电商项目前端架构详解

图灵学院
郭嘉

1. 常见的前端开发模式详解
2. 基于NODEJS的前端开发生态
3. 前后端分离带来的问题分析
4. 前端代码调试实战

前端开发模式

1.前后端没有区分由程序员一人包干

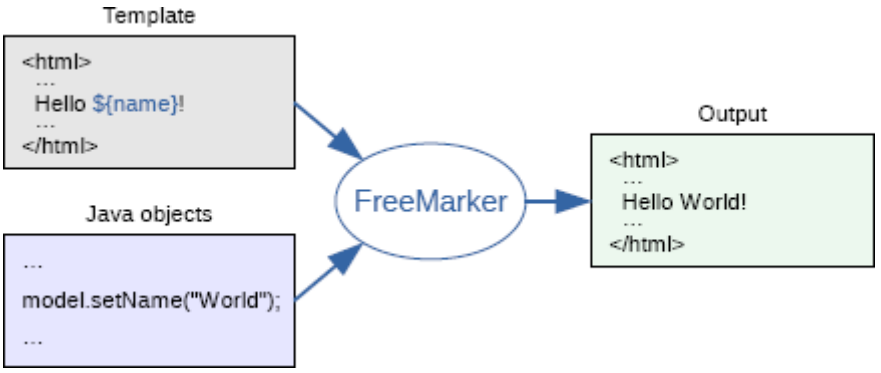
渲染模式：后端渲染，以jsp，asp, php 为代表的后端直接输出html形式，全部由后端工程师搞定
优点：开发模式简单
缺点：程序员职责过多，需要关注页面的展示，后台业务的逻辑处理

2.借助于FTL（Template Language），Velocity，Thymeleaf进行模板的编写,在模板中，前端工程师专注于如何展现数据，后端程序员展示什么样的数据及业务逻辑的处理，开启了功能性的职责划分。

渲染模式：同上

优点：分离了部分后端程序员的写页面的工作

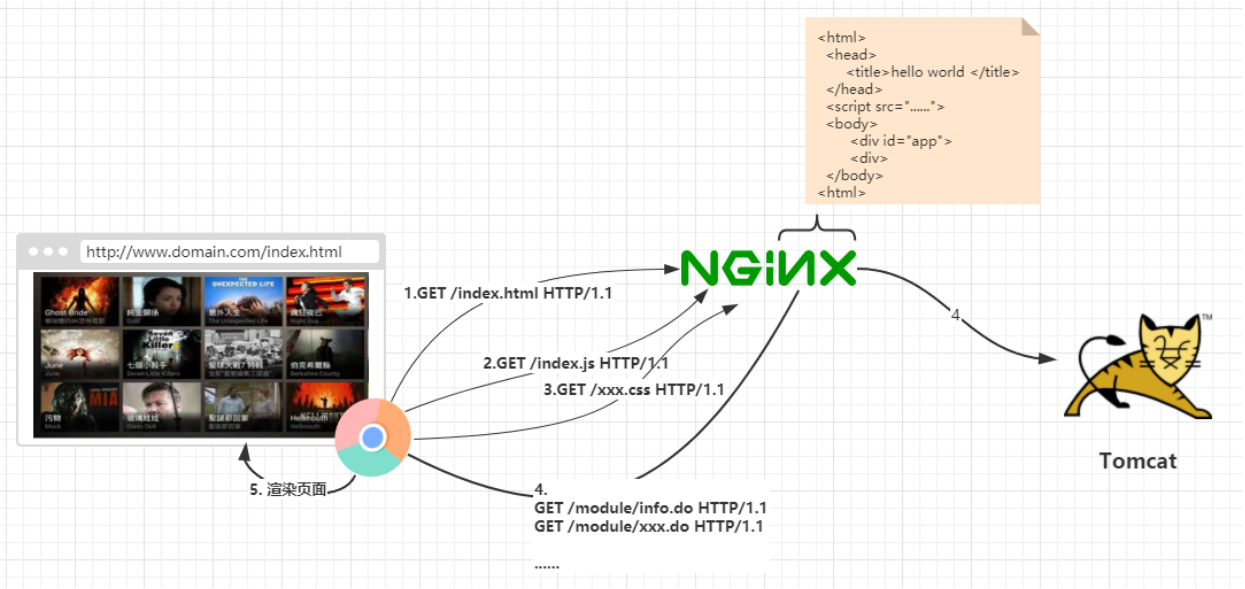
缺点：后端需要关注与前端页面的展现，如果要应对不同的业务需求，后台接口需要同步修改



3. 单页Web应用（single page web application，SPA），就是只有一张Web页面的应用，是加载单个HTML 页面并在用户与应用程序交互时动态更新该页面的Web应用程序。

渲染模式：

前端渲染：主要使用js来渲染页面的大部分内容的前端架构



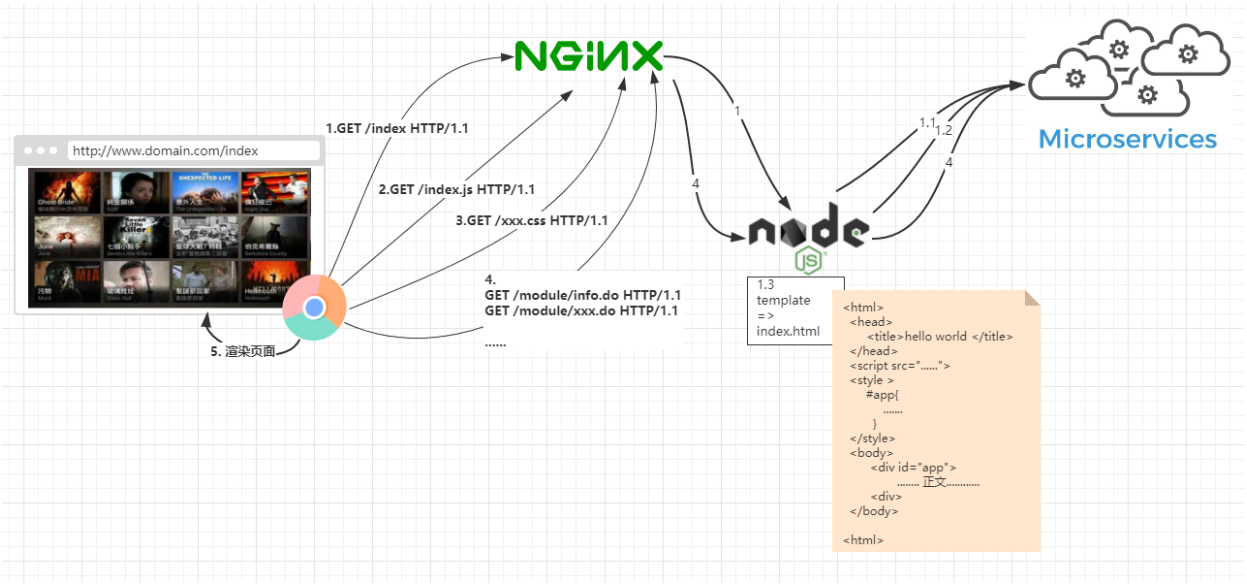
优点：完全前后端分离，职责分离，工程分离

缺点：

纯前端渲染，对seo不友好，首屏响应时间比较长

服务端渲染（同构渲染）：纯浏览器渲染会导致白屏的时间增加，应为请求需要先获取html，css，js，请求后端数据，执行渲染页面。前面执行过程过长容易导致白屏。可

以让请求html这一步过程返回的html就是已经渲染好的页面，浏览器可以直接显示，这样减少白屏时间，提升用户体验。能同时编写服务端渲染和前端渲染就成了问题，后端不擅长写页面，前端不懂后台，这时node.js 整好能解决这个问题。



3.基于NodeJs 的web前端工程

3.1 Node.js 是什么？

Node.js is a JavaScript runtime built on Chrome's V8

类比于java来说，Node.js 类似于 JRE，它是一个运行环境，是javascript 语言的宿主环境

3.2 Node.js 在web 中的应用场景

a. 构建前端工程（代码压缩，css兼容性的前缀，sass代码构建成css样式，浏览器兼容性问题等一系列自动化流程，babel,gulp,grunt,webpack,sass... ）

b. web 应用服务器：语言友好,使用 javascript语言，这样前端工程师是就可以·无缝切换到后端的开发当中

项目实战前端架构

1. 安装Node.js

```
https://nodejs.org/en/
```

2. 安装镜像

全局安装：

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
```

3. 本地安装插件:

```
npm install axios --save-dev
```

注意：

```
 打开了powershell命令行之后,输入
 set-ExecutionPolicy RemoteSigned
 然后更改权限为A

```

4. 代码获取：

```
git clone
```

5. 安装依赖

```
cnpm install
```

6. 运行项目

```
cnpm run serve
```

前后分离带来的跨域问题

同源策略（Same-origin policy）

什么是同源策略？

浏览器端对请求的处理中，如果两个 URL 的**协议、域名**和**端口**都相同，我们就称这两个 URL 同源

同源

```
http://www.domaina.com/index
http://www.domaina.com/module/path1
```

非同源

```
http://www.domaina.com/index
https://www.domaina.com/module/path1
```

```
http://www.domaina.com/index
http://www.domainb.com/module/path1
```

```
http://www.domaina.com/index
http://www.domaina.com:8081/module/path1
```

两个相同的源之间浏览器默认其是可以相互访问资源和操作 DOM 的。两个不同的源之间若想要相互访问资源或者操作 DOM，那么会有一套基础的安全策略的制约。具体有如下两方面的限制

1. 安全性： 浏览器要防止当前站点的私密数据不会向其他站点发送

如当前站点的Cookie,LocalStorage,IndexDb 不会被发送到其他站点或被其他站点脚本读取到

无法跨域获取Dom，无法发送Ajax请求。

2. 可用性：大型站点的图片，音视频等资源，希望部署在独立服务器上，为缓解当前服务的压力，开放某些特定的方式，访问非同源站点

如：<script><img><iframe><link><vedio>等，可以同src属性跨域访问

允许跨域提交表单/或重定向请求

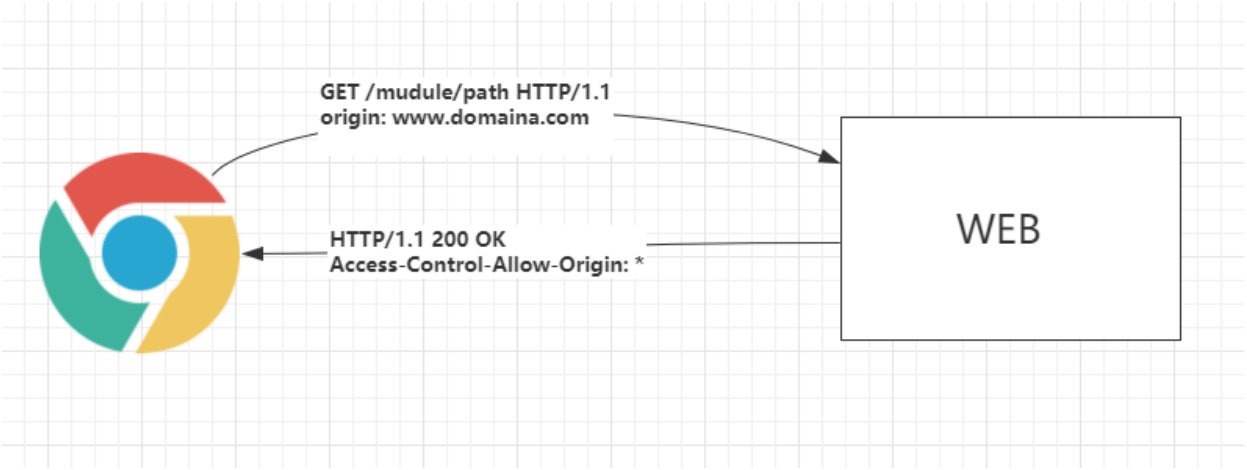
解决方案：

1. 服务端解决方案:

跨域请求分两种情况

a. 简单请求：

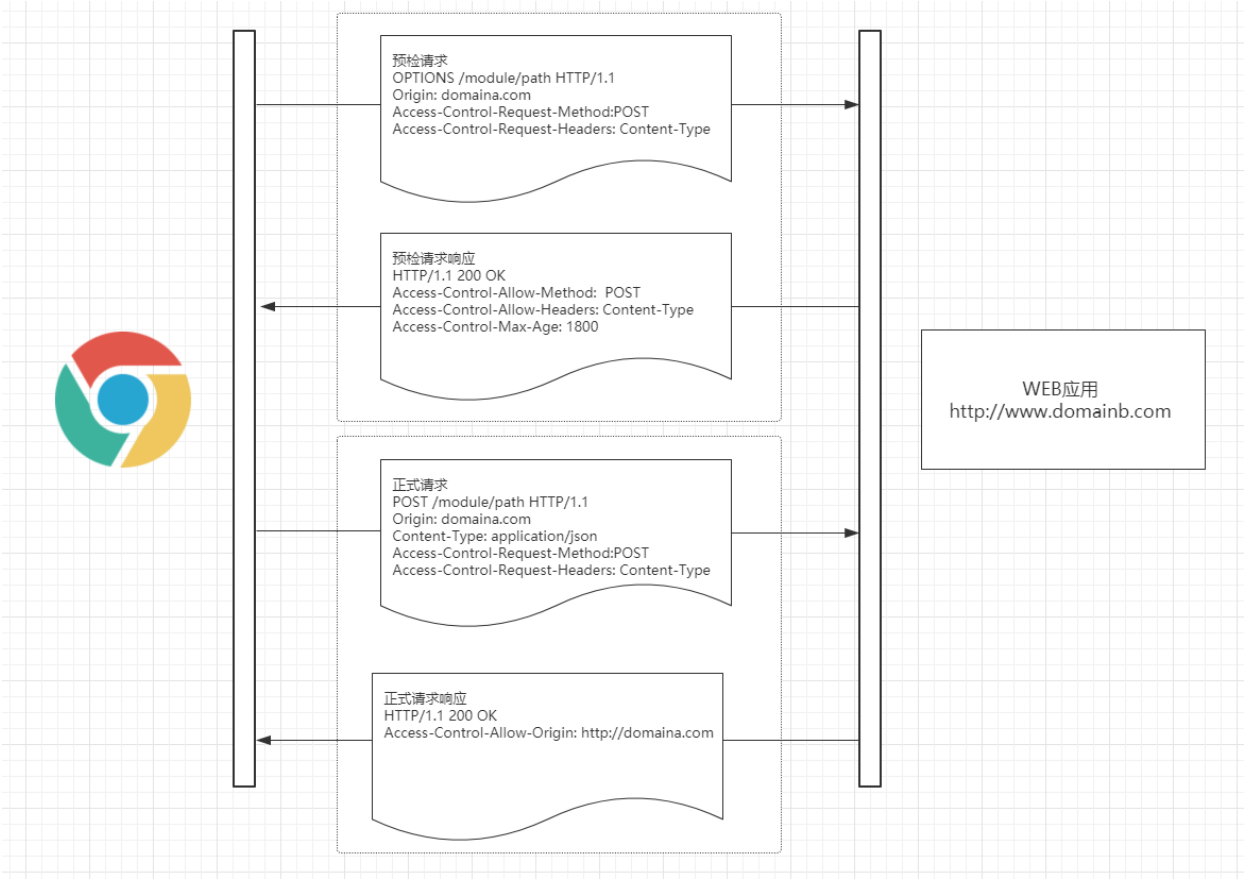
- 请求方法使用GET/HEAD/POST请求之一
- 仅能使用CORS安全的头部，Accept,Accept-Language, Content-Language,Content-Type
- Content-Type的值只能是：text/plain,multipart/form-data,application/x-www-form- urlencoded三者其中之一



服务端解决方案： 在http响应头中添加 Access-Control-Allow-Origin 头，值为信任的站点。

b. 复杂请求

不符合简单请求条件的即为复杂请求，访问跨域资源前，需要发起preflight预检请求（OPTIONS请求）询问何种请求是被允许的，预检请求失败，则不会发起正式的业务请求，预检请求成功，然后发起正式请求。



springboot

```
https://spring.io/guides/gs/rest-service-cors/#controller-method-cors-configuration
```

2. 代理服务器，反向代理接口请求

apache http server / nginx

```
location /api {
 rewrite ^/api/(.*)$ /$1 break;
 proxy_pass http://localhost:8081/;
}
```

3. jsonp 方式

前端路由：

为什么需要路由

- Ajax的方式实现了页面无跳转的数据更新，但是如果调用层级太深，当用户刷新页面时，数　　据恢复到初始状态，影响用户体验。
- Ajax的调用路径不会被浏览器记住，不支持前进/后退

实现方式：hash / history

Hash路由：url的hash以 # 开头，hash值得变化不会刷新页面，可以通过hash值得变化，触发hashchange事件，来处理对应的回调

History：History 路由是基于 HTML5 规范，在 HTML5 规范中提供了 history.pushState || history.replaceState 来进行路由控制