前后端分离实践有感

2018-01-15 15:54:31 分类：Web开发 (/category/WebDev1024/)

来自：边城的博客 (https://my.oschina.net/jamesfancy/blog/1604237)，作者公众号"边城客栈"，Githubhttps://github.com/jamesfancy (https://github.com/jamesfancy)，微博@边城客栈-JFan

摘要: 前后端分离并不是什么新鲜事，到处都是前后端分离的实践。然而一些历史项目在从一体化 Web 设计转向前后端分离的架构时，仍然不可避免的会遇到各种各样的问题。由于层出不穷的问题，甚至会有团队质疑，一体化好好的，为什么要前后端分离？

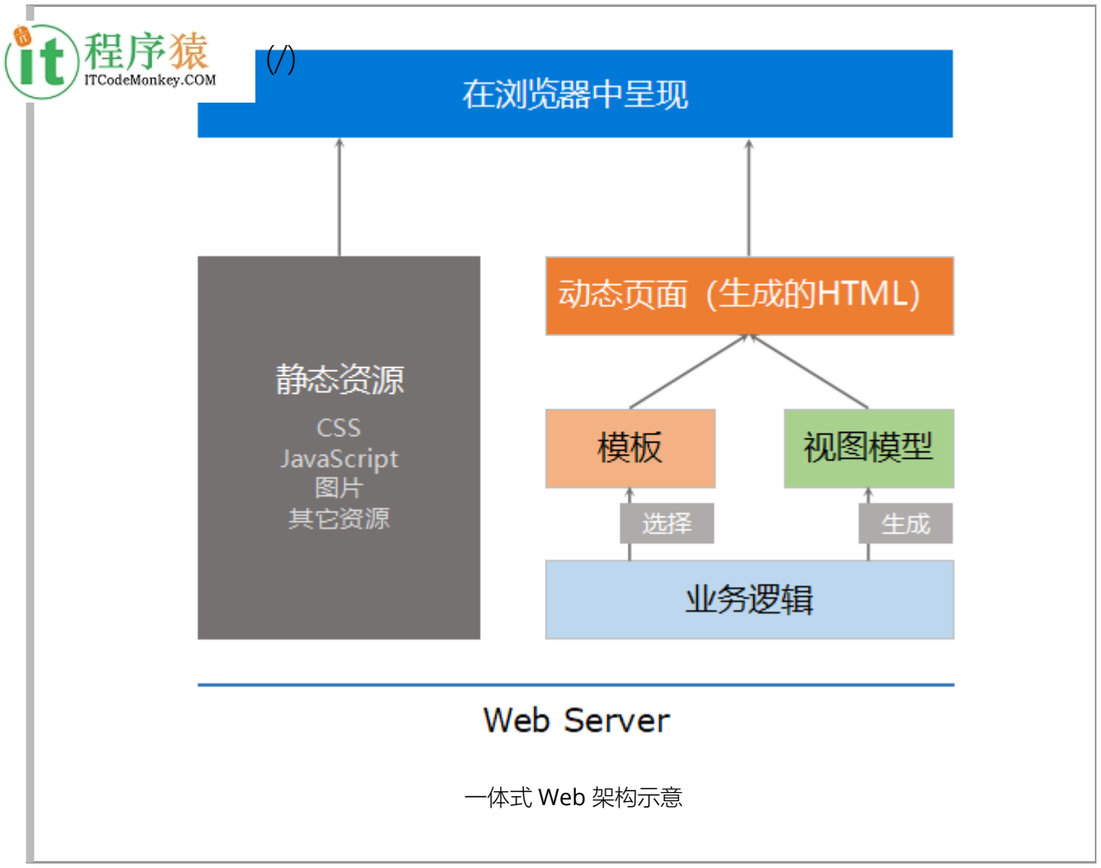
前后端分离实践有感

前后端分离并不是什么新鲜事，到处都是前后端分离的实践。然而一些历史项目在从一体化 Web 设计转向前后端分离的架构时，仍然不可避免的会遇到各种各样的问题。由于层出不穷的问题，甚至会有团队质疑，一体化好好的，为什么要前后端分离？

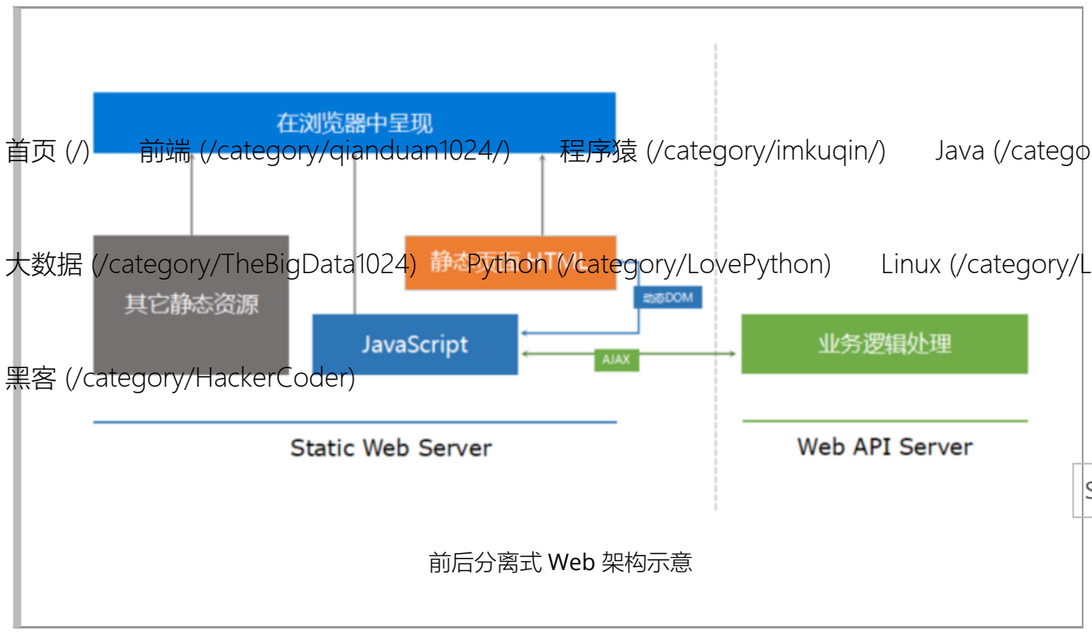
说到底，并不是前后分离不好，只是可能不适合，或者说……设计思维还没有转变过来……

最新文章

1 程序员的创业陷阱：接私...
2 Java命令学习系列（四）...
3 2017年数据库技术盘点 (/a...
4 黑客正在销售合法的代码...
5 阿里P8何召卫：为啥35岁...



一体式 Web 架构示意



前后分离式 Web 架构示意

为什么要前后端分离

首页 (/) 前端 (/category/qianduan1024/) 程序猿 (/category/imkuqin/) Java (/category/JavaCoder1024/)
大数据 (/category/TheBigData1024) Python (/category/LovePython) Linux (/category/LoveLinux1024)
黑客 (/category/HackerCoder)
Search Go

程序猿 ITCodeMonkey.COM

更现实的问题是什么时候需要前后端分离，即前后端分离的应用场景。

(/)

了 2011 年左右，公司在以 .NET 开发团队为主的基础上扩展了 Java 团队，两个团队虽然是在做不同的产品，但是仍然存在大量重复性的开发，比如用 ASP.NET WebPage 写了组织机构相关的页面，用 JSP 又要再写一遍。在这种情况下，团队就开始思考这样一个方案：如果前端实现与后端技术无关，那页面呈现的部分就可以共用，不同的后端技术只需要实现后端业务逻辑就好。

方案根本要解决的问题是把数据和页面剥离开来。应对这种需求的技术是现成的，前端采用静态网页相关的技术，HTML + CSS + JavaScript，通过 AJAX 技术调用后端提供的业务接口。前后端协商好接口方式通过 HTTP 提供，统一使用 POST 谓词。接口数据结构使用 XML 实现，前端 jQuery 解析 XML 很方便，后端对 XML 的处理工具就更多了……后来由于后端 JSON库（比如 Newtonsoft JSON.NET、jackson、Gson 等）崛起，前端处理 JSON 也更容易（`JSON.parse()` 和 `JSON.stringify()`），就将数据结构换成了 JSON 实现。

这种架构从本质上来说就是 SOA（面向服务的架构）。当后端不提供页面，只是纯粹的通过 Web API 来提供数据和业务交互能力之后，Web 前端就成了纯粹的客户端角色，与 WinForm、移动终端应用属于同样的角色，可以把它们合在一起，统称为**前端**。以前的一体化架构需要定制页面来实现 Web 应用，同时又定义一套 WebService/WSDL 来对 WinForm 和移动终端提供服务。转换为新的架构之后，可以统一使用 Web API 形式为所有类型的前端提供服务。至于某些类型的前端对这个 Web API 进行的 RPC 封装，那又是另外一回事了。

通过这样的架构改造，前后端实际就已经分离开了。抛开其它类型的前端不提，这里只讨论 Web 前端和后端。由于分离，Web 前端在开发的时候压根不需要了解后端是用的什么技术，只需要后端提供了什么样的接口可以用来做什么事情就好，什么 C#/ASP.NET、Java/JEE、数据库……这些技术可以统统不去了解。而后端的 .NET 团队和 Java 团队也脱离了逻辑无关的美学思维，不需要面对美工精细的界面设计约束，也不需要在思考逻辑实现的同时还要去考虑页面上怎么布局的问题，只需要处理自己擅长的逻辑和数据就好。

前后端分离之后，两端的开发人员都轻松不少，由于技术和业务都更专注，开发效率也提高了。分离带来的好处渐渐体现出来：

1、前后职责分离

前端倾向于呈现，着重处理用户体验相关的问题；后端则倾处于业务逻辑、数据处理和持久化等。在设计清晰的情况下，后端只需要以数据为中心对业务处理算法负责，并按约定为前端提供 API 接口；而前端使用这些接口对用户体验负责。

2、前后技术分离

程序猿 ITCodeMonkey.COM

技术，也不关心后端具体用什么技术来实现，只需要会 HTML/CSS/JavaScript 就能(/)心后端开发技术，除了省去学习前端技术的麻烦，连 Web 框架的学习研究都只需要关注 Web API 就好，而不用去关注基于页面视图的 MVC 技术（并不是说不需要 MVC，Web API 的接口部分的数据结构呈现也是 View），不用考虑特别复杂的数据组织和呈现。

3、前后分离带来了用户用户体验和业务处理解耦

前端可以根据用户不同时期的体验需求迅速改版，后端对此毫无压力。同理，后端进行的业务逻辑升级，数据持久方案变更，只要不影响到接口，前端可以毫不知情。当然如果需求变更引起接口变化的时候，前后端又需要坐在一起同步信息了。

4、前后分离，可以分别归约两端的设计

后端只提供 API 服务，不考虑页面呈现的问题。实现 SOA 架构的 API 可以服务于各种前端，而不仅仅是 Web 前端，可以做到一套服务，各端使用；同时对于前端来说，不依赖后端技术的前端部分可以独立部署，也可以应于 Hybrid 架构，嵌入各种“壳”（比如 Electron、Codorva 等），迅速实现多终端。

前后分离架构

任何技术方案都不是银弹，前后分离不仅带来好处，也带来矛盾。我们在实践初期，由于前端团队力量相对薄弱，同时按照惯例，所有业务处理几乎都是由后端（原来的技术骨干）来设计和定义的，前端处理过程中常常发现接口定义不符合用户操作流程，AJAX 异步请求过多等问题。毕竟后端思维和前端思维还是有所不同——前端思维倾向于用户体验，而后端思维则更倾向于业务的技术实现。

除此之外，前后分离在安全性上的要求也略有不同。由于前后分离本质上是一种 SOA 架构，所以在授权上也需要按 SOA 架构的方式来思考。Cookie/Session 的方式虽然可用，但并不是特别合适，相对来说，基于 Token 的认证则更适合一些。采用基于 Token 的认证就意味着后端的认证部分需要重写……后端当然不想重写，于是会将皮球踢给前端来让前端想办法实现基于 Cookie/Session 的认证……于是前端开始报怨（悲剧）……

谁来主导

这些矛盾的出现，归根结底在于设计不够清晰明确。毫无疑问，在开发过程中，主导者应该是架构师或者设计师。然而实际场景中，架构师或者设计师往往也是开发人员，所以他们的主要技术栈会极大的影响前后端在整个项目中的主次作用。这位骨干处于哪端，开发的便捷性就会向哪端倾斜。这是一个不好的现象，但是我们不得不面对这样的现状，我相信很多不太大的团队也面临着类似的问题。

如果没有良好的流程规范，通常前端接触的到角色会比后端更多（多数应用型项目/产品，并非所有情况）。

程序猿 ITCodeMonkey.COM

- 受到项目/产品经理或客户的直接影响：这个地方应该放个按钮，那个操作应该这(/)
- 前端还要与美工对接——这样的设计不好实现，是否可以改成那样？客户要求必须这么操作，但是这个设计做不到；
- 前端还要跟后端对接，对于某些应用，甚至是多个后端

换句话说，前端可以成为项目沟通的中心，所以比后端更合适承担主导的角色。

接口设计

接口分后端服务实现和前端调用两个部分，技术都是成熟技术，并不难，接口设计才是难点。前面提到前后端会产生一些矛盾。从前端的角度来看，重点关注的是用户体验，包括用户在进行业务操作时的流动方向和相关处理；而从后端的角度来看，重点关注的是数据完整、有效、安全。矛盾在于双方关注点不同，信息不对称，还各有私心。解决这些矛盾的着眼点就是接口设计。

接口设计时，其粒度的大小往往代表了前后端工作量的大小（非绝对，这和整体架构有关）。接口粒度太小，前端要处理的事情就多，尤其是对各种异步处理就可能会感到应接不暇；粒度太大，就会出现高耦合，降低灵活性和扩展性，当然这种情况下后端的工作就轻松不了。业务层面的东西涉及到具体的产品，这里不多做讨论。这里主要讨论一点点技术层面的东西。

就形式上来说，Web API 可以定义成 REST，也可以是 RPC，只要前后端商议确定下来就行。更重要的是在输入参数和输出结果上，最好一开始就有相对固定的定义，这往往取决于前端架构或采用的 UI 框架。

常见请求参数的数据形式有如下一些：

- 键值对，用于 URL 中的 QueryString 或者 POST 等方法的 Payload
- XML/JSON/...，通常用于 POST 等方法的 Payload，也可以使用 multipart 传递
- ROUTE，由后端路由解析 URL 取得，在 RESTful 中常用

而服务器响应的数据形式就五花八门各式各样了，通常一个完整的响应至少需要包含状态码、消息、数据三个部分的内容，其中

- 状态码，HTTP 状态码或响应数据中特定的状态属性
- 消息，通常是放在响应内容中，作为数据的一部分
- 数据，根据接口协议，可能是各种格式，当前最流行的是 JSON

我们在实践中使用 JSON 形式，最初定义了这样一种形式

程序猿 ITCodeMonkey.COM

```
    "number",
    "string",
    "any"
}
```

`code` 主要用于指导前端进行一些特殊的操作，比如 `0` 表示 API 调用成功，`非0` 表示调用失败，其中 `1` 表示需要登录、`2` 表示未获取授权……对于这个定义，前端拿到响应之后，就可以在应用框架层进行一些常规处理，比如当 `code` 为 `1` 的时候，弹出登录窗口请用户在当前页面登录，而当 `code` 为 `2` 的时候，则弹出消息提示并后附链接引导用户获取授权。

> 参阅：前后分离模型之封装 Api 调用 (https://segmentfault.com/a/1190000012040777)

一开始这样做并没有什么问题，直到前端框架换用了 jQuery EasyUI。以 EasyUI 为例的好多 UI 库都支持为组件配置数据 URL，它会自动通过 AJAX 来获取数据，但对数据结构有要求。如果仍然采用之前设计的响应结构，就需要为组件定义数据过滤器（filter）来处理响应结果，这样做写 filter 以及为组件声明 filter 的工作量也是不小的。为了减少这部分工作量我们决定改一改接口。

新的接口是一种可变结构，正常情况下返回 UI 需要的数据结构，出错的情况则响应一个类型于原定结构的数据结构：

```
{
    "error": {
        "identity": "special identity string",
        "code": "number",
        "message": "string",
        "data": "any"
    }
}
```

对于新响应数据结构，前端框架只需要判断一下是否存在 `error` 属性，如果存在，检查其 `identity` 属性是否为指定的特殊值（比如某个特定的 GUID），然后再使用其 `code` 和 `message` 属性处理错误。这个错误判断过程略为复杂一些，但可以由前端应用框架统一处理。

如果使用 RESTful 风格的接口，部分状态码可以用 HTTP 状态码代替，比如 401 表示需要登录，403 就可以表示没有获得授权，500 表示程序处理过程中发生错误。当然，虽然 HTTP 状态码与 RESTful 风格更配，但是非 RESTful 风格也可以使用 HTTP 状态码来代替 `error.code`。

用户认证

认证方案很多，比如 Cookie/Session 在某些环境下仍然可行、也可以使用基于 Token 和 OAuth 或者 JWT，甚至是自己实现基于 Token 的认证方式。

程序猿 ITCodeMonkey.COM

基于 Cookie/Session 的认证方案

(/)ssion 认证方案并非不可行，只不过有一些限制。如果前端部分和后端部分同源，比如页面发布在 `http://domain.name/`，而 Web API 发布在 `http://domain.name/api/`，这种情况下，原来的一体式 Web 方案所采用的 Cookie/Session 方案可以直接迁移过来，毫无压力。但是如果前面发布和 API 发布不同源，这种方法处理起来就复杂了。

然后一般前后端分离的开发方式，不管是开发阶段还是发布阶段，不同源的可能性占绝大比例，所以认证方案通常会使用与 Cookie 无关的方案。

基于 OAuth 的认证方案

目前各大网站的开放式接口都是 SOA 架构，如果把这些开放式接口看作提供服务方（服务端），而把使用这些开放式接口的应用看作客户端，那么就可以产生这样一种和前后分离对应的关系：

```
前端 ⇌ 客户端
    ⇣
  （基于 OAuth 的认证）
    ⇡
后端 ⇌ 服务端
```

所以，开放式接口广泛使用的 OAuth 方案用于前后分离是可行的，但在具体实施上却并不是那么容易。尤其是在安全性上，由于前端是完全暴露在外的，与 OAuth 通常实施的环境（后端⇌服务端）相比，要注意的是首次认证不是使用已注册的 AppID 和 AppToken，而是使用用户名和密码。

基于 Token/JWT 的认证方案

虽然这个方案放在最后，但这个方案却是目前前后端分离最适合的方案。基于 Token 的认证方案，各种讨论由来已久，而 JWT 是相对较为成熟，也得到多数人认可的一种。从 jwt.io (https://jwt.io/) 上可以找到各种技术栈的 JWT 实现，应用起来也比较方便。

话虽如此，JWT 方案和以前使用的 Cookie/Session 在处理上还是有较大的差别，需要一定的学习成本。有人担心 JWT 的数据量太大。这确实是一个问题，但是硬件并不贵，4G 也开始进入不限流量阶段，一般应用中不用太在意这个问题。

前后分离的测试

前后分离之后，前端的测试将以用户体验测试和集成测试为主，而后端则主要是进行单元测试和 Web API 接口测试。与一体化的 Web 应用相比，多了一层接口测试，这一层测试可以完全自动化，一旦完成测试开发，就能在很大程度上控制住业务处理和数据错误。这样一来，集成测试的工作量会相对单一也容易得多。

前端测试的工作相对来说减轻不了多少，前后分离之后的前端部分承担了原来的集成测试工作。但是在假设 Web API 正确的情况下进行集成测试，工作量是可以减轻不少的，用例可以只关注前端体验性的问题，比如呈现是否正确，跳转是否正确，用户的操作步骤是否符合要求以及提示信息是否准确等等。

程序猿 ITCodeMonkey.COM

证这部分工作在项目时间紧迫的情况下甚至都可以完全抛给 Web API 去处理。不管开发中都有一个共识：永远不要相信前端！既然后端必须保证数据的安全性和有效性，那么前端省略这一步骤并不会对后端造成什么实质性的威胁，最多只是用户体验差一点。但是，如果前后端都要做数据有效性验证，那一定要严格按照文档来进行，不然很容易出现前后端数据验证不一致的情况（这不是前后分离的问题，一体化架构同样存在这个问题）。

小结

总的来说，前后分离所带来的好处还是很明显的。但是具体实施的时候需要一个全新的思考方式，而不是基于原有一体化 Web 开发方式来进行思考。前后分离的开放方式将开发人员从复杂的技术组合中解放出来，大家都可以更专注于自己擅长的领域来进行开发，但同时也对前后端团队的沟通交流提出了更高的要求，前后端团队必须要一同设计出相对稳定的 Web API 接口（这部分工作其实不管是否前后端分离都是少不了的，只是前后分离的架构对此要求更高，更明确地要求接口不只存在于人的记忆中，更要文档化、持久化）。

推荐↓↓↓

Web开发

上一篇： 前端性能优化 —— 移动端浏览器优化策略 (/article/1915.html)

下一篇：3分钟读懂区块链 (/article/1924.html)

© 2017-2018 IT程序猿 闽ICP备08108865号-1 合作联系：QQ：616316321，微信：itcodemonkey