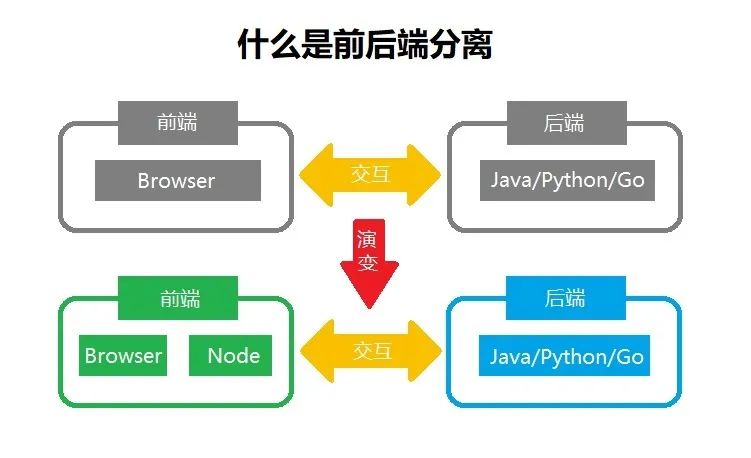
https://mp.weixin.qq.com/s/U4DCOeVKwfXaFmI-Yx\_\_Rg

**前后端分离的接口规范，我们是这样做的！**



## 1. 前言

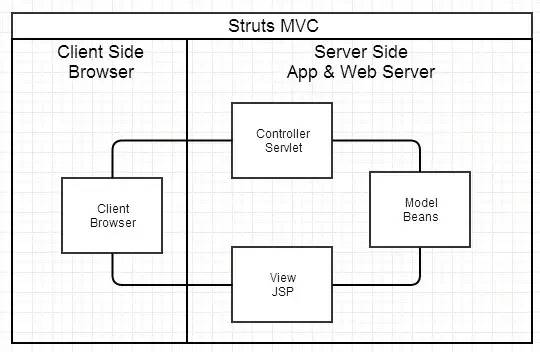
随着互联网的高速发展，前端页面的展示、交互体验越来越灵活、炫丽，响应体验也要求越来越高，后端服务的高并发、高可用、高性能、高扩展等特性的要求也愈加苛刻，从而导致前后端研发各自专注于自己擅长的领域深耕细作。

然而带来的另一个问题：前后端的对接界面双方却关注甚少，没有任何接口约定规范情况下各自撸起袖子就是干，导致我们在产品项目开发过程中，前后端的接口联调对接 工作量占比在30%-50%左右，甚至会更高。往往前后端接口联调对接及系统间的联调对接都是整个产品项目研发的软肋。

本文的主要初衷就是规范约定先行，尽量避免沟通联调产生的不必要的问题，让大家身心愉快地专注于各自擅长的领域。

## 2. 为何要分离

目前现有前后端开发模式：“后端为主的MVC时代”，如下图所示：



后端为主的MVC时代

代码可维护性得到明显好转，MVC 是个非常好的协作模式，从架构层面让开发者懂得什么代码应该写在什么地方。为了让 View 层更简单干脆，还可以选择 Velocity、Freemaker 等模板，使得模板里写不了 Java 代码。看起来是功能变弱了，但正是这种限制使得前后端分工更清晰。然而依旧并不是那么清晰，这个阶段的典型问题是：

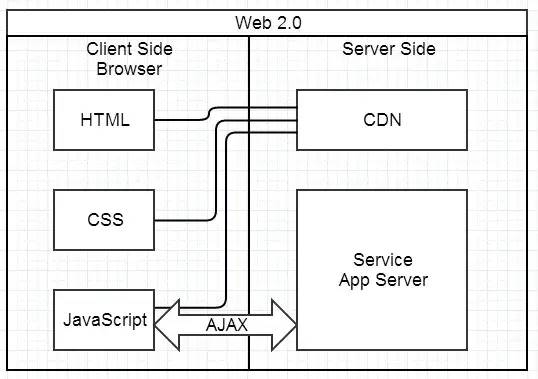
1. **前端开发重度依赖开发环境，开发效率低** 。这种架构下，前后端协作有两种模式：**一种是前端写demo，写好后，让后端去套模板** 。淘宝早期包括现在依旧有大量业务线是这种模式。好处很明显，demo 可以本地开发，很高效。不足是还需要后端套模板，有可能套错，套完后还需要前端确定，来回沟通调整的成本比较大。**另一种协作模式是前端负责浏览器端的所有开发和服务器端的 View 层模板开发，支付宝是这种模式。** 好处是 UI 相关的代码都是前端去写就好，后端不用太关注，不足就是前端开发重度绑定后端环境，环境成为影响前端开发效率的重要因素。
2. **前后端职责依旧纠缠不清** 。Velocity 模板还是蛮强大的，变量、逻辑、宏等特性，依旧可以通过拿到的上下文变量来实现各种业务逻辑。这样，只要前端弱势一点，往往就会被后端要求在模板层写出不少业务代码。**还有一个很大的灰色地带是 Controller，页面路由等功能本应该是前端最关注的，但却是由后端来实现** 。Controller 本身与 Model 往往也会纠缠不清，看了让人咬牙的业务代码经常会出现在 Controller 层。这些问题不能全归结于程序员的素养，否则 JSP 就够了。
3. **对前端发挥的局限** 。性能优化如果只在前端做空间非常有限，于是我们经常需要后端合作才能碰撞出火花，但由于后端框架限制，我们很难使用Comet、Bigpipe等技术方案来优化性能。

**总上所述，就跟為什麼要代碼重構一樣：**

1. 关注点分离
2. 职责分离
3. 对的人做对的事
4. 更好的共建模式
5. 快速的反应变化

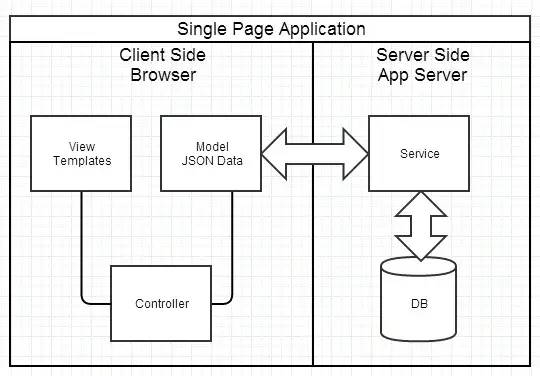
## 3. 什么是分离

我们现在要做的前后分离第一阶段：“基于 Ajax 带来的 SPA 时代”，如图：



基于 Ajax 带来的 SPA 时代

这种模式下，**前后端的分工非常清晰，前后端的关键协作点是 Ajax 接口。** 看起来是如此美妙，但回过头来看看的话，这与 JSP 时代区别不大。复杂度从服务端的 JSP 里移到了浏览器的 JavaScript，浏览器端变得很复杂。类似 Spring MVC，**这个时代开始出现浏览器端的分层架构：**



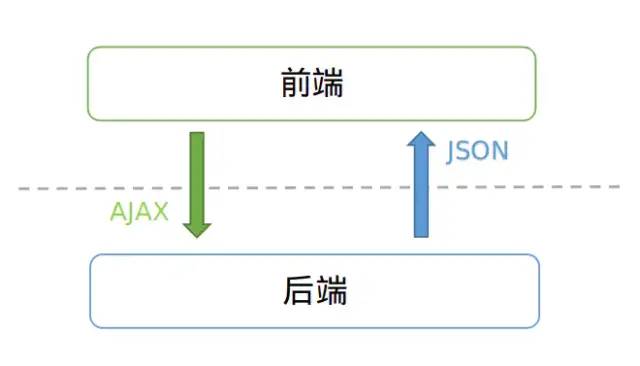
浏览器端的分层架构

对于这一SPA阶段，前后端分离有几个重要挑战：

1. **前后端接口的约定。** 如果后端的接口一塌糊涂，如果后端的业务模型不够稳定，那么前端开发会很痛苦。这一块在业界有 API Blueprint 等方案来约定和沉淀接口，==在阿里，不少团队也有类似尝试，通过接口规则、接口平台等方式来做。有了和后端一起沉淀的接口规则，还可以用来模拟数据，使得前后端可以在约定接口后实现高效并行开发。== 相信这一块会越做越好。
2. **前端开发的复杂度控制。** SPA 应用大多以功能交互型为主，JavaScript 代码过十万行很正常。大量 JS 代码的组织，与 View 层的绑定等，都不是容易的事情。典型的解决方案是业界的 Backbone，但 Backbone 做的事还很有限，依旧存在大量空白区域需要挑战。

## 4. 如何做分离

### 4.1 职责分离



职责分离

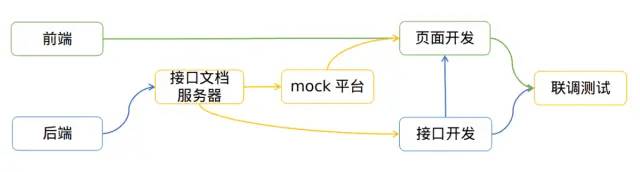
1. 前后端仅仅通过异步接口(AJAX/JSONP)来编程
2. 前后端都各自有自己的开发流程，构建工具，测试集合
3. 关注点分离，前后端变得相对独立并松耦合

| **后端** | **前端** |
| --- | --- |
| 提供数据 | 接收数据，返回数据 |
| 处理业务逻辑 | 处理渲染逻辑 |
| Server-side MVC架构 | Client-side MV\* 架构 |
| 代码跑在服务器上 | 代码跑在浏览器上 |

### 4.2 开发流程

1. 后端编写和维护接口文档，在 API 变化时更新接口文档
2. 后端根据接口文档进行接口开发
3. 前端根据接口文档进行开发 + Mock平台
4. 开发完成后联调和提交测试

**Mock 服务器根据接口文档自动生成 Mock 数据，实现了接口文档即API：**

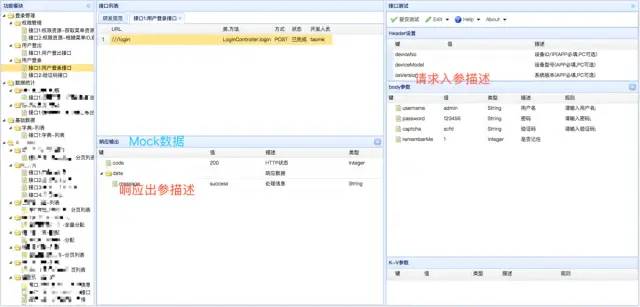


开发流程

### 4.3 具体实施

现在已基本完成了，接口方面的实施：

1. 接口文档服务器：可实现接口变更实时同步给前端展示；
2. Mock接口数据平台：可实现接口变更实时Mock数据给前端使用；
3. 接口规范定义：很重要，接口定义的好坏直接影响到前端的工作量和实现逻辑；具体定义规范见下节；



## 5. 接口规范V1.0.0

### 5.1 规范原则

1. **接口返回数据即显示：前端仅做渲染逻辑处理；**
2. **渲染逻辑禁止跨多个接口调用；**
3. **前端关注交互、渲染逻辑，尽量避免业务逻辑处理的出现；**
4. **请求响应传输数据格式：JSON，JSON数据尽量简单轻量，避免多级JSON的出现；**

### 5.2 基本格式

#### 5.2.1 请求基本格式

GET请求、POST请求==必须包含key为body的入参，所有请求数据包装为JSON格式，并存放到入参body中==，示例如下：

1. **GET请求：**

xxx/login?body={"username":"admin","password":"123456","captcha":"scfd","rememberMe":1}

1. **POST**请求



#### 5.2.2 响应基本格式

{  
  code: 200,  
  data: {  
    message: "success"  
  }  
}

1. code : 请求处理状态

200: 请求处理成功

500: 请求处理失败

401: 请求未认证，跳转登录页

406: 请求未授权，跳转未授权提示页

1. data.message: 请求处理消息

code=200 且 data.message="success": 请求处理成功

code=200 且 data.message!="success": 请求处理成功, 普通消息提示：message内容

code=500: 请求处理失败，警告消息提示：message内容

### 5.3 响应实体格式

{  
  code: 200,  
  data: {  
    message: "success",  
    entity: {  
      id: 1,  
      name: "XXX",  
      code: "XXX"  
    }  
  }  
}

data.entity: 响应返回的实体数据

### 5.4 响应列表格式

{  
  code: 200,  
  data: {  
    message: "success",  
    list: [  
      {  
        id: 1,  
        name: "XXX",  
        code: "XXX"  
      },  
      {  
        id: 2,  
        name: "XXX",  
        code: "XXX"  
      }  
    ]  
  }  
}

data.list: 响应返回的列表数据

### 5.5 响应分页格式

{  
  code: 200,  
  data: {  
    recordCount: 2,  
    message: "success",  
    totalCount: 2,  
    pageNo: 1,  
    pageSize: 10,  
    list: [  
      {  
        id: 1,  
        name: "XXX",  
        code: "H001"  
      },  
      {  
        id: 2,  
        name: "XXX",  
        code: "H001"  
      }  
    ],  
    totalPage: 1  
  }  
}

data.recordCount: 当前页记录数 data.totalCount: 总记录数 data.pageNo: 当前页码 data.pageSize: 每页大小 data.totalPage: 总页数

### 5.6 特殊内容规范

#### 5.6.1 下拉框、复选框、单选框

由后端接口统一逻辑判定是否选中，通过isSelect标示是否选中，示例如下：

{  
  code: 200,  
  data: {  
    message: "success",  
    list: [  
      {  
        id: 1,  
        name: "XXX",  
        code: "XXX",  
        isSelect: 1  
      },  
      {  
        id: 1,  
        name: "XXX",  
        code: "XXX",  
        isSelect: 0  
      }  
    ]  
  }  
}

禁止下拉框、复选框、单选框判定选中逻辑由前端来处理，统一由后端逻辑判定选中返回给前端展示；

#### 5.6.2 Boolean类型

关于Boolean类型，JSON数据传输中一律使用1/0来标示，1为是/True，0为否/False；

#### 5.6.3 日期类型

关于日期类型，JSON数据传输中一律使用字符串，具体日期格式因业务而定；

## 6. 未来的大前端

目前我们现在用的前后端分离模式属于第一阶段，由于使用到的一些技术jquery等，对于一些页面展示、数据渲染还是比较复杂，不能够很好的达到复用。对于前端还是有很大的工作量。

下一阶段可以在前端工程化方面，对技术框架的选择、前端模块化重用方面，可多做考量。也就是要迎来“==前端为主的 MV\* 时代==”。大多数的公司也基本都处于这个分离阶段。

最后阶段就是==Node 带来的全栈时代==，完全有前端来控制页面，URL，Controller，路由等，后端的应用就逐步弱化为真正的数据服务+业务服务，做且仅能做的是提供数据、处理业务逻辑，关注高可用、高并发等。