基于 Node.js 的10亿PV级移动 Webapp 开发

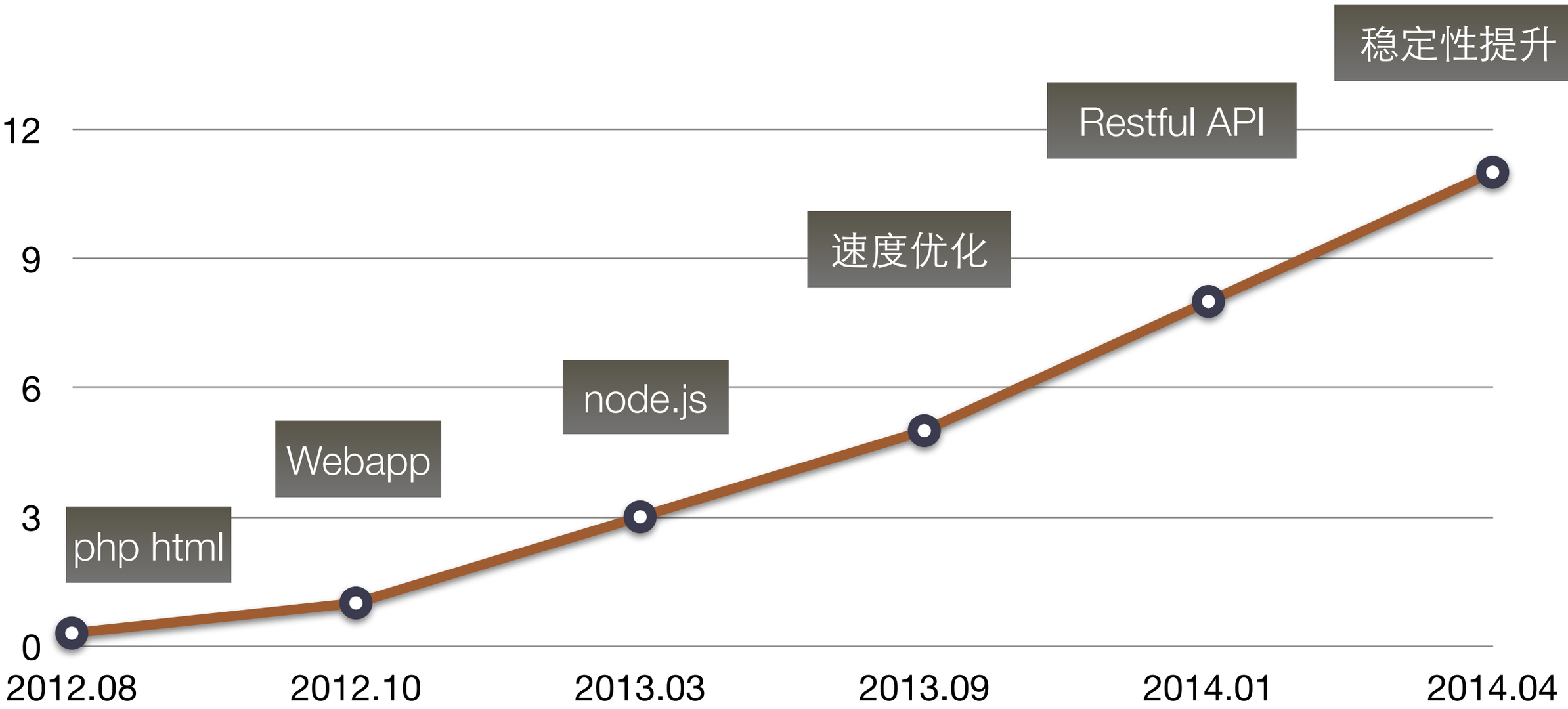
By 徐一智@百度 (xuyizhi@gmail.com)

关于我

- zezhou (徐一智)
- 喜欢：Web 开发, Machine Learning, Python
- 目前做移动前端架构方面的工作
- github (@zezhou) 微博 (@我是阿徐)

反馈交流：*xuyizhi@gmail.com*

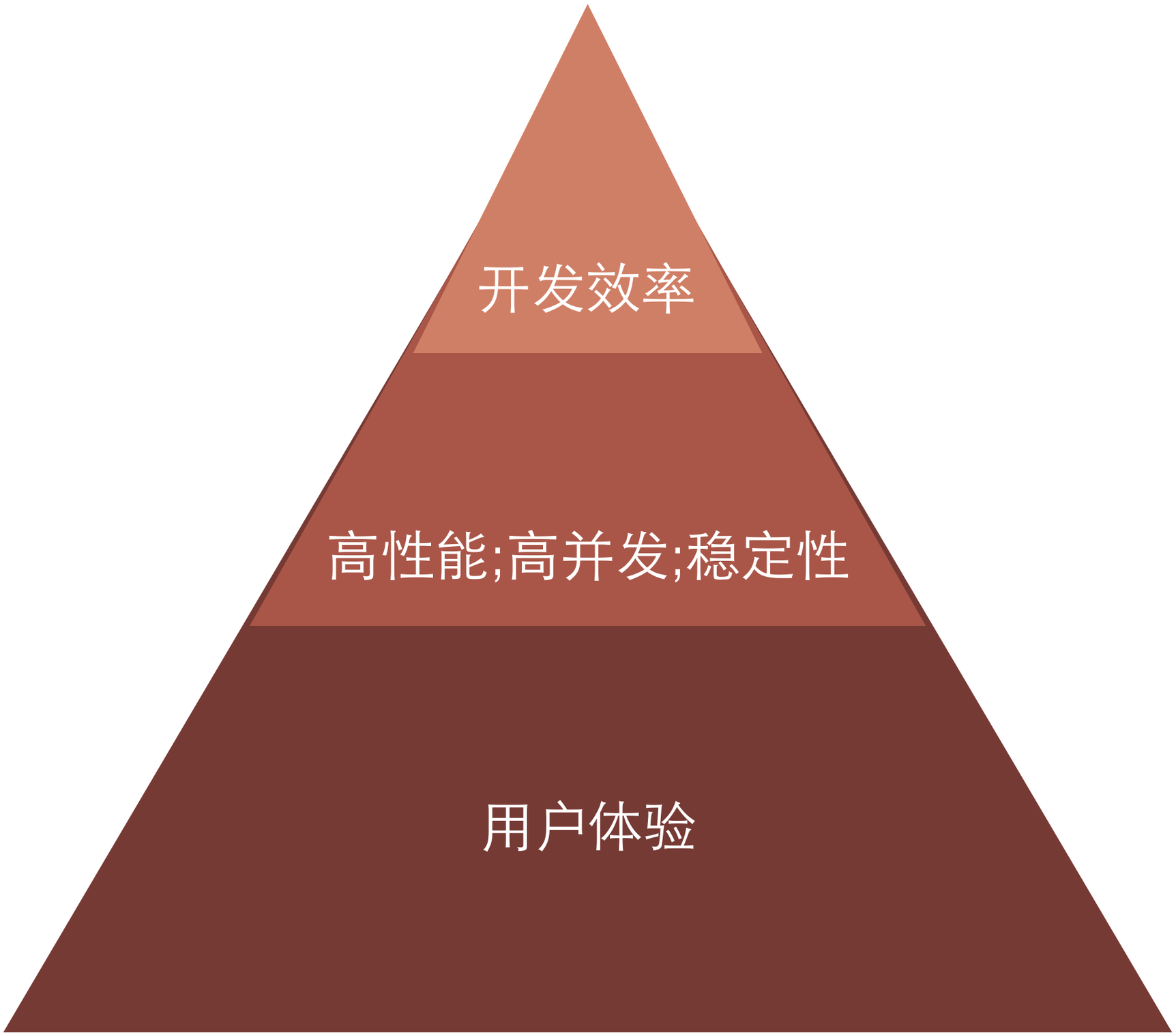
10亿级PV移动产品发展历程



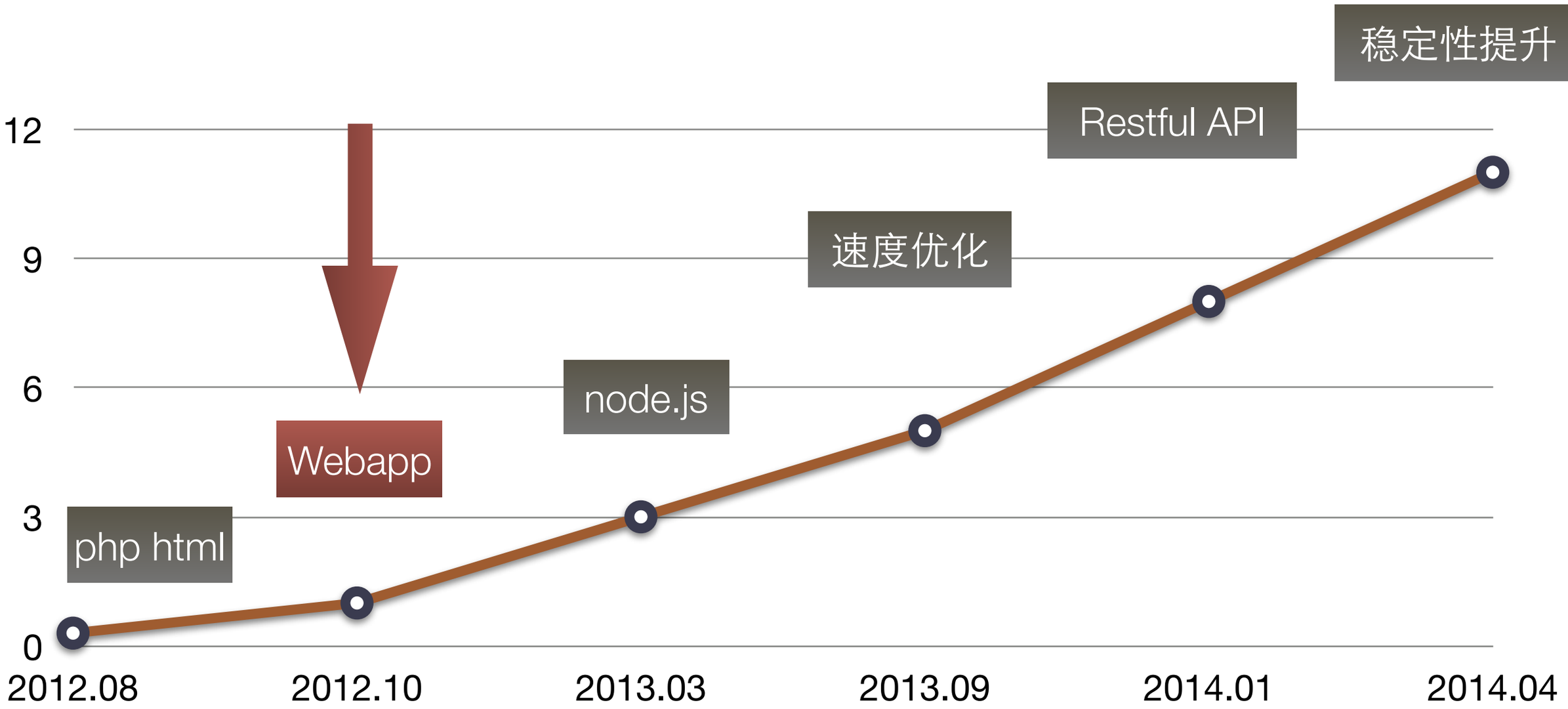
前端开发职责


前端
UI
Webapp
Node.js
后端
按钮
样式
排版
数据
事件
业务逻辑
Web Server
业务逻辑
页面渲染
检索
用户信息
数据挖掘

产品成长过程中遇到哪些问题?



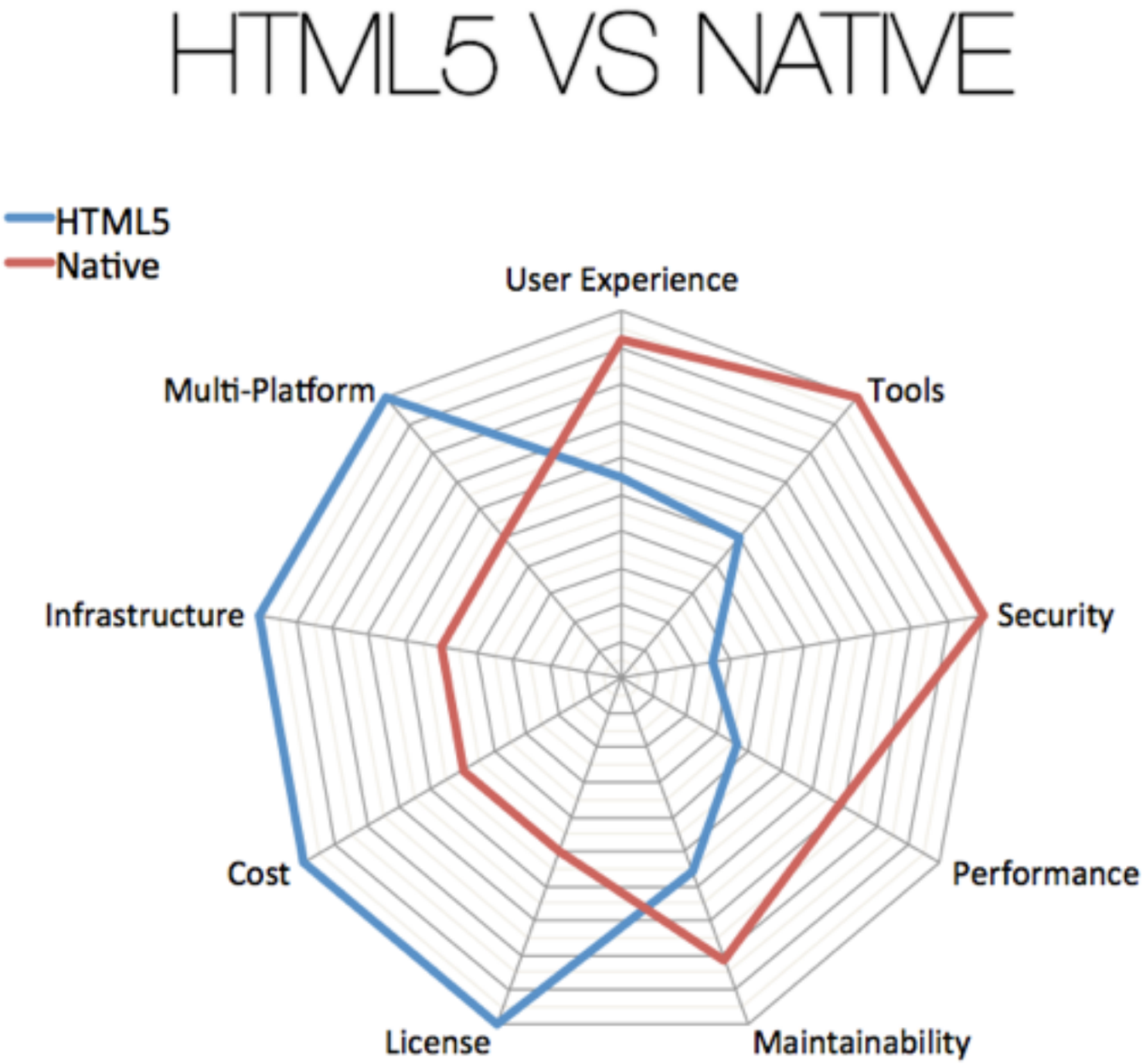
挑战一：创造极致的用户体验



移动下需要什么样的体验?

- 触屏操作
- 适合手机屏幕阅读
- Native 的流畅体验
- 适应移动网络环境(速度快, 容错，省流量)

选型

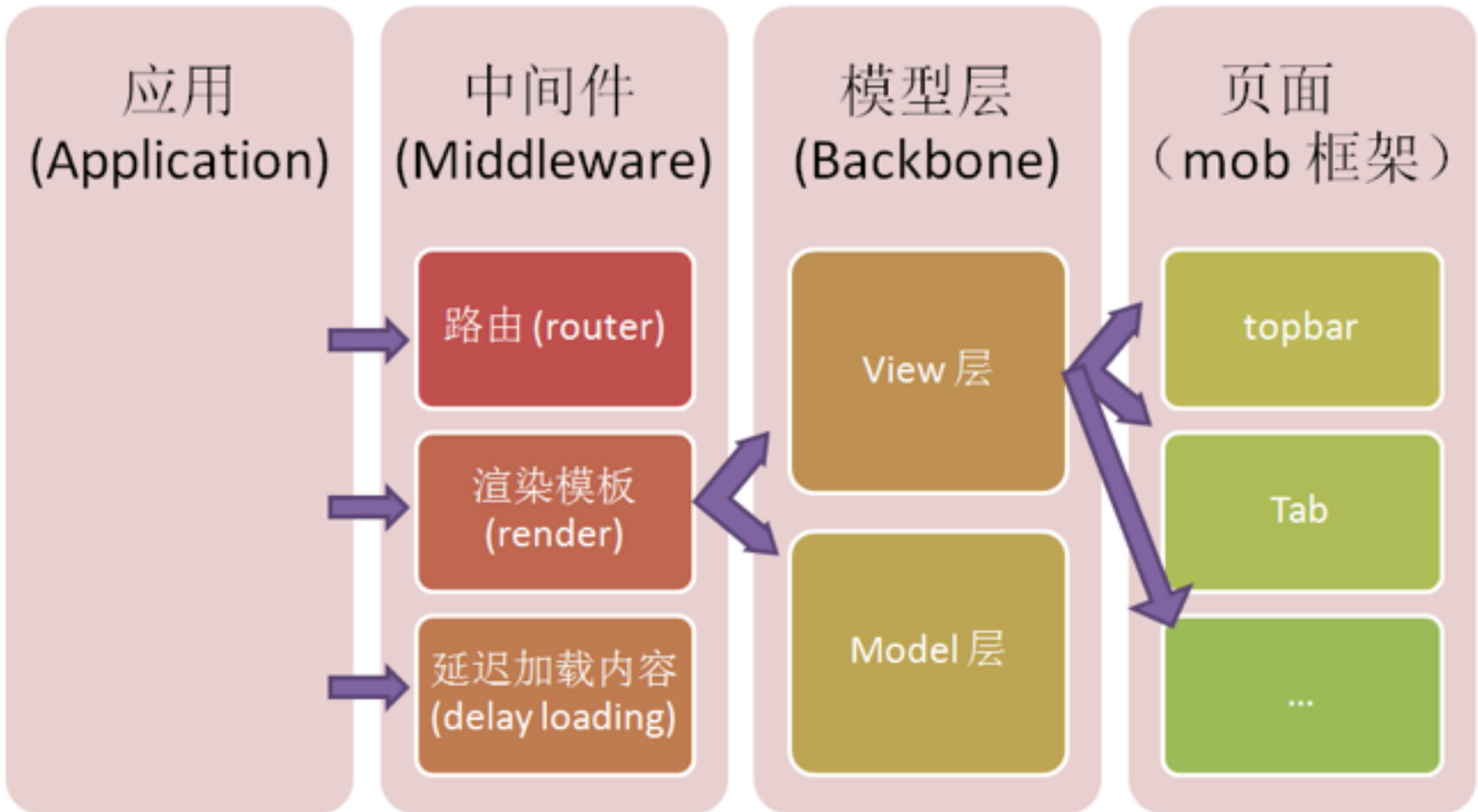

HTML5 VS NATIVE
HTML5
Native
User Experience
Tools
Security
Performance
Maintainability
License
Cost
Infrastructure
Multi-Platform

- Web 导流方便

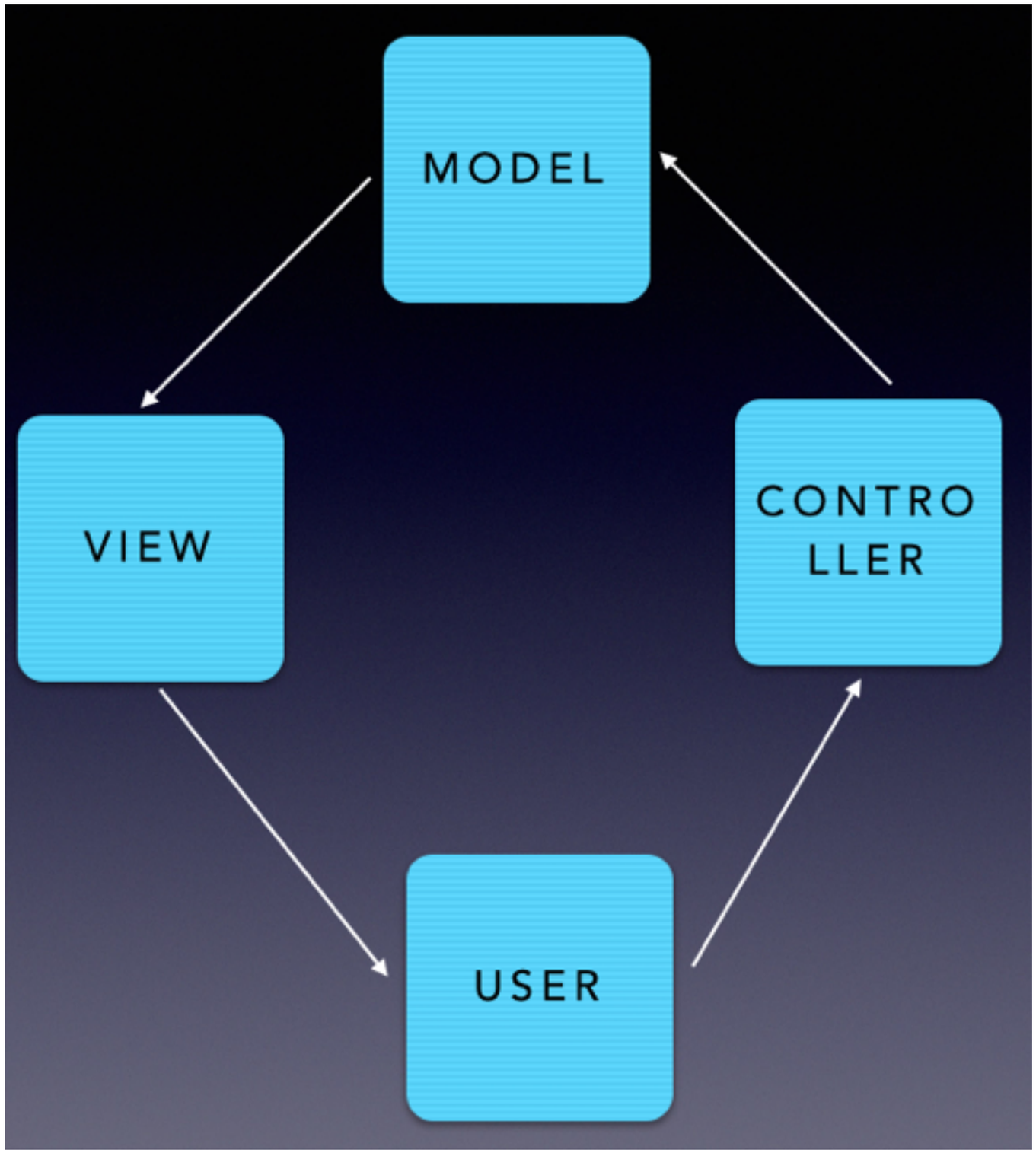
单页应用开发难点

- 自己管理状态 (路由，render, template)，复杂度和 Web 页面相比上升n倍
- 追求native般的操作流畅性
- 移动环境下的新问题 (移动2G,3G,4G网络的适配, android, ios的适配)

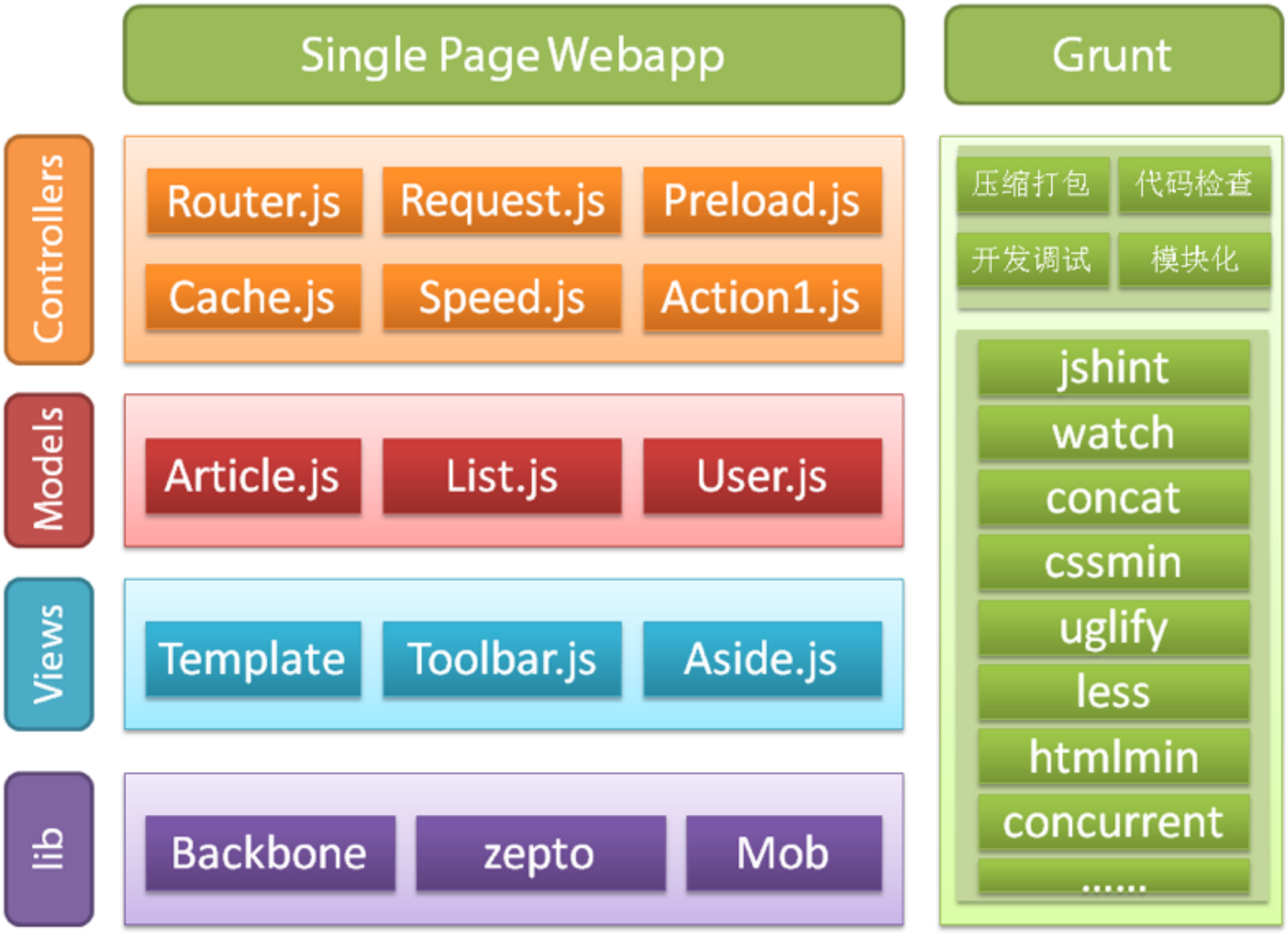
降低代码复杂度: Web UI 分层架构



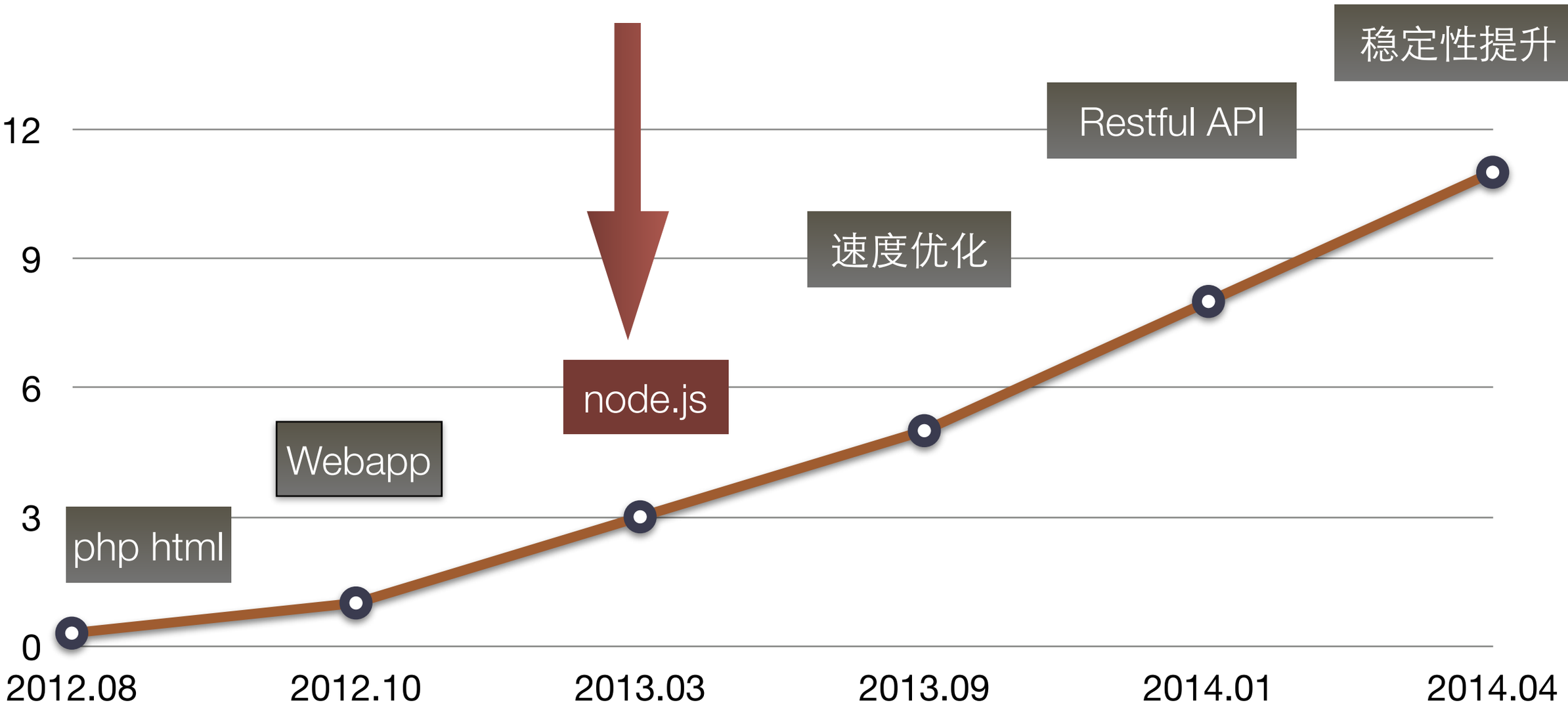
简化各层级之间关系：单向调用



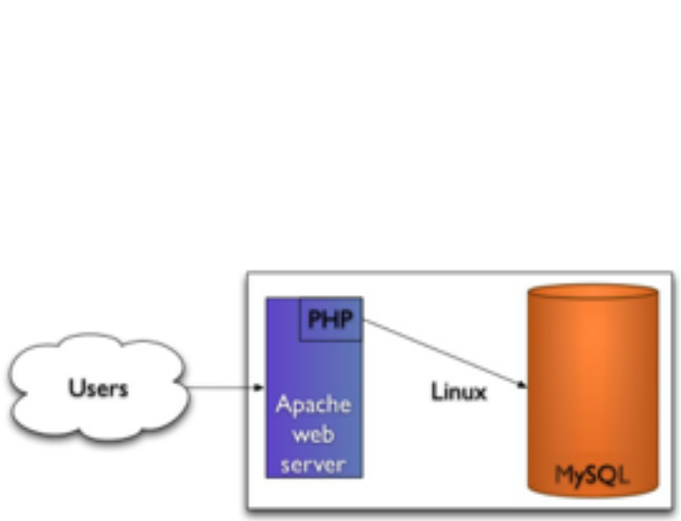
Single Page Webapp 结构图

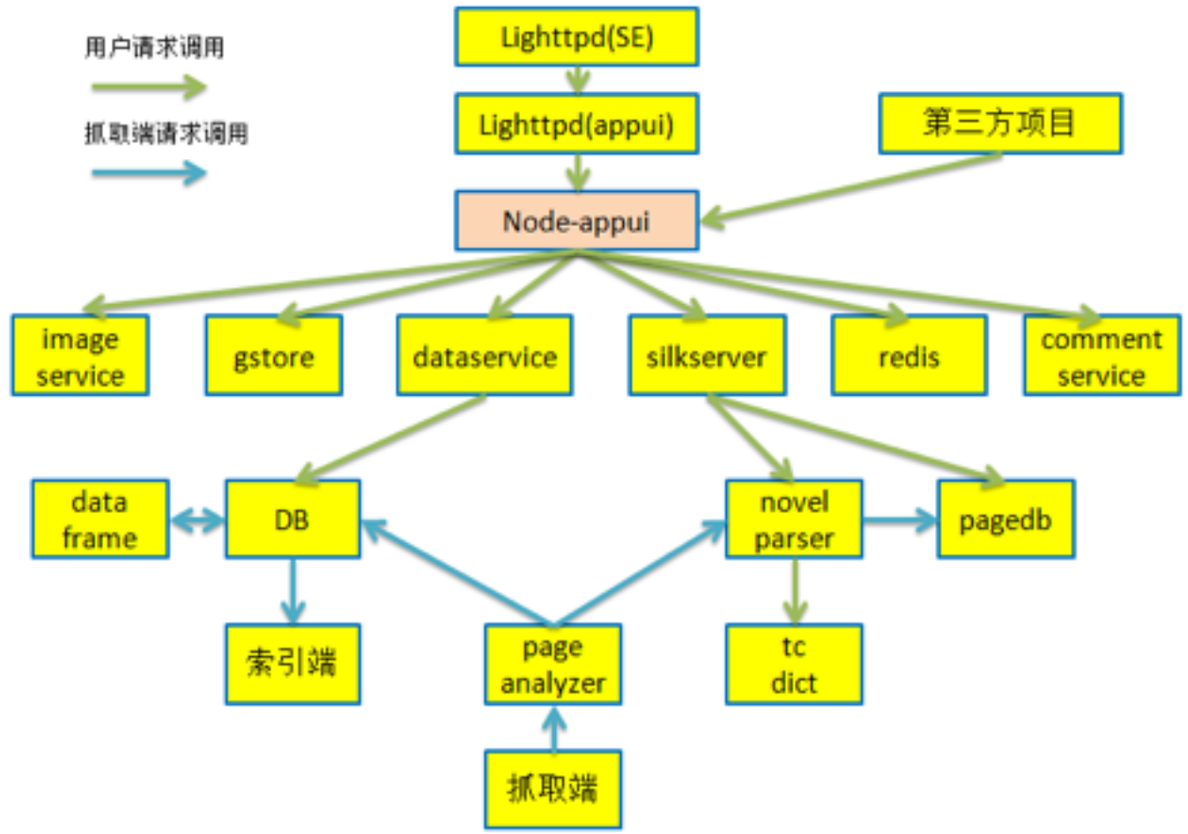


挑战二：支持产品的指数级增长



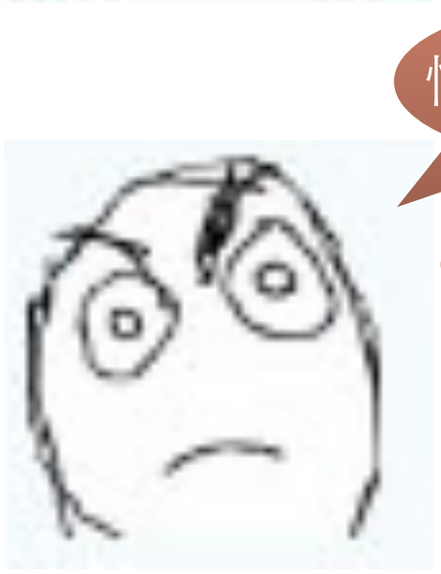
App Server 架构变迁



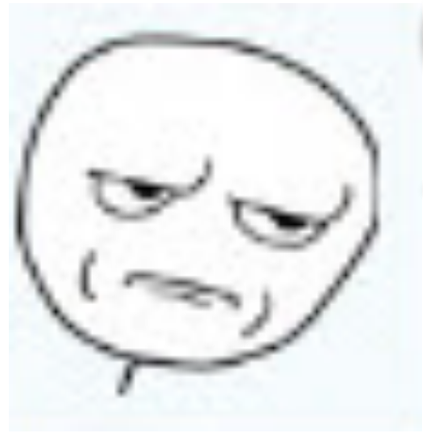


PV上涨，性能问题浮现

RD



OP



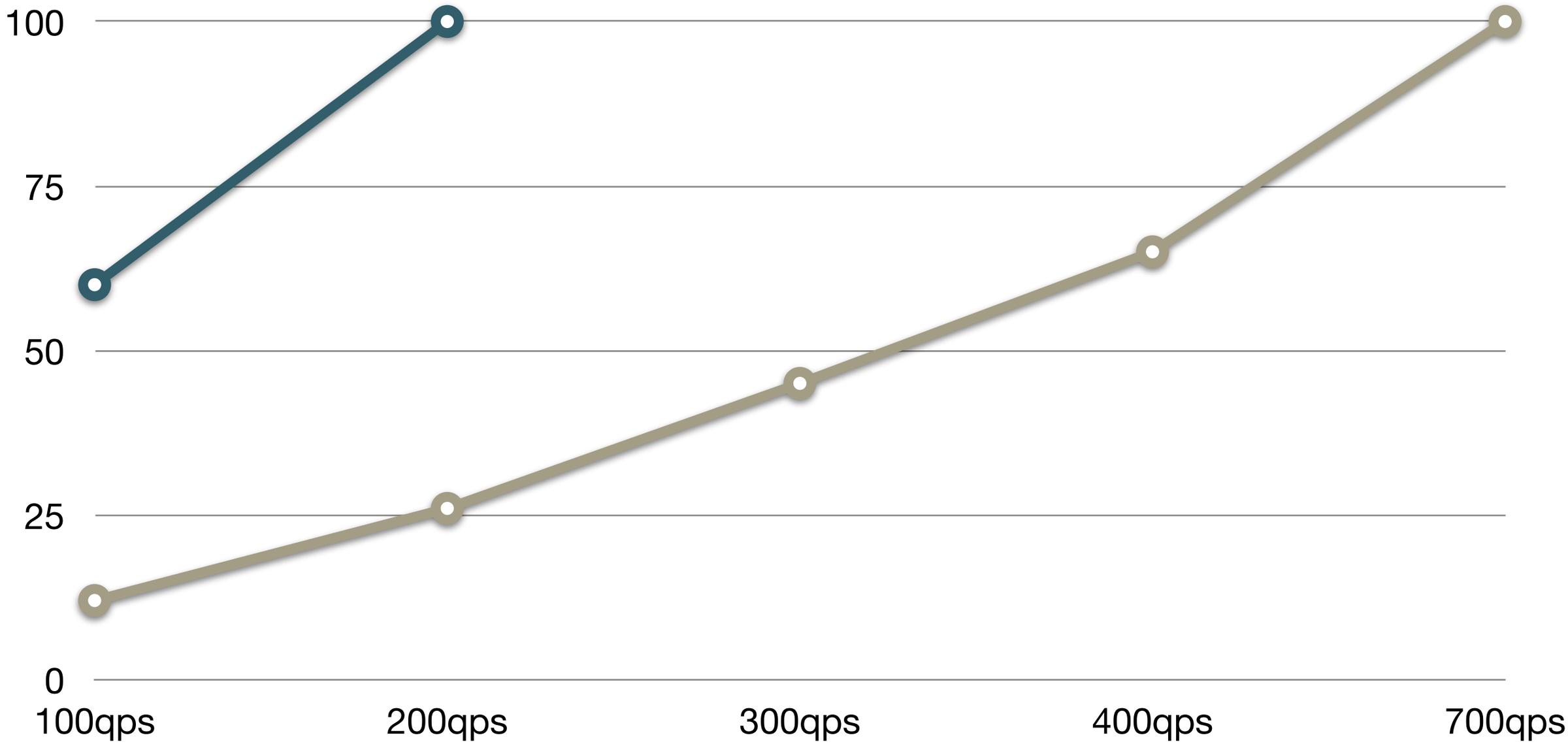
解决性能问题：服务端架构从PHP迁移到Node.js

同步阻塞（排队）

异步非阻塞（叫号）

单进程单线程，V8 + libuv，天生高并发

PHP迁移Node.js之后qps增加了7倍



产品功能增加，系统复杂度膨胀

某个版本的数据流走向

增强代码可维护性

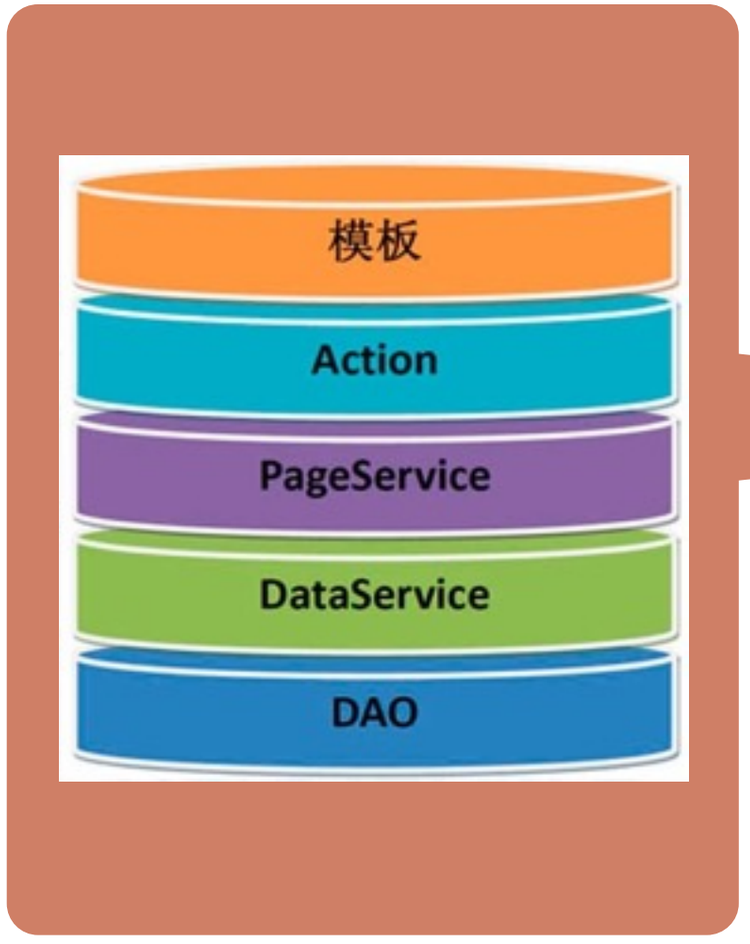
```
//传统回调方式
var titles = []
getCatalog(null, function(err, catalog){ // 回调1
   getArticles(err, function(articles){ // 回调2
      getTitles(err, fetechedTitles){ // 回调3
         titles = fetechedTitles
      }
   })
})
```

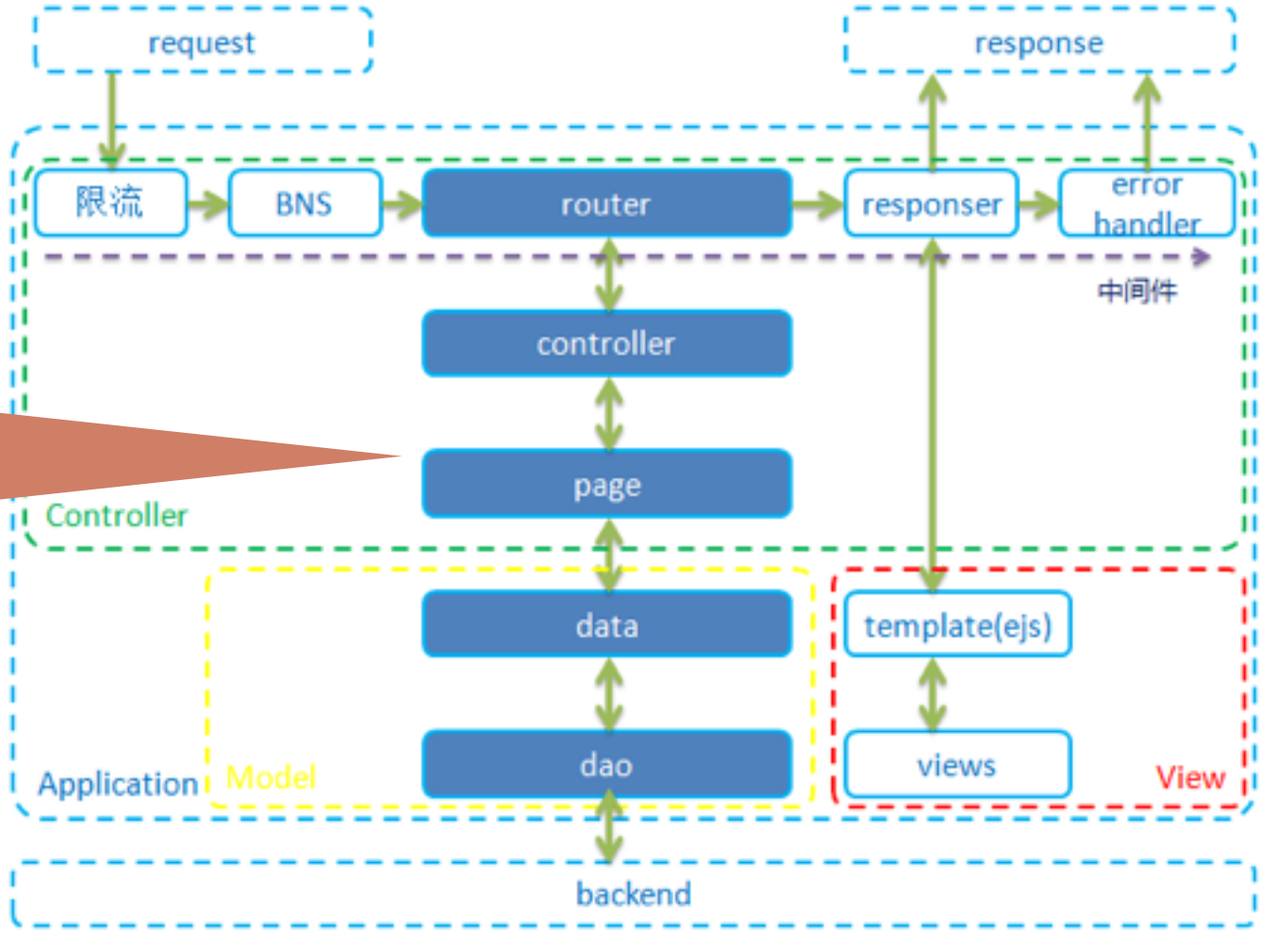
```
//async 方式
var titles = []
async.waterfall([
      getcatalog,
      getarticles,
      getTitles
   ],
   function(res){
      titles = res
   }
)
```

```
// es6 & koa 方式
var titles = []
co(function *() {
   var catalog = yield getCatalog(gid)
   var articles = yield getArticles(catalog)
   titles = yield getTitles(articles)
});
```

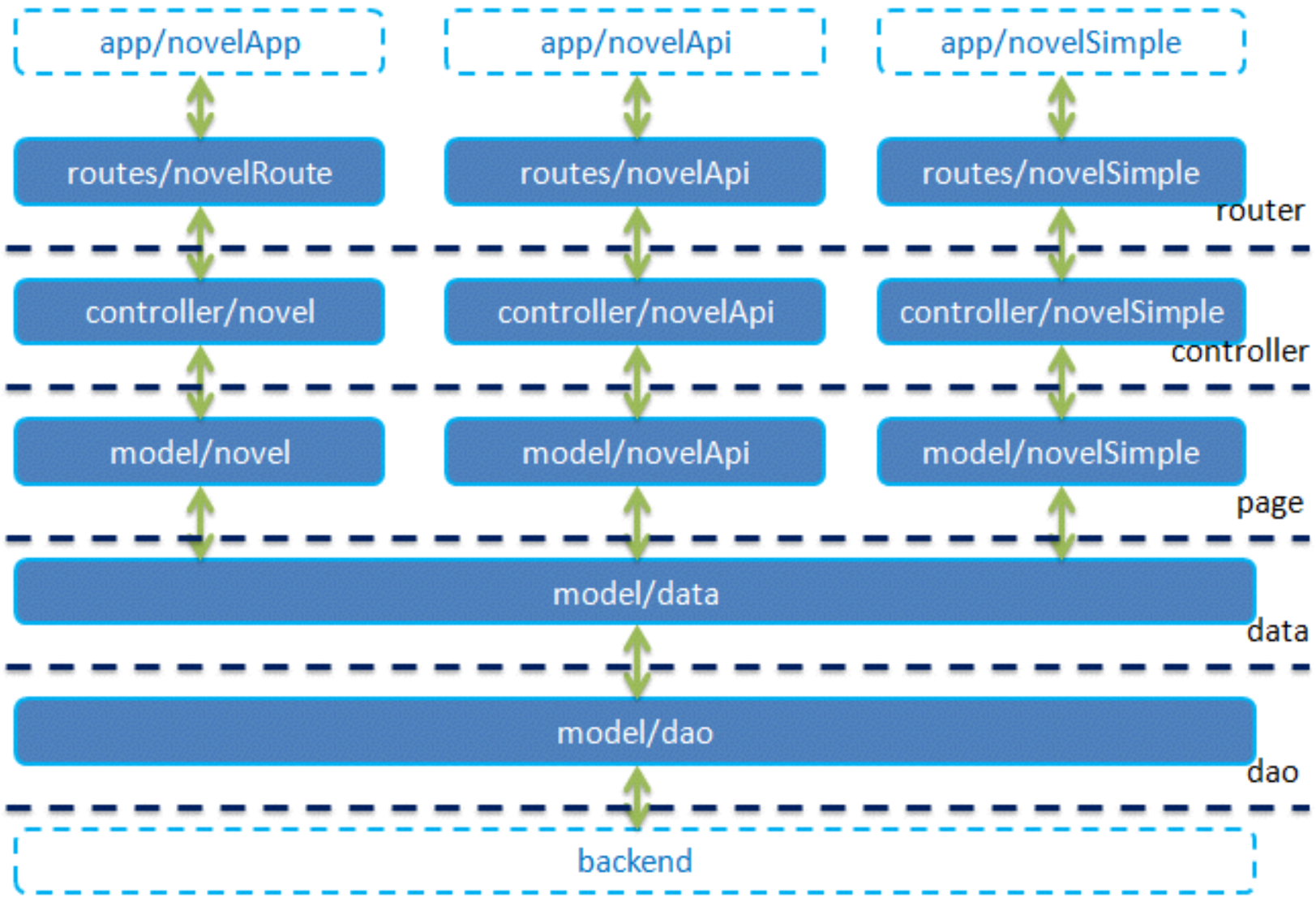
- 减少回调 (回调只允许一次, async, 层之间交互用 event)
- ES6 的 Generator, Koa, co

MVC分层架构降低复杂度

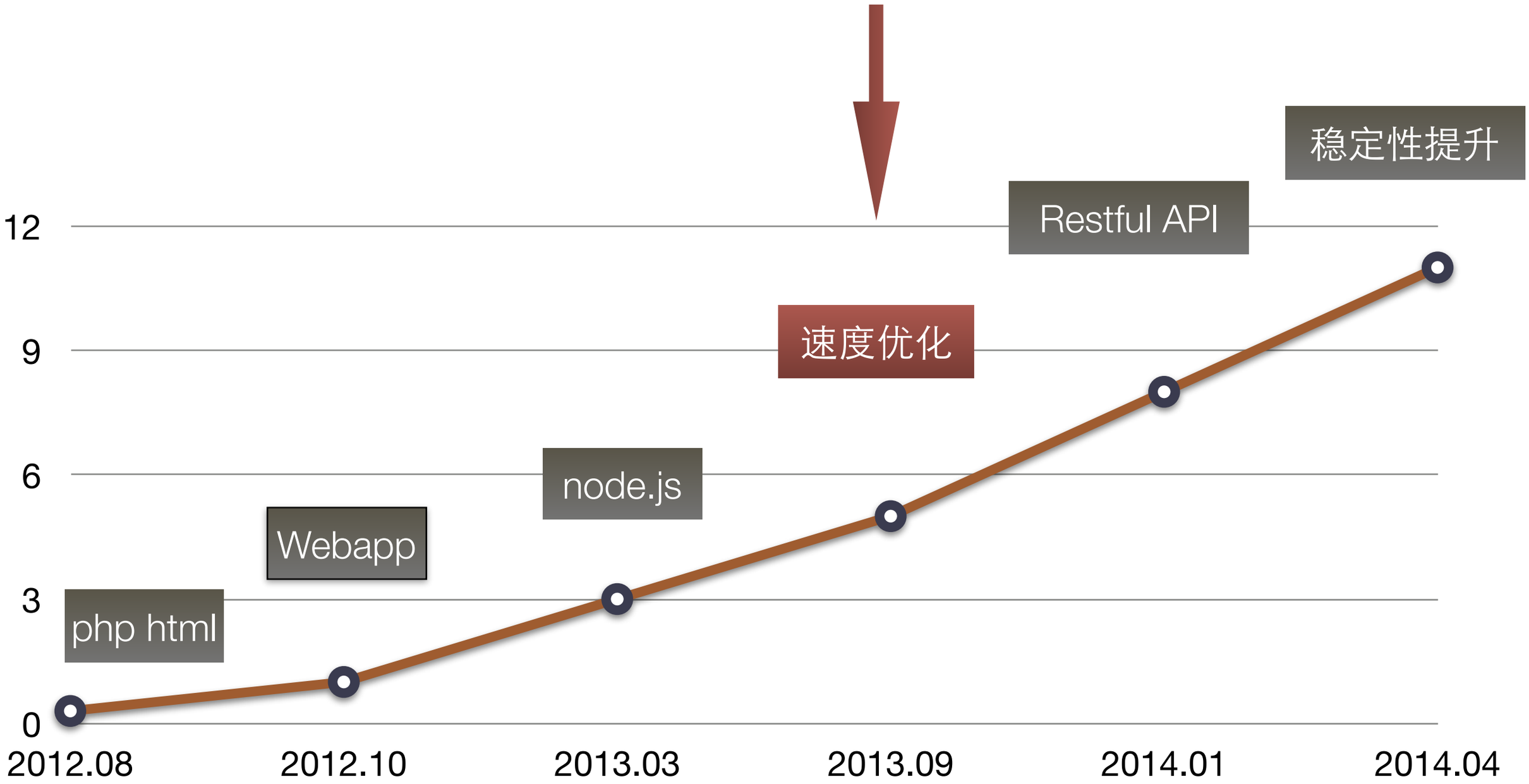




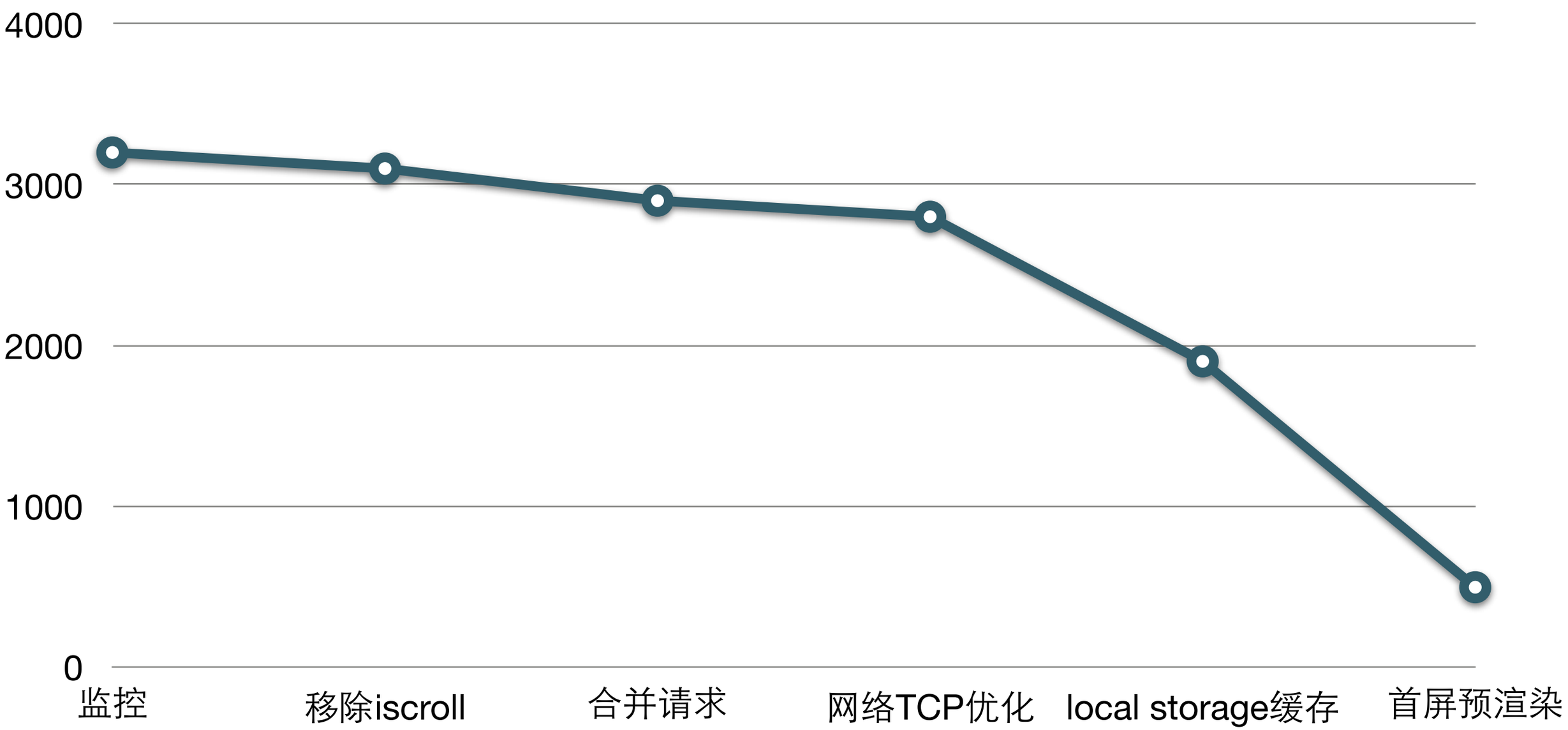
被上层调用，调用下层（之间event通讯）



挑战三：快


稳定性提升
Restful API
速度优化
node.js
Webapp
php html
0
3
6
9
2012.08
2012.10
2013.03
2013.09
2014.01
2014.04

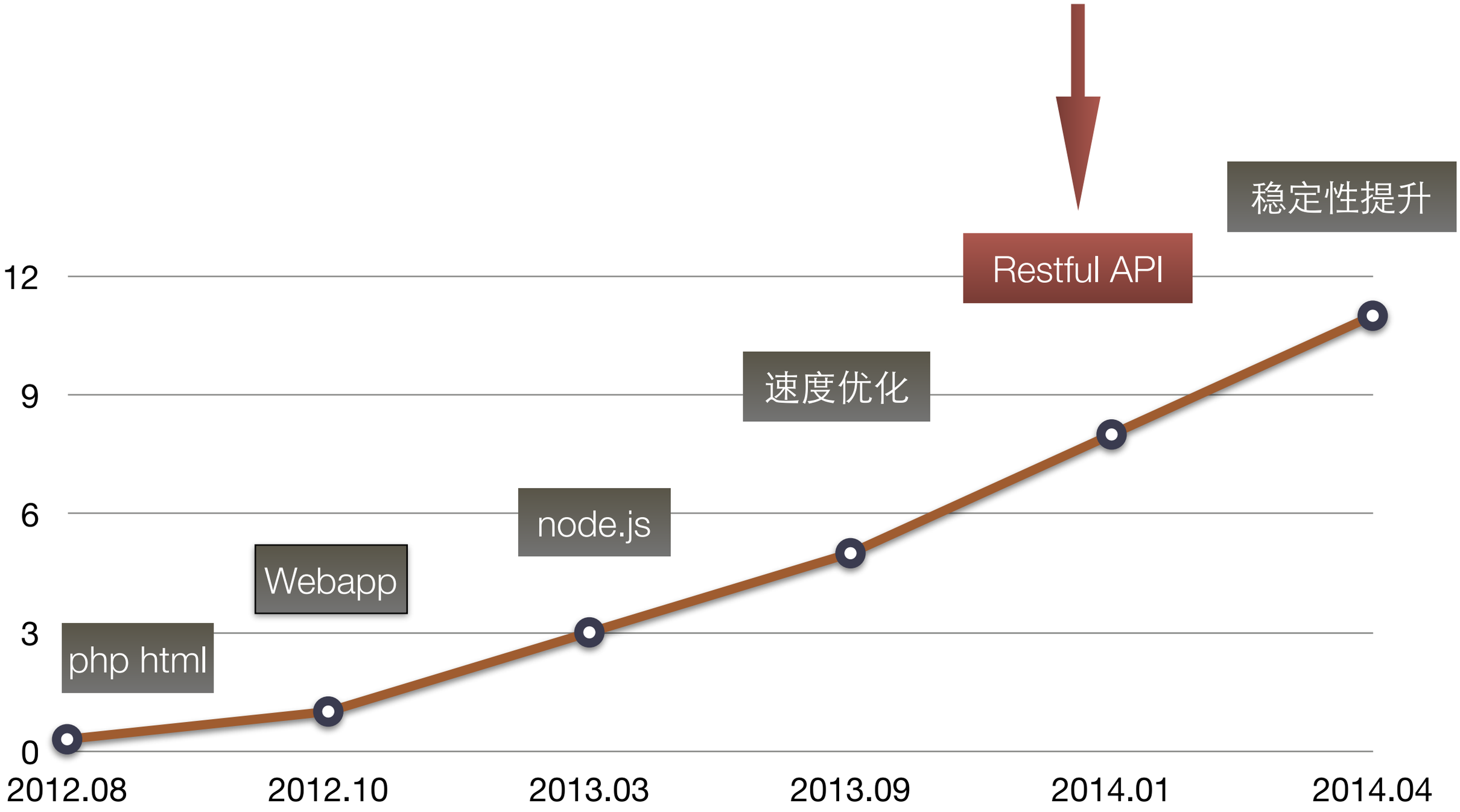
首屏加载速度优化


4000
3000
2000
1000
0
监控
移除iscroll
合并请求
网络TCP优化
local storage缓存
首屏预渲染

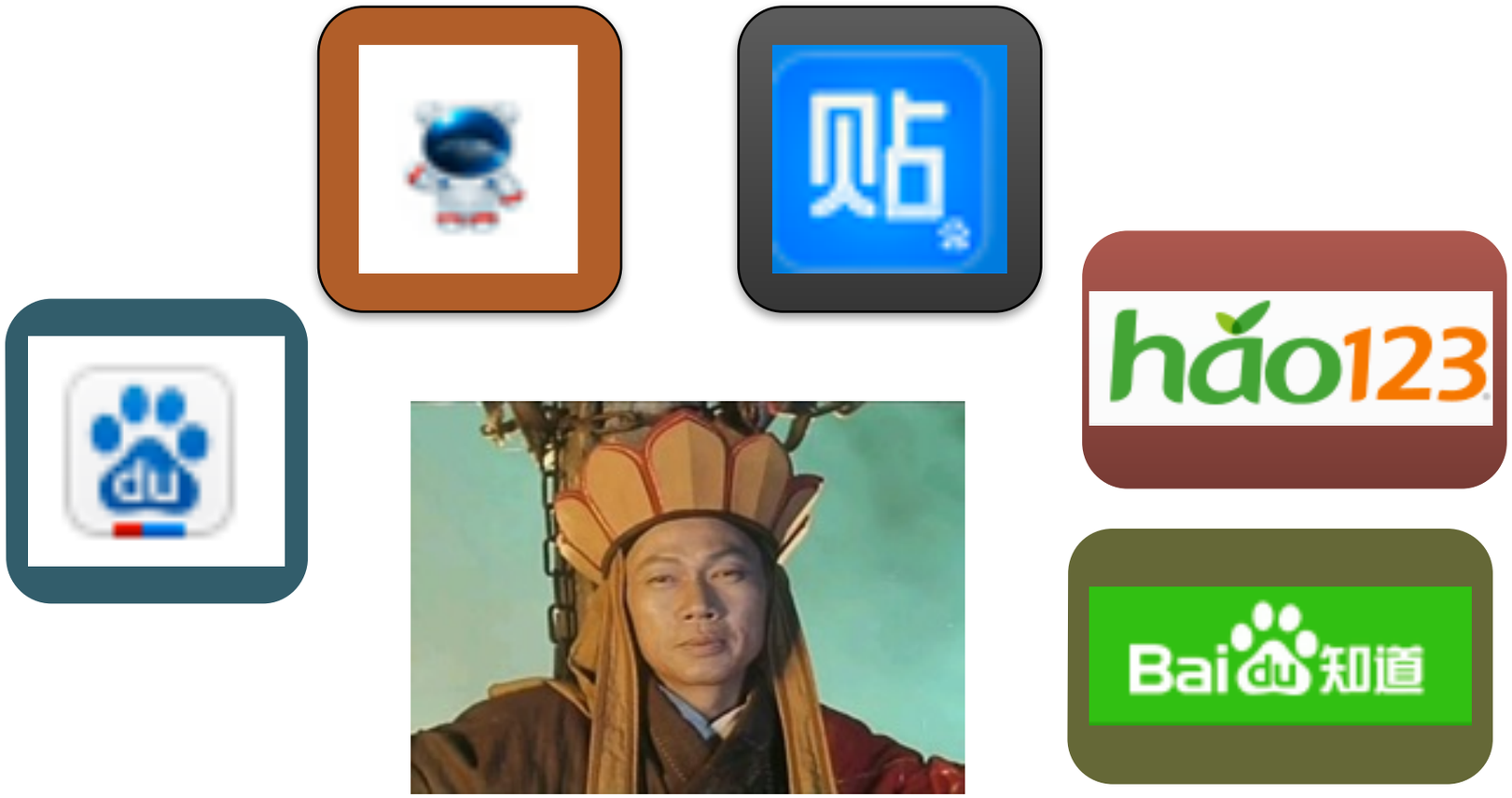
前端异常监控


异常发生的文件列表：
(1) undefined 异常PV：637 异常UV：669 异常发生次数：4,596
(2) htt 异常PV：294 异常UV：296
异常发生的浏览器：
47.79%
0 5 10 15 20 25 30 35 40 45
UC Browser Safari
比用户先发现问题
最近半月趋势：
10k
5k
0k
2014-10-27
2014-10-31
2014-11-04
异常PV 异常UV 异常次数

挑战四：1VN


稳定性提升
Restful API
速度优化
node.js
Webapp
php html
9
6
3
0
2012.08
2012.10
2013.03
2013.09
2014.01
2014.04

服务API化



解决对外合作问题

服务API化

关键一 Restful

注意 常见的一种设计错误，就是URI包含动词，因为"资源"表示一种实体，所以应该是名词，中show是动词，这个URI就设计错了，正确的写法应该是/posts/1，然后用GET方法表示show款，从账户1向账户2汇款500元，错误的URI是：

```
POST /accounts/1/transfer/500/to/2
```

正确的写法是把动词transfer改成名词transaction，资源不能是动词，但是可以是一种服务：

关键二 文档

[redacted]基本信息

包括[redacted]是否可离线，不包括[redacted]信息等

```
GET  http://[redacted].baidu.com/[redacted]/{gid,gid,gid....}
```

示例：

```
http://[redacted].baidu.com/[redacted]/3278655874,3961103225
```

```
{
        data: [
                {
                        [redacted],
                        [redacted],
                        [redacted],
                        coverImage: "http://tc1.baidu-1img.cn/tim
%40tpl_size%40&src=http%3A%2F%2Fbj.bs.baidu.com%2F[redacted]
```

提升开发效率

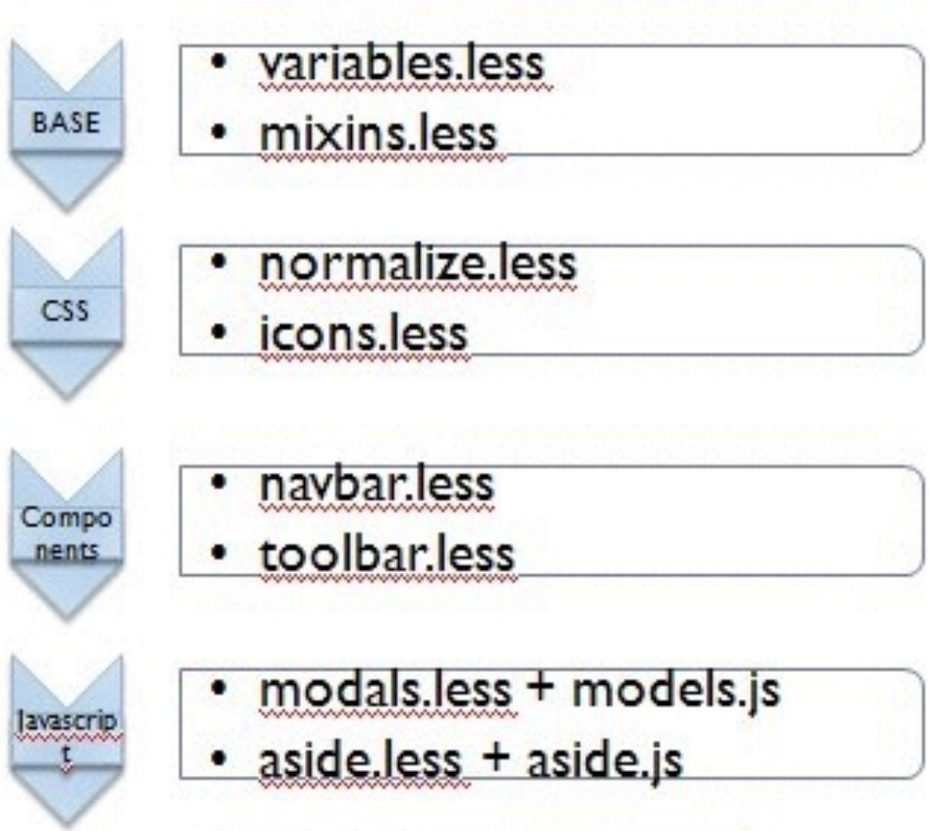
UI的抽象,复用

Mob UI 框架

齐全的一整套组件

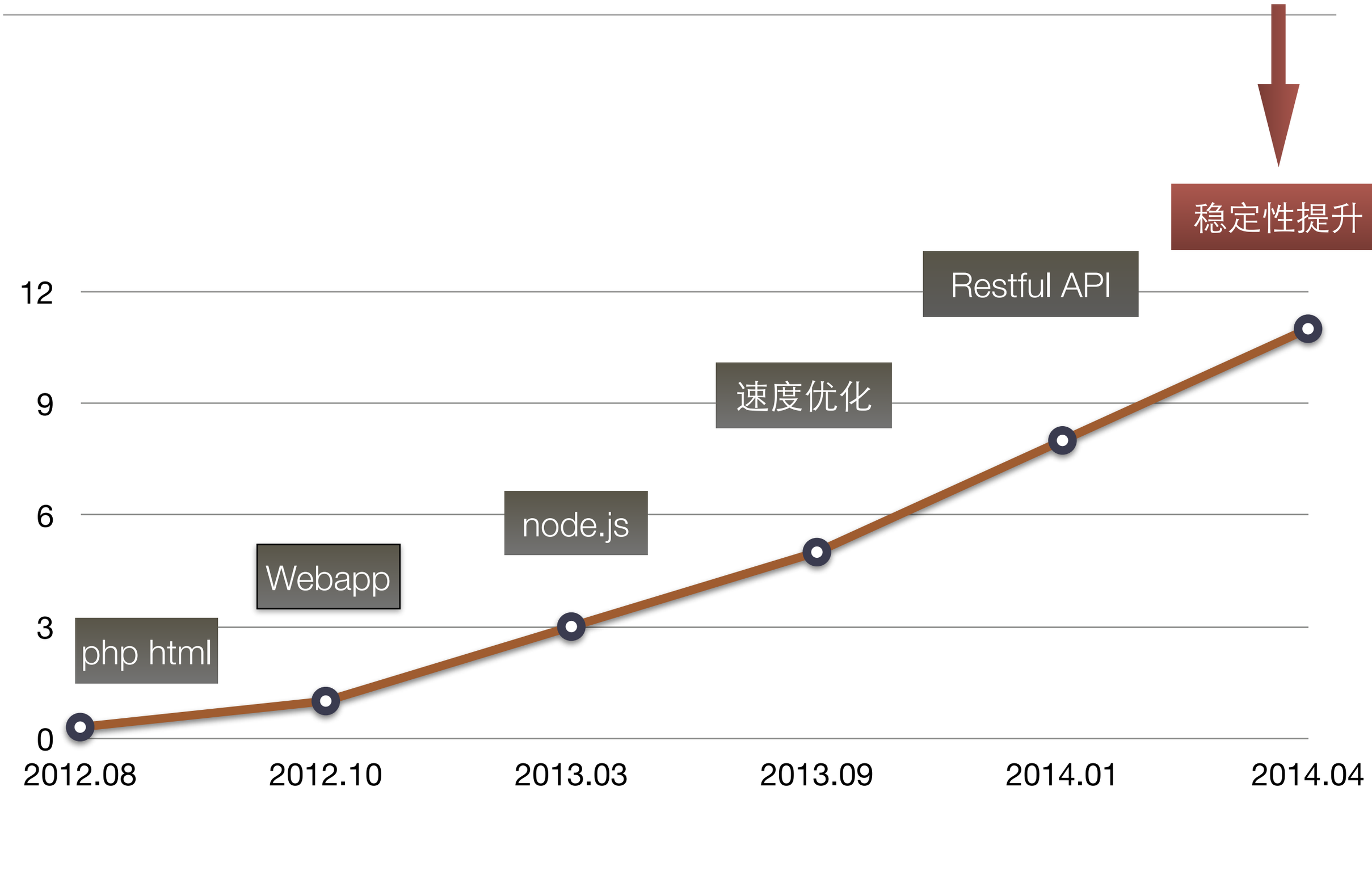
Sass {less}

可编程前端架构



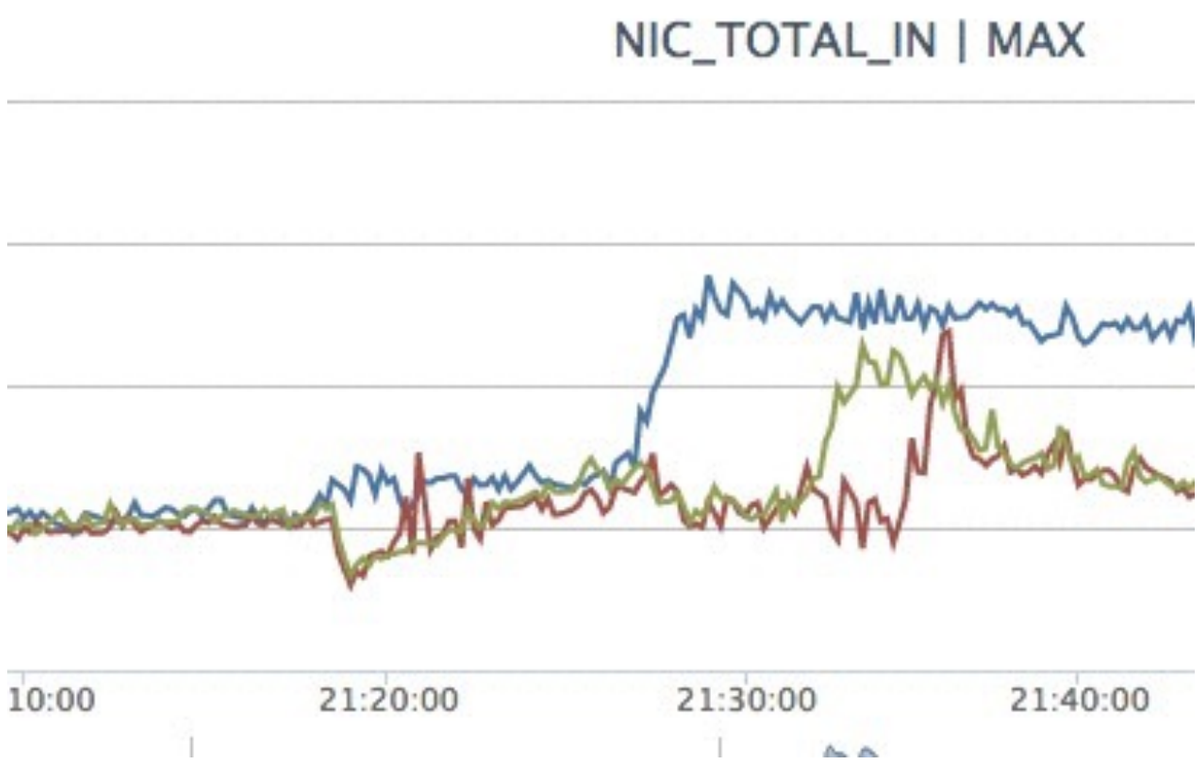
分层设计

挑战五：超大流量下的服务稳定性



流量继续上涨，发生雪崩

一个机房，一台机器出问题造成整个系统崩溃



出问题时的应对

- 制定降级预案，优雅降级
- 简单问题快速上线修复
- 查看公司运维系统, 监控系统，日志平台，报表
- 查看分层日志快速定位：query log, request log, error log, node error log
- 错误追踪: logid 串起前后端各个模块的日志
- 内存泄露: heapdump diff
- 不要解析大文本等耗cpu性能的的工作

问题预防 - Design for Failure

- 上线: 先小流量上线单台机器，经过高峰期考验再全流量上线
- 模块隔离
- 流量设置阈值防止下游雪崩
- 对于重试等策略通过判断状态码等进行优化
- 评估系统各子模块流量，对子模块进行监控，达到最高容量80%就报警扩容

在你的产品中也用上
10亿级PV的前端架构

一个软件工程师的人生

好了，重头再来…
这次我一定会老老实实
做个正确的东西出来！

不久之后…

我擦……
又是这样……

Mob UI 框架

- mobile 版 bootstrap (类似于 zepto 和 jquery 的关系)
- 重点在 mobile 专属组件(下滑刷新，触摸)
- 和 agularjs, reactjs, backbone 结合开发webapp
- http://mobframe.github.io/mob/

Lark Node.js 框架

- 专注于做工业级Node.js的架构
- 对线上众多10亿级Node.js产品的实践经验的总结
- 即开即用
- https://github.com/larkjs