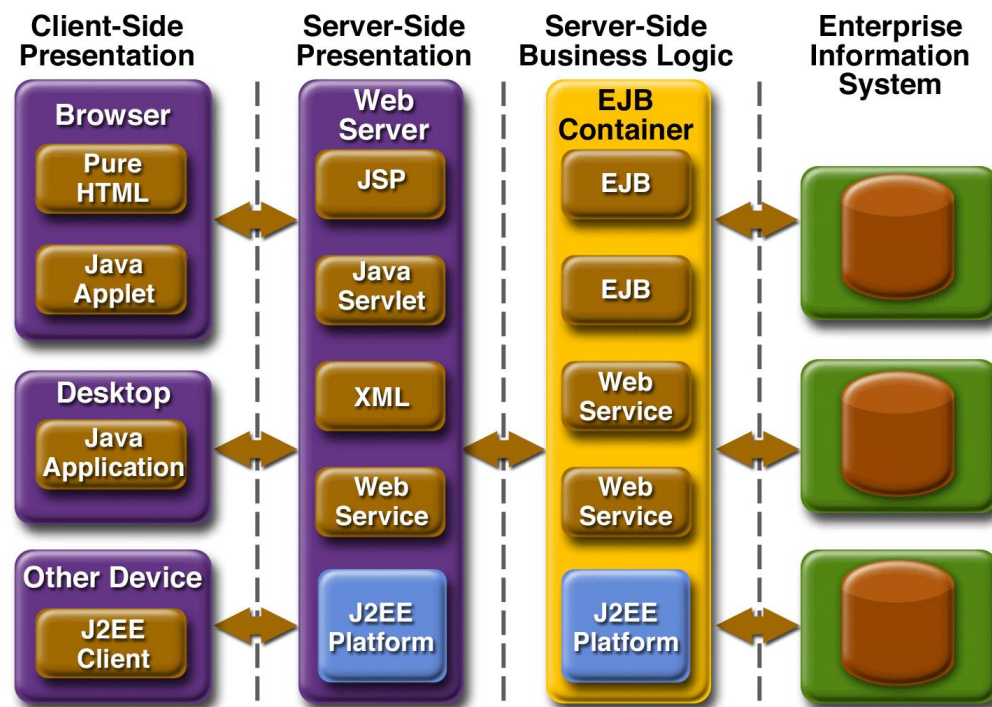
How CGI works?



1. CGI程序部署在Web Server上
 1. Servlet挂在Apache
2. HTTP请求到达时，Web Server调用CGI程序，重定向其标准输出，并将一些参数写入环境变量或者标准输入
3. CGI通过输入的数据进行处理，并将返回结果写入标准输出
4. Web Server返回标准输出流里的输入给客户端

Java 2 Platform

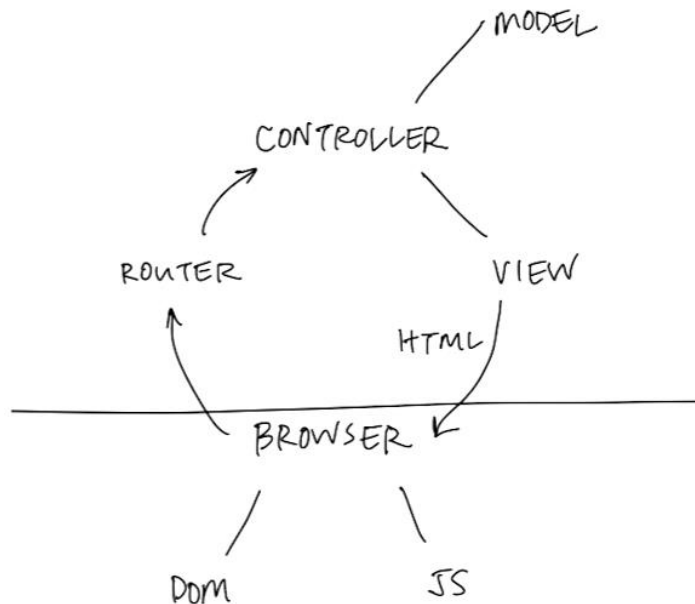
- 1999年，Sun公司，Java 1.2，J2EE/J2SE/J2ME
 - J2: Java 2 Platform
 - J2SE: Standard Edition: Java基础类
 - 数据库，网络...
 - J2ME: Micro Edition: 嵌入式电子产品...
 - 手机，机顶盒...
 - Android?
- J2EE: Enterprise Edition: 企业级应用
 - Servlet, JSP, EJB...
- 生态系统:
 - Tomcat: 自带Web Server的容器
 - Spring: 2002年，原本用于替代EJB
- Web企业开发的标准



J2EE架构

快速开发

- “企业级开发” 繁琐，复杂
- “约定大于配置”
- 简单轻量的架构，快速开发
- 2005年Ruby on Rails
 - GitHub, Hulu...
- 2005年Python Django
- 2010年Python Flask



一个简单的MVC架构

Flask is Fun

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

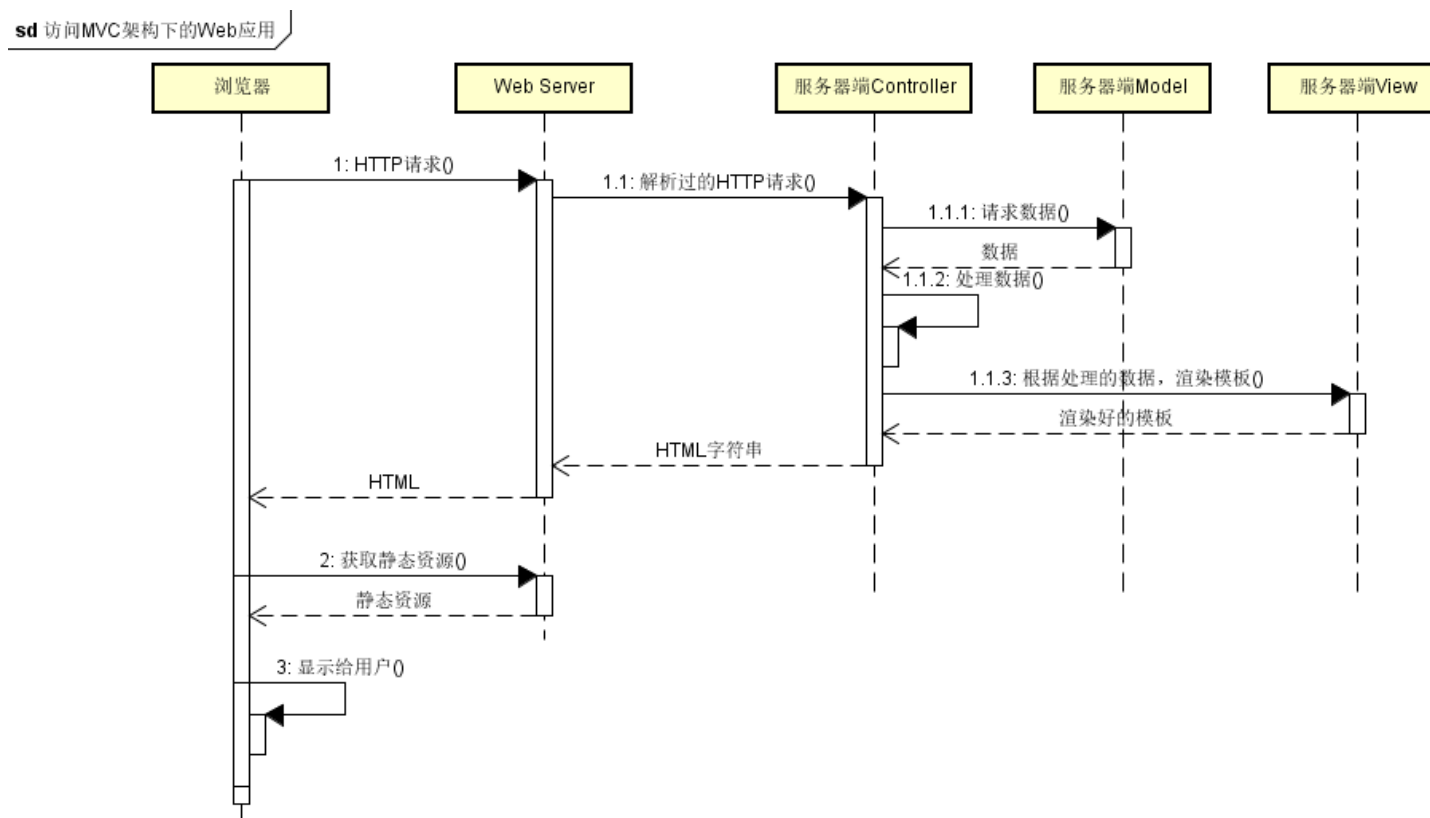
And Easy to Setup

```
$ pip install Flask
$ FLASK_APP=hello.py flask run
* Running on http://localhost:5000/
```

5行代码启动一个网站

访问MVC架构的Web应用

- MVC
- 模板引擎



模板引擎和渲染模板

- 模板即**HTML**的模板
- JSP, jinja2(Flask), Razor(ASP.NET MVC)
- Vue? Angular?

```
<div class="form-group">
  
  <input type="text" class="form-control" name="captcha" placeholder="验证码" />
</div>
```

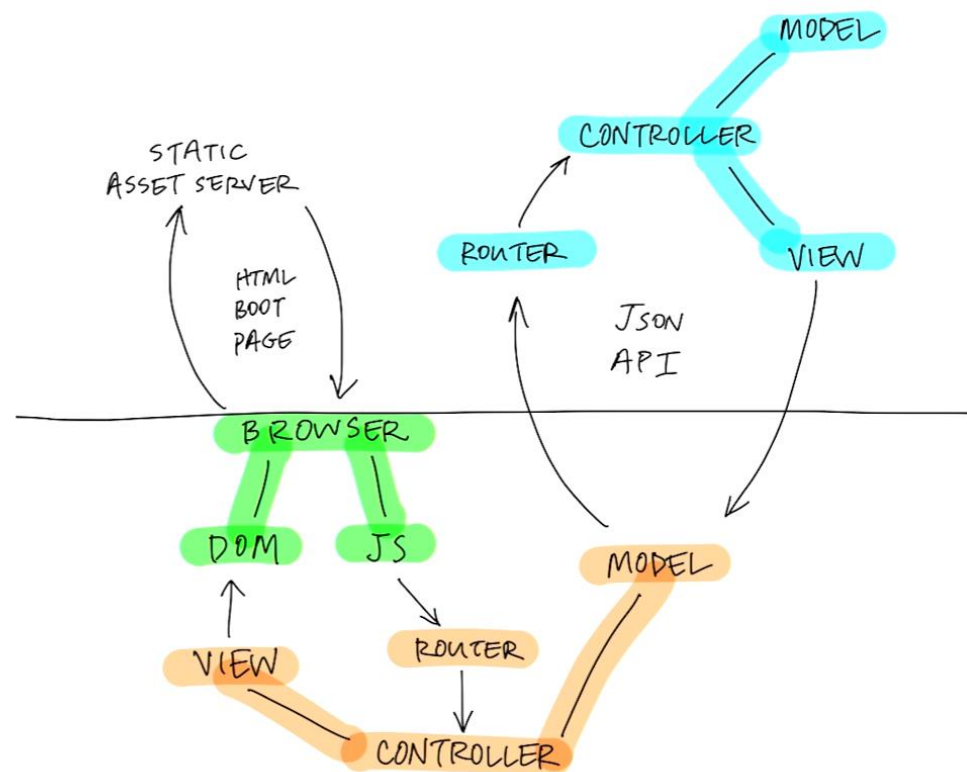
模板占位符

```
@app.route("/login",methods=["GET","POST"])
def logins():
    login_url="http://cer.nju.edu.cn/amserver/UI/Login"
    filename=datetime.datetime.now().timestamp()
    captcha_path = os.path.join(os.path.dirname(__file__), os.path.join("static","temp","{0}.jpg".format(filename)))
    if session.get("captcha_path") and os.path.exists(session.get("captcha_path")): ...
    if request.method=="GET": ...
    return render_template("/login.html",img=url_for("static",filename="temp/{0}.jpg".format(filename)))
```

在Controller处理数据后，将数据传入模板
将模板的占用符填充成真正的数据
将模板转换为真正的HTML字符串
即“模板渲染”

Node.js

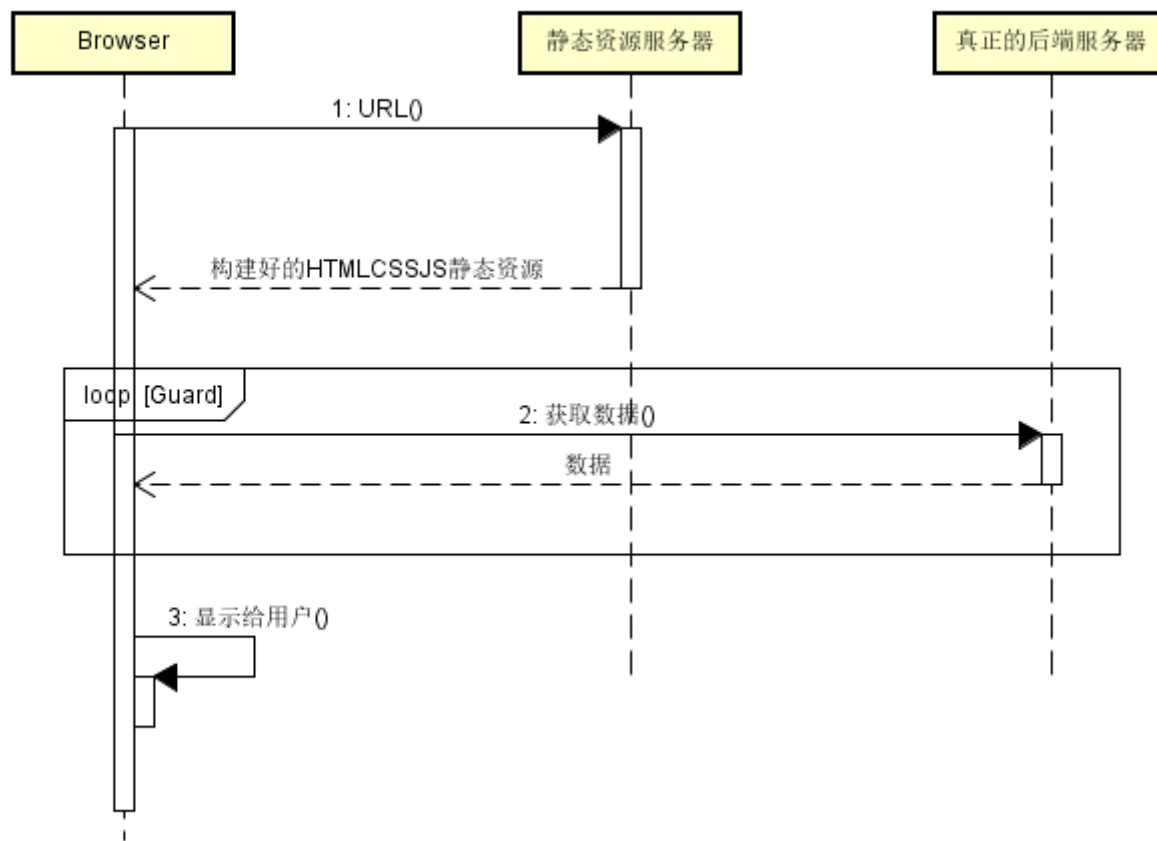
- 2009年，Node.js运行时，在服务器上运行JS
- 前端工程化的基础
- 前后端分离开始发展
 - 前端：渲染页面
 - 后端：提供数据



前后端分离下的架构

访问前后端分离下的Web应用

sd 访问前后端分离下的Web应用



前端技术的变化

前端技术的变化

- 2006: HTML/CSS/JS, 静态内容
 - RIA (Rich Internet Application) : Flash, Applet, Silverlight (2007)
- 2005-2006: AJAX, jQuery, 前端能力和复杂度提高
- 2009: Node.js, 前端MVC/MVVM框架
- 2013-现在: React/Vue/Angular, 同构应用, 服务器端渲染

-2006

- 静态内容
- HTML写结构，CSS写样式，JS写简单的交互
- CGI/服务器程序拼接好字符串，浏览器解析
- Rich Internet Application
 - Flash
 - Applet
 - Silverlight（2007）
 - 问题：标准化，性能，安全

```
PrintWriter writer = response.getWriter();
response.setContentType("text/html; charset=utf-8");
writer.write("S: "<html>" +
            "<head><title>hello</title></head>" +
            "<body><h1>Hello World</h1><body>" +
            "</html>");
```

在Servlet中拼接HTML字符串

Rich Internet Application

```
import java.applet.Applet;
import java.awt.*;

// Applet code for the "Hello, world!" example.
// This should be saved in a file named as "HelloWorld.java".
public class HelloWorld extends Applet {
    // This method is mandatory, but can be empty (i.e., have no actual code).
    public void init() {}

    // This method is mandatory, but can be empty (i.e., have no actual code).
    public void stop() {}

    // Print a message on the screen (x=20, y=10).
    public void paint(Graphics g) {
        g.drawString("Hello, world!", 20, 10);

        // Draws a circle on the screen (x=40, y=30).
        g.drawArc(40, 30, 20, 20, 0, 360);
    }
}
```

```
<!DOCTYPE HTML PUBLIC
    "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE>HelloWorld_example.html</TITLE>
</HEAD>
<BODY>
<H1>A Java applet example</H1>
<P>Here it is: <APPLET code="HelloWorld.class" WIDTH="200" HEIGHT="40">
This is where HelloWorld.class runs.</APPLET></P>
</BODY>
</HTML>
```

Applet



```
<embed name="kd" src="/imageyx/player/kd.swf"
quality="high" allowScriptAccess="always"
pluginspage="http://www.adobe.com/shockwave/download/downlo
ad.cgi?P1_Prod_Version=ShockwaveFlash" type="application/x-
shockwave-flash" width="50" height="26"></embed>
```

Flash

AJAX

- 2005年AJAX, Asynchronous JavaScript and XML
- 实现局部刷新, 降低网络开销, 提升用户体验
- Outlook Mail, Gmail...

传统提交表单的方式

浏览器刷新, 等待服务器响应

一个表单

123 123 submit

按下submit

服务器响应
整个HTML

一个表单

用户名 密码 submit

提交成功!

使用AJAX实现局部刷新

浏览器不刷新

一个表单

123 123 submit

按下submit

一个表单

用户名 密码 提交中

服务器响应
只包含关键
数据

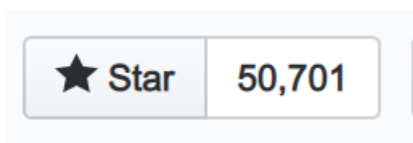
一个表单

用户名 密码 submit

提交成功!

jQuery

- 2006年
- 享誉世界的JavaScript库
- 解决了诸多痛点
 - 浏览器兼容问题
 - 简洁强大的API设计
 - 动画，ajax等强大功能
 - 插件系统



```
> $(".download-main").addClass("123").css("color", "green")
< ▶ jQuery.fn.init [div.download-main.123, prevObject: jQuery.fn.init(1)
> document.querySelector(".download-main").classList.add("123")
< undefined
> document.querySelector(".download-main").style.color = "green"
< "green"
```

简洁易用统一的DOM API

```
> $.ajax({ url: "/test", method: "POST", body: { bodyContent: "bodyContent" }}).done((data) => console.log(data))
< ▶ {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}
✖ ▶ POST http://api.jquery.com/test 404 (Not Found) jquery-1.11.3.j
> var xhr = new XMLHttpRequest();
< undefined
> xhr.open("POST", "/test", true);
< undefined
> xhr.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
< undefined
> xhr.onload = (data) => console.log(data);
< (data) => console.log(data)
> xhr.send(JSON.stringify({bodyContent: "bodyContent"}))
< undefined
✖ ▶ POST http://api.jquery.com/test 404 (Not Found) VM:
▶ ProgressEvent {isTrusted: true, LengthComputable: false, Loaded: 9439, total: 0, type: "Load", ...} VM:
```

简单易用的AJAX

Node.js

- 2008年，Chrome V8 JavaScript引擎
 - JavaScript运行效率的突破
- 2009年，基于V8的Node.js运行时
- 使JavaScript能够运行在浏览器之外
- 前端工程化、前后端分离、JavaScript全栈的根本基础
- 相关项目：
 - Node Package Manager (npm)
 - Webpack
 - Express

```
serve.ts x
1  const pathObj = require("path");
2  const rootDir = pathObj.join(__dirname, "../dist");
3  const port = 3000;
4  const express = require("express");
5
6  const app = express();
7
8  app.use(express.static(rootDir));
9
10 app.listen(port);
11
12 console.log('Serving production files from: '+rootDir);
13 console.log(`Listening at http://localhost:${port}`);
14 console.log('Press Ctrl + C to stop.');
```

使用Express启动一个服务器

前端MVC/MVVM框架

- MVC/MVVM从后端移向前端

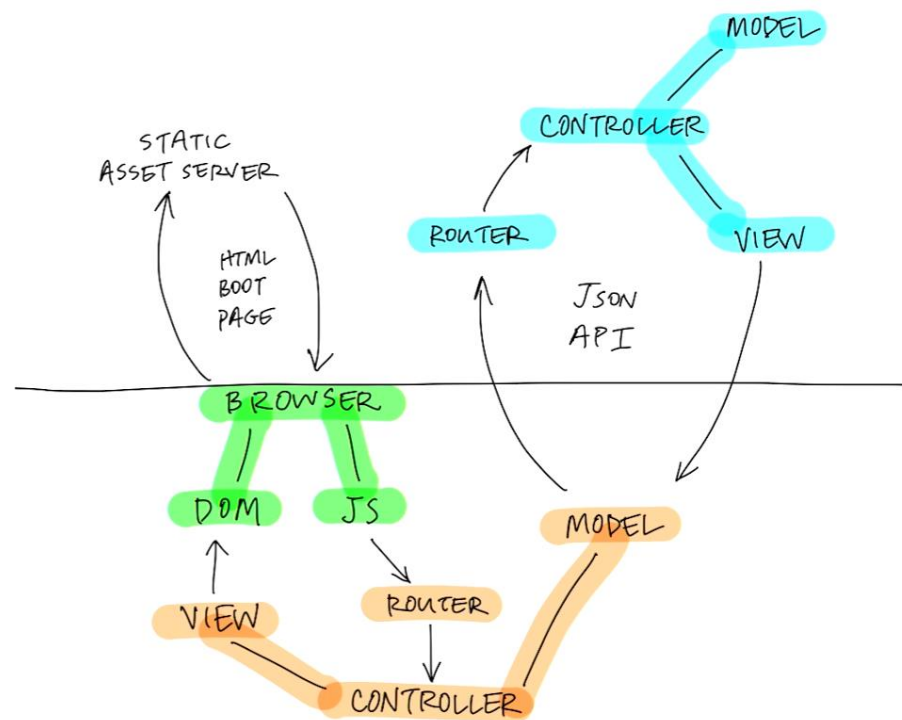
- 原因

- V8使JS执行效率大幅提升
 - 前端功能逐渐变得更加复杂
 - Node和NPM提供的足够的基础设施
 - 模块化设施（CommonJS），生态系统，测试调试设施

- 带来的好处

- 减少服务器压力
 - 减少网络请求
 - 提高用户体验
 - 快速迭代

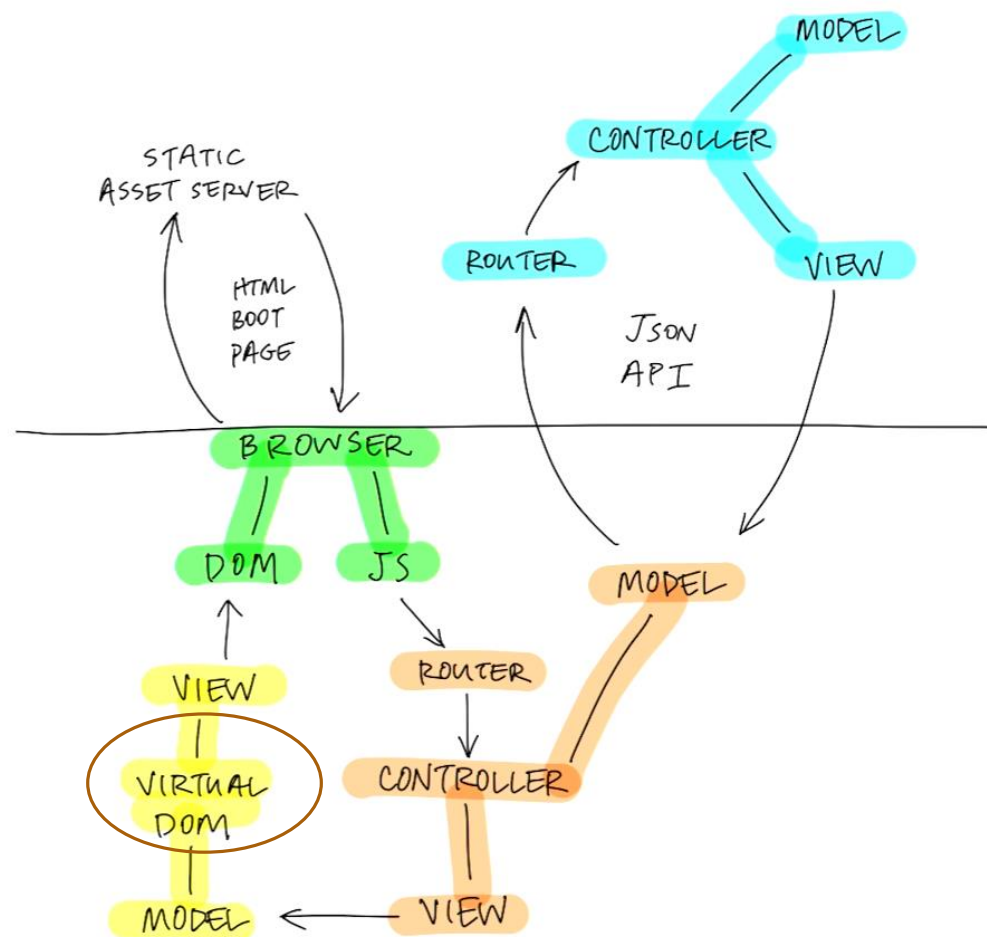
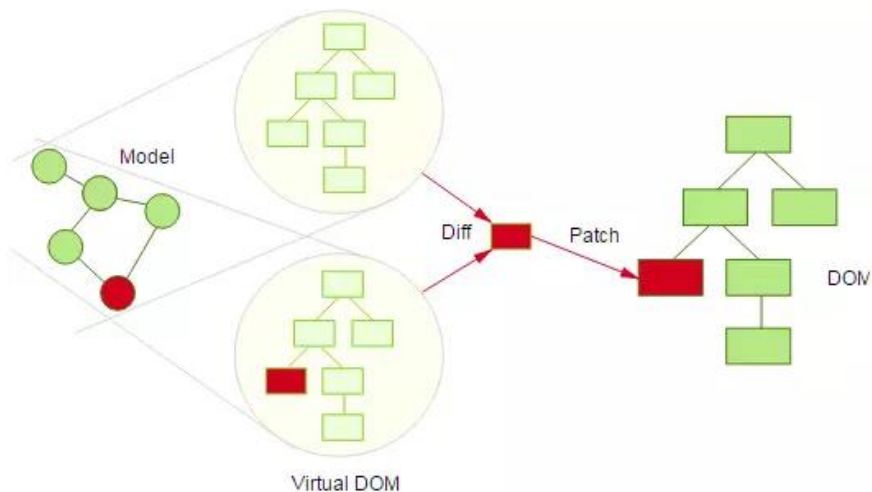
- Backbone, Ember.js, Angular.js



MVC下的架构

Virtual DOM

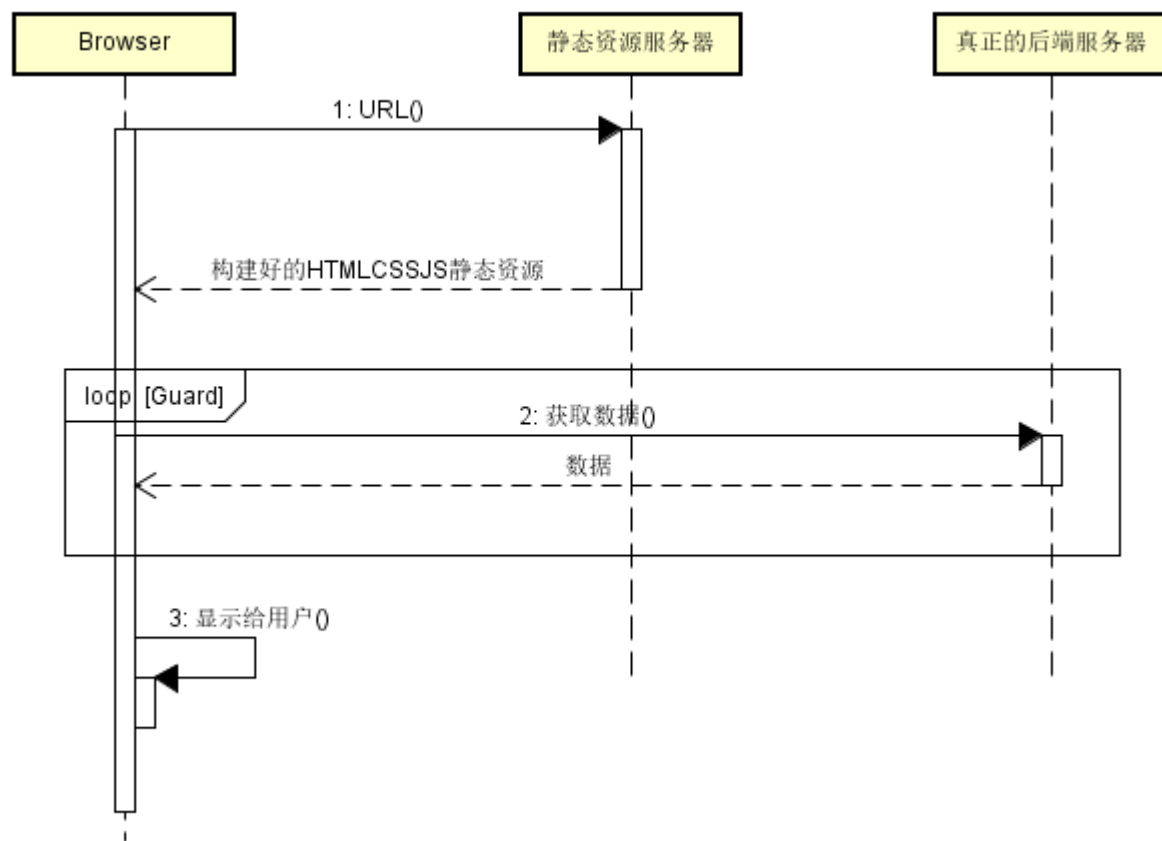
- 框架自己维护一套DOM模型
- 在框架更新时和真实DOM进行diff
- 对不一样的地方进行PATCH
- 减少DOM操作，提高性能
- 三大框架均采用Virtual DOM
- jQuery?



有Virtual DOM的架构

客户端渲染

sd 访问前后端分离下的Web应用



- 用户无论输入什么URL，静态资源服务器总是返回同样的HTML/CSS/JS资源
- HTML的唯一作用是调用JS
- **JS中的逻辑渲染出整个页面**
- 用户交互后，由JS发起获取数据的请求，处理后更新DOM
- 问题
 - JS包大小膨胀
 - 首屏加载时间
 - JS运行性能
 - 爬虫，搜索引擎

爬虫问题



用户眼中的网站

```
<!DOCTYPE html> <html> <head> <meta charset=utf-8> <meta name=viewport content="width=device-width, initial-scale=1"> <title>A+Quant</title> <script>!function(i) {if(i. search) {var a= {};i. search. slice(1). split("&"). forEach(function(i) {var l=i. split("=");a[l[0]]=l. slice(1). join("="). replace(/~/g, "&")}}, void 0!==a. p&window. history. replaceState(null, null, i. pathname. slice(0, -1)+(a. p||"")+ (a. q?"?" +a. q:"")+i. hash)}} (window. location)</script> <link rel="shortcut icon" href="/logo.png"><link href="/13.css" rel="stylesheet"><link href="/11.css" rel="stylesheet"></head> <body> <div id=root> <p>Initializing...</p> </div> <script type="text/javascript" src="/index.66320.js"></script><script type="text/javascript" src="/vendors~main.66320.js"></script><script type="text/javascript" src="/main.66320.js"></script></body> </html>
```

搜索引擎和爬虫眼中的网站

大部分搜索引擎无法检索到网站上的内容

JS大小膨胀

- 将所有资源打包进一个JS
- 影响
 - 加载不需要的资源
 - 影响首屏加载时间
 - 用户从发起网络请求到看到界面的时间
 - 解析大量JS的性能问题

index.html	2019/1/5 15:58	Chrome HTML D...	1 KB
index.js	2019/1/5 15:58	JavaScript 文件	15,181 KB
index.js.map	2019/1/5 15:58	Linker Address ...	37,731 KB
manifest.js	2019/1/5 15:58	JavaScript 文件	2 KB
manifest.js.map	2019/1/5 15:58	Linker Address ...	15 KB
vendor.js	2019/1/5 15:58	JavaScript 文件	648 KB
vendor.js.map	2019/1/5 15:58	Linker Address ...	4,060 KB

没有做代码分割的前端代码：过大的JS

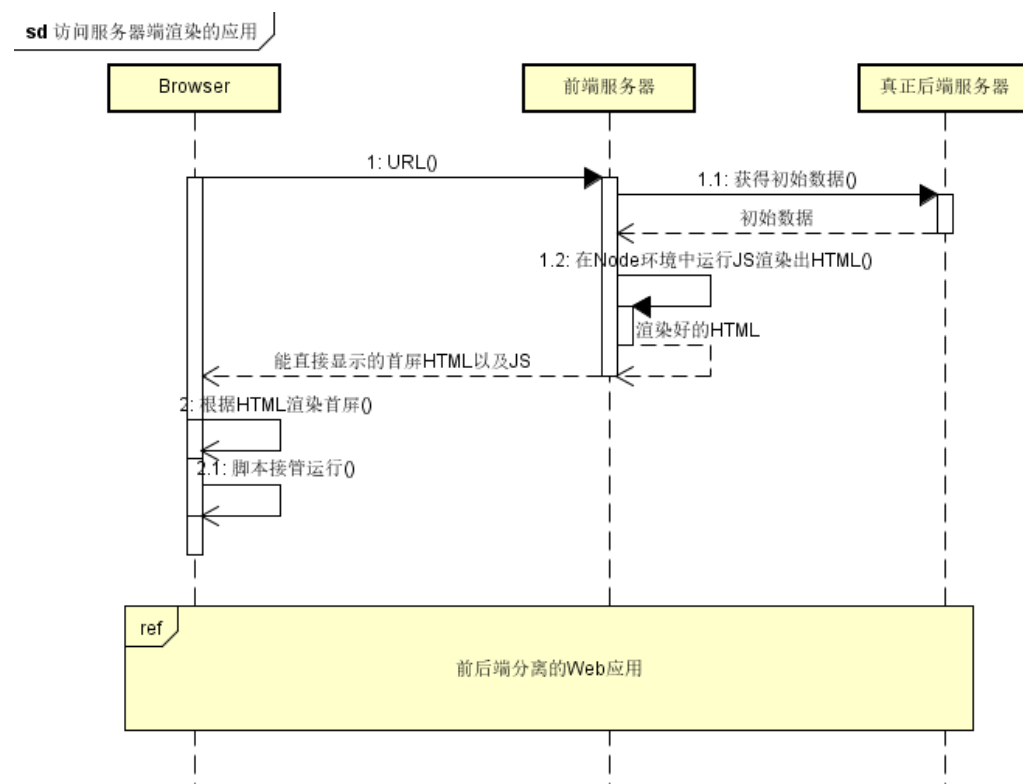
Name	Status	Type	Initiator	Size	Time	Water
inject.js	200	script	content.js:/6	(from disk ...)	23 ms	
favicon.ico	404	text/html	Other	399 B	29 ms	
manifest.js?f04c5c1a061df9f7381f	200	script	(index)	1.8 KB	45 ms	
localhost	200	document	Other	690 B	46 ms	
data:image/jpeg;bas...	200	jpeg	(index)	(from mem...)	120 ms	
vendor.js?2a5936ef539bf6b0f6bd	200	script	(index)	648 KB	5.10 s	
index.js?d4bb66341cc85c5e0031	200	script	(index)	14.8 MB	1.0 min	

7 requests | 15.5 MB transferred | Finish: 1.1 min | DOMContentLoaded: 1.1 min | Load: 1.1 min

需要太多网络流量和加载时间

服务器端渲染和同构应用

- 先把前端应用在服务器上渲染出HTML，再发给浏览器
- 浏览器加载出HTML后，再由脚本接管运行
- 解决首屏速度，爬虫问题
- 问题：
 - 浏览器环境和Node环境不同
 - 同构应用：能同时在Node和浏览器环境运行的应用
 - 状态同步
 - Vuex/Redux等状态管理工具



服务器渲染实例

知乎

[首页](#)

发现

话题

郑钧炮轰如今的音乐排行榜




提问

推荐

关注

热榜

如何看待百度因员工虚报打车发票一事开除员工？

老狼： 谢邀。虚假报销在很多公司都是高压线，一旦被核实就启动了响应的流程，一般都是零容忍而以开除了事。不赞成阴谋论。看待大公司不能把他想做一个人，大公... [阅读全文](#) 

▲ 贊同 61



15 条评论



台享




文藏



感谢



如何看待南京航空航天大学副教授张明宝证明黎曼猜想?

dhchen: 谢邀,看了摘要,似乎主要的工具是洛必达法则。我个人建议他投稿到有一定国际声誉的期刊,在此之前我认为它应该注意下面几件事 第一,他把洛必达... [阅读全文](#) 

▲ 赞同 621



84 条评论



台亭



收藏



感谢



浏览器初始得到的HTML 搜索引擎和爬虫看到的网站

用户看到的界面

现在的前端技术

HTML5

- 2014年10月确定标准
- 极大增强了Web应用的功能
 - 语义化标签
 - Article, header, footer...
 - 新的API
 - Canvas, video, audio, drag&drop, 地理位置, 剪切板.....
- 持续演进
- 标准化
 - ActiveX, Flash, Silverlight...

React, Vue, Angular

	React	Vue	Angular
发布时间	2013年	2014年	2016年
团队	Facebook	Evan You	Google
特点	Virtual DOM , JSX, 函数式	简单易上手, 轻量高效	高度工程化

工程化设施

- TypeScript/Flow: 给JS增加类型
- Babel: 构建时将新版本的ES语法转换成旧版本的语法
- Webpack及其生态系统: JS打包和预处理
 - Webpack自带: 静态代码优化, 代码分割
 - UglifyJS: “丑化JS”, 压缩JS大小, 静态代码优化
 - Cache-loader: 给构建加入cache, 加快构建速度
 - Hot Reload: 热重载, 写代码保存后, 不刷新网页即可看到效果
 - Webpack-dev-server: 测试服务器
 - ...
- Postcss: CSS预处理器
- Jest, Mocha等: 前端测试
- Mock Server
- Jslint, tslint: 代码风格检查
- ...

工程化设施

```
interface Props {  
  data: Matching[];  
}  
  
export class CurrentPosition extends React.Component<Props, {}> {  
  
  render() {  
    this.props.data[0].  
    return (  
      <Card style={{  
        <MatchingPage  
        percentage Matching (MatchingList... number  
        quotaId Matching (MatchingList.t... string  
        quotaName Matching (MatchingList... string  
        totalValue Matching (MatchingList... number  
        type Matching (MatchingList.ts... TranType
```

TypeScript类型提示

Put in next-gen JavaScript	Get browser-compatible JavaScript out
<pre>[1, 2, 3].map(n => n ** 2);</pre>	<pre>[1, 2, 3].map(function (n) { return Math.pow(n, 2); });</pre>

babel

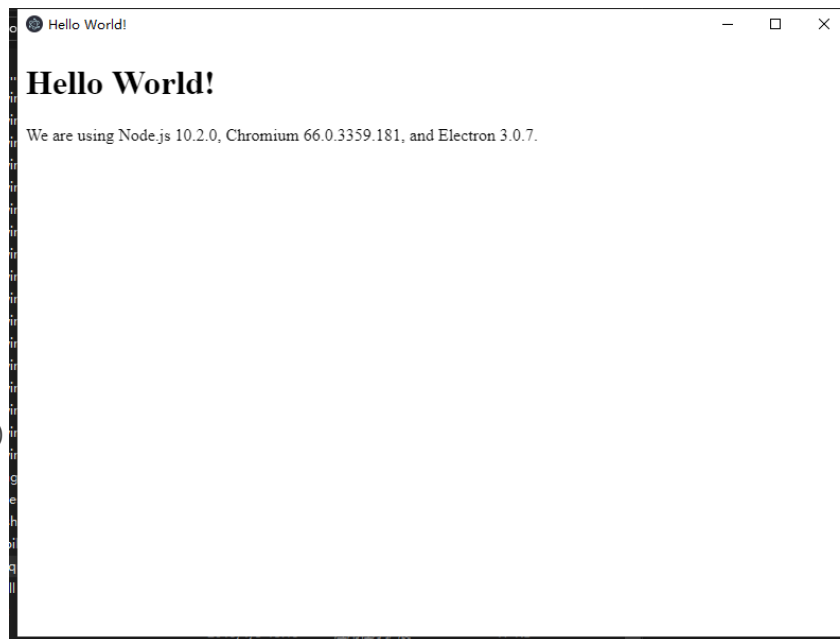
```
function aFunc(aParamter, bParamter){  
  var i=0;  
  for (var index=1;i<100;i++) {  
    setTimeout(console.log(i), 0);  
  }  
}
```

```
function aFunc(o,n){for(var i=0;100>i;i++)setTimeout(console.log(i),0)}
```

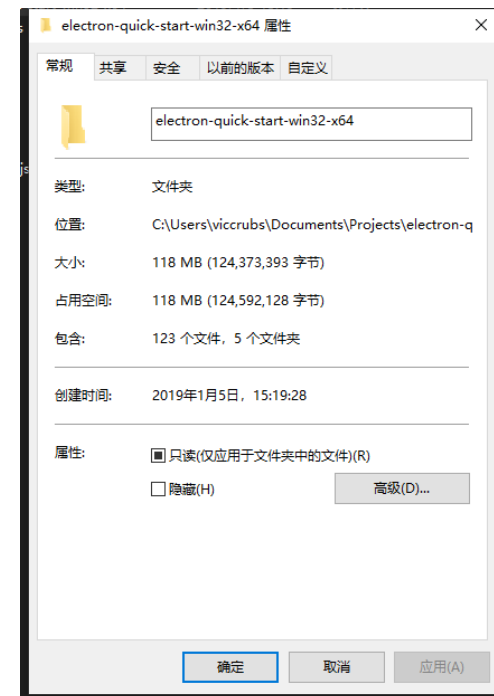
uglify

Electron

- 使用HTML/CSS/JS开发原生桌面应用
- 优点
 - 使用成熟的HTML/CSS/JS生态系统
 - 兼容Windows/macOS/Linux多个平台
- 缺点
 - 打包大小过大（毕竟需要Chromium内核）
 - 速度慢
- 示例
 - Visual Studio Code, GitHub Desktop...



Hello World

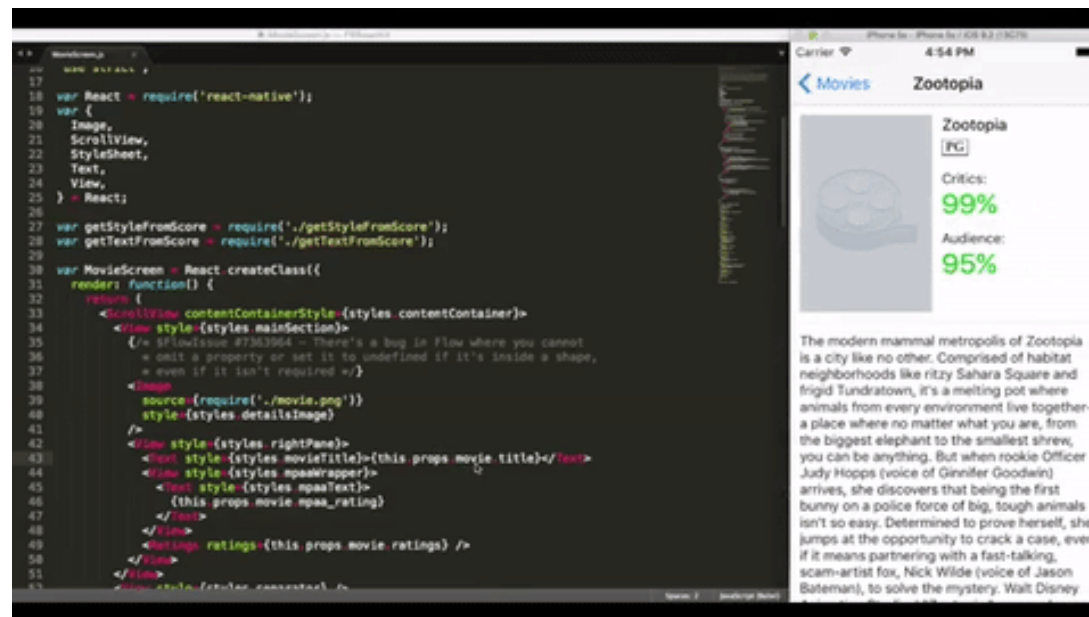


左侧项目
打包大小

...

React Native/NativeScript

- 使用React/Vue/Angular编写原生移动App
- 生成原生控件，不是WebView
- 问题：
 - 从写2个平台到写3个平台？
- 后续：Flutter



React Native的官网演示

未来值得关注的 前端技术

WebAssembly

- JavaScript的性能问题

- WebAssembly

- 一种机器码，很接近各个平台原生汇编的语法
- 各个浏览器正在逐渐支持
- XXX Compiles to WebAssembly
 - LLVM, C#, Java, Rust...

```
1 export function f(x: i32): i32 {  
2     if (x === 1 || x === 2) {  
3         return 1;  
4     }  
5     return f(x - 1) + f(x - 2)  
6 }
```

TypeScript的子集



```
1 func $src/asm/module/f (param f64) (result f64)  
2 (local i32)  
3     get_local 0  
4     f64.const 1  
5     f64.eq  
6     tee_local 1  
7     if i32  
8         get_local 1  
9     else  
10        get_local 0  
11        f64.const 2  
12        f64.eq  
13    end  
14    i32.const 1  
15    i32.and  
16    if  
17        f64.const 1  
18        return  
19    end  
20    get_local 0  
21    f64.const 1  
22    f64.sub  
23    call 0  
24    get_local 0  
25    f64.const 2  
26    f64.sub  
27    call 0  
28    f64.add  
29    end
```

编译成的字节码

Asm.js

- asm.js
 - 使用简单的类型标注
 - 给支持asm.js的引擎提供优化空间
 - Firefox

```
var a = 1;  
  
var x = a | 0; // x 是32位整数  
var y = +a; // y 是64位浮点数
```

类型标注

XXX Compiles to JavaScript

- C++, Kotlin, F#, OCaml...



Open source (MIT license) LLVM-based C++ to JavaScript compiler

C++ ⇒ LLVM ⇒ Emscripten ⇒ JavaScript

Another example:

```
float array[5000]; // C++
int main() {
  for (int i = 0; i < 5000; ++i) {
    array[i] += 1.0f;
  }
}
```

⇒ Emscripten ⇒

```
var buffer = new ArrayBuffer(32768); // JavaScript
var HEAPF32 = new Float32Array(buffer);
function main() {
  var a = 0, b = 0;
  do {
    a = (8 + (b << 2)) | 0;
    HEAPF32[a >> 2] = +HEAPF32[a >> 2] + 1.0;
    b = (b + 1) | 0;
  } while ((b | 0) < 5000);
}
```

C++(LLVM) to JavaScript

```
F# HTML CSS
4 // MODEL
5
6 type Model =
7     { Value : string }
8
9 type Msg =
10     | ChangeValue of string
11
12 let init () = { Value = "" }, Cmd.none
13
14 // UPDATE
15
16 let update (msg:Msg) (model:Model) =
17     match msg with
18     | ChangeValue newValue ->
19         { Value = newValue }, Cmd.none
20
21 // VIEW (rendered with React)
22
23 let view model dispatch =
24     div [ Class "main-container" ]
25     [ input [ Class "input"
26         Value model.Value
27         OnChange (fun ev -> ev.target.value |> string |> ChangeValue |> dispatch)
28         span [ ]
29             [ str "Hello, "
30               str model.Value
31               str "!" ] ]
32
33 // App
34 Program.mkProgram init update view
35 |> Program.withConsoleTrace
36 |> Program.withReactSynchronous "elmish-app"
37 |> Program.run
38
39 Live sample Code
40
41 import { union, record, string } from "fable-library/Reflection";
42 import { Union, declare, Record } from "fable-library/Types";
43 import { withReactSynchronous } from "fable-repl-lib/Elmish.React";
44 import { div, span, str, input, Props$0024EDOMAttr as Props$00240024EDOMAttr, Props$0024Cmd as Props$00240024Cmd } from "fable-repl-lib/src/cmd";
45 import { Cmd$$$none as Cmd$0024$0024$0024none } from "fable-repl-lib/src/cmd";
46 import { ProgramModule$$$run as ProgramModule$0024$0024$0024run, ProgramModule$$$view as ProgramModule$0024$0024$0024view } from "fable-repl-lib/src/cmd";
47 export const Model = declare(function Elmish_SimpleInput_Model(arg1) {
48     this.Value = arg1;
49 }, Record);
50 export function Model$reflection() {
51     return record("Elmish.SimpleInput.Model", [], Model, () => [["Value", string]]);
52 }
53 export const Msg = declare(function Elmish_SimpleInput_Msg(tag, name, ...fields) {
54     Union.call(this, tag, name, ...fields);
55 }, Union);
56 export function Msg$reflection() {
57     return union("Elmish.SimpleInput.Msg", [], Msg, () => [["ChangeValue", [string]]]);
58 }
59 export function init() {
60     return [new Model(""), Cmd$0024$0024$0024none()];
61 }
62 export function update(msg, model) {
63     const newValue = msg.fields[0];
64     return [new Model(newValue), Cmd$0024$0024$0024none()];
65 }
66 export function view(model$$$1, dispatch) {
67     return div([new Props$00240024HTMLAttr(23, "className", "main-container")], [input(
68         dispatch(new Msg(0, "ChangeValue", String(ev.target.value)));
69     )], span([, [str("Hello, "), str(model$$$1.Value), str("!")]]));
70 }
71 ProgramModule$0024$0024$0024run(withReactSynchronous("elmish-app", ProgramModule$0024$0024$0024view, update, view));
```

F# to JavaScript(Fable)

Progressive Web Application

- 本地缓存
- 在离线下也能打开应用
- 支持推送通知
- 支持安装在设备上
- 及时更新
- 示例: Twitter
- 类似框架
 - 微信/支付宝小程序
 - 快应用



像本地应用一样运行在各个平台

Web Components

- 标准的HTML组件化解决方案
 - 和React/Vue/Angular有本质不同
 - 标准支持，无需预处理
- 示例：
 - YouTube
 - Polymer框架

```
renderer">  
▼ <ytd-grid-video-renderer lockup class="style-  
scope ytd-grid-renderer"> == $0  
▼ <div id="dismissable" class="style-scope ytd-  
grid-video-renderer">  
▼ <ytd-thumbnail use-hovered-property width=  
"210" class="style-scope ytd-grid-video-
```

YouTube的一个自定义组件

```
1 class PopUpInfo extends HTMLElement {  
2   constructor() {  
3     // Always call super first in constructor  
4     super();  
5  
6     // write element functionality in here  
7  
8     ...  
9   }  
10 }
```

使用JS定义组件

```
1 <popup-info img="img/alt.png" text="Your card validation code (CVC)  
2   is an extra security feature – it is the last 3 or 4 numbers on the  
3   back of your card.">
```

直接在HTML里使用

WebSocket

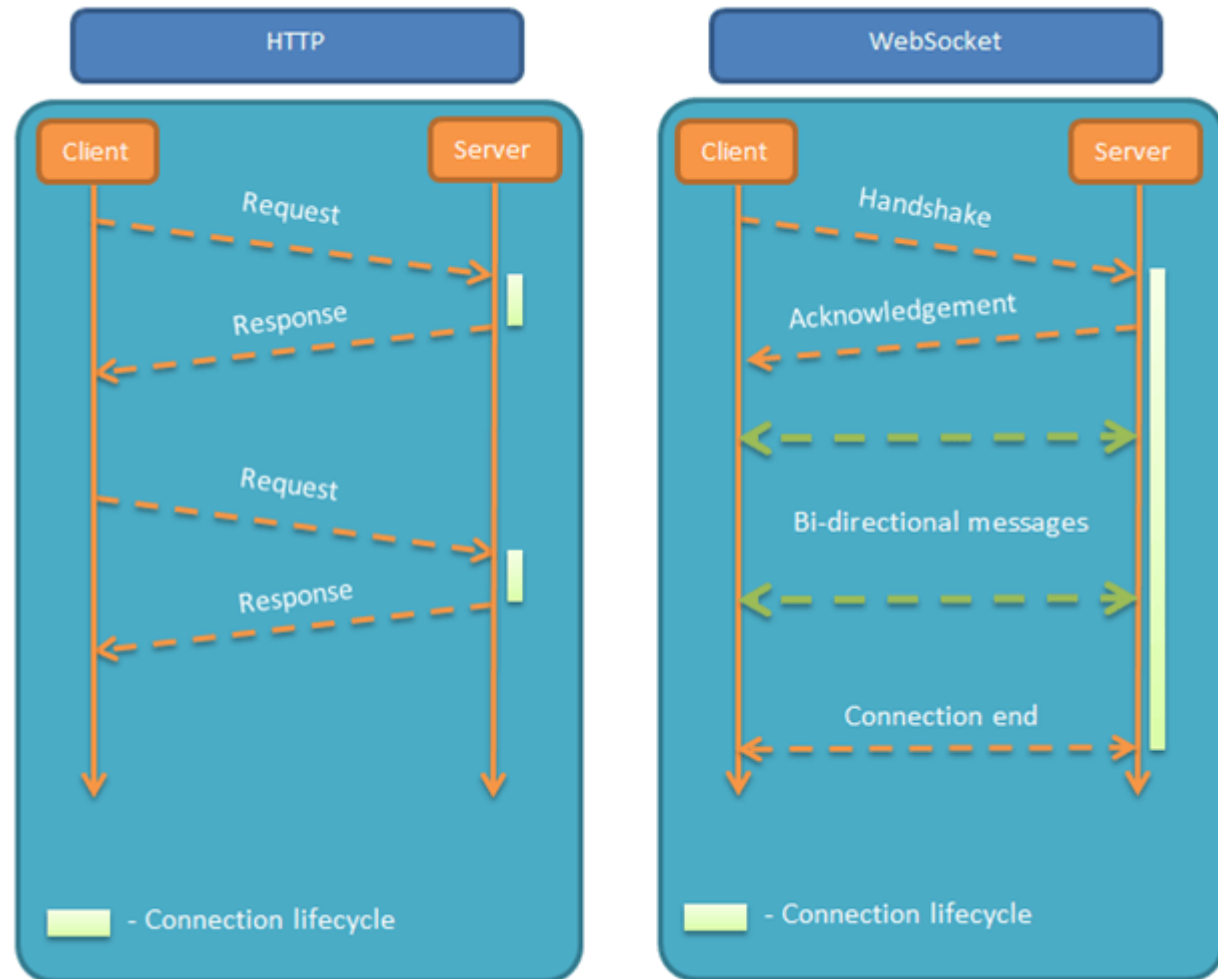
- HTTP是无状态的协议
- (HTTP/2之前)服务器不能主动推送信息
 - 浪费资源
- WebSocket
 - 像Socket一样，允许客户端和服务端双向交流
- 例子：
 - Microsoft Bot Framework
 - <http://njumscbot.azurewebsites.net/>

General

Request URL: wss://webchat.botframework.com/v3/directline/conversations/14sh0Z5QAnH685pN1uBZCb/stream?watermark=-&t=wFKfTKGwxms.dAA.MQA0AHMAaABPAFoANQBRAEEAbgBIADYA0AA1AHAATgAxAHUAQgBaAEMAYgA.8G7ubdGk1AE.6v6tSWN0sqY.NWrY1v0ed1_JzcTH4T-6rJRG_-k2Jtt8WZbWnSAxybA

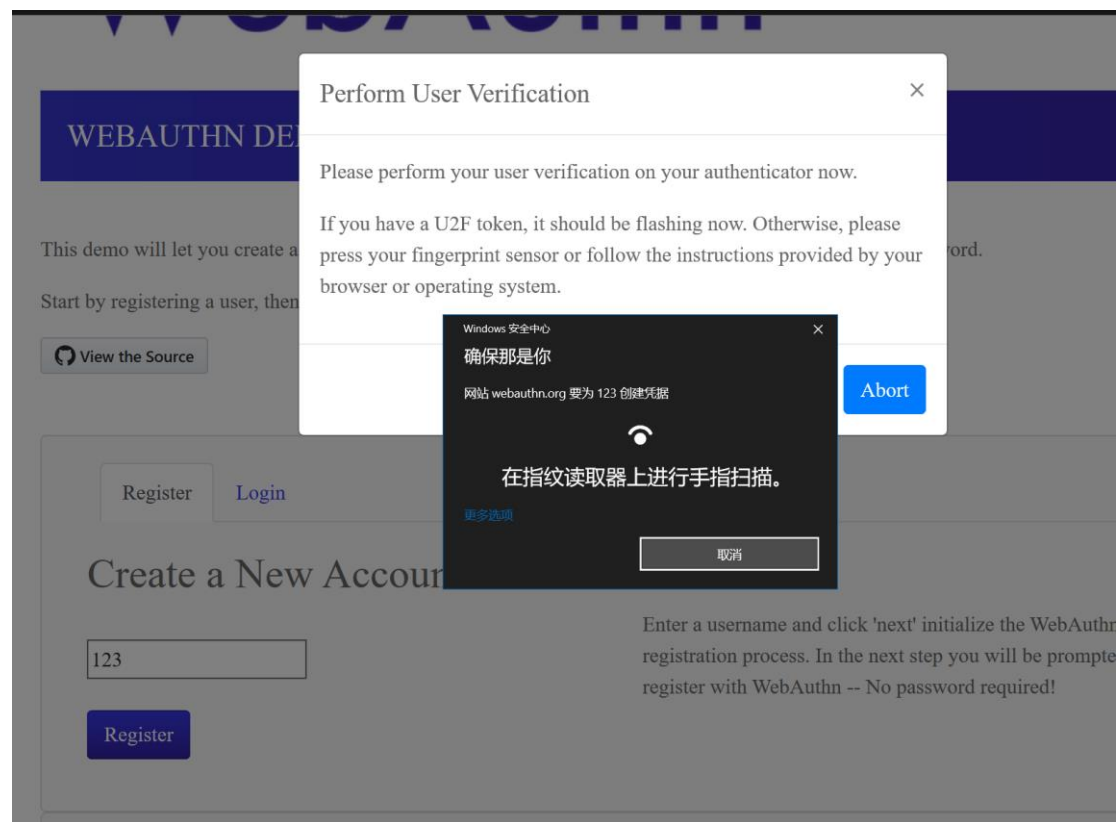
Request Method: GET

Status Code: 101 Switching Protocols



WebAuthn

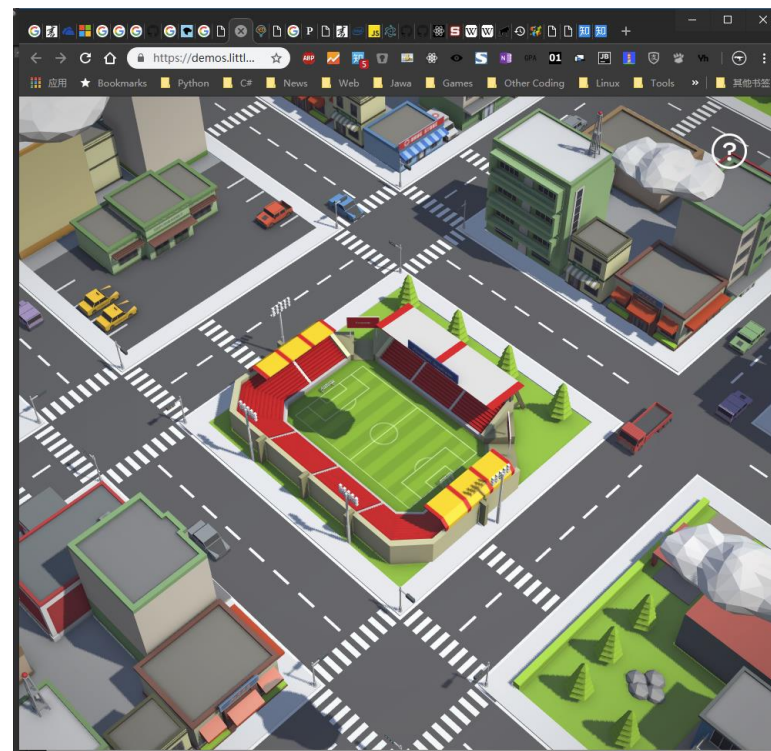
- Web Authentication and Authorization
- W3C标准
- 通过生物识别装置进行鉴权
- 2018年正式成为标准
- 已经获得大多数浏览器的支持
- <https://webauthn.org>



在Edge上，借助WebAuthn，通过Windows Hello注册

WebGL和Three.js

- 在浏览器中运行3D场景
- 云游戏？



Three.js制作的在浏览器中运行的可交互3D场景

谢谢！
