

# 架构师

ARCHITECT



## 人物 | People

陈本峰：HTML5跨屏前端框架  
Amaze UI的开源之道

## 推荐文章 | Article

谷歌的诀窍：如何取消验证码

颠覆式前端UI开发框架：React

## 专题 | Topic

天猫11.11：搜索引擎实时秒级更新

小米11.11：海量数据压力下的推送服务

京东11.11：交易系统的关键技术



# 卷首语

又到年末，时光荏苒，InfoQ 在中国已经走过七个春秋，有幸参与并见证了中国 IT 产业的高速发展。技术变革是推动产业发展的主要力量，我们尝试站在技术浪潮的前沿，让国内的架构师、开发者了解和借鉴整个技术社区的成果。HTML 5、Node.js、Cloud、Big Data、NoSQL、DevOps、Agile、Open Source、Docker……一长串的列表，InfoQ 是国内最早关注并推广这些技术和方法的社区之一。

随着国内 IT 行业的崛起，中国的技术力量在世界上有了更多的话语权，我们则更加关注国内技术趋势的发展、技术人的成长。目前，InfoQ 的线上内容包括新闻、文章、专栏、访谈、演讲等，线下内容包括 QClub 技术沙龙、QCon 全球软件开发大会、ArchSummit 全球架构师峰会等，我们努力通过各种形式来构建国内技术社区交流的平台，让更多的人在技术浪潮中得到锻炼并最终受益。

除了传统的内容和活动，InfoQ 根据读者的需求，推出了精品栏目“架构师”月刊、“中国顶尖技术团队访谈录”等电子书，定期将网站内容精选成册，发布给广大读者。另外，针对 IT 领域的热点事件，我们也及时地进行追踪报道，比如“双十一”技术专题。

本月“架构师”汇集近期各个栏目的部分文章，希望以精华的形式呈献给大家，以此向崛起的中国技术力量致敬！

不忘初心，执着前行。

InfoQ 中国总编辑 崔康





促进软件开发领域知识与创新的传播



中文 | 英文 | 日文 | 葡文 | .....

# 目 录

人物 | People

[陈本峰：HTML5 跨屏前端框架 Amaze UI 的开源之道](#)

观点 | Opinion

[Android 5.0 技术新趋势](#)

[.NET 开源核心运行时，且行且珍惜](#)

专题 | Topic

[天猫 11.11：搜索引擎实时秒级更新](#)

[小米 11.11：海量数据压力下的推送服务](#)

[天猫 11.11：移动端性能提升两倍](#)

[京东 11.11：交易系统的关键技术](#)

[蘑菇街 11.11：流量超过预期，系统曾被冲垮](#)

[天猫 11.11：多终端下的一致性体验](#)

推荐文章 | Article

[谷歌的诀窍：如何取消验证码](#)

[扩展性即服务](#)

[颠覆式前端 UI 开发框架：React](#)



# 陈本峰：HTML5 跨屏前端框架 Amaze UI 的开源之道

作者 崔康

## 编者按

对陈本峰的采访，源于技术圈内的一个饭局，虽然大家对他的云适配创业经历很感兴趣，但是他却在自我介绍中反复提到了“开源”和“Amaze UI”，言谈举止中透露着对国内开源社区发展的关注和热情参与，特别是他领导的 Amaze UI 开源项目在正式上线 2 个月之后，在 Github 上就取得了 1000 多个 star 的关注度，这样的成绩在国内屈指可数。于是，便有了对陈本峰的采访和下面的内容。

## Amaze UI 的前世今生

在中关村创业大街旁边的办公室中，Amaze UI 项目的领导者、云适配 CEO 陈本峰欣然接受了 InfoQ 中国总编辑崔康和高级编辑郭蕾的专访。陈本峰首先谈起了 [Amaze UI](#) 开源的背景，[云适配](#)是专注前端技术的公司，在把网站从电脑版转到手机版的时候，需要一个手机的前端框架。从创业初期开始积累，Amaze UI 逐渐发展起来。

早期的时候，对这种移动端的 Web 界面框架比较少。国外的这块移动版做得相当的好，我觉得他们兼容性做得不错，但是速度非常慢，打开非常慢，所以无法忍受。于是我们当时决定自己做，所以就有了这套框架（Amaze UI），一直以来都是给云适配整套体系用的，后来我们觉得这套东西可以剥离出来，让别人也能用。所有人在开发移动网站的时候，都应该需要这么一个框架，因为这样可以大幅度提高开发效率，不需要重新发明，这是一个基本的思想。



在开源之前，陈本峰的团队主要做了 3 件事情，这些看似微不足道的步骤为开源项目打下了良好的基础：

- 代码精进化
- 文档规范化
- 启动内测

## 国内开源项目尚处萌芽期

随着国内技术社区的发展，国产开源项目越来越多，但真正运营成功并取得广泛关注的例子并不多。在笔者抛出这个问题之后，陈本峰显然已经早有答案。他指出，从国外的开源经验来看，一个项目要想成功，必须有一个专职的研发团队来做。虽然我们谈开源，经常说靠社区的力量，但是最核心的推动力还需要是专职团队，并且这个专职团队是真的为社区服务的。

国内很多开源项目，大多是开发者自己的兴趣爱好，并不是公司层面来支持项目的，经过一段时间之后，开发者工作调动或者公司业务重心转移，就会导致项目夭折了。陈本峰分析过 Github 上的一些开源项目，刚启动的时候，更新频率很高，一个月有几十次，但是到后来，基本两年都没有更新一次了。这种状况无法给潜在的开发者信心。

对于开发者来说，评估开源项目可用性主要有两点，一是社区支持度，二是活跃度。这两项不达标，就没人敢用。陈本峰认为，如果开源是一个公司行为，而不是个人兴趣爱好，那么活跃度可以有保障。

## 如何成功做好开源项目

即使公司提供支持，开源项目就可以成功了吗？显然不是！陈本峰强调，开源要用一种服务社区的心态去做，而不只是服务自己公司内部。

虽然 Amaze UI 是云适配整个大产品体系的一部分，但是开源之后，Amaze UI 团队就只需要去考虑外面社区的开发者的需求，把我们自己的内部团队也当成跟外面社区等同的客户去对待，不因为是内部的团队就优先处理，而是看这个需求到底是大家普遍认为一个比较需要的高的需求，还是它就是我们的一个特殊行为。如果是我们的一个特殊行为，就应该慎重。让我们内部的团队基于开源基础上自己去改，改完以后由开源项目团队判断是接受还是拒绝掉，完全独立地由两套团队去运行，他们的目标也就非常明确。云适配跟 Amaze UI 开源的目标是各自明确的，互相独立的。我们在做的过程中，也发现外面社区有很多比较强烈的需求，比如我们在做 Amaze UI 里面发现最大的一个需求就是除了 Amaze UI 本身提供的一些功



能，客户需要一些第三方的英文插件。其实这个需求本身在云适配这块是没那么强的，因为云适配这块主要还是针对移动端，而客户需要的英文插件是开发者在做一个横跨手机、平板、PC三屏的网页时会需要的。并不是云适配的强烈需求，但是在我们的2.0阶段最重要的工作就是做这件事情，这就是一个非常典型的例子，说明我们是优先考虑外面开发者的，然后才是考虑内部开发者，或者就等同对待。

这并不是纸上谈兵，Amaze UI项目如今由两位全职的开发者在推动和维护，都是云适配的员工。

## 用产品的标准做开源，解决移动化难题

对于开源项目的定位问题，Amaze UI虽然提供英文的相关介绍，但是它更是为国内开发者优化的，陈本峰分析了几个原因：

- 本土化的支持，Bootstrap没有做专门针对中文的支持。字体是网页里面非常重要的一块，直接决定了网页展示出来的体验的好坏。  
Bootstrap里面是没有定义中文字体的，这就会导致每个浏览器都会根据默认设置去选一个字体。比如说IE在XP和Windows8下字体就是不一样的，在苹果下面字体又不一样了，然后在各种手机浏览器上面字体都不一样。最后导致做出来的网页可能在各种浏览器、各种操作系统下面看起来效果都不一样，是完全不可控的。而且可能会导致排版格式掉。AmazeUI里面很严格的定义了中文字体，做到在各种设备、操作系统、浏览器下看到的效果基本上是一样的，比如说中文字体，我们用的是雅黑。在Windows底下是雅黑，但是在苹果底下是没有雅黑这个字体的，那我们就用最接近的黑体去做。Bootstrap基本上是用13号字，我们是用14号字，字号的大小也会导致排版不一样。Bootstrap不太可能去加中文字体，因为如果一旦加中文字体，就还要加日文字符、韩文字符、法文字符，Bootstrap就会变得巨大无比了，这肯定也不是产品设计的初衷。还有对本土浏览器的支持，当时做Amaze UI的另外一个想法源于浏览器的兼容性，对于多数前端开发者来说，或者都是一个梦魇。可能开发一个网页用一个月的时间，但是做浏览器的兼容可能要花两个月的时间，甚至都做不完，面向的浏览器太多了。这些工作没有必要让开发者一遍又一遍的重复。所以当时我们想做一个开源产品，大家基于这个产品，把浏览器兼容性都调整一下，以后使用这个产品就行了，节省了大量的工作。国内浏览器和国外浏览器也是很不一样的，像360、搜狗等，而且国内有双核浏览器，这也是国外不存在的。针对这些中国特色，我们会做一些调整和一些特别的优化，这也是我们跟Bootstrap一个比较大的区别。
- 移动优先，Amaze UI一开始为移动端开发的，所以非常考虑在移动端的表现，要让整个代码体积尽量小。另外尽量的采用CSS3的动画，动



画效果以前在 PC 端，都是用 JS 版做，一方面要下载 JS 代码。另外一方面是它对机器的要求比较高。因为 JS 需要大量的 CPU 运算。移动端的浏览器，相对来说都比较现代，都是支持 Html5 的，使用 CSS3 动画就会节省代码，因为一行 CSS3 代码可以解决一个动画的问题，代码体积会小很多。第二，因为 CSS 动画是浏览器原生支持的，所以会有硬件加速。硬件的运算能力是要比用 JS 软件上的能力要强很多的，所以整个移动端的体验会好。

- 组件化，Amaze UI 非常强调组件这个概念，近两年有一个非常热的技术叫做 Web component，它的意思是说，网页的每一个构件，都可以封装成一个组件，这种技术在后端已经非常成熟了。比如说 Node.js 里面的 NPM，可以用来管理各种各样的包。Ruby 的 Gem，Python 也有，但是前端是没有的。在前端大家的做法是非常原始的，比如要做轮播图，就是拷贝大段的 Html5、CSS 和 JS，这个很大的问题就是，只要拷错一行，就不工作了。另外一个是更新的问题，来源的组件更新了，本地的代码也需要更新，可能一个修改就直接导致更新不了。所以前端整个开发的技术还是相对比较原始的状态的，所以 Google 在 2012 年的时候提出了 Web component 的概念，这个概念发展的非常快，W3C 已经把它列入标准的开发范围了，现在已经在推进这个 Web 标准，我们设想未来的 Web 前端开发，应该是基于这种组件式的，所以我们也做了很多组件。Bootstrap 并没有朝这个方向去走，它更多的是强调它这种框架的底层架构，而我们强调组件。而且这种组件是非常具有本土化特质的，比如说我们上面有百度地图的组件，国外用 Google。我们的客服的组件，都是第三方部分提供商，或者一些视频播放的组件，视频播放组件可能会播放土豆优酷的视频，在国外会用 Youtube。未来我们希望做成类似于 Node.js 的 NPM 的包管理的系统，程序开发者需要什么组件，一个命令行直接就下载下来了。

说到 Amaze UI 对待开源的认真态度，版本路线图的规划也是一个例子。陈本峰介绍说：

我们是比较严格的按照国外比较先进的开源项目的运营方式，比如说我们会找 Bug，去分级，分成 P0、P1、P2、P3、P4。P0 是前面要解决的，P1 会在下一个发布版本会解决，P2 我们会在下一个版本有时间的条件下去做，没有时间会往后推。P3 属于这种未来可以讨论头脑风暴的，用户提交上来一个 Bug，一个 issue，我们马上就会做一个级别的判定，这样子提这个 Bug 的开发者会知道，他的这个问题大概是会在什么时间阶段会被解决，而不是说就是大家提上来了，我们把所有的 Bug 拢在一起，而是清清楚楚去把问题分类，确定会在哪个版本解决。版本规划也是，每一个版本的工作重点都分的很清楚，有很清晰的规划图，清晰的 Bug 管理系统，让开发者觉得这个项目比较靠谱，认真对待的每一个版本，很认真对待用户



提出来的每一个问题的，而不是含含糊糊的，让用户根本没有期待。

## 开源亦能双赢

目前国内很多企业做开源都是抱着试试看的态度，那么开源仅仅是做活雷锋吗？陈本峰认为从商业角度，Amaze UI 和云适配也是受益的：

- 本身云适配业务是让用户把网站转到手机端，所以我们对兼容性、适配性是非常关注的。但是单凭一己之力，是没法做到兼容性非常完善的。我们开源出去，如果这个产品有别人用，那别人也来贡献代码，这样也能够反过来帮助云适配这个产品，能够做到更好的兼容性、适配性，对我们产品的提升是有直接帮助的，所以我们愿意去做这件事情，也值得投入做这件事情。
- 做招聘，Amaze UI 的开发者全部是前端的开发者。我们去招人的时候，就在 Amaze UI 上打广告，大家会觉得 Amaze UI 这家公司前端工程师，是一个不错的选择，就会愿意来，成本跟猎头费的成本也差不多。从这点来说，对我们就是招人肯定是有帮助的。目前 Amaze UI 在招募技术爱好者，也欢迎大家参与。
- 对于公司这个品牌来说，如果我们做了一个全中国最流行的一个前端框架，那么云适配以后做跟网站相关的一些业务，肯定会得到别人的认可，这也一个品牌上的关注。所以我们会去做这件事情。对于另外一家公司，他可能就没有这么大动力了，比如说如果是一家做电商的公司，他可能不会有那么直接的帮助，那招人直接花猎头费了，他也不做网站相关的业务。可能跟云适配自己本身这个特定的业务是非常相关的。

## 开源的未来出路

开源项目有哪些出路？陈本峰分析了国外的例子：

开源项目参与模式现在在国外是比较成熟的，基本上国外 2B（To Business，面向企业，以下简称 2B）产品的公司基本都是做开源的，我觉得他们的这个商业模式有几种，一个就是做收费技术支持，然后就是做培训，我们在做的过程中已经有这种参与，已经有人找过我们去给客户做收费培训。技术支持也可以做，这是比较容易看到的。还有就是去做一些系统集成的解决方案的，像 IBM，IBM 做了这种大量的开源，像 Java、Eclipse，基本上做这种解决方案，当然解决方案里面利用最高的还是它的硬件。当 IBM 的软件不是主要收入来源的时候，他就愿意去做开源的软件，加上自己的硬件卖出去。像 Google 做开源的目的是通过开源这种方式促进各



种人去使用互联网，越来越多的人使用移动互联网，他的移动搜索就赚钱了，为什么 Google 会去做一个开源的浏览器 Chrome？

Google 的商业模式在于流量变现，只要世界上有越来越多的人上网，就有越来越多的流量，那他就能变现，这是 Google 的一个商业逻辑，那这些都是跟他支撑业务是有关系的，如果纯支撑业务没关系，那就是培训，还有技术支持，Redhead 就是这种模式。

谈到对未来国内的这种开源发展趋势的看法，陈本峰对 2B 市场的开源报以比较大的期望：

我觉得开源在国内市场里比以前是要好很多了，市场繁荣多了，今天的 Github 上面有越来越多的中国开发者出现了，随着这个行业的发展，未来开源这个事情会在国内会越来越流行。当然我觉得可能主力应该还是那些大的互联网公司，因为国外主力也都是像 Google、Facebook 和雅虎这些公司。现在还处在一个萌芽期，哪一天它真的能爆发，就是看这些大佬们在这块开始发力的时候。可能也是因为在中国做 2B 的大公司是非常少的，国外这种 2B 的大的上市公司是非常多的。在中国整个 2B 的企业还没有完全起来，这个也限制它开源时期的发展，为什么呢？因为首先这个企业有很多内部系统的集成的需求，如果不是开源，他没法知道这个产品是不是跟他内部的现有的这些产品能够很好的融合在一起。那你开源之后，他自己先拿过来研究一下，是不是结合的好，所以这个时候，把产品开源，其实是一种变相的推广手段。第二个考虑到一些安全性的问题，开源之后客户也可以消除对安全性的隐忧，像我们云适配也是，它也是针对企业，是个 2B 的产品，我们基本上也还是按这条路来走的，所以我觉得国内开源能够飞速发展，就是有两个条件，一个就是 BAT 一些大公司开始介入、开始投入，第二个就是国内 2B 的公司开始参与进来。

在本文发布之际，Amaze UI 发布了 2.0 正式版，感兴趣的读者可以访问其[官方网站](#)了解详情。



# Android 5.0 技术新趋势

作者 范怀宇

由于 Android 的版本分裂比较严重，整个新系统升级可能需要一两年甚至更长时间。所以目前使用 Android 5.0 的大部分是喜欢尝鲜的用户，同时现在市场上能够很好支持 Android 5.0 的应用又很少，如果开发者能捕捉这个机会，从这些用户那里得到更好的反馈，或者更好的证明机会，都是非常有价值的。

在 Android 5.0 发布之后，我认为有几个地方可以让开发者做得更好，比如可以建立一个更好看的应用，或者能够解决在应用中最大的性能问题——电量消耗的问题，等等。同时 Android 5.0 也带来了更多的可能性，让开发者可以做出不一样的应用，以及能够在更多的设备上部署服务。下面我分别介绍一下我对 Android 5.0 可能带来的技术趋势的看法。

## 更美的应用——Material Design

在 Android 5.0 的宣传中强调最多的一个亮点就是「Material Design」，从设计语言来讲，这个特点是能在将整个素材铺平的同时还遵循一定的物理材质的需求。这样的设计可以让应用感觉更活泼、具有更丰富的颜色，有无处不在的阴影，以及动画效果更真实等等。对于技术来说，Material Design 解决了两个非常大的问题，其一是「阴影」，它所有的阴影都是默认系统实现的——只要配置 Z 轴的高度，所有的阴影都可以通过系统默认实现。

另一个是「动画」，可以说 Android 5.0 将动画应用到了各个角落，实际上在这方面 Android 做了相当多的技术工作，使这个效果不是简单的贴图，实现出来的效果有点像游戏里面的投影，也就是真实的投影效果。

为了实现这些效果，Android 曾经尝试过不同的方案，最后使用的方案是当 Z 轴比较矮的时候，使用一个「面光源」，相当于一面的光打下来，如果 Z 轴比较高的时候，可能会加一个「点光源」，尽可能使整个投影变得真实，当然它没有办法支持在图形学中比较复杂的「光线追踪」——连续的反射效果，因为有个试验结果是只要开启「光线追踪」 10 秒就会导致手机变得滚烫，以现在的手机性能还没有办法很完美地支持。

而在这一技术的背后有个发展过程，大约是在 2001 年的时候，Android 开始做硬件加速技术，它的计算同样以硬件加速，会在 GPU 上进行差值计算，



用以实现阴影特效，动画也是同理。现在 Android 5.0 将动画做到无所不在，比如给一个页面做了阴影，阴影的抬升有动画，界面的切换有动画，每一次点击、每一次拖动、每一次滚动都有动画。

现在最典型的动画是「水波纹」，现在 Android 5.0 上点任何东西都有一个水波漾起的动画效果。当然，开发者也可以指定各种各样的动画，比如一些基于路径的动画和界面切换时的动画，整个动画的实现技术也是依赖于 Android 在两年前做的一个后台的独立渲染线程，它能够实现异步渲染这些动画。

对于老版本而言，如果想要实现类似的阴影效果或动画效果，让整个界面变得有动感，单靠自己的能力做是特别难的。Android 5.0 提供了所有的这些开发支持，只要用 Android 5.0 SDK，用一些支持包，就可以配上它的风格、调色板、以及一部分控件，配置一些动画，就可以很快地建设出一个特别有「Material Design」风格的界面。现在真正支持「Material Design」的应用还比较少，如果用这样的方式来构建自己的产品，其实可以给那些偏 Geek 的用户一个特别好的感觉。

## 更省电的应用——Project Volta

大家以前做 Android 开发的时候可能涉及最多的问题就是「为什么你的应用那么耗电？」其实整个耗电模型在 Android 模型里的计算是非常复杂的。

Volta 这个项目就是要在 Android 5.0 解决耗电问题。这次的解决办法还是很特别的——就是给开发者提供了能力，如果开发者能够很好利用这些能力，就能在系统上面跑得更快。Android 提供的能力是新的后台任务系统，非常像 iOS 系统，可以配置一组触发条件，比如网络变化、电量消耗到了什么程度、设备是不是进入了休眠状态，这些东西可以触发一个后台任务，这个后台任务的执行也是限时的，如果在时限之内没有完成，系统会将其取消，这个过程特别像 iOS 整个的机制。它可能取代了类似常驻后台、定时任务或者需要更复杂的策略才能保持后台运行的东西，实现更省电的需求。

更重要的内容是 Project Volta 提供了一组调试工具，因为以前可以在 Android 上面调试内存、调试界面等等，唯独很难知道电量耗在哪儿了，这个调试工具加上可视化工具可以展现你的界面上各个元器件以及各个环节的耗电情况，例如在哪个时间段的耗电量是最多的，这时候有什么应用在运行，什么设备是大量耗电的。通过这样的分析，能够更好地了解应用为什么耗电，帮助开发者解决问题。

## 系统服务

### Screen Capturing



每次 Android 新版本升级都伴随着一件事就是将原来只有系统有的能力开放给了开发者，开发者可以用这些能力来建设不一样的应用。比如豌豆荚做过的一些尝试，我们想做一些游戏截屏、录屏或者是一些 OCR 的分析，目的是想分析图片里的一些的东西。这在 Android 老的版本里是不支持的，除非 Root 才能用的这样的能力，但是新的系统服务已经支持了截屏和录屏。这样的话，如果发挥你的想象力，要做一些游戏录屏或者其他应用，这就提供了一个新的机会。

## App Usages

应用的使用信息也是一样的，我们原来想知道手机上那个设备用户最喜欢用哪个应用，什么时候用，这些信息都没有，现在 Android 5.0 提供了这样的接口，开发者不仅可以知道用户在各个应用上耗费的时间，甚至可以知道什么时候用什么时间，整个切换事件都是可以查到的，这些信息原来也只有 Android 自己知道，在这个版本里这些信息全部开放给了开发者，这样能够更好地发挥开发者的想象。

## Recent Screen

新的 Recent Screen 取代了原来的 Recent Tasks，特点是原来就像一个最近使用的应用列表，现在相当于最近使用的页面列表。对于豌豆荚来说，我们希望用户不只是看到自己什么时候用了豌豆荚，而是看到用户在豌豆荚里浏览了三个应用，可以随时回到那三个应用里面，类似这样的功能。其实这样可以给开发者提供一个更好的用户入口，让用户回到你的应用时更方便，或者能力更强。

## 更多设备支持

Android 5.0 在更多设备上都有了支持，不仅在了手机上，现在可以伴随着 Android Wear、Android TV、Android Auto，这三个现在都有 SDK，整个发展状态还都比较好。

Android Wear 已经有了很多 App，Android TV 上也有很多游戏和应用，Android Auto 发布了第一版 SDK。比如说在 Android 5.0 里面增强了通知栏，用户看到手机上是一个普通的通知栏，但同样的通知栏到手表上去看，比如打开 Android Wear，发现可以上下翻动、左右翻动邮件，在这样的设备上能获得一个更好的体验，同样它提供了 TV input Framework (TIF) 框架。

比如对于流媒体来说，可以非常快速地把内容部署在 Android TV 上，开发者做一个 Android TV 的 App，用户可以在 Android TV 上用这样的内容。



同样，通过新的 MediaPlay Session 接口，可以方便地把用户手机上的流媒体放到汽车上，可以实现一进汽车就自动播放、在汽车的车载系统上控制手机的内容，这些新的接口其实都是提供了很多新的机会。现在这样的 App 还偏少，比如目前看 Android Wear 的 App 可能做得最多的就是卖各种各样的表盘，实际上这个方面还有很多的可能性，当然它是一个特别新的领域，存在着一定的未知性，如果现在看这样的机会，可能给你提供的是很多新的产品机会。

以上就是我感觉目前 Android 5.0 对于移动开发需要关注的点，总体来说，如果需要做一个新产品，开发者需要看到一些新的技术可能性，看到一些新的技术的趋势，希望这个分享能够对从事开发的朋友们有所帮助。

---

## 作者简介

范怀宇，豌豆荚应用平台技术负责人。范怀宇 2011 年初加入豌豆荚，先后负责过豌豆荚 Windows2.0 版本、豌豆荚云服务等业务，目前负责豌豆荚应用平台和基础技术相关业务。范怀宇长期专注于 Android 相关技术的研究，著有《Android 开发精要》一书。



# .NET 开源核心运行时， 且行且珍惜

作者 崔康

## 背景

InfoQ 中文站此前[报道](#)过，2014 年 11 月 12 日，ASP.NET 之父、微软云计算与企业级产品工程部执行副总裁 Scott Guthrie，在 Connect 全球开发者在线会议上宣布，微软将开源全部.NET 核心运行时，并将.NET 扩展为可在 Linux 和 Mac OS 平台上运行。.NET 核心运行时将基于 MIT 开源许可协议发布，其中将包括执行.NET 代码所需的一切项目——CLR、JIT 编译器、垃圾收集器（GC）和核心.NET 基础类库。此外，微软还发布了 Visual Studio Community2013，这是 Visual Studio 的一个新的免费版本。

关于此次事件的报道和评论，已经有了很多版本。不过，最近经常有朋友问我的看法，索性就梳理了下思路，把想到的可能影响和启发都写了出来，博大家一看。

---

## 不是终点

记得.NET 宣布开源的当天，编辑群里就一阵骚动，看得出大家还是很激动的。笔者最初也是吃了一惊，不过后来静心一想，这也是水到渠成的事情了。从时间轴看，近几年，.NET 家族已经先后开源了多个成员，包括 ASP.NET、C#编译器等，再加上这次的.NET 核心运行时，至此，服务器端框架已经全部开源（或者很快开源）。所以，这次事件只是把微软的.NET 开源战略推上了高潮，但肯定不是结束。

宣布开源是个新起点，如何把开源社区构建起来，生态系统培育起来，才是关键。开源需要精心的运营，每年的开源项目不断涌现，而在不断消失。.NET 开源项目不需要考虑失败的问题，开发者关注的是跨平台支持何时落地、开源社区如何运转起来。



## 大势所趋

开源是大势所趋，就在 5 年前，各大公司和厂商还在讨论是否应该开源，但现在大家讨论的则是如何更好地开源。随着互联网时代的发展和成熟，开源已经成为标配。从 Github 的 [.NET Core](#) 开源项目来看，目前已经有将近 6000 个 star，考虑到开源一个多月的时间，这样的成绩已经说明开发社区对微软开源的支持。

以前，开源是一种保障软件项目生存的方式，而现在很多开源项目已经成为企业的一种商业和运营策略，生存已经不是问题，如何活的更好才是关键。就像某 I 做 Linux 开源是为了卖服务和硬件，Google 做 Android 开源是为了获取移动互联网入口一样，微软做.NET 开源也是为了其“移动优先、云优先”战略利益服务的。看到网上有人说开源是微软的阴谋论，笔者不是很赞同。现在 IT 界玩的都是阳谋了，为了公司利益无可厚非，只要不损害社区的利益，更何况开源本身是件好事情。

## 曾经的两强争霸

在 2005 年以前，Java 和.NET 平分秋色，由于微软的强大支持，在社区等很多方面.NET 甚至比 Java 更出色，毕竟 Java 在曾经的 Sun 的领导下并没有太多亮点，多亏了开源支持，才保持着鲜亮的生命力。当时，在企业级开发领域，Java EE 和.NET 都已经受到来自开发社区的诟病。但是随后而来的移动互联网浪潮给了 Java 另一次生命，谷歌开源安卓系统，借此掌控移动互联网入口，其开发语言 Java 也成为众多移动开发者的首选，要么 iOS 上的 Object-C 语言，要么安卓的 Java，而且大部分是两者都要。微软则选择了一条相对封闭的策略，即使联合诺基亚，Windows Phone 的市场占有率依然不高，这直接导致.NET 的使用率也不高。

笔者还记得，在两强争霸的年代，国内少有的几个技术论坛中，.NET 和 Java 两派吵得不可开交，立场鲜明，从工具支持吵到语言特性，又吵到函数库支持。现在想想，多个竞争者并存的年代对开发者是好事，有比较有选择，而且可以让竞争者倾听开发者的声音。

## 跨平台支持

此次.NET 开源和对 Linux、Mac OS 的支持，为移动开发者提供了新的选择，未来可以不再依赖于 Java，.NET 支持的 C#、C++ 等可以成为移动开发语言。作为后来者的 C# 语言的优势明显，Java 一直在追赶，现在有了跨平台支持，开发者可以自由选择。



从.NET 开源实现项目 Mono 的博客上可以看到，Mono 计划与.NET 团队合作，把.NET Core 的代码融到 Mono 项目中，同时把 Mono 中的平台依赖代码贡献给.NET。

相比 Java 体系，目前.NET 只开源了服务器端框架，而 Java 开源的是全部系统，包括客户端和开发工具。未来微软开源的路是否更加宽广，值得观察。

## 培养生态系统

生态系统的培养需要一段比较长的时间，Java 开源将近 10 年了，才形成如此成熟的生态圈，.NET 需要有耐心和执行力才能走得更远。既然是生态系统，那么一定要制定并维持好生态系统中的游戏规则。这个系统不单单是开发者，还需要包括他背后的就职公司、他开发的产品面向的客户、开发者的合作伙伴等等，当然还有微软。一个平台想要凝聚力，不外乎从物质和精神两个方面分析。物质方面，开发者能够从这个平台上获取利益，比如这个平台是支持开发者创业的、多语言跨平台的、具备快速部署优势的、适合迭代开发的，生态系统的成功案例多，具备这些特质，会让这个平台的开发者最终受益。从精神方面，要让开发者有参与感，他在开发方面的经验和知识能够通过生态系统分享出去，影响其他人和公司，并且他的意见能够得到及时的反馈，那么这种参与感会吸引开发者。关键点包括创业孵化器、制定和维护生存法则、塑造合作共赢的业务模式、让圈内的利益相关者受益。

最近几年，微软对于开源越来越支持，但是给开发社区的形象还没有转变过来。毕竟，微软已经做行业老大很多年，其在闭源方面的印象给开发者影响太深刻了，很难在短时间内扭转。需要持续的运营投入和影响。

其实微软过去在社区运营方面很有经验，包括各种线上线下的活动，还有 MVP 等奖励机制，在开源方面，微软可以继续从前的动作。有两点建议：一是重视本土化，关于微软开源项目的介绍、进展、分析和案例要及时告知中国开发者；二是培养成功案例，榜样的力量是无穷的，开源之后的赢家在哪里？甜头在哪里？树立这样的榜样，开发社区都会看到。

## 开源选择

开源的历史很久远了，以 Java 为例，开源快 10 年了，它的服务器端、客户端、开发工具都是开源的，虽然历经了 Sun、Oracle 等公司的发展，但一直保持旺盛的生命力。特别是在移动互联网崛起之后，Java 获得了新生，发展迅猛。开发者在选择开源平台和项目中，最担心的不是这个项目能不能满足我的需求，是这个项目靠不靠谱，会不会一直做下去，这是最关键的。这就是为什么绝大多数成功的开源项目背后都是有大公司的支持，有很多人全职做开源项目，人员稳定，路线图清晰，开发者放心，敢用你的产品。其次才是满足需求，赢得信任感。



开源的好处是，吸收整个技术社区的力量促进开源项目的发展和竞争力，第二个好处是，提高开发者的参与感和信任度，有利于吸引开发者，构建良好的生态系统。劣势，对于一些没有厂商支持的开源项目，路线图不明确，有纷争，存在分裂或者失败的机会。对比.NET 开源，开发者不必担心这个项目会半途而废，可以更关注项目本身的发展和特性。

## 云优先战略

今年，Azure 成为首个在中国落地的全球公有云平台，其支持 Linux 和很多其他开源软件，这本身就说明微软目前的态度非常开放，从 Windows 到 Azure，战略意图很明显。从全球来看，Azure 占全球 PaaS 市场份额的 64%，排第一。微软开源的态度，可以吸引更多开发者使用 Azure 平台，毕竟 Azure 不是一个微软专有技术的平台。

## 未来，且行且珍惜

笔者对.NET 开源抱有比较高的期待，在云计算、大数据、移动互联网和物联网的时代，Java 一家独大不利于社区的发展，微软已经不再是过去封闭的巨头，开始拥抱开源，翻看微软过去的历史，我从不怀疑微软的强大的执行力，时间是一个关键的因素，开发者需要看到微软在承诺开源之后的迅速行动，开源社区和生态快速搭建起来，跨平台官方支持版本快速发布出来，抓住移动互联网的契机，把整个.NET 新局面打开，千万不要虎头蛇尾，且行且珍惜，开发者是最聪明的。



# 天猫 11.11：搜索引擎实时秒级更新

作者 郭蕾

搜索是很多用户在天猫购物时的第一入口，搜索结果会根据销量、库存、人气对商品进行排序，而商品的显示顺序往往会影响用户的选择，所以保证搜索结果的实时性和准确性非常重要。在电商系统中，特别是在“双十一”这样的高并发场景下，如何准确展示搜索结果显得尤为重要。在今年的“双十一”活动中，InfoQ有幸采访到了阿里巴巴集团搜索引擎的三位负责人仁基、桂南和控傅，与他们共同探讨了搜索引擎背后的细节。以下内容根据本次采访整理而成。

阿里巴巴的搜索引擎承担着全集团的搜索业务，包括淘宝、天猫、1688 等系统，对比传统的搜索引擎，阿里集团的搜索引擎有一些比较大的突破性、创造性的工作。传统的搜索引擎，只可以做到离线全量、增量构建索引，而阿里的搜索引擎已经是演变成一个能够做到离线、增量、实时三个等级的搜索引擎。电商平台最大的一个特点就是短时高并发，像双十一这样的活动中，搜索引擎需要考虑如何让流量发挥更大的价值。传统的搜索引擎解决短时高并发的思路是添加缓存层以减少搜索引擎的访问量，而这样的解决方案，天猫之前也有使用，但是缓存会有延迟，实时搜索的需求根本无法满足。所以为了解决实时的问题，阿里的搜索引擎去掉了应用层和业务层的缓存，重点优化和提升引擎层的服务能力。为了兼顾实时性和吞吐量，搜索引擎实现了全量、增量、实时三种更新通路。通过三种方式的灵活组合，在保证了海量数据定期全量更新的同时提供了秒级实时更新能力，避免了数据延迟，提升了用户体验。

## 系统架构

从整体上来看，阿里搜索引擎的架构图如下。从上到下，分别是应用层、业务层、搜索引擎层、离线处理层和 DB 层，应用层其实就是调用方，大的来看可以分为 Web、App、Wap。业务层会针对相应的业务对搜索结果进行整理，如 Android 和 iOS 的搜索结果显示是不一样的。搜索引擎层有点类似传统系统的搜索引擎，阿里巴巴的搜索引擎会在搜索的基础上根据用户习惯提供个性化的搜索结果。索引层主要包括全量索引和流式计算，全量索引其实就是一个基于 Hadoop/HBase 的离线集群，而流式计算是阿里自己研发的一套系统。之所以没有选用 Storm，是因为在这一层中，光有计算是不够的，



还需要有数据的存储（开源解决方案 HBase）。如果使用 Storm，接下来会面临一个问题，Storm 是一个集群，HBase 又会是一个集群，这样，Storm 的 Disk 以及 HBase 的 CPU 其实都没有充分利用到，所以阿里的方案是 Hadoop Yarn 与 HBase 混合部署，把两个集群合并在一起，既可以做大规模的数据处理，也可以做流式计算，通过这样的方式，可以将离线和实时计算更好地融合。最底层的数据源层会把用户、商品、交易信息同步到上层的 HBase 集群中。



## 流式计算

Storm 是一个无状态的流式计算框架，而无状态的流式计算体系，更适合做简单的统计分析，比如针对成交维度或者点击维度做计数。而阿里自研的流式计算框架 iStream，已经不再是简单的、无状态的流式计算概念。iStream 借助 HBase 集群存储用户状态，以完成一些相对复杂的模型的计算。同时模型的计算结果可以通过相应的接口直接推送到上层的搜索引擎中，以服务每一条流量的排序变化。

## 排序链

在搜索引擎层，不仅包括商品的搜索引擎，还会包括其它层面的服务（如架构图所示）。商品搜索引擎中包含商品、店铺、活动等维度的信息，而图中的个性化服务旨在为用户提供个性化的搜索体验，个性化服务会根据用户的



实时行为反馈搜索结果。而 QP (Query Planner) 会对用户的搜索请求进行分析（搜索词、搜索场景、页面）进一步个性化搜索服务。在搜索引擎层，通过这三个系统的互相配合为上层的业务层提供个性化的搜索数据。

不同的业务对应的搜索排序结果不同，阿里搜索引擎中排序部分是通过类似链式处理的方式实现的，内部称为排序链。排序链是由不同的用户特征对应的算法插件组合而成，算法插件是单独存在的，可以根据具体情况组合到不同业务的排序链中。目前在线上运行的排序链有几十条，系统会根据不同的业务、用户、场景、Query 选择不同的排序逻辑

## 双十一优化

而在双十一这样的高并发活动中，搜索引擎需要保证流量的合理分配，比如搜索结果中不能显示售罄的商品。但是对于一些热门商品，从库存充足到售罄可能是几分钟的时间。为了保证搜索结果的实时性，阿里搜索引擎架构针对这样的场景做了优化，去掉了不能感知业务变化的缓存（业务层），重点优化搜索引擎层的缓存。以商品售罄的场景为例，当商品售罄时，业务系统会发送异步消息通知离线集群，离线集群通过流式计算将更新同步到引擎，而当引擎返回搜索结果时，会在缓存的基础上对结果进行二次过滤，从而保证搜索结果的实时性和准确性。

另外，在今年双十一中，天猫搜索底层第一次使用精确到更新粒度的 SKU (Stock Keeping Unit) 引擎代替之前的宝贝引擎，底层引擎索引量较之前翻了几番。天猫从召回逻辑、前端的属性展示、筛选以及搜索结果页到详情页的联动，向用户提供了精确度更高、更细致的搜索购物体验。对于标类产品，基于 SKU 引擎的搜索导购缩短了用户的搜索购物路径，比如搜索 iPhone 5s 后，SKU 引擎会显示对应的销售属性，方便用户选择。此外在 SKU 引擎的基础上，天猫还实现了用户的尺码个性化，在包含确定尺码信息的类目中，如鞋、文胸，天猫可以匹配用户的尺码个性化信息，将适合的商品优先展示给用户。InfoQ 会在后续文章中与相关技术专家剖析 sku 引擎的设计思路与实现，敬请期待。



# 小米 11.11：海量数据压力下的推送服务

作者 臧秀涛

11.11 大促，随着移动端业务量的急剧提升，像小米推送这样的基础服务也经受了巨大的考验。11月12日，小米的项目总监汪轩然在[微博](#)上宣布，“小米推送服务共发出 9.65 亿条消息，平均每分钟发送 67 万条。更值得一提的是，后台监控显示，推送服务后台系统在全天运作非常平稳，没有任何卡顿拥堵现象，让各种促销、返利、订单更新消息第一时间触达用户。”

汪轩然，2007 年毕业于清华大学计算机系，后加入微软亚洲工程院，曾参与 WP7 上的浏览器的开发。2010 年 7 月加入小米，曾担任米聊安卓团队的团队主管，现在在小米任项目总监，负责小米的开发者服务，掌管推送服务、统计服务和移动广告联盟三大业务，旨在为小米搭建一个移动 App 业务的互联网生态圈。

我们联系了汪轩然，就小米推送服务的架构、特点、性能等问题对他进行了采访，以下内容根据本次采访整理而成。

## 基础技术架构

**协议是推送服务的核心。**小米推送服务所采用的协议是由之前的米聊演变过来的，而米聊从一开始就选择使用 XMPP 协议，之后开发团队对 XMPP 协议做过几轮精简和重构。现在 XMPP 部分只是作为一个数据的传输层，之上跑着各种独立的业务，每个业务称为一个“channel”；每个 channel 上跑的数据格式可以是不一样的。消息推送服务是其中一个 channel，这个 channel 上传输的数据是通过 Thrift 进行二进制化的协议格式。

再来看一下小米推送服务的服务端架构。下图是后台服务端的一个基本架构图。整个服务端包含如下几层：

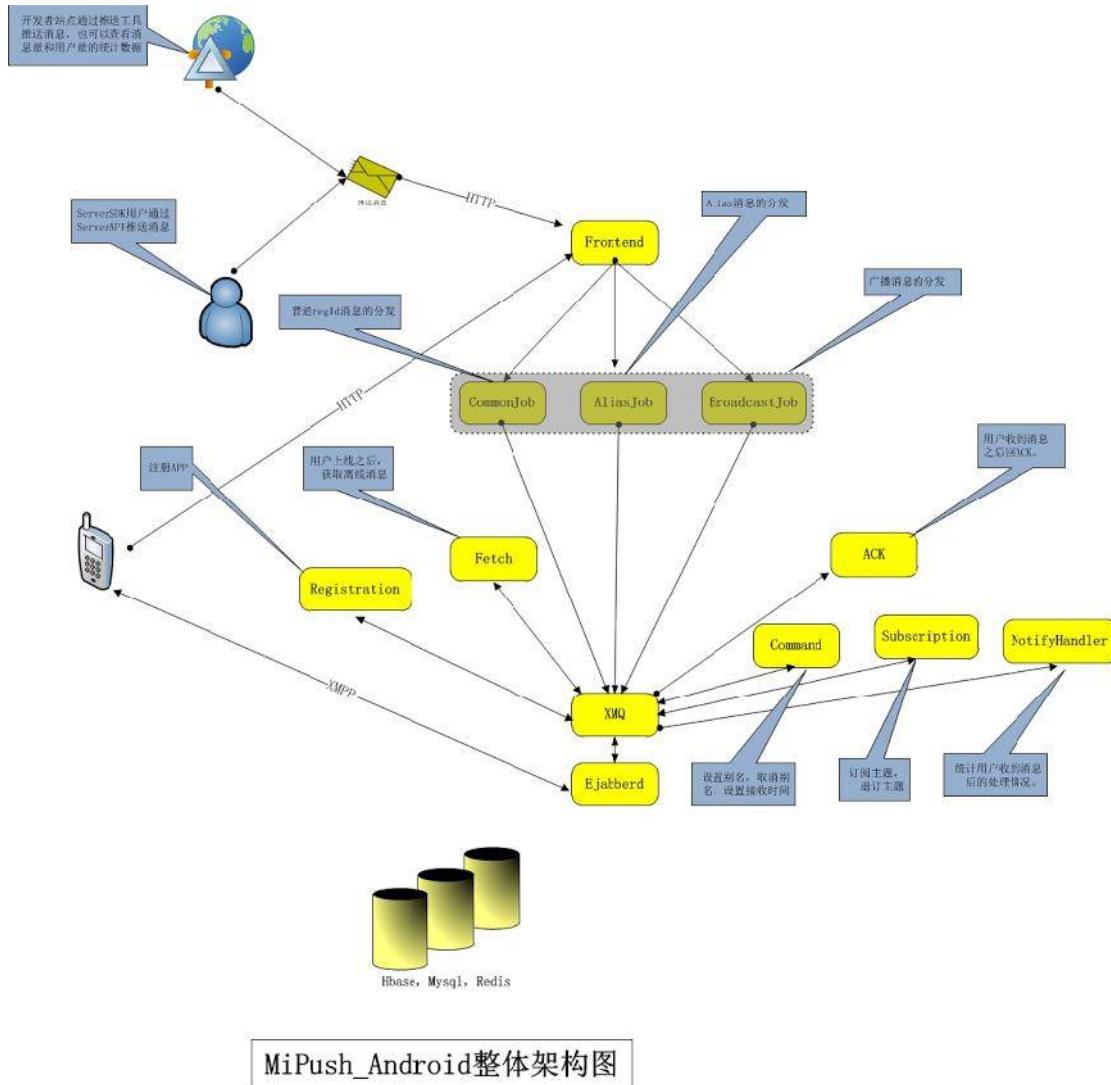
1. **XMPP 前端：**用于维护跟客户端之间的长连接，使用 EJabberd 项目来处理来自客户端的 XMPP 请求，同时通过 XMQ 模块来处理推送服务特有的 XMPP 消息协议。



## 专题 | Topic

2. 中间层：业务逻辑层，主要用于将消息请求异步化、创建和维护消息队列、以及处理客户端的一些命令请求（注册、设置别名、设置 topic 等）。
3. HTTP 前端：这一层负责对接来自第三方 App 的服务器的发消息的 HTTPS 请求，以及来自客户端生成账号的 HTTPS 请求。

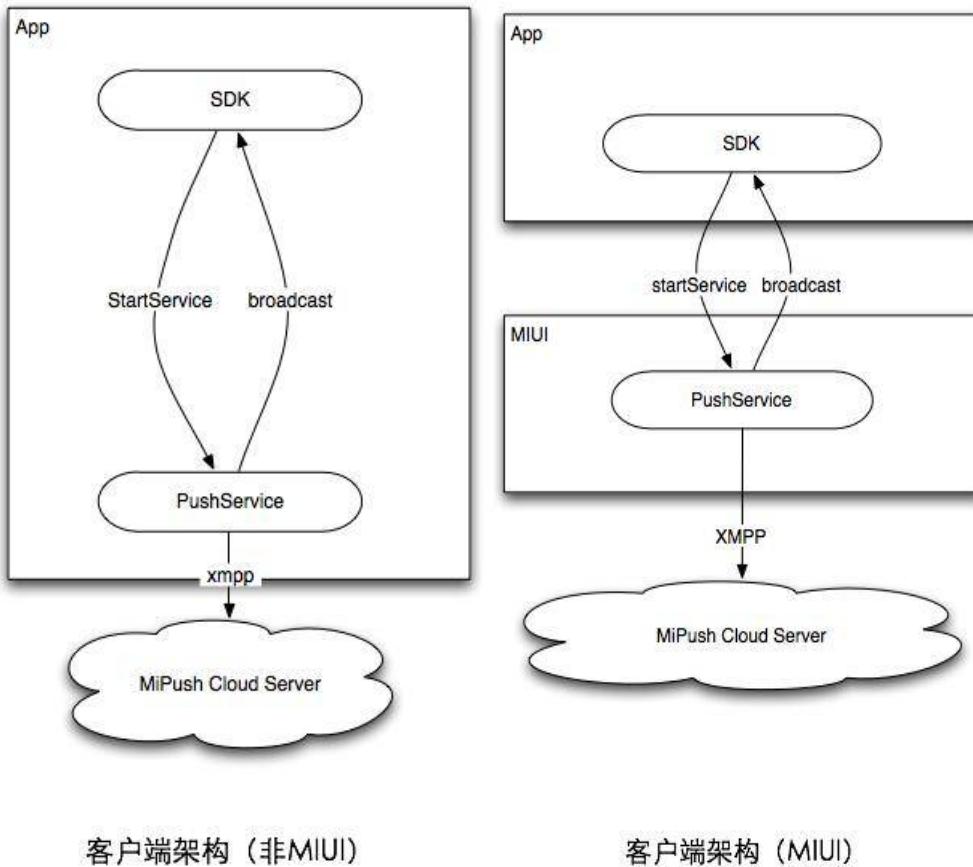
再就是数据存储，这里采用了小米的统一 HBase 存储，同时还使用 MySQL 来保存一些量不大，但需要复杂过滤条件的数据（topic 等），并且为了降低对 HBase 的压力，中间还加了一层 Redis 作为缓存。



最后看一下客户端架构。客户端 SDK 主要包含两个层次：SDK 层和 PushService 层。前者提供了面向 App 接入的接口、回调方法以及对 Thrift 的数据进行反序列化的处理逻辑；后者用于维护 XMPP 长连接和收发消息。两层之间使用 Intent 方式来传输数据。值得一提的是，在 MIUI 系统上，



PushService 层是系统共用的，即 MIUI 系统提供了一个统一的 PushService 管理模块，不需要每个应用单独启动自己的 PushService。



## 功能实现

小米推送服务支持单发和群发消息两种推送方式。单发消息支持针对 regID 和别名两种方式，regID 是小米推送服务后台根据设备标识+appID+时间戳生成，为了减少设备碰撞概率，设备标识我们采用的依据是 imei+AndroidID+build 序列号。别名是 App 在客户端设置上报的，便于应用将自己的设备/用户标识符同我们的 regID 作关联，这样 App 就不需要在后台维护 regID 跟设备/用户的对应关系了。群发消息采用打标签的方式来区分，客户端和服务端都可以给指定设备设置标签，发消息的时候，只需选取指定标签发送即可，小米推送后台会将标签所对应的设备展开。一个标签支持的设备数无上限。

那小米推送服务的**稳定性**是如何保证的呢？小米推送服务采用多机房方案，平时流量均摊，一旦某个机房出现故障，流量无缝切换到其它机房，并且单



一个机房的容量能保证提供无损服务。目前是双机房部署，预计明年会扩展第三个机房。

**安全性**也是小米推送服务重点考虑的一个因素。数据传输过程中，得益于推送服务采用的双层协议方案，消息会采取双重加密，第一重是 XMPP 传输层，保证数据在网络传输的过程中不会被篡改、监听。第二重是在 Thrift 二进制层，用以保证消息到达 Service 之后，通过 broadcast 发送给 App 进程的过程中不会被截获和伪造。第二重加密往往会被其它第三方推送服务忽略，但其风险同样很大。

## 性能指标

11.11 大促，所面对的请求量是在小米推送服务的设计容量之内的，目前设计和机器规模可以支持峰值每分钟 1000 万条消息；平时业务量至少每分钟 40 万，峰值每分钟 600 万条消息。

推送消息量平时波动很大，所以开发团队准备着流量随时可能忽增 200% 的情况，并在线下做好压力测试和优化；如果流量特别大，还有以下应对措施：

1. 异步排队处理，此时消息送达时间可能会比平时稍慢，但不会对整个系统有太大冲击；
2. 消息有优先级，广播消息会以低优先级处理；
3. 限流，控制开发者发送消息的频率；
4. 扩容，如果机器负载过高或者某个服务有瓶颈，可以很快速地增加机器，部署服务，增强系统处理能力。

## 小米推送服务所经历的重构

软件系统在开发和演进过程中，经常会经历较大规模的重构。小米推送服务有两次比较大的重构。

一是开发语言从 Erlang 转为 Java。小米原来的消息系统是使用 Erlang 开发的，所以推送系统的第一版也是基于 Erlang；但是 Erlang 的社区不够活跃，开发人员很难找，学习曲线陡，支持工具和类库少，所以后来开发团队选择了使用 Java 重新开发；迁移到 Java 后，对开发人员的要求降低，各种工具和类库较多，大大提高了开发效率。

二是无处不在的 Cache。客户端使用小米推送服务的 SDK，开发者使用 API 的情况千变万化，很多场景是意料之外的；需要对调用频繁的业务添加 Cache，尽可能在本地进程内处理；例如，对于客户端调用 API 设置别名和订阅 topic，先检查 Cache 是否已经设置过，只有没有设置才往后端服务发送；优化后，后台服务的业务压力大大减少。



## 在开发小米推送过程中的一些感悟

1. 服务要支持水平扩展，尽可能实现为无状态，或者使用一致性哈希进行划分；方便扩容，可以保证即使系统暂时有性能瓶颈也能通过加机器解决。
2. 监控先行，能够很方便地采集、分析服务器的负载和业务的请求量、percentile、slow log，能够清楚了解到系统的瓶颈，有针对性地改进。
3. 不要过早优化，先实现功能并尽快上线，根据监控数据对关键地方进行优化。
4. 敏捷开发，快速迭代，日拱一卒，每天都有简短的站立会议，能够迅速响应变化，持续改进系统。



# 天猫 11.11：移动端性能提升两倍

作者 郭蕾

据阿里巴巴提供的数据显示，在双十一开始后的三分钟内，天猫平台的销售额就超过 10 亿元，其中移动端占比超 70%。惊人的数据背后需要有强大的技术做支撑，移动客户端需要保证在高并发场景以及不同的网络环境下为用户提供顺畅的购物体验。在双十一当天，InfoQ 有幸采访到了阿里巴巴无线事业部技术总监南天，并与他探讨了阿里巴巴在移动端方面的技术突破。南天目前负责手机淘宝和移动基础平台的研发建设，在移动方面有非常丰富的经验。

**InfoQ：**刚才大家一直在讨论移动端的交易额，非常震惊。这么大的交易量，能具体说说手淘针对双十一做了哪些优化么？

**南天：**我们确实做了很多的优化，但我觉得不能简单的说是针对双十一的优化，今年手机淘宝（以下简称手淘）发生了很大的变化。对一个网站来说，双十一的挑战在于高流量到来的时候，峰值系统的稳定性、响应能力和可伸缩性。但是对于移动端来说，我们需要保证高并发访问下客户端的流畅性。从产品形态来看，手机淘宝过去就是一个简单的移动端购物类工具，提供的功能大体是搜索、下单、查看商品信息。而现在的手淘进一步的扩大了业务范围，电脑上所能看到的所有产品形态，都会移植到手淘，以满足消费者多样化的消费需求。

当然，随着业务的快速膨胀，代码量也在快速增长，之前的架构已经不能很好的支持业务的快速发展。今年上半年我们做了手淘成立以来最大的一次技术架构的重构，引入了很多服务端架构的重构思想，把整个手淘 APP 重构为一个大的、容器式的开放平台。这样的架构既能够快速集成各个团队的开发成果，也能隔离功能模块，部分模块出问题不会影响整体的 APP 性能、稳定性。随着这样的变化而来的是整个开发模式的变化。手淘从原来集中式的研发，变成了可以做多个



团队并行迭代的研发模式。这样的改变之后，整个研发的效率、研发的质量都有极大程度的提升。

**InfoQ：**您上面从架构、团队等方面进行了剖析，能介绍下技术方面的一些突破么？

**南天：**我们今年针对手淘的消费者端做了很多的性能优化。单从性能方面而言主要可以分为两部分，一个是基础网络优化，由于地区和设备的不同，移动端的网络环境也各不相同。我们针对网络协议做了精简和优化，同时也尝试引入了 SPDY，以最小化网络延迟，提升网络速度，优化用户的网络使用体验。通过对网络的优化，上半年网络的性能提升有近两倍，效果非常棒。

第二个是移动端特有的一些优化，如安装包大小、功耗、网耗。很多用户在下载新应用时，会考虑到安装包的大小。基于之前的架构，我们将手淘改造为一个插件式的平台，平台可以按需加载相应的模块。同时通过图片压缩、代码去重等将 Android 的安装包从 40M 缩减到 27M。从这个点上来说，我们探索到了一个可以持续优化安装包大小的思路，未来手淘的安装包大小可能还会继续往下降，比如降到 20M 以下。除了这个之外，还包括功耗、网耗方面的优化。在每一次版本上线之前，我们会通过模拟用户来模拟真实用户的使用情况，进而监控它的消耗是多少。另外我们会监控应用静默期的电量和流量消耗数据，通过对监控数据的分析，来确定应用功耗和网耗上是恶化还是提升，以决定这个版本要做怎样的改进、要不要如期发布等。

**InfoQ：**我留意到手机淘宝不仅支持新发布的 iPhone 6，并且还支持很多低端的 Android 机，适配方面做的这么好，能不能和我们的读者分享下经验？

**南天：**首先手淘有一个 100 多台真机的适配实验室，针对这 100 多台机器，我们会通过脚本跑自动化的测试来模拟用户的点击，以统计某个版本在不同的机型、不同的操作系统上的运行情况。这一块是自动化的过程。除此之外，我们会扫描 API 在不同机型、操作系统上的兼容情况并记录到知识库，从扫描结果中我们可以看到某个 API 的表现是否符合我们的预期，如果不符预期，我们会杜绝相应 API 的使用。同时我们会监控灰度期用户的反馈。用户使用什么样的机型、网络在什么地方碰到什么问题，我们快速针对这个反馈做改进。



另外，对于一些低端的机型做了特殊处理，我们会单独派发某个稳定版本的 APK，这个稳定版本基本不再会添加额外的功能。同时我们也会配合网络方面的优化来提供一个更稳定的购物体验，比如在 2G、3G、弱移动网络的情况下，可能加载的内容以及交互方式都会不一样。

**InfoQ：**手淘的活动页面有非常多的图片内容，但是在双十一高峰期间用户体验还是特别流畅，这块有用什么特殊的技术么？

**南天：**所有的活动页面，我们都是基于 HTML 5 的技术做的。但是 HTML5 在移动端的体验有很多问题。无论是从渲染还是网络，HTML5 的调用栈都非常长，我们针对其做了很多的优化。比如图片，在不同网络环境下加载的图片其实是不一样的，我们会针对图像的编码做一些优化，给用户一个清晰度更高，但又兼顾用户网络性能平衡的选择。

同时我们也会使用优化过的网络库替代 WebKit 的通讯方式，并优化 HTML5 渲染图片的过程。在手淘中，所有的 HTML5 页面，都是放在自己的 HTML5 容器中，这个 HTML5 容器提供了很多跟 Native 互动的通讯能力，解决了对 GPU 的调用问题，以提升渲染能力。当然，通过 HTML5 容器的方式，页面的兼容问题也会随之解决。

图片部分的处理其实还是之前提到的针对内存按需加载的方式，我们会根据手机内存来确定一次加载多少个图片，剩余的图片可能会做异步预加载处理。同时我们对很多原生的前端控件也做了修改，比如像滑屏的控件，它本身是当滑屏停止时才去做图片加载。而经过我们修改后，它就可以随着滑屏的过程实时加载图片，这极大程度提升了我们的浏览体验。

**InfoQ：**双十一中有很多的营销活动，非常有意思。我想这样的游戏也渐会成为营销的趋势。能介绍下手淘在游戏方面的一些工作么？

**南天：**是的，我们基于 HTML5 的容器实现了营销互动的小游戏。移动端存在很多 PC 互联网不具备的能力，比如传感器、语音交互、图像能力。我们今年的开发过程中，也非常强调如何把移动端的能力更好地和我们的业务相结合。以红包么哒为例，客户端可以扫描人脸，通过图像识别技术，我们能够识别人脸，并将人脸替换为一个红包，以增加营销活动的趣味性。这是在产品层面上的一些创新。



技术上不同的平台会对应不同的技术，比如 Android，我们用了 Box 2D（愤怒的小鸟使用的物理引擎），它相对来说还是比较成熟的，但是使用中我们也碰到很多问题，比如 Box 2D 对外封装的一些 API 并不太友好，所以针对不同的机型，我们也做了很多优化。在 iOS 中，我们使用到了 Sprite Kit，借助它我们可以用很小的成本构建一个重力的世界。可能这些组件都会有很多不成熟的地方，比如会导致应用 Crash，我们也重点努力去解决这些问题。总体来看，游戏方面的技术选型还是以开源或者原生的组件为主。

InfoQ 接下来将会详细剖析阿里巴巴在移动客户端的几个突破性技术，包括网络优化、游戏引擎、HTML 5 容器，敬请期待。另外，读者可以关注南天团队的微信公众号（微信号：AlibabaMTT），他们将会定期分享最新的移动开发技术。



# 京东 11.11：交易系统的 关键技术

作者 崔康

电商的 11.11 大促，既是一场全民运动，也是顶级团队和技术的对决。为了深入剖析 11.11 背后的技术力量，InfoQ 派出了多位编辑亲临各大电商的 11.11 指挥部现场，对一线的技术专家做了各个领域的专访。本篇新闻就是对京东商城技术研发体系交易平台副总监王晓钟的采访报道。

王晓钟介绍说，11.11 大促，基本的原则是保证主要的交易系统没有任何故障，这是多部门合作的结果。运维部门从网络层开始就准备了很多预案和容量规划，负责处理外部的流量，特别是恶意流量的甄别和处理。

## 交易系统瓶颈

谈到交易系统的瓶颈，他认为，随着系统数据和状态的变化，它不是一个固定的点。以线上交易为例，依赖的逻辑分支特别多，包括用户信息、商品信息、价格计算、库存信息、虚拟资产使用情况等等，这些信息都来源于不同的服务，所以整个交易系统的逻辑特别复杂。在前期准备中，京东内部进行了大量的压力测试，尝试找出可能的瓶颈点；在实际运行过程中，工程师密切监控，每个热点都有预案。处理的思路要么是提前准备很多组机器，分担流量，要么是临时开启一些细微的降级，增加一些缓存。

## 监控系统

京东的实时监控系统基本上都是通过日志分析来完成，包括软硬件多个维度，硬件包括 CPU 使用率、网络连接数等等；软件包括某个接口的响应时间、异常抛出的次数等等，当然监控系统也要做 11.11 备战，比如对于重要系统的数据隔离等等。关于响应时间，目前瓶颈都在公网上，国内的互联网质量比较差。从服务器这边讲，每一百次调用中，最差的一次也只有 15 毫秒，但是到了公网上，根据测试，好的也是 100 多毫秒，差的甚至到 1 秒左右。监控系统是京东自己开发的，所有的第三方的东西，它都造一个通用的轮子，京东以前还是一辆大卡车，用通用的轮子就可以了。目前这个业务量可以说



已经是一辆跑车了，对轮胎的要求特别高，所以轮子都自己定制的，适合自己的业务、系统、软件、硬件，包括适合自己的人和管理。

## 云平台与容量规划

京东的底层系统其实分为两块，一块是内部的虚拟云，有不少系统是在用虚拟云系统；还有一部分，比如说交易，像这种交易也有一部分在用虚拟云，有一部分在用硬件，都是不一样的，看业务是否适合。因为云不适合所有的业务场景。比如说有状态的应用，对数据一致性要求高的，它就不太适合。有些像购物车的价格计算，完全没有状态，就适合。

云平台，第一，部署和管理上和以前相比要方便很多；第二，在故障处理上，有很多底层的可以自动切换的功能，合理的调配资源。

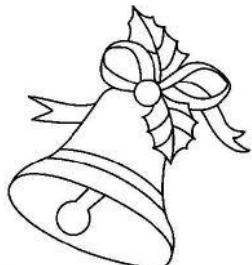
容量规划，主要依靠各个团队对于自己的未来业务的预测，京东的团队主要依靠自己的大数据系统。从大数据团队里获取业务增长数据的趋势。大家对这些趋势进行分析，可以合理的判断自己用户的增长量大致是多少。比如对用户团队来说，关注的是用户的登陆量和注册量，对交易团队来说，关心的是订单的增长量。商品团队关心的每天商品的更新量，还有商品的增长量，每个团队不一样。他们根据自己量的趋势来做自己的容量规划。

## 数据一致性

交易系统数据一致性，交易依赖的用户、商品、促销、库存这些接口，首先没办法做异步，只有同步的来做，而且如果做同步的话，分布式事务是很难绕过去的话题，对电商来说，简直是一个技术上的恶梦。目前京东的做法很简单，提交的时候就做强一致性检查。提交完了，在强一致性检查的基础上，再做异步的一致性检查。某一单如果没有提交成功，那没有一致性问题。只要这一单成功了，系统肯定会保证数据一致性。举个具体例子，交易一旦成功，用户余额如果扣了，那肯定就是扣了。不会存在交易成功，但余额实际上没扣的情况。还有优惠券也是一样的，必须是强一致性的。

## 线上测试

做线上的性能测试怎么不影响其他用户？举个例子，假设线上是两组系统，如果要做线上的性能压测，会把用户导到其中的一组上去。物理上和做压测的那一组隔绝。因为对交易来说，现在交易已经做到分布式交易，各个组之间，包括数据都做了隔绝。如果一组压力过大出现问题，哪怕整组宕机的情况下，其他几组机群还是好的，快速的把入口流量一切换，保证用户体验。这一招就可以用在线上性能测试中。每次线上性能压测的时候，把用户导到



一组隔绝的机器上，用户在上面跑。然后剩下的那几组就用各种工具进行压测，跑出的数据特别真实。

## 秒杀隔离

秒杀系统是今年从主交易系统中剥离出来，服务器和数据系统都是独立部署的，秒杀用的库存和商品信息都是单独推送到秒杀系统里，完全隔离。系统本身又做了很多针对秒杀的优化。



# 蘑菇街 11.11：流量超过预期，系统曾被冲垮

作者 郭蕾

以导购起家的女性时尚电商蘑菇街目前已经上线自己的电商交易平台，全面转型为垂直电商。据官方数据显示，蘑菇街双十一交易额已突破一亿元，并提前完成了此前的预定目标。蘑菇街今年第一次自己做交易，虽说总的交易量只有天猫几分钟的量，但峰值时期系统的压力同样不小，并且他们也才刚刚开始。那蘑菇街是怎么准备这场战争的？相应做了哪些预案？带着这些问题，InfoQ 专门采访了蘑菇街的 CTO 岳旭强。岳旭强于 2004 年加入淘宝，参与并主导核心业务系统搭建，2007 年开始负责淘宝服务化体系建设。2010 年底离职创业，组建并带领蘑菇街技术团队，支撑蘑菇街的高速发展和业务转型，有丰富的架构设计和技术团队管理经验。

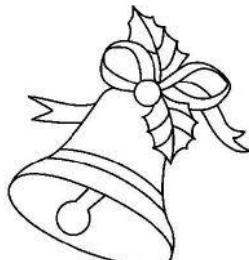
**InfoQ：**请介绍下双十一当天蘑菇街的交易情况，当天有多少的成交额？

**岳旭强：**截至 11 月 11 日 24 时，蘑菇街总成交额超过 3.37 亿，移动端成交占比持续稳定在 75% 以上，客单超过 250 元。双十一当天的成交额近 2 亿，虽然也就天猫几分钟的交易量，但是我们还是很激动，今年也是蘑菇街新的开始。

**InfoQ：**嗯，其实也很不错，蘑菇街第一次做交易就能交出了这么好的一份答案。双十一高峰时期系统的压力也不小吧，为了保证系统在这样的短时高并发场景下的稳定性以及高可用性，蘑菇街在架构方面做了哪些调整？

**岳旭强：**我们从去年年底一直到今年上半年一直在做架构方面调整。以前蘑菇街整个架构是混在一起的，没有什么架构可言，因为那个时候最主要的诉求就是灵活、简单、快速实现需求，所以技术方面没有太多东西。可以说从 2010 年底开始做蘑菇街一直到去年下半年，整个架构上没有太大的优化和调整，所有的东西就跟小网站一样的，代码都写在一起，只有一个大的代码库，也没有什么太多分层。

随着蘑菇街从导购网站向垂直电商的转型，系统也就开始变得越来越复杂。从去年年底开始，我们就着手架构方面的调



整。其实所有的系统都一样，当大到一定程度后所要做的第一件事情都是拆分，也就是微服务化。拆分主要是从两个层面去做，我们先从垂直层面（业务）做了一些拆分，把交易、资金、评价、商品等服务化，每一个是单独的系统，系统与系统之间通过接口通信。

水平层面的话，我们把比较重要的应用的核心服务抽取出 来，比如说资金业务、交易业务、商品业务。以前它们是一个单独的系统，可能从 Web 访问到中间的业务处理，再到 DB 都是在一起的，拆分之后，Web 服务和业务处理就被隔离开来，业务处理的服务也会被其它业务调用。拆分之后中间也就会用一些异步的消息机制来提高系统的稳定性。

拆分之后系统之间相互隔离，一定程度上提高了系统的稳定性。比如这次双十一活动中，11 号凌晨的时候活动系统垮了，但只是活动的相关业务不能访问，其它的业务不会受影响。

当然为了保证拆分后系统的稳定性，我们也开发了很多基础的组件，比如我们跨语言通讯的框架、缓存集群方案、内部系统负载均衡方案、数据库中间层。

**InfoQ：**交易场景下，数据库的读操作远大于写操作，这个时候数据库的负载也特别大。针对这次双十一，蘑菇街在数据库方面做了哪些优化？

**岳旭强：**数据库这块其实我们也是按照服务、业务来拆分的，比如负责交易的团队会全权负责交易系统的数据库，交易的数据库是独立的一个集群，使用的是 MySQL，有做读写分离，主要是使用我们自己研发的中间层 donquixote（堂吉柯德），它的主要功能是连接池、屏蔽非法 SQL、读写分离。

针对这次双十一，我们主要做了两件事，一个是硬件层面的扩容。一个是数据库 SQL 查询优化，我们大概花了两个月时间，把所有的 SQL 查询全部优化了一遍，虽然每一个都比较细小，但是最后综合起来的效果非常明显。同样的配置下，没有优化之前每秒处理的订单数大概是 900 单，优化之后每秒的可以处理的订单翻了一倍，1700 单。

**InfoQ：**秒杀这样的场景下，蘑菇街是如何保证数据的一致性的？



**岳旭强：**这个其实蛮麻烦的，以前蘑菇街做导购社区的时候，一致性的问题基本不用考虑。但是做交易的话，这个事就变得非常复杂，

秒杀这样的场景下，主要需要解决超卖和错单的问题。解决超卖，我们主要将秒杀交易全部异步化，库存、优惠券的扣除都是异步队列消费时处理，同时保证最先入队的人先下单。解决错单的问题，会在交易下单时，第一步 push 异步队列，第二步日志打点，后台处理同时校验日志、队列中的消息，以保证数据的正确性。

**InfoQ：**在双十一之前蘑菇街应该也做了很多次演练，双十一高峰期有没有出现演练时没有预想到的情况？

**岳旭强：**哈哈，当然有。一开始我们根据去年的交易情况以及最近的业务量对双十一的交易量做了个预估，我们认为一天 1 个亿差不多了，数据库按照六七月份数据库的线上峰值为基准，放大八倍压测，Web 服务器按照十五倍压测，这是双十一前的准备。但是没想到 10 号凌晨就达到我们的预案值了，当时很紧张，但是也很开心。我们先是紧急做了处理，砍掉了很多短期影响不是很大的功能和业务，尽量的优化和减少系统的压力。接下来的一天时间又对数据库做了优化调整，包括限流等应急措施。Web 那块我们觉得 8 台机器问题不大，并且扩展也很容易之前的准备也比较充分，所以也没有把精力放在这上面。

11 号还没到 0 点的时候，11: 56，活动系统就被冲垮了。限流保护都没有产生作用，瞬间被冲垮了，因为限流是需要延后一小点时间，数据过来后才做限制操作的，当时还没来得及，几台机器就垮掉了，这是没想到的。当时我们加了几台机器，终于在 0: 11 的时候恢复正常了。第二天我们回放当天的日志，活动系统超过了我们当时预案的 10 多倍。这 10 多分钟，按照 10 号的量，可能损失就有 1000 万。

这两天我们也会复盘，总结一下这次遇到的问题，后续也希望能分享给 InfoQ 的读者。



# 天猫 11.11：多终端下的 一致性体验

作者 郭蕾

今年的双十一电商大战完满结束，各大电商也随后相继公布了各自的订单数量和销售总额。在这一次的电商大战中，各大公司已经从单纯的价格较量延伸到平台稳定性和用户体验上的竞争，而这些较量背后需要有相应的强大技术做支撑。其中，Web 前端直接和用户交互，它的稳定性、流畅性直接决定着整个系统的用户体验。InfoQ 也采访了天猫的前端架构负责人鬼道，以期与读者分享天猫在 Web 前端方面的成熟经验。

**InfoQ：**我们可以看到在整个双十一的过程中，与用户打交道的前端页面访问基本都很流畅。能介绍下天猫 Web 端的架构么？

**鬼道：**MAP（tMall fe Architecture & Publication）是天猫 Web 端的架构代号。MAP 涵盖了一张页面的代码管理、开发环境、模板、数据接口、发布、终端判断、线上监控、性能标准等各个方面。在本次双十一中，我们全链路的性能都处于竞品的顶尖水平，如首页第一、搜索结果页第二、商品详情页第一。

另外，在移动端，我们使用了一项叫 Dynative 的技术，当使用天猫客户端访问双十一的 Web 页面时，客户端会自动生成一个 Native 实现的 View，大大提升交互的流畅性和加载性能。另外我们也有同学在负责研究跨终端的组件方案，让一套组件可以同时运行在手机、Pad 和 PC 上，当然在客户端下时它同样能转换成一个 Native 组件，从而保证一致和优秀的用户体验。

在性能监控上，我们不但能区分不同地域用户的性能情况，还能以网络类型、客户端、访问类型（直接访问、刷新、返回）、网络协议（HTTP/1、HTTP/2、SPDY）、操作系统等过滤条件分析应用性能。在性能指标上，不但有首字节时间、首次渲染时间、Dom ready、onload 这些常规指标，还提供重定向、DNS 查询、TCP 链接建立、服务端响应、动画帧



数、内存占用等指标。对于应用的加载性能除了能提供以上提到的关键性指标外，我们还能提供应用加载过程截图以及基于全网真实用户环境的加载瀑布图。

**InfoQ：**为了保证业务峰值时用户顺滑的用户体验，淘宝对前端页面做了哪些特殊处理？

**鬼道：**如何保证在峰值时的用户体验，这也是我们最近重点的事情。从大的方面来说，主要有两方面。一是业务降级，通过监控重要接口的访问情况，一旦访问量超过警戒值，为保证核心链路的稳定性，会将其他来源的访问按照预设的方案进行降级。二是 CDN，CDN 是静态资源的主要载体，阿里前端的静态资源默认都部署在 CDN 上，这个我一会再谈。另外后端系统针对系统读写特性也有相应的动静分离策略、容灾策略和异常隔离策略。

**InfoQ：**不管是在首页还是商品页中，图片都占了很大一部分，在如此大的高并发访问情况下，网站的图片仍然可以在短时间内显示出来，真是一件不可思议的事情，图片这块，淘宝有用什么先进技术么？

**鬼道：**我们在全国各个省部署了大量的 CDN 服务器，把短时间内高并发的请求转化成了分布式的处理，由每个省市本地的服务器来处理各自的请求，避免了大量数据跨长途骨干网往返；另外，我们在操作系统的内核和 TCP 协议栈层面也针对网络拥塞、丢包的情况做了优化，快速发现丢包缩短 TCP 协议的应变时间；通过精准的 IP 库区分用户区域，确保用户访问到离自己最近的服务器。

同时，天猫客户端会根据网络状态展示不同质量的图片，如 Wi-Fi 下展示高清图片，非 Wi-Fi 网络下展示普通图片，这个策略在图片质量和响应速度上做了很好的折中。另外，减少图片请求在移动端上效果尤其明显。Icon Font 用于维护一组图标，在享受矢量图形缩放和文字颜色可变的优势上，也是减少图片请求的一种方式；未使用 Icon Font 的图标会通过 split 工具自动合并成大图，同样可以减少图片请求数量。天猫的主要页面上都已经做到在高清屏（移动和 PC 设备）上展示高清图片，非高清屏上降级展示普通图片。图片格式方面，我们也有新的尝试，在天猫商品详情页面上也部分启用了 WebP，主要是考虑 WebP 相对 JPG 有更高的压缩率，但 WebP 在移动端解压速度的问题也不容忽视。

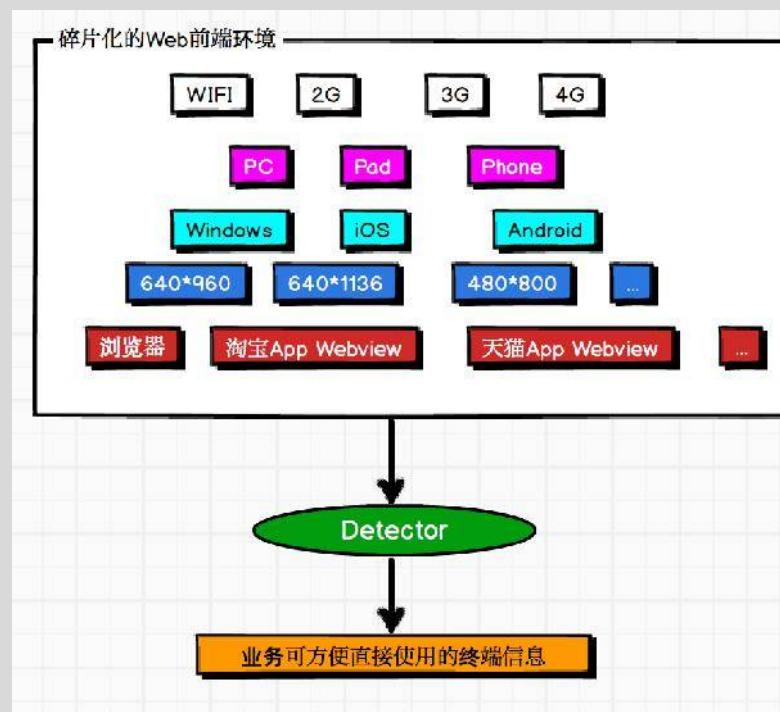


**InfoQ:** 不同地区的用户可能访问的速度也不一样，天猫在前端这块有相应的监控手段么？

**鬼道:** 有监控手段，我们会实时分析访问日志，上报各个区域的用户请求响应时间，在发现异常时通过 DNS 调度将用户导流到正常的服务器上。

**InfoQ:** 用户可能会在不同的屏幕上访问天猫，天猫是如何保证跨终端的一致性的？

**鬼道:** 在一致性上我们有一个很重要的基础设施 Detector。随着无线互联网的发展，业务上需要在不同的端给用户呈现不同的内容，前端所面临的终端环境也越来越复杂和碎片化，这就要求前端具备方便获取准确、一致的各种终端信息的能力，为了解决跨终端一致性问题，在双十一之前，我们就设计和开发了多终端判断基础模块 Detector，通过 Detector，我们实现了双十一期间成百上千页面的一致终端判断，逻辑跳转，不同终端内容输出的艰巨任务。



Detector 是技术基础，用户感知到的一致性需要依靠上层应用在设计层面的一致性保证，由于天猫不同终端上的界面来自同一个设计团队，这有利于设计风格和理念保持一致。为了更好地延续设计风格，天猫内部有一个 MUI 项目，就是将常用的界面元素封装成组件库。



InfoQ：在本次双十一活动中，有哪些好的经验可以分享给我们的读者？

**鬼道：**经验有很多，我挑些重点的来分享给大家。首先 Hybrid API 的应用可以降低跨客户端开发成本。双十一前我们推动发布了阿里 Hybrid API 1.0，它提供一套能够运行在阿里多个客户端和独立浏览器的 18 组共 26 个 API，目前已支持天猫、淘宝、支付宝客户端，既包括常用的传感器 API 也有窗口管理、登录、支付等 API。双十一期间应用于天猫狂欢城、天猫店铺互动、手机淘宝互动分会场等多个业务。

游戏引擎 Hilo。Hilo 是阿里研发的互动游戏引擎，支持 PC 端和移动端几乎所有主流浏览器，也通过 Flash 支持 IE 6、7。Hilo 可以极大降低互动游戏的开发成本，适合阿里的业务场景。双十一期间应用于天猫狂欢城、天猫店铺互动、手机淘宝互动分会场等多个业务。

应用稳定性监控。对于应用的稳定性，我们有覆盖所有端的监控系统，在 Native 端我们有实时的 Crash 监控平台，在 Web 端我们有实时的 JS 报错监控平台。任何客户端的报错都能实时体现，例如我们发现 iPhone 客户端在双 11 时 Crash 率大幅上升，我们当晚立刻修复了问题，Crash 率恢复到了正常水平。

双十一天猫首页开闭幕式。相比去年而言，本次开幕式的挑战主要是动画变得更加复杂，根据场景选用合适的动画：CSS 3 动画用于处理开始出现时的头像波浪，CSS 3 动画适用于大批量处理、动画方式较为简单的场景；JS 动画用于处理中间两个场景，JS 动画使用了 KISSY 框架的 Anim 模块，比较方便；Canvas 动画主要应用于倒计时阶段的粒子效果。

关注上下游系统及链路。双十一需要保证是对客户的整体体验稳定、流畅，任何一个环节出错都对这个目标有影响；前端处于用户体验的最前端，用户的体验从这里开始，任何的差错也会从这里产生；加上前端是所有后面系统服务的“消费者”，从消费者的角度关注和检查，而不仅仅依赖服务方自身的监控和保证，能最大限度减少出问题的概率。

充分的预演和测试。随着移动的浪潮，前端所面临的终端环境也越来越复杂，不再是仅仅关注几种浏览器，加上系统碎片化的问题，所需要测试的客户端越来越多，仅仅依赖于 QA



和前端的自测在日常中基本能保证主流的终端稳定，但在双十一这样的场景中，就显得不足。

依赖于程序而非系统或人。双十一其他每个时间段都有一定的状态变化，从预热到正式到下线，每个阶段的操作我们都靠程序控制，提前上线，而不是依赖于人或者系统的定时发布。没有人可以做到 100% 不出错，没有系统可以保证 100% 稳定，将出问题的几率控制到最小。



# 谷歌的诀窍：如何取消验证码

作者 张天雷

谷歌最近研发出一种新方法，该方法能够通过一个简单的方法区分机器人和人，从而取消网站上那些烦人的验证码，并带给用户更好的新体验。

验证码（CAPTCHA）是 Completely Automated Public Turing test to tell Computers and Humans Apart 的缩写，意思是完全自动地区分计算机和人类的图灵测试。很多年来，互联网上的注册、验证等都是通过那些独特的验证码来防止机器人对服务器进行暴力占用。通过扭曲的验证码符号，它可以防止恶意破解密码、恶意刷票、论坛灌水等等不太正常的行为。同时可以防止一些非正规用户对程序进行暴力破解。

然而，验证码带给真正人的用户体验往往是非常烦人而且繁杂的。为了上一个论坛网站，用户需要首先通过验证码注册账户，接着通过验证码登录网站，最后通过验证码回复用户所感兴趣的帖子。同时，过度扭曲的验证码往往很不容易识别出来，从而增加了用户尝试的次数，更加让这个过程变得臃肿。为了解决这个问题，谷歌的相关人员前后进行多次研究，并且提出了相关方法。

在一年前，谷歌提出了一种改进方法，即视觉或者视屏的迷宫，叫做 reCAPTCHA。这种方式在图灵测试中对人类和机器人的区别率达到了 99.8%，这是一种非常有效的改进。然而，在 reCAPTCHA 方法提出之后，相关人员又对机器人识别做出了更加智能的改进，从而让机器人适应了新的验证方式，这时，现有的验证码就无法再做到有效区别机器和人了。

因此，谷歌在最近提出了一种新的分析方式来判别人类还是机器人。他们改进了这种方法，让那些使用验证码的网站可能再也不需要用户来输入一个验证码了。用户只需要点击一个复选框（如下图）即可完成验证工作。如果这样还不能判定是机器人还是人类，那么这种方式会使用老的 reCAPTCHA 方式来验证。根据谷歌内部透露，早期使用新的 reCAPTCHA 的用户群体，在使用新方式的时候，约 60% 的 WordPress、80% 的 Humble Bundle 用户只是用了复选框点击判断，并没有看到老的 reCAPTCHA 界面。





这种方式的具体原理是在用户使用网络的时候就开始收集大量信息，包括 IP 地址、Cookies 等，然后通过这些数据来判定用户现在与过去的使用方式是否一致，从而确定到底是不是机器人。谷歌还会根据用户鼠标在复选框上的移动方式、点击方式等细微的地方，确定人与机器人的区别。还有一些其他的细微变量也帮助了谷歌判断，但是目前谷歌没有透露，以防止那些机器人的软件破解这种方式。

当需要使用这种新的验证方式的时候，网站开发者需要从谷歌获取一个密钥对。新的 reCAPTCHA API 同时为移动设备提供了一套自动、清晰的组件，该 API 禁用了 JavaScript 来防止对信息的盗取。大部分现行的浏览器，像 Chrome 3+、Firefox 3+、IE 7+、Opera 10.10+以及 Safari 4+都已支持该 API。



# 扩展性即服务

作者 谢丽

当前，在构建可扩展的新闻推送功能方面，业内有一种向外部托管组件迁移的趋势。[Stream.io](#) 公司首席执行官 [Thierry Schellenbach](#) 发表了一篇[博文](#)，从搜索、新闻推送和实时功能三个方面对比了开源解决方案和托管解决方案，并探讨了这种趋势产生的原因。

在搜索服务器方面，他对比了 [ElasticSearch](#) 和 [Algolia](#)。其中，前者是一个开源解决方案，后者通过托管模型提供专利搜索技术。ElasticSearch 的安装配置非常简单，只需几天就可以完成，不过用户技术栈中会多一种组件，而且后续需要处理升级、实例停机等问题。Algolia 只是简单地提供托管 API，并负责监控和运维，用户可以立即享受服务升级带来的好处，而且只需花几个小时就可以为自己的应用添加可扩展的高性能搜索组件。此外，Algolia 还提供[分布式搜索网络](#)，它可以在世界范围内复制搜索索引，降低搜索请求的响应延迟。除了方便用户外，这种托管模型也有益于 Algolia 本身。他们可以快速迭代，并立即将更新推送给客户，并且无需维护一个客户端与服务器端的版本兼容矩阵。

在新闻推送和活动流方面，他对比了 [Stream Framework](#) 和 GetStream.io。其中，前者是一个由他编写的开源框架，后者是其公司提供的托管服务。使用 Stream Framework，开发人员用几天或几周的时间就可以为应用添加一个可扩展的新闻推送功能。不过，安装、设置和维护像 Cassandra、Redis、Celery 和 RabbitMQ 这样的服务也非常耗时，而且增加了技术栈的复杂度。而借助 GetStream.io，开发人员仅用几个小时就可以完成同样的功能。

在实时功能方面，他对比了开源库 [Faye](#) 和托管服务 [PubNub](#)。不管使用哪一种，开发人员都可以在几分钟内实现实时功能。但 Faye 需要做很多调整才能处理有许多并发的实时连接。

诚然，托管组件有诸多优点。但从根本上讲，Thierry 认为，有两个原因推动了这种趋势：一是开发人员和企业越来越善于使用微服务架构；二是云提供商使多区域支持的设置更简单。

即便如此，也有一些企业仍在观望。Thierry 总结了以下四个方面的原因：

- 1) 托管解决方案无法定制；
- 2) 安全因素；



- 3) 担心外部组件问题可能会导致应用故障;
- 4) 担心被供应商锁定。

最后，Thierry 总结道，这种趋势催生了诸如 Algolia、Stream.io 和 PubNub 等提供可扩展托管组件的初创公司。而从开发人员的角度来讲，这种趋势产生了两个结果：一是开发人员向应用添加可扩展组件的时间成本大大降低；二是可以及时获得高级功能。



# 颠覆式前端 UI 开发框架： React

作者 王沛

基于 HTML 的前端界面开发正变得越来越复杂，其本质问题基本都可以归结于如何将来自于服务器端或者用户输入的动态数据高效的反映到复杂的用户界面上。而来自 Facebook 的 [React](#) 框架正是完全面向此问题的一个解决方案，按官网描述，其出发点为：用于开发数据不断变化的大型应用程序（Building large applications with data that changes over time）。相比传统型的前端开发，React 开辟了一个相当另类的途径，实现了前端界面的高效率高性能开发。

首先，对于 React，有一些认识误区，这里先总结一下：

- React 不是一个完整的 MVC 框架，最多可以认为是 MVC 中的 V (View)，甚至 React 并不非常认可 MVC 开发模式；
- React 的服务器端 Render 能力只能算是一个锦上添花的功能，并不是其核心出发点，事实上 React 官方站点几乎没有提及其实现于服务器端的应用；
- 有人拿 React 和 Web Component 相提并论，但两者并不是完全的竞争关系，你完全可以用 React 去开发一个真正的 Web Component；
- React 不是一个新的模板语言，JSX 只是一个表象，没有 JSX 的 React 也能工作。

## 1. React 的原理

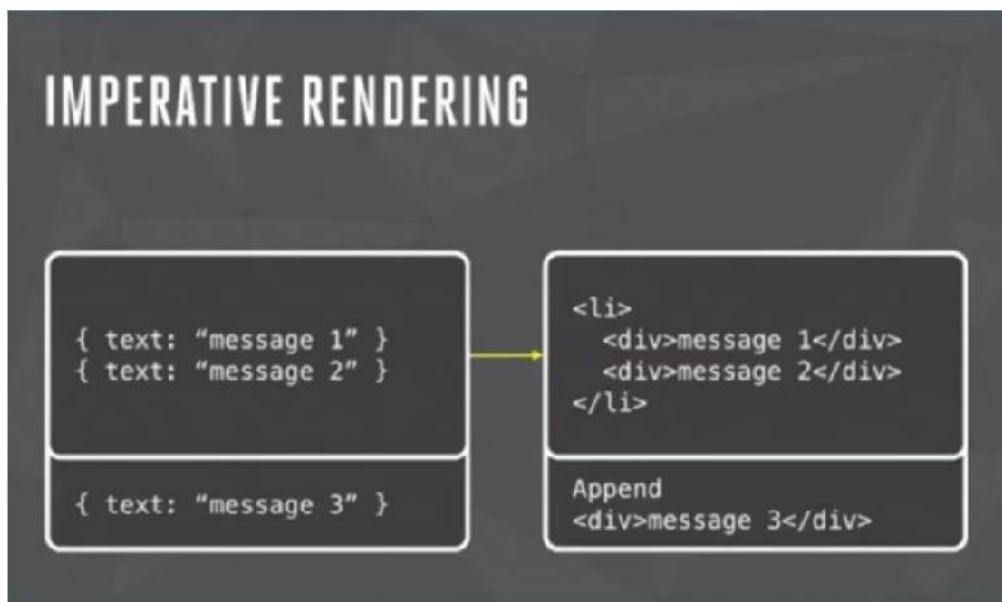
在 Web 开发中，我们总需要将变化的数据实时反应到 UI 上，这时就需要对 DOM 进行操作。而复杂或频繁的 DOM 操作通常是性能瓶颈产生的原因（如何进行高性能的复杂 DOM 操作通常是衡量一个前端开发人员技能的重要指标）。React 为此引入了虚拟 DOM (Virtual DOM) 的机制：在浏览器端用 Javascript 实现了一套 DOM API。基于 React 进行开发时所有的 DOM 构造都是通过虚拟 DOM 进行，每当数据变化时，React 都会重新构建整个 DOM 树，然后 React 将当前整个 DOM 树和上一次的 DOM 树进行对比，得到 DOM 结构的区别，然后仅仅将需要变化的部分进行实际的浏览器 DOM 更新。而且 React 能够批处理虚拟 DOM 的刷新，在一个事件循环 (Event

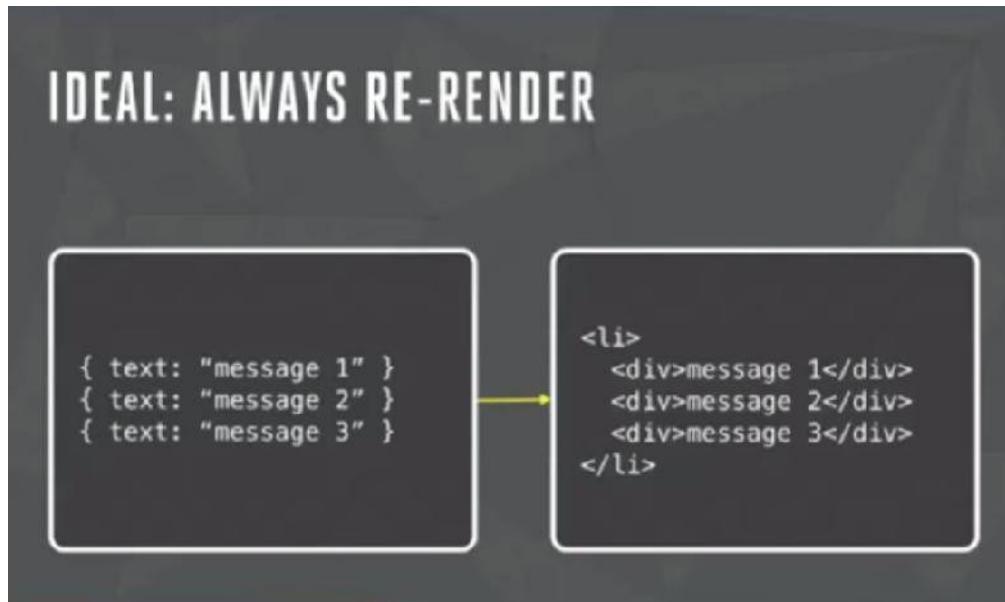


Loop) 内的两次数据变化会被合并，例如你连续的先将节点内容从 A 变成 B，然后又从 B 变成 A，React 会认为 UI 不发生任何变化，而如果通过手动控制，这种逻辑通常是极其复杂的。尽管每一次都需要构造完整的虚拟 DOM 树，但是因为虚拟 DOM 是内存数据，性能是极高的，而对实际 DOM 进行操作的仅仅是 Diff 部分，因而能达到提高性能的目的。这样，在保证性能的同时，开发者将不再需要关注某个数据的变化如何更新到一个或多个具体的 DOM 元素，而只需要关心在任意一个数据状态下，整个界面是如何 Render 的。

如果你像在 90 年代那样写过服务器端 Render 的纯 Web 页面那么应该知道，服务器端所要做的就是根据数据 Render 出 HTML 送到浏览器端。如果这时因为用户的一个点击需要改变某个状态文字，那么也是通过刷新整个页面来完成的。服务器端并不需要知道是哪一小段 HTML 发生了变化，而只需要根据数据刷新整个页面。换句话说，任何 UI 的变化都是通过整体刷新来完成的。而 React 将这种开发模式以高性能的方式带到了前端，每做一点界面的更新，你都可以认为刷新了整个页面。至于如何进行局部更新以保证性能，则是 React 框架要完成的事情。

借用 [Facebook 介绍 React 的视频](#) 中聊天应用的例子，当一条新的消息过来时，传统开发的思路如上图，你的开发过程需要知道哪条数据过来了，如何将新的 DOM 结点添加到当前 DOM 树上；而基于 React 的开发思路如下图，你永远只需要关心数据整体，两次数据之间的 UI 如何变化，则完全交给框架去做。





可以看到，使用 React 大大降低了逻辑复杂性，意味着开发难度降低，可能产生 Bug 的机会也更少。至于 React 如何做到将原来  $O(n^3)$  复杂度的 Diff 算法降低到  $O(n)$ ，大家可以参考[这篇文章](#)。

## 2. 组件化的开发思路

虚拟 DOM 不仅带来了简单的 UI 开发逻辑，同时也带来了组件化开发的思想，所谓组件，即封装起来的具有独立功能的 UI 部件。React 推荐以组件的方式去重新思考 UI 构成，将 UI 上每一个功能相对独立的模块定义成组件，然后将小的组件通过组合或者嵌套的方式构成大的组件，最终完成整体 UI 的构建。例如，Facebook 的 [instagram.com](https://www.instagram.com) 整站都采用了 React 来开发，整个页面就是一个大的组件，其中包含了嵌套的大量其它组件，大家有兴趣可以看下它背后的代码。

如果说 MVC 的思想让你做到视图-数据-控制器的分离，那么组件化的思考方式则是带来了 UI 功能模块之间的分离。我们通过一个典型的 Blog 评论界面来看 MVC 和组件化开发思路的区别。

对于 MVC 开发模式来说，开发者将三者定义成不同的类，实现了表现，数据，控制的分离。开发者更多的是从技术的角度来对 UI 进行拆分，实现松耦合。



### Comments

---

**Nate**  
This is a great article! Reply

**Kevin**  
This is a great article! Reply

**Ben**  
This is a great article! Reply

Leave a comment...

Send

**Template**

```

1 <div class="comment-box">
2   <h3>Comments</h3>
3   <ul class="comment-list">
4     {#comments.items}
5       <li>
6         <img src={thumbnails}>
7         <label>{username}</label>
8         <p>{comment}</p>
9         <button>Reply</button>
10      </li>
11    {/comments.items}
12  </ul>
13
14  <div class="form-box">
15    <textarea></textarea>
16    <button>Send</button>
17  </div>
18 </div>

```

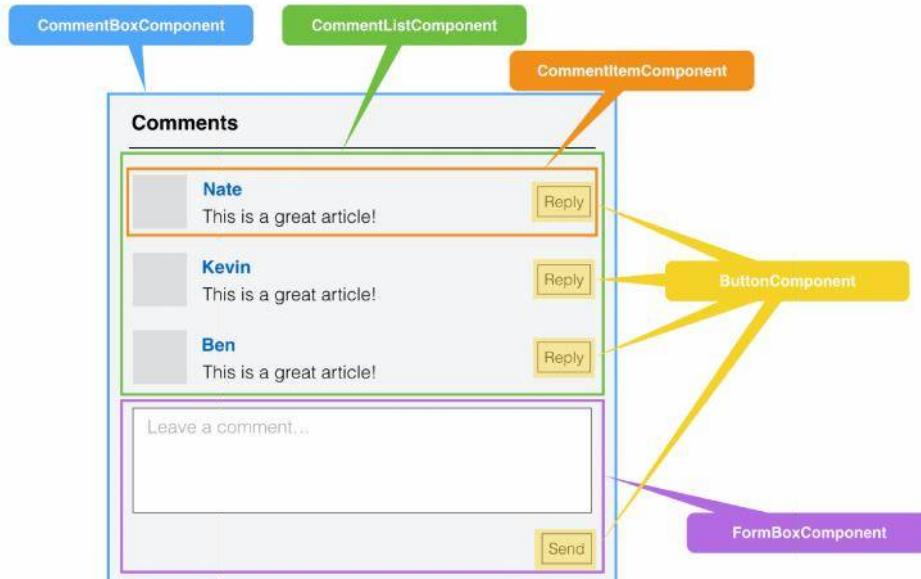
**Controller**

```

1 var controller = new Controller();
2   addComment: function () {
3     ...
4   },
5   replyComment: function () {
6     ...
7   },
8   userNameClick: function () {
9     ...
10   }
11 }
12
13 //...
14
15 })();

```

对于 React 而言，则完全是一个新的思路，开发者从功能的角度出发，将 UI 分成不同的组件，每个组件都独立封装。



在 React 中，你按照界面模块自然划分的方式来组织和编写你的代码，对于评论界面而言，整个 UI 是一个通过小组件构成的大组件，每个组件只关心自己部分的逻辑，彼此独立。这样最外层的界面的 Render 只需要如下代码：



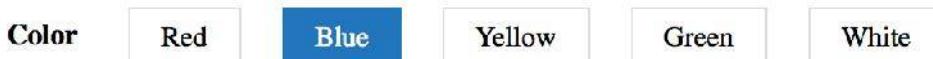
```
var CommentBox = React.createClass({
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
        <CommentList />
        <CommentForm />
      </div>
    );
  }
});
```

通过这种方式，每个组件的 UI 和逻辑都定义在组件内部，和外部完全通过 API 来交互，通过组合的方式来实现复杂的功能。React 认为一个组件应该具有如下特征：

- 1) 可组合（Composeable）：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有（own）它创建的子组件，通过这个特性，一个复杂的 UI 可以拆分成多个简单的 UI 组件；
- 2) 可重用（Reusable）：每个组件都是具有独立功能的，它可以被使用在多个 UI 场景；
- 3) 可维护（Maintainable）：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护；
- 4) 可测试（Testable）：因为每个组件都是独立的，那么对于各个组件分别测试显然要比对于整个 UI 进行测试容易的多。

### 3. 一个 React 组件开发的例子：Tab 选择器

上面从总体上介绍了 React 带来的全新的前端开发方法，以及其带来的影响，并没有介绍如何使用。为了让大家对其有一个具体的印象，这里实际来开发一个简单的组件：Tab 选择器。网店的产品页面通常需要这样的控件来选择产品属性，例如选择衣服的颜色。这个控件接受一个数据源展示多个 Tab 供点击，点击后就选中了某个颜色，界面通常如下图所示。



按传统方式，我们可以用如下代码来实现一个 jQuery 插件：



```
1  $.fn.TabSelector = function (options) {
2    var arr = ['<div class="tab-selector">'];
3    arr.push('<label>', options.label, '</label>');
4    arr.push('<ul>');
5    options.data.forEach(function (item) {
6      arr.push('<li data-value="' + item.value + '">');
7      arr.push(item.name);
8      arr.push('</li>');
9    });
10   arr.push('</ul></div>');
11
12   this.html(arr.join(''));
13
14   var lastSelected = null;
15   this.on('click', 'li', function () {
16     var $this = $(this);
17     if (lastSelected) {
18       lastSelected.removeClass('selected');
19     }
20     $this.addClass('selected');
21     lastSelected = $this;
22   });
23
24   return this;
25 }
```

用 React 方式，代码如下：

```
1  var TabSelector = React.createClass({
2    getInitialState: function() {
3      return {selected: this.props.selected};
4    },
5
6    handleOnClick: function (evt) {
7      this.setState({selected: evt.target.getAttribute('data-value')});
8    },
9
10   render: function() {
11     var tabs = this.props.data.map(function (item) {
12       var selected = item.value == this.state.selected ? 'selected' : '';
13       return <li data-value={item.value}>
14         className={selected}
15         onClick={this.handleClick}
16         >{item.name}</li>
17       ;
18     }, this);
19
20     return <div className="tab-selector">
21       <label>{this.props.label}</label>
22       <ul>
23         {tabs}
24       </ul>
25     </div>
26   ;
27 }
28});
```



通过比较可以看到，jQuery 插件方式，开发者首先需要考虑控件第一次 Render 出来时的 DOM 构建；其次，需要知道如何切换 UI 上的选中状态。

而 React 的方式，开发者仅仅需要考虑整体界面的 DOM 构建，不再需要关心局部更新，每次在一个 React 的 Component 上调用 setState 方法，都会触发 render 来重建整个界面。从开发思想的角度看，你可以认为每次数据的更新都会做整体的完全刷新。逻辑简单而直接。

如果我们再多考虑一步，控件的值不只在初始化和点击时可以设置，而且还可以通过程序动态的去设置。那么对于 jQuery 的方案而言，我们需要额外的方法和入口去做对应的 UI 更新。而对于 React 方式，则无需做任何改变，外部只需调用 setState 方法改变它的状态即可。这就是简化 UI 逻辑带来的好处。

完整的代码和演示已上传在 Github 上：<https://github.com/supnate/react-tab-selector>，大家可以实际试用一下。

## 4. 结论

如上所述，React 是一个全新思路的前端 UI 框架，它完全接管了 UI 开发中最为复杂的局部更新部分，擅长在复杂场景下保证高性能；同时，它引入了基于组件的开发思想，从另一个角度来重新审视 UI 的构成。通过这种方法，不仅能够提高开发效率，而且可以让代码更容易理解，维护和测试。

Facebook 以这样一种方式将沉淀多年的前端开发经验和技术的积累完全开源出来，值得所有前端开发者去借鉴和学习。并且 React 在发布一年的时间里就获得了极大的关注，Github 上拥有超过 1 万的 Star，相信其对前端开发的方向，甚至 Web Component 的标准，都将产生一定的影响。



# 封面植物——雪松



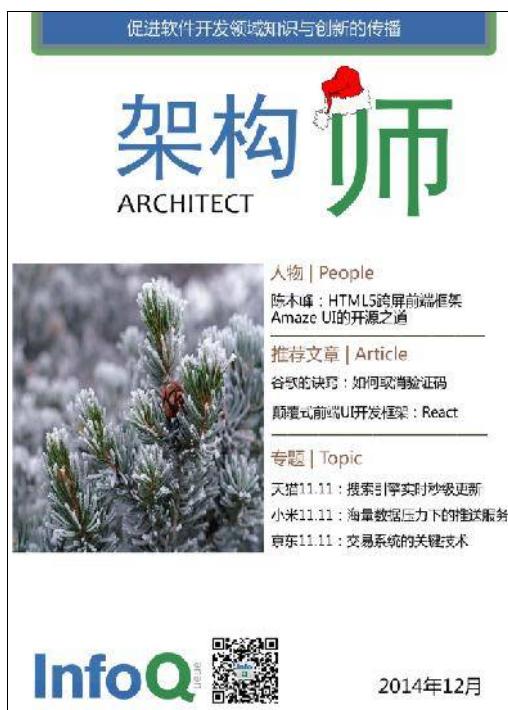
雪松（学名：Cedrus deodara (Roxb.) G. Don）是松科雪松属植物。常绿乔木，树冠尖塔形，大枝平展，小枝略下垂。由于球果形状相似，与杉树最为接近。叶针形，长8~60厘米，质硬，灰绿色或银灰色，在长枝上散生，短枝上簇生。原产于喜玛拉雅山脉海拔1500~3200米的地带和地中海沿岸1000~2200米的地带。10~11月开花。阿特拉斯雪松的叶丛。

球果翌年成熟，椭圆状卵形，熟时赤褐色。产于亚洲西部、喜马拉雅山西部和非洲，地中海沿岸，中国只有一种喜玛拉雅雪松，分布于西藏南部及印度和阿富汗。适生于中国年雨量600~1000毫米的温带和亚热带地区，长江中下游各地栽培生长良好。雪松喜光，亦有一定耐荫能力，喜凉爽湿润气候，对温度变化的适应力相当强，适生于土层深厚，排水良好的中性或微酸性土壤，忌水湿，土中积水往往生长不良，甚至死亡。雪松高可达80米，高大挺拔，侧枝平伸，枝叶浓密，枝下高极低，树冠呈坐地尖塔形，终年苍翠，姿态雄美。它与南洋杉、日本金松同为世界著名的三大观赏树种，雪松还有“树木皇后”之美称。

用雪松布置圣诞树它有以下几大优点：

- 1、雪松的适应性强：它无论是盆栽、缸栽、都能适应；
- 2、雪松比较耐寒、耐旱，南北气候都能适应；
- 3、雪松冠形饱满、枝条匀称不密不松、整体呈正三角形；
- 4、雪松占地面积小，室内室外均可布置；
- 5、每年的十一月至来年的二月是雪松移栽成活率比较高的季节；
- 6、雪松的适应环境性，外观形状苍翠挺拔有它独特的寓意。





## 架构师 12 月刊

每月 8 号出版

本期主编：崔康

流程编辑：丁晓昀

发行人：霍泰稳

读者反馈/投稿：[editors@cn.infoq.com](mailto:editors@cn.infoq.com)

新浪微博：<http://weibo.com/infoqchina>

商务合作：[sales@cn.infoq.com](mailto:sales@cn.infoq.com)

## 本期主编：崔康

InfoQ 中国总编辑，致力于中国 IT 领域知识与创新的传播，目前负责 InfoQ 整体内容的品牌和质量，同时担任 QCon、ArchSummit 大会的总策划。技术人出身，毕业于天津大学计算机学院，在加入 InfoQ 之前，在某大型外企长期担任协作软件平台的技术负责人和架构师，在相关技术领域积累了一定的经验。从 2008 年开始参与国内社区的技术传播，在多家科技媒体先后发表过数十篇文章，并出版多本译著，总产量超过 50 万字。可以通过 [tyler.cui@infoq.com](mailto:tyler.cui@infoq.com) 或者微博（“崔康 Tyler”）与他联系。



# 架构师

[www.infoq.com/cn/architect](http://www.infoq.com/cn/architect)

每月8号出版

