

前后端分离实践@淘宝

清羽

我



- 清羽（周骞）
- 淘宝UED / FEM
- <http://weibo.com/carffuca/>
- qingyu@taobao.com

1. 引子

前后端分离

实践@淘宝

为什么分离



曾经.....

- 我是后端，套页面套页面.....这么没技术含量的工作不知道是谁想出来的。眼花缭乱的HTML结构真让人痛苦。



曾经.....

- 我是前端，我希望尝试一下bigpipe，页面体验性能的提升会让我KPI更好吧：) 但我们的后端是Java的，我不懂，他们又懒得去管这个.....



曾经.....

- 我是测试，这个Bug提给前端还是后端呢？看上去是展现错误，但也可能是逻辑本身的问题。哎，不管了，先提给最好说话的那个，让他去定位一下吧。



我们希望.....

关注点分离

职责分离

对的人做对的事情

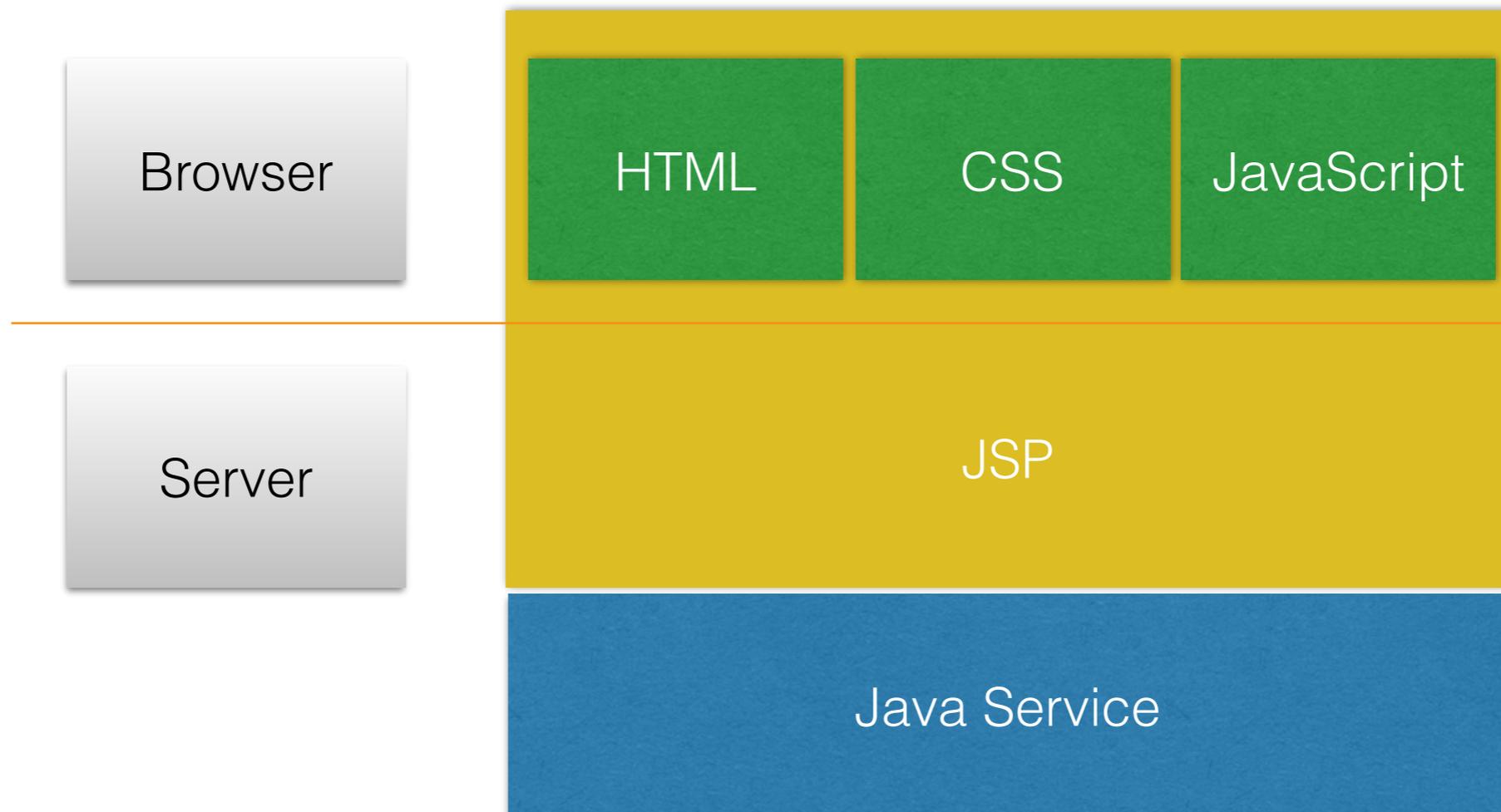
更好的共建模式

快速的响应变化

2. 历史

我们如何走到这里？

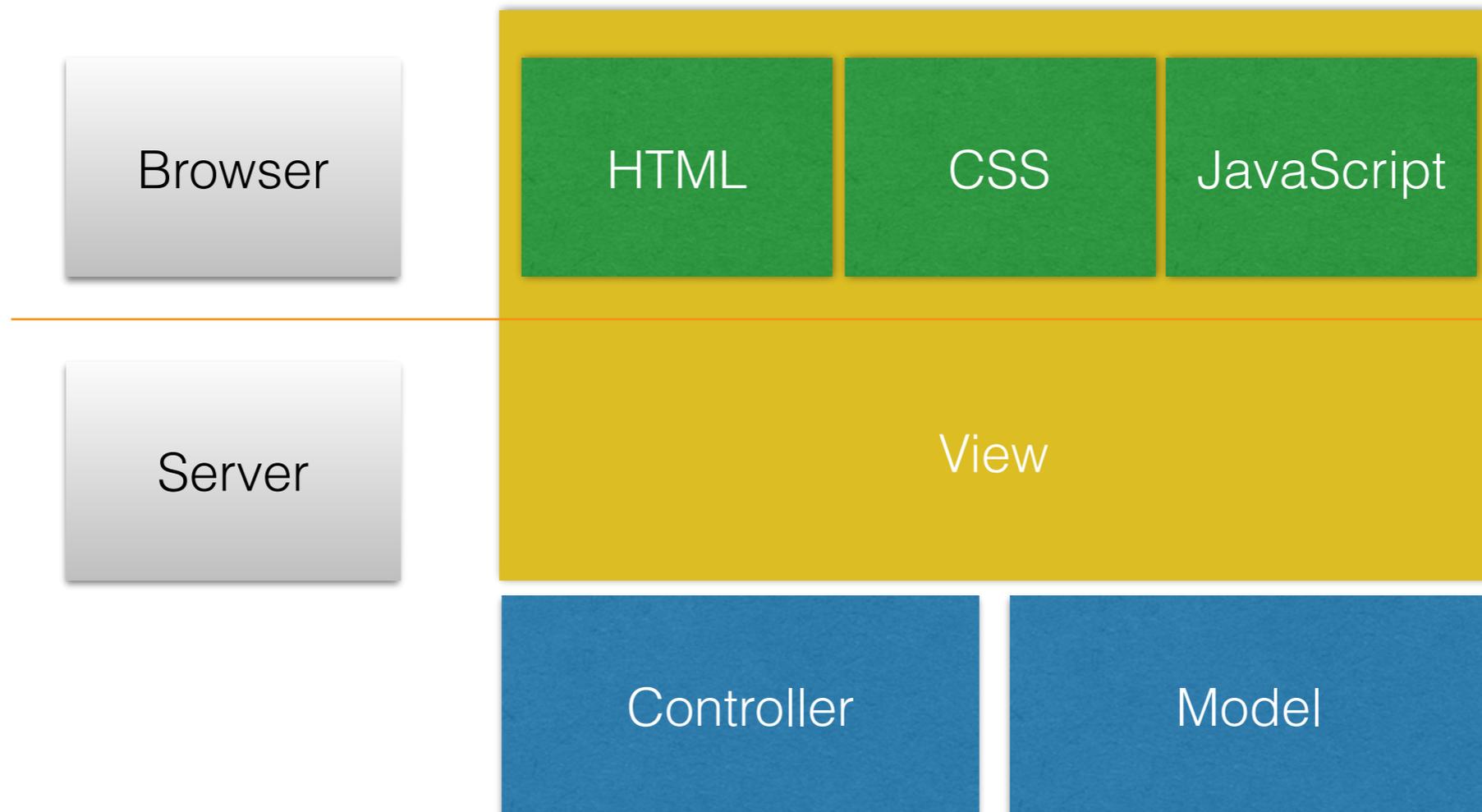
Web开发混沌时代



Web开发混沌时代

- 前端？ 后端？ 请叫我程序员。
- 展现逻辑与业务逻辑混杂，耦合度非常高。

后端MVC时代



后端MVC时代

- 前后端的合作主要靠**约定**。
- 展现逻辑？业务逻辑？在页面，在Model层，更多在**View层**。
- 前端后端在工作在服务端View层**高度耦合**。

纠结的View

- 前端写Demo，后端套页面。
- 后端需要写HTML，而前端需要Review结果是否正确。
- 前端写View层，后端只管数据。
- 前端需要熟悉后端模版语言、了解后端架构、搭建后端环境。

后端MVC时代

沟通成本上升

代码渐渐走向腐烂

维护成本上升

无法正确快速的响应变化

3. 转折

我们还是**前后端分离**
吧！

分离

前端

展现逻辑

页面渲染

后端

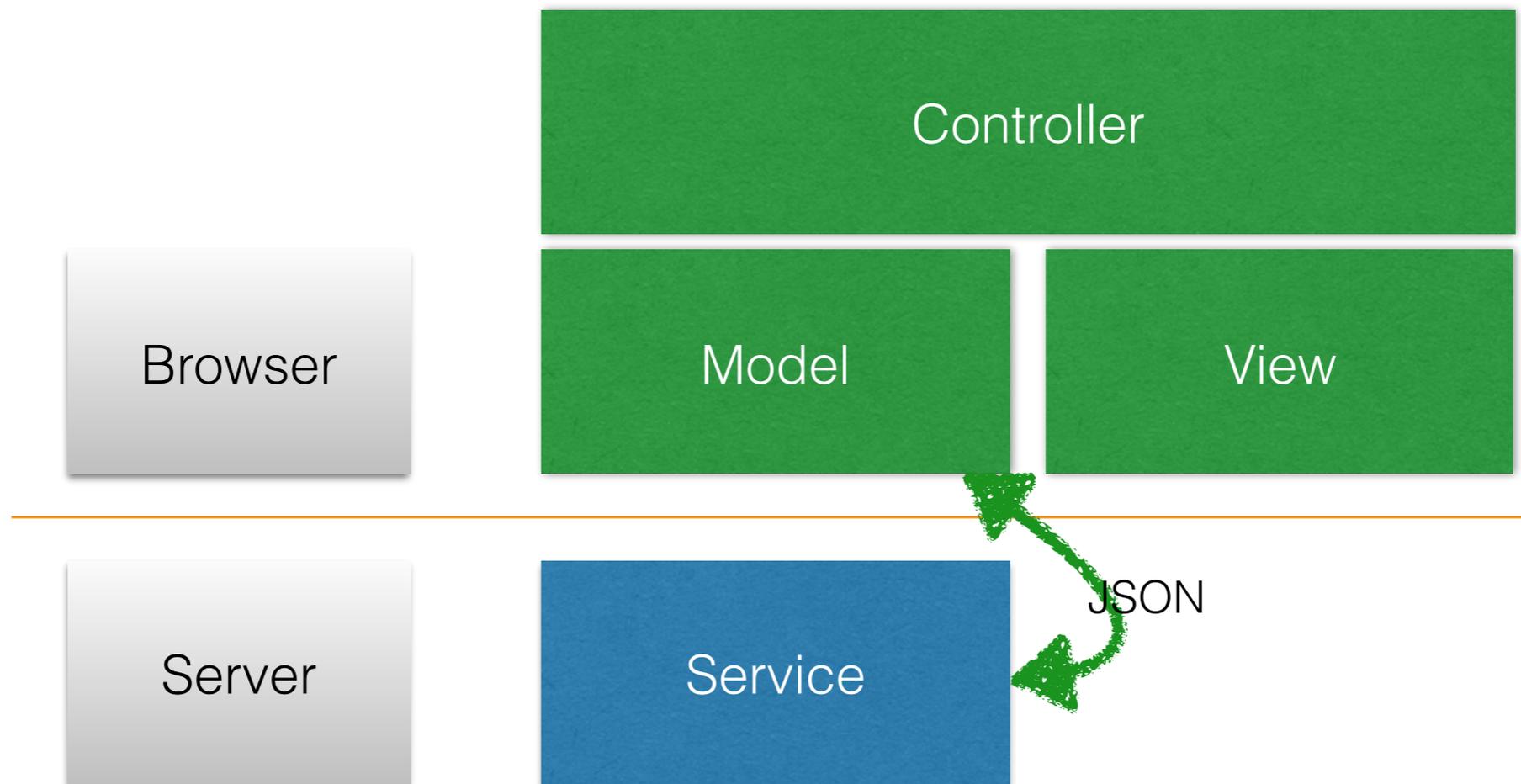
业务逻辑

提供数据

前后端分离

从AJAX到SPA到前端MV*

前后端分离



Backbone, EmberJS, KnockoutJS, AngularJS , React, etc.

前后端分离

后端	前端
提供数据，处理数据	接受数据，返回数据
处理业务逻辑	处理展现逻辑
Service-side MVC	Client-side MV*
代码跑在服务器上	代码跑在浏览器上

前后端分离

- 前后端分工明确，工作的耦合度较低。
- 以Browser、Server物理拆封成两块，开发部署相对独立。

前后端分离

- “我们的 Web APP 首次打开白屏时间很长，能不能优化一下性能？”
- 《Optimizing Single Page Apps for SEO》
- 代码重用问题
-

前后端分离

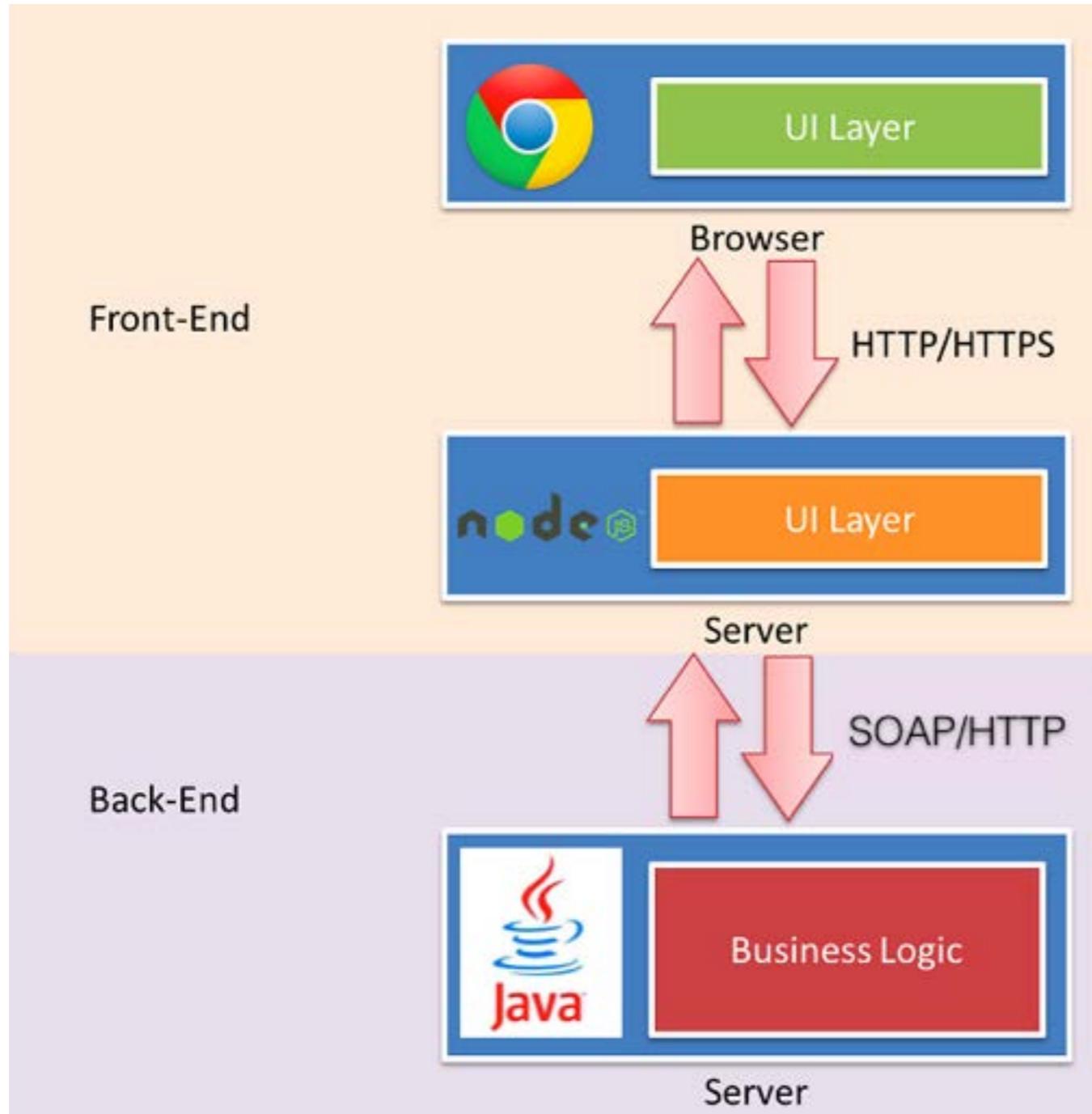


<https://www.flickr.com/photos/dmjames58/8964986373>

4. 主题

基于Node.js的前后端分离

基于Node.js的前后端分离



<http://www.nczonline.net/blog/2013/10/07/node-js-and-the-new-web-front-end/>

重新定义前后端

按照工作职责划分还是物理环境
来划分？

Why Node.js?

- 现有前端知识体系的补充，**学习成本低**。
- 在模版和部分逻辑上可以做到**浏览器服务器共用**。
- 事件驱动、非阻塞I/O，适合**IO密集型业务**。
- 执行**速度**也不差。

业界同行

- Node version of the app doubled the number of requests/second.
- Response time dropped 35% or 200 milliseconds.
- From 1 to 12 Node apps in six months.



业界同行

- Easily serving 50,000 requests/minutes (this was 6 months ago and before rolling out to 48 countries).
- Node services are now handling the same amount of traffic as before but with less hardware.
- Page load times decreased by a whopping 50%.



业界同行

- On average, Node services handle around 1,680,000 - 2,000,000 requests/minute.
- Yahoo has around 200 developers writing Node code.

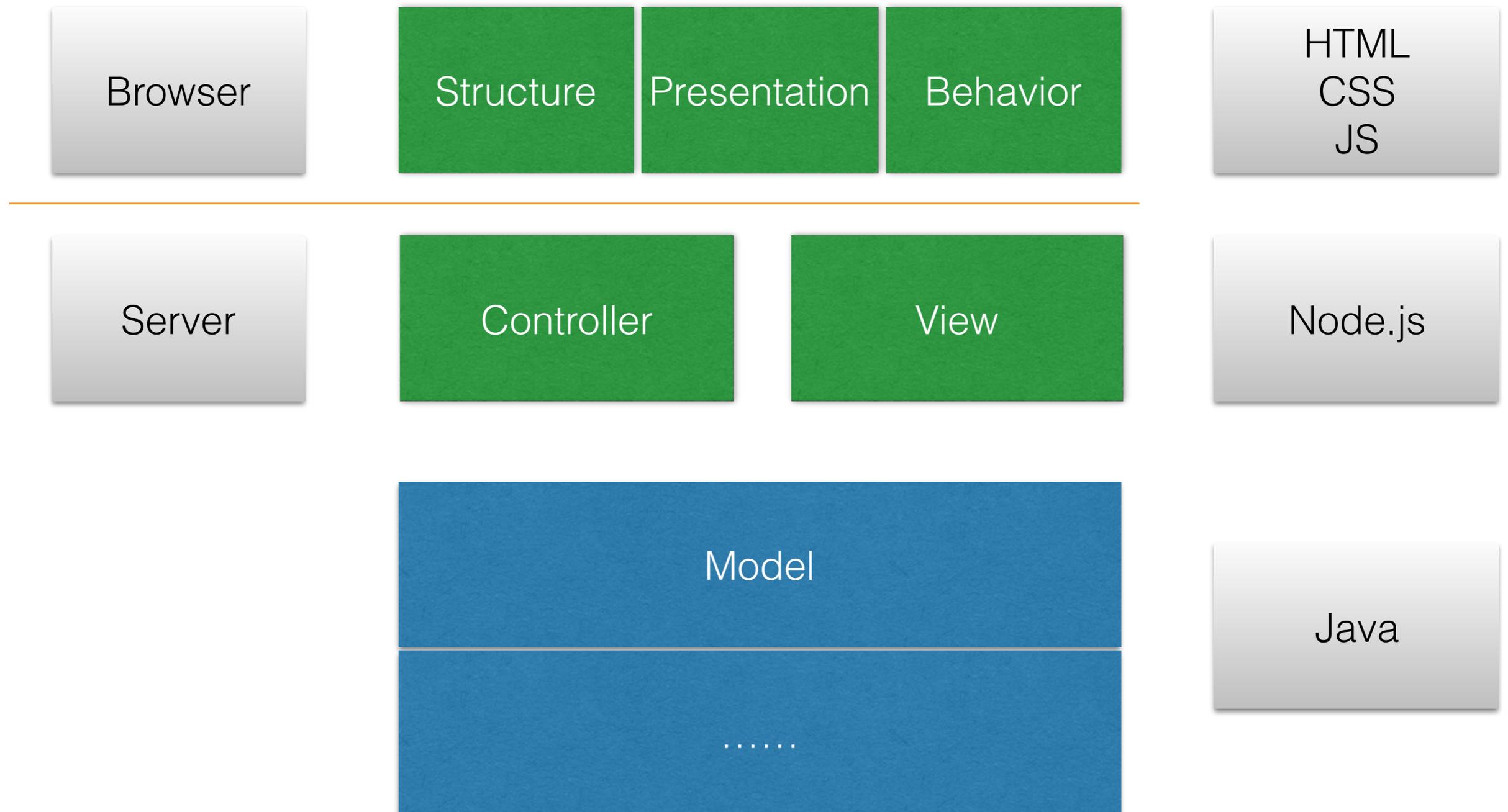
The image shows the classic Yahoo! logo in a bold, blue, sans-serif font. The word "YAHOO!" is written in all caps, with an exclamation point at the end.

回到主题

Node替换Java VS Node与Java协同

Node全栈工程师 VS 前端工程师 + 后端工程师

基于Node.js的前后端分离



基于Node.js的前后端分离

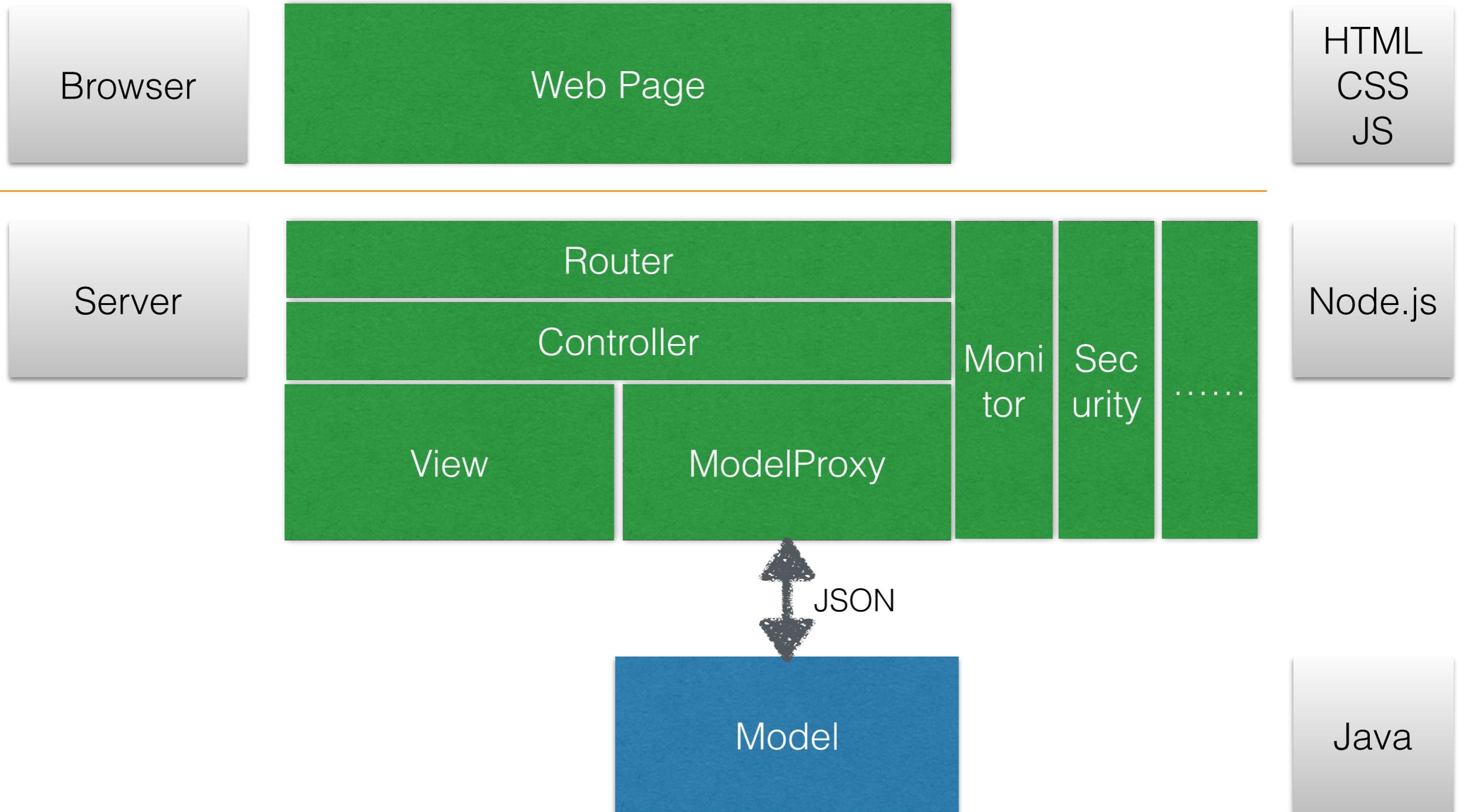
后端	前端
服务器	浏览器
JAVA	HTML+JS+CSS
提供数据接口	浏览器端JS
封装业务逻辑	页面样式
保障服务稳定	交互体验
	Node Server端JS 转发数据，合并数据 渲染页面，输出控制 路由设计，控制逻辑

5. 实践

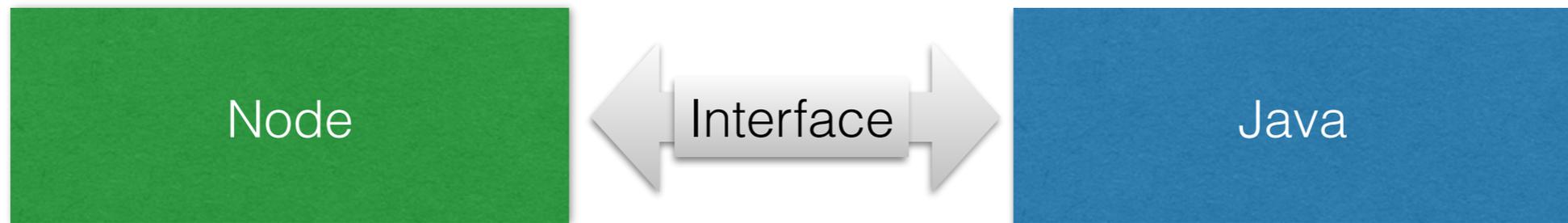
一切都是这么顺利？



实践

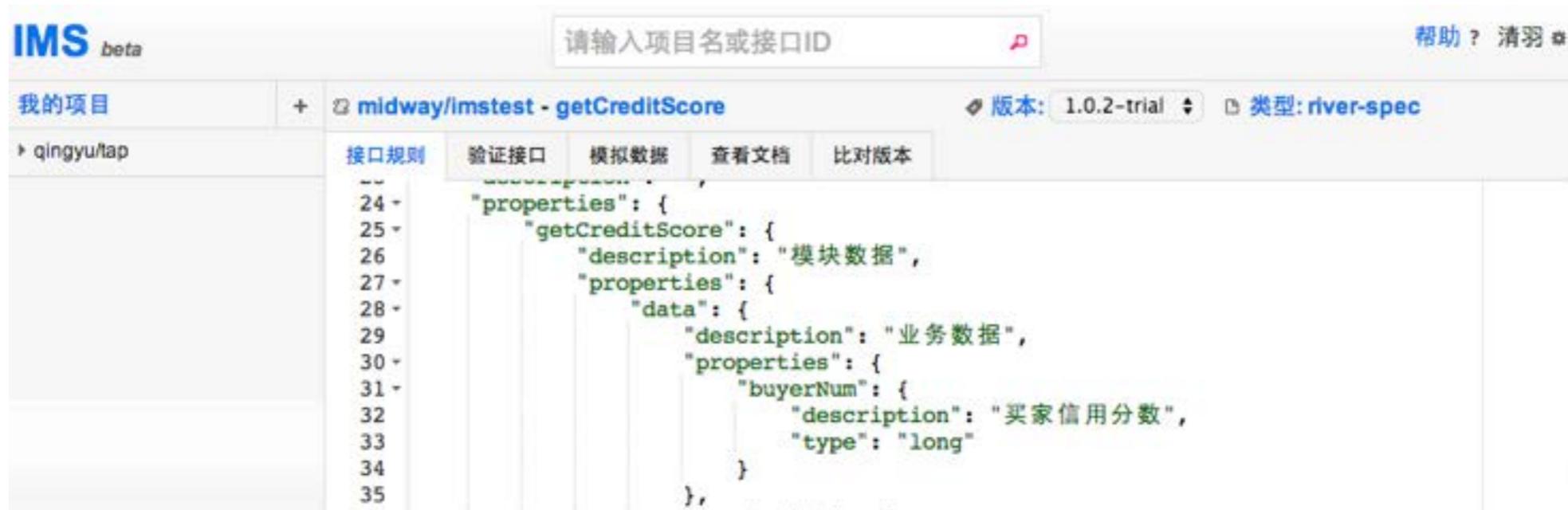


接口



- 基于JSON Schema规范
- 增强 请求 与 响应的条件描述 (River-spec)
- 扩展format, 增加业务定义
- 接口先行

接口



- **版本管理**: 比较、合并、历史记录
- **接口验证**: 数据格式校验、回归测试
- **数据模拟**: 并行开发、边界条件测试

接口

- “使用前后端接口先行，让bug量减少、bug定位修复时间**大大缩短**。”——项目开发
- 使用HTTP本地调用，**RT < 15ms**，是可接受的范围。

部署

- 同集群部署：Java、Node.js 的进程都跑在一台机器上。
- 分集群部署：Java 一个集群，Node.js 一个集群。

部署

	同机部署 3* (Node+Java)	分机部署 1*Node+2*Java
QPS	360 (3* 120)	413
RT	125ms	193ms
load	3.8 (3~4.5)	Node:3.8(3~4.5) Java:3.6(3~4)

分机部署QPS提升约15%，而RT时间却增加约50%。

部署



同集群部署

分集群部署

发布时打成一个包，形成强约束，方便一起发布，有问题时一并回滚

发布与回滚时需要负载均衡入口来精确控制机器上线状态，分别操作两个集群

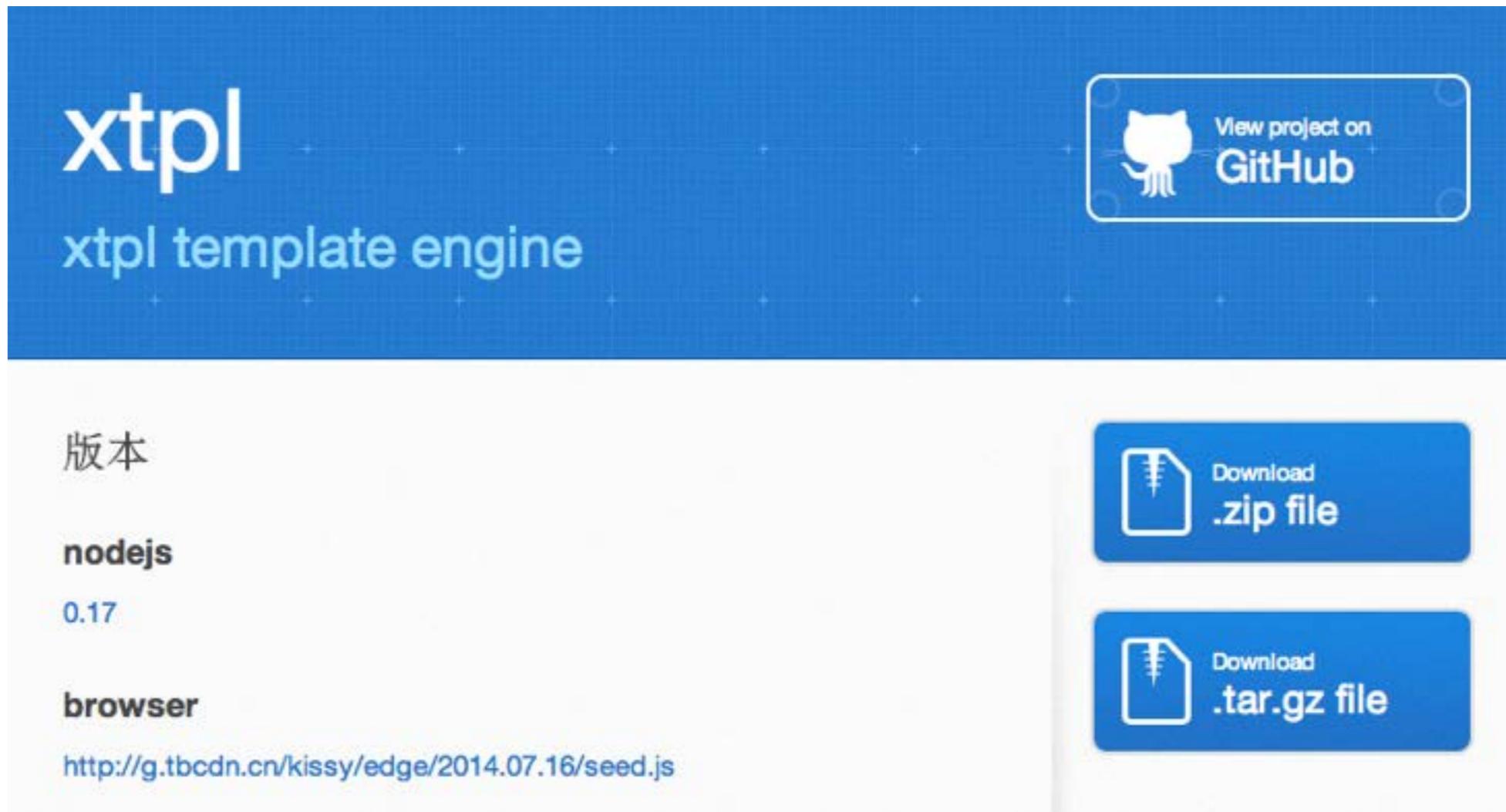
通讯在本地进行，网络性能优于经过集群之间的内网交换机

通讯在集群之间发生，有轻微的网络通讯成本

Node.js 与 Java 的配比不能做到一对多或者多对一

Node.js 与 Java 的配比可以做到多对一或者一对多

模版



The screenshot shows the xtpl website interface. At the top left, the logo 'xtpl' is displayed in white on a blue background, with the text 'xtpl template engine' below it. To the right, there is a button with the GitHub logo and the text 'View project on GitHub'. Below the blue header, the page is divided into two columns. The left column contains the text '版本' (Version), 'nodejs', '0.17', 'browser', and a URL 'http://g.tbcdn.cn/kissy/edge/2014.07.16/seed.js'. The right column contains two blue buttons: 'Download .zip file' and 'Download .tar.gz file', each with a document icon.

xtpl
xtpl template engine

View project on
GitHub

版本

nodejs
0.17

browser

<http://g.tbcdn.cn/kissy/edge/2014.07.16/seed.js>

Download
.zip file

Download
.tar.gz file

Powered By 

模版



xtpl

133,811 ops/sec \pm 3.11%

dust

166,525 ops/sec \pm 2.55%

jade

32,796 ops/sec \pm 2.84%

nunjucks

80,385 ops/sec \pm 2.39%

ejs

65,900 ops/sec \pm 2.03%

XTPL: <http://kissyteam.github.io/xtpl/>

模版

CMS调用

国际化

数据埋点

安全处理

XTPL

框架化

[首页](#) [快速上手](#) [API](#) [案例](#) [常见问题](#) [联系我们](#)

Fork me on GitLab

MIDWAY 1.2.9

基于 Node.js 的 前后端分离 解决方案

两分钟搭建Midway应用



高效

前后端分离的开发模式，让前端和后端各司其职，简化开发流程，大幅提升开发效率。



稳定

完善的异常处理机制以及多进程架构，结合AliMonitor平台，让应用更稳定。

框架化

```
'use strict';  
  
route.get('/', HomeController.index);  
  
module.exports = route;
```

```
{{extend ("./layout/default-layout") }}  
  
{{#block ("content") }}  
    {{include ("./page/home-page") }}  
{{/block}}
```

```
'use strict';  
  
var ModelProxy = require('midway').getPlugin('dataproxy'),  
    HomeData = new ModelProxy({  
    fetch: 'Midway.Home'  
});  
  
exports.index = function(req, res, next) {  
  
    HomeData.fetch()  
        .done(function(data) {  
            res.render('home', data);  
        })  
        .fail(function(err) {  
            next(err);  
        });  
  
};
```

反馈

- “模版由前端自己写，自己维护，能更好的把控展现层的**代码质量**。”——项目前端
- “终于在项目中实施了一次bigpipe，基于Node.js**简单**多了，项目性能和体验真不错。”——项目前端

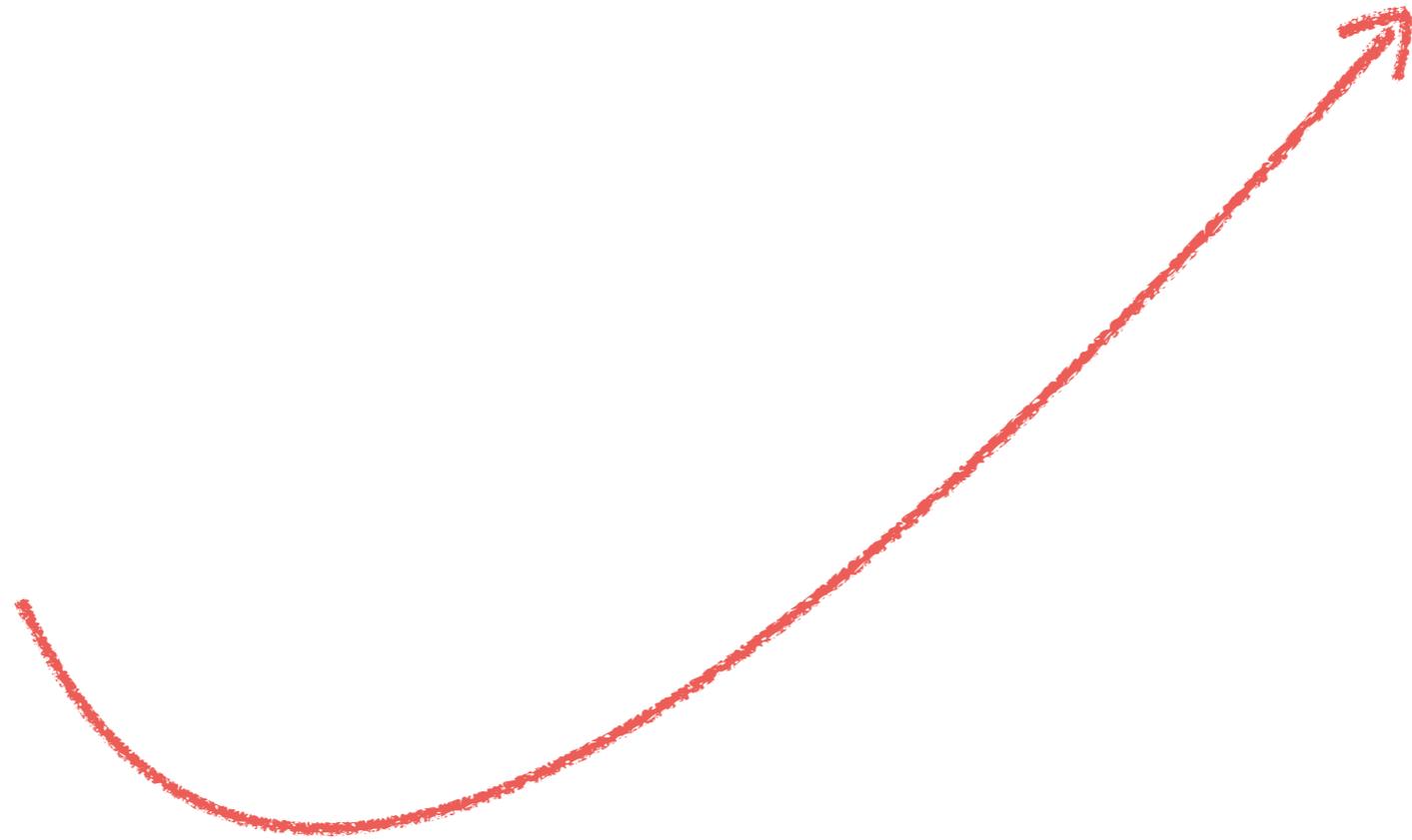
反馈

- “Midway让我们的产品RT时间降低了30%！”——项目开发
- “前端的工作增加了20% - 30%，但后端的工作减少了30%。Bug变少，定位和修复的效率显著提升。”——项目PM
- “Midway让我觉得自己更像程序员了……”——项目前端

挑战

- 前端需要开始关注服务器端的运维工作。
- 接口先行，需要开发、前端、测试具备这样的意识，并纳入开发流程。
- 短期来看前端工作量增加。

挑战



效率真的提升？

挑战

不仅是提升效率，

更重要的是**解放生产力！**

解放生产力

- 案例一：在一个偏静态的门户页面
 - 替换掉原来的PHP环境，QPS提升约8倍。
 - 产出静态页面同步到CDN，并定时刷新。

解放生产力

- 案例二：一个PV量大且多异步请求的页面
 - 在Node.js端实用Bigpipe，分批输出。
 - 拆分Java大接口为独立小接口，并发请求，从串行到并行，缩短请求时间。
 - 首屏服务端渲染，次屏浏览器渲染，前后端部分共享模版。

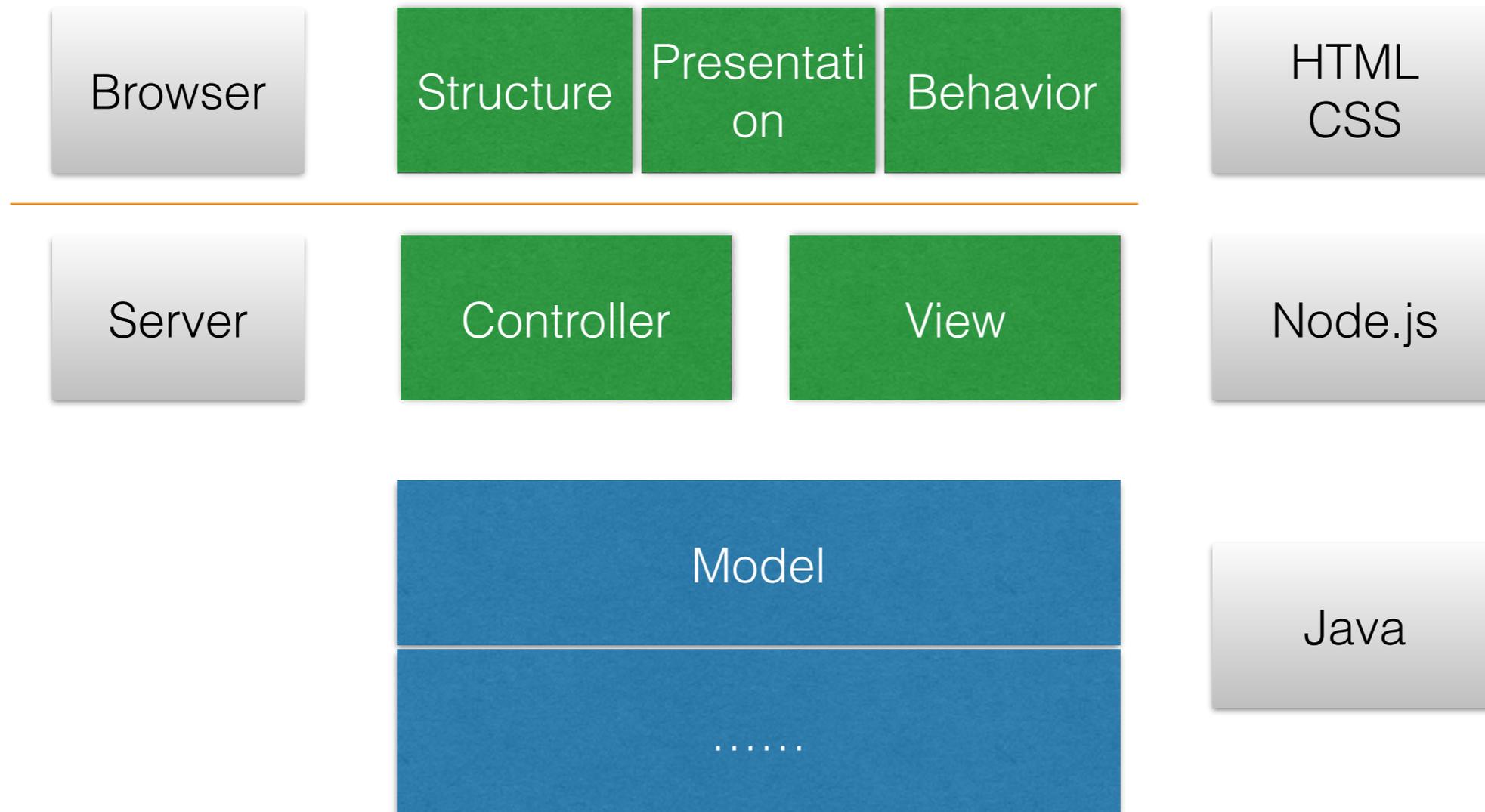
解放生产力

- 多终端输出方案
- 页面自动化测试方案
- JS Convert: coffee script, dart, etc.
- ES6 Compiler: Traceur-Compiler, JSDC, etc.
- CSS Compiler: sass, less, stylus, etc.

解放生产力

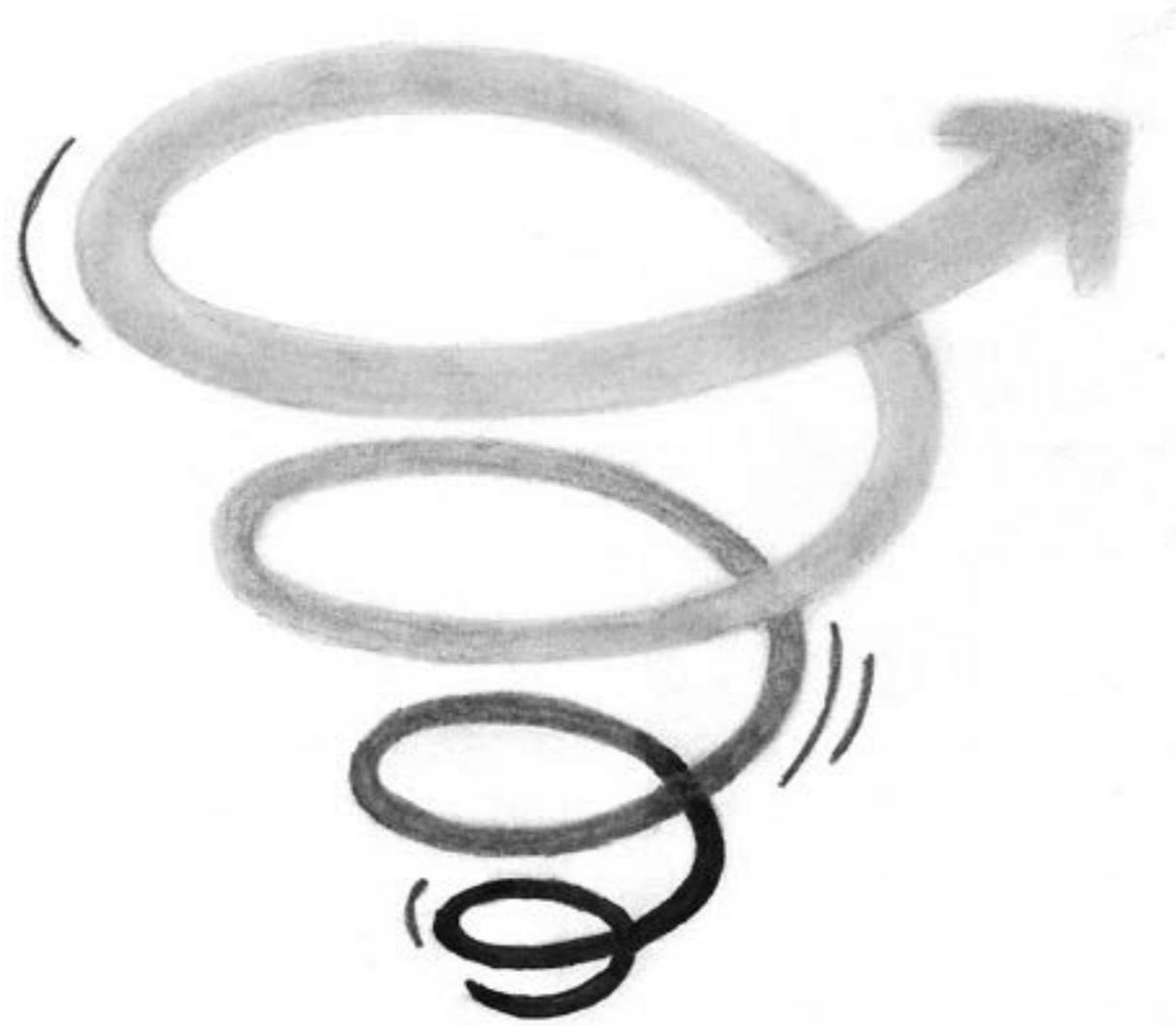
基于Node.js的前后端分离方案，让这些离我们更近。

6. 总结



分离是为了职责专注，改变是为了产品更好。

总结



发展的螺旋式上升。

总结



面对新的浪潮，我们准备好了吗？

我们需要

基于Node.js的前后端分离

前端工程师 后端工程师 运维工程师

清羽 (qingyu@taobao.com)

THANK YOU!