

# 架构师

10月 ARCHITECT



## 特别专题

构建iOS持续集成平台

自动化构建和依赖管理

测试框架

CI服务器与自动化部署

## 主编有话说

测试的平均斯坦与极端斯坦

围绕自动化测试开展持续集成

从Groovy到Java 8

中国IT高管谈创新型组织

InfoQ  
架构



每月8号出版

# 卷首语

## 侘·寂

在乔布斯离去两周年的日子，寻觅着从网络、媒体中获得的对乔布斯的印象，首先映入脑海的是乔布斯那张经典照片的描述：



我那时单身，我所需要的也就是一杯茶，一盏灯和一个音乐播放器；

试图用「禅」去概括这个画面，总觉得不妥，于是搜索到了『侘·寂』，感觉就对了。侘·寂是日本美学意识的一个组成部分，一般指的是朴素又安静的事物。这么说可能范围太广又不好理解，举一个例子可能就清楚了：

什么是侘·寂？落叶的庭院扫的一干二净之后，还要轻轻把树摇一下，抖落几片叶子，这才是Wabi Sabi的境界。

所以说对乔布斯，用「侘·寂」来描述他的一生也是十分合适的，人生起落，事态无常，不变的是对美好事情的追求，在人上的巅峰悄然离去，为人们所惋惜，但是留下的那个巅峰缺一直为后人所攀登，这就是「不圆满的美」！

说起这个话题是想到了它与程序或者产品之间的关系：开始的时候我们对所要追求的事情没有一个成型的概念，于是在某个阶段我们做出了一个东西，如果按照

最后的标准说可以打78分，很多人就开始在剩下的两分上作文章，即使圆满了也是80分，而另一些人不同，他们知道现在的提升空间绝不是这两分，上限还很远，于是100分的产品，他们拿到了98分；

不要在开始的阶段就把细节打造的完美无缺，因为你不知道满分是多少！

本期主编：水羽哲



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

阿里日照：做关系型数据库比做key-value挑战更大

## 观点 | View

你为什么要关注行为驱动开发

DevOps：是否必须简化基础架构？

## 本期专题：构建iOS持续集成平台 | Topic

构建iOS持续集成平台（一）——自动化构建和依赖管理

构建iOS持续集成平台（二）——测试框架

构建iOS持续集成平台（三）——CI服务器与自动化部署

## 推荐文章 | Article

中国IT高管谈创新型组织

从Groovy到Java 8

## 特别专栏 | Column

测试专栏主编：侯伯薇

软件测试中的黑天鹅（三）——测试的平均斯坦与极端斯坦

敏捷自动化测试（4）——围绕自动化测试开展持续集成

## 避开那些坑 | Void

DevOps之7大习惯

前端工程与性能优化（上）：静态资源版本更新与缓存

前端工程与性能优化（下）：静态资源管理与模板框架

## 新品推荐 | Product

Ceylon整装待发

企业软件开发者继续使用.NET 4.0

RAD Studio XE5支持Android、iOS和REST 客户端

Visual Studio 2013 RC版本已提供下载，并且宣布了正式发布的日期

Node.js 进入移动领域：StrongLoop 推出开源的 mBaaS

用于Visual Studio的免费PowerShell工具

# QClub

我们影响有影响力的人

北京 上海 广州 大连 西安 太原 成都 杭州 武汉 南京 深圳...

QClub

邀请  
业内知名专家

自由开放的  
讨论氛围

讨论时下  
热点话题

定期举办的线下活动

结识  
圈内技术好友

InfoQ



中文 | 英文 | 日文 | 葡文 | .....

● 想成为技术**牛人**? ● 想获得技术**干货**? ● 想结识技术**圈内朋友**?

# 百度技术沙龙

与技术大咖一起，讨论时下技术热点



牛人



大咖



圈内朋友



热点



干货

百度技术沙龙第43期

## 如何把好移动互联网产品的质量关

10月19日，上海IC咖啡



[salon.baidu-tech.com](http://salon.baidu-tech.com)

畅想 • 交流 • 争鸣 • 聚会



百度技术沙龙是由百度主办，InfoQ负责策划、组织、实施的线下技术交流活动，每月一期，每期一个主题，由2场演讲以及Open Space开放讨论环节组成。旨在为中高端技术人员提供一个自由的技术交流和分享的平台。每期沙龙会邀请1名百度讲师分享百度在特定技术领域的成果及实践经验，同时还会邀请1名优秀的互联网公司或企业技术负责人对同一话题进行分享。活动主要面向开发者、技术负责人、项目经理、架构师等IT技术人员。

新浪微博：  
[infoqchina](http://infoqchina)

Bai<sup>du</sup>百度

InfoQ

## 人物 | People

# 阿里日照：做关系型数据库比做key-value挑战更大

作者 [杨赛](#)

OceanBase是一个分布式的[关系型数据库](#)，是阿里巴巴自主研发的开源项目之一。根据其[Github](#)页面上的介绍，OceanBase实现了跨行跨表的事务，支持数千亿条记录、数百TB数据上的SQL操作，截止到2012年8月为止，OceanBase数据库支持了阿里巴巴集团旗下多个重要业务的数据存储，支持业务包括收藏夹、直通车报表、天猫评价等，截止2013年4月份，OceanBase线上业务的数据量已经超过一千亿条。

在2013年4月[支付宝官方博客上的一篇访谈](#)中，[阿里正祥](#)介绍了OceanBase的一些设计思路和技术进展。2013年7月的[阿里技术嘉年华](#)上，阿里高级技术专家日照带来了一场分享，介绍OceanBase 4.0，即OceanBase SQL版本的整体架构，质量保证和运维。会议期间，InfoQ编辑与日照进行了一些沟通，了解一些OceanBase的应用状态和团队情况。

**嘉宾简介：**杨传辉，花名[日照](#)，阿里高级技术专家，热于分享底层技术并从分享中学习。负责OceanBase开发，关注云计算和分布式数据库，之前在百度从事大规模分布式存储、计算系统等底层基础设施构建工作。

**InfoQ：**先介绍一下你个人在数据库、存储系统开发方面的经历吧。在这个过程中，感觉做数据库开发这个工作本身经历了哪些变化？

日照：我之前在百度做了两年半左右的云存储、云计算系统的开发。到阿里以后三年多的时间一直做OceanBase。以前是做纯粹的分布式系统，现在做的OceanBase则是融合了分布式系统和传统的关系数据库这两块东西。

我自己的经历，一直都是比较专的，基本上是一直做一件事情。

**InfoQ：**从做数据库的角度，做这种**Key-value**和做关系数据库有什么区别？

日照：关系数据库的复杂度比普通的Key-Value系统要大一个数量级。从技术的角度看，关系数据库内部的SQL执行、事务、多版本并发控制、数据一致性

都非常复杂。另外一点，那就是大家对你的看法。只要你说要做关系数据库，用户就会将你做的系统和Oracle、MySQL这样的通用数据库做对比，要求支持各种业务场景、易于运维，等等。相反，如果做Key-Value系统的话，只需要在某种业务场景下有优势就可以了。

**InfoQ：**听你这么说，重头做一个关系数据库会比较吃亏，还不如在**MySQL**的基础上进行改进？

日照：MySQL发展了这么多年，有很多值得借鉴的地方，同时也面临一些问题。关系数据库设计之初假设都是基于传统的机械磁盘，而现在SSD这样的新硬件发展很快，大有取代机械磁盘之势。另外，虽然MySQL在单机层面做了大量的工作，但是在主备同步和可扩展性上存在很大的问题。基于MySQL做改进的好处是可以很快应用到生产上，阿里也有这样的团队。OceanBase则是希望从根本上解决这些问题，技术挑战要比直接改进MySQL大一个数量级，不过从阿里数据库的规模上看，这样的投入是值得的。

**InfoQ：****OceanBase 0.4**版本主要的改动在哪些方面？这些方面的开发优先级是如何确立的？

日照：OceanBase 0.4出来之前只支持API接口，0.4版本最大的一个改动就是支持SQL。支持SQL以后，底层的实现机制也做了很大的改变，例如支持查询计划执行、行锁、多版本并发控制，等等，另外，二级索引功能也在开发中。另外的改进就是性能优化，0.4之前我们的主要精力在开发功能，没有精力优化，0.4专门成立了一个性能优化小组，整体性能上提升了一个数量级。对于查询简单且数据量较大的业务场景，OceanBase的性能是比MySQL要好的。

OceanBase开发的需求有两个来源：一个是开发版本计划的功能，另外一个是业务方提出的需求。OceanBase大体上按照预定的计划开发新功能，不过也会根据业务方的需求调整优先级。

**InfoQ：****OceanBase**现在主要的客户有哪些？

日照：OceanBase有40几个业务方，这些业务方来自淘宝、天猫、支付宝等

各个集团子公司。

我们能够感受到的有淘宝收藏夹，底层存储全部采用OceanBase；也有天猫评价，双十一大促的时候用得很多；另外，所有广告主的报表信息都存放在OceanBase。支付宝也有一些业务，比如余额包的部分数据存放在OceanBase。

**InfoQ：**所以，现在在量大的业务场景，数据库运维团队都会推荐**OceanBase**吗？

日照：这是我所希望的，但也不完全是这样子，对于不需要事务的场景，运维团队也会考虑HBase。现在很多是一个混合的方案，因为没有任何一个系统有百分之百的优势。我觉得再过一两年左右，在量大的场景里，OceanBase会有更大的优势。OceanBase的一个特点就是，我们进步非常快，比如从0.3到0.4也就一年多点时间，这段时间SQL功能从无到有，性能也提升了一个数量级，而其他系统基本上变化不大。

**InfoQ：****OceanBase**团队有多少人？分别是什么角色？

日照：我们这个团队基本上三年时间都只做了OceanBase这么一个事情。最开始的时候没有那么多人，大概十几个开发。到现在为止，整个OceanBase团队是三十几个开发，五个左右测试，五个左右专职运维。另外，运维这里也有一些变化。以前是由OceanBase专职运维负责所有线上业务接入和运维，但是随着业务线越来越多，整个数据库的运维团队都会来运维OceanBase，和现在集团MySQL运维的做法类似。

**InfoQ：**从开发**OceanBase**的角度来说，参与到这个项目的开发者需要具备哪些方面的技能，或者背景？目前，你们是如何招揽到这种技能和背景的人才的？

日照：我们团队有一套成熟的人才培养机制，主要靠培养素质好的应届生。优秀的应届生放到我们这儿，有理论学习，也有实践工作，分布式数据库开发能力提升得很快。社招我们也做，不过做得比较少，国内做类似工作的人太少了。

。

**InfoQ：**素质比较好的人怎么来评估呢？

日照：我们公司有一套统一的招聘标准，我们自己还会在统一的招聘标准上加一些东西。对于应届生，我们会看他的实践能力，他学习成绩，包括是不是足够的踏实，是不是足够的聪明，动手能力，写代码的能力，有没有花时间专门静下心来做某一类型的项目。

**InfoQ：**没有对特定的语言的要求？

日照：没有什么特定语言的要求。我们更关注的是基础知识和潜质，语言只是可以很快学到的一项技能。

**InfoQ：**进来之后大家喜欢做这个吗？

日照：应届生里面有很多愿意做底层的，因为他们觉得很有技术含量，呵呵。我们整个项目组三年以来一直都在忙新版本、支持业务，从来没停过。新同学加入后不断会有新的技术挑战，所以，大家做得还是蛮开心的。

**InfoQ：**中文项目的发展有很多局限性。**OceanBase**是否有国际化的打算？

日照：中文项目发展的局限性是有的。OceanBase如果一直用中文，是会有一些人来看代码，但是很少人会去给这个代码做贡献。同时，OceanBase现在还是一个快速发展的时期，这个时候外部的committer要参与进来是比较难的，因为变化是比较大的。再过一两年的时间，相对稳定一些，到那个时候，我们还是希望吸收国际上一些英语国家的committer。之前OceanBase开源的消息在Twitter里面放出来的时候，已经引起了很多的关注，有一些HBase的committer都知道我们这个项目。

**InfoQ：****OceanBase**目前主要针对阿里集团内部的业务提供服务。未来是否计划通过某种途径提供对第三方企业、个人的服务支持，比如像**TAE**那样？

日照：现在已经有一些银行在跟我们进行一些非正式的合作。比如说交行，他

会有一些非核心的分析型业务，放到OceanBase里面来搞，我们也安排了一个专职同学花大约20%的时间回答他们的问题。

另外，阿里云也有一些RDS客户的业务更适合OceanBase，他希望能够接入OceanBase。对他来说，OceanBase和MySQL用起来是一样的，都是兼容的。虽然OceanBase的功能不如MySQL多，但是他并没有用到那些复杂的功能，他就管哪个东西更便宜。

目前这个阶段，我们还没有把阿里集团内部的业务完全支持好，因此，这个时候我们90%以上的精力都放在集团内部，只花少量精力了解外部需求。等到OceanBase足够成熟了，我们会提供外部服务，大概在一两年之后。

原文链接：<http://www.infoq.com/cn/news/2013/09/relational-and-key-value-db>

## 相关内容

- [James Phillips谈从关系型数据库转到NoSQL](#)
- [NuoDB发布可扩展的云关系型数据库](#)
- [MetaModel——跨多种数据存储提供统一的数据访问](#)
- [开发多语言持久性应用程序](#)
- [从关系数据库向NoSQL迁移：采访Couchbase的产品管理主管Dipti Borkar](#)

# 观点 | View

## 你为什么要关注行为驱动开发

作者 [Jan Stenberg](#)，译者 [孙镜涛](#)

敏捷让我们不再需要将精确的需求提前转化为精细计划的工作，但是随着流程后期的大量发现和误解我们依然会有大量的浪费，哪怕是在短期冲刺中也是如此，[Matt Wynne](#) 在一个最近的[行为驱动开发概要](#)中如是说。

[Cucumber](#)（一款开源的BDD工具）的首席开发者Matt进一步解释说，BDD要解决的一个关键问题是提升问题领域人员和解决方案领域人员之间的交流。BDD的目的是为了创建并发展一个这两个领域之间相互理解的公共区域，创建一种公共语言，一种普及的语言（正如[领域驱动设计](#)中所定义的那样）。

[用户故事](#)是一个敏捷工具，它能够帮助我们描述请求的功能。为了在将用户故事转换成工作功能之前更好地理解它们，BDD使用具体的示例揭示我们如何使用这些描述的功能。这也是在早期阶段发现边缘案例的一种廉价方式，而不是在后期、解决它们的成本更加昂贵时再发现它们。

BDD强调通过示例揭示问题领域，它的一个优点是所有相关人员都能参与进来，包括用户和利益相关者以及开发人员和测试人员；示例并不是技术性的，基本上你需要的所有就是笔和纸或者一些其他类似的东西。

在探讨解决方案领域的时候，Matt的经验是：随着项目的发展，团队通常会发现向代码库中引入一个变化会越来越难。Matt将代码库比作厨房，一个良好组织的干净厨房能够更加容易地生产高质量的食物。保持一个厨房干净意味着需要经常打扫，把脏东西拿走，同时一个专业的工作环境也需要同样的策略。在一个代码库中，与清扫厨房相等的工作是重构，改变设计但不需要改变它的行为。随着重构的进行，代码库会更加有组织性，更加容易理解和改变。

为了确保重构期间的行为不会发生变化，我们需要自动化测试，例如测试驱动开发（TDD），TDD和BDD这两者都使用示例阐明需求。

如果没有干净的代码那么你就不会保持敏捷。

BDD是由[Dan North](#)在2006年左右提出的，他不仅书写了一篇[介绍](#)还从[BDD的观点出发编写了一些故事](#)。

[Specification by Example](#)就是一种定义与BDD密切相关的需求的方式。

[Cucumber](#) 是一款行为驱动开发的开源工具，现在支持9种编程语言，包括基于JVM的语言。同时针对企业的新版本的[Cucumber Pro](#) 已于最近发布。

查看英文原文：[Why You Should Care About Behaviour-Driven Development](#)

原文链接：<http://www.infoq.com/cn/news/2013/09/care-about-bdd>

## 相关内容

- [BDD工具Cucumber开发团队扩大且修复了大量Bug](#)
- [度量驱动开发](#)
- [虚拟座谈会：代码测试比率、测试驱动开发及行为驱动开发](#)
- [行为驱动开发关注点从数据库转向领域模型](#)
- [.NET工具和实践调查结果](#)

## 观点 | View

# DevOps：是否必须简化基础架构？

作者 [Matthias Marschall](#)，译者 李彬

在运行一个[DevOps专家小组](#)的过程中，RedMonk的分析师[James Governor](#)提出了这样一个问题：为了引入DevOps，我们是否必须简化基础架构？

小组成员们都赞同这样的观点：我们必须改变我们的系统和组织机构，以减少投放市场的时间。哪怕现存的基础架构依旧保留，我们也将需要尽我们所能地实现自动化，以便加速创新。

[Opscode](#) ([Chef](#)的创造者) 的首席开发官[Adam Jacob](#)表示，新环境其实并没有变得更简单，只是设计有别。这些不同的设计有助于加速事物运转，从而显得更加简单。

来自[Puppetlabs](#) ([Puppet](#)的缔造者) 的[Luke Kanies](#)确信，“每个人都需要做出改变”。他希望每个人都能够构建工具，用来帮助人们构建新技术并摆脱旧技术。他提出了这样的问题：如何更快速地切换到新技术？他的方法是，在业务中寻找这样的个领域，在这个领域中，我们不需要太多的投入就能够取得显著成果。他注意到，通过将他们的基础架构中的某些特定部分自动化，客户端投放市场的时间间隔，从过去的每隔数月减少到现在的每隔数周甚至更短。

IBM Rational云架构师Robbie Minshall表示，人们对云计算提供商所实现的一切感到惊奇，而且他们希望相同的事情也能够发生在自己的数据中心身上。鉴于现存的基础架构和大型主机并不会消逝，也无法迁移到云中，他认为，重要的是找出如何在其他方面进行创新，并让这些东西与现存的基础架构互相配合。快速前进的事物需要与迟缓的那些有所联系。

来自[Netflix](#)的[Adrian Cockcroft](#)强调，更快速进入市场是取胜之道。

我们无需将整个基础架构简化，就可以引入DevOps。但是小组的专家们同意，在对企业来说，将关键部分自动化以便加速投放市场是必不可少的。我们需要能够在运行复杂业务的同时进行创新。

查看英文原文：[DevOps: possible without simplifying your infrastructure?](#)

原文链接：<http://www.infoq.com/cn/news/2013/09/devops-expert-panel>

# 专题推荐语

## 构建iOS持续集成平台

2000年Matin Fowler发表文章Continuous Integration；2007年，Paul Duvall, Steve Matyas和Andrew Glover合著的《Continuous Integration: Improving Software Quality and Reducing Risk》出版发行，该书获得了2008年的图灵大奖。持续集成理念经过10多年的发展，已经成为了业界的标准。在Java, Ruby的世界已经诞生了非常成熟的持续集成工具和实践，而对于iOS领域来说，因为技术本身相对比较年轻和苹果与生俱来的封闭思想，在持续集成方面的发展相对滞后一些，但是，随着越来越多的iOS开发者的涌入，以及各个互联网巨头加大对iOS开发的投入，诞生了一大批非常好用的持续集成工具和服务，本专栏的目的就是介绍一下如何有效的利用这些类库，服务快速构建一个iOS开发环境下的持续集成平台。



# 本期专题：构建iOS持续集成平台 | Topic

## 构建iOS持续集成平台（一）——自动化构建和依赖管理

作者 [刘先宁](#)

2000年Martin Fowler发表文章Continuous Integration 【1】；2007年，Paul Duvall, Steve Matyas和Andrew Glover合著的《Continuous Integration: Improving Software Quality and Reducing Risk》 【2】出版发行，该书获得了2008年的图灵大奖。持续集成理念经过10多年的发展，已经成为了业界的标准。在Java, Ruby的世界已经诞生了非常成熟的持续集成工具和实践，而对于iOS领域来说，因为技术本身相对比较年轻和苹果与生俱来的封闭思想，在持续集成方面的发展相对滞后一些，但是，随着越来越多的iOS开发者的涌入，以及各个互联网巨头加大对iOS开发的投入，诞生了一大批非常好用的持续集成工具和服务，本文的目的就是介绍一下如何有效的利用这些类库，服务快速构建一个iOS开发环境下的持续集成平台。

### 自动化构建

在MartinFowler的文章[1]中关于自动化的构建定义如下：

Anyone should be able to bring **in** a virgin machine, check the sources **out** of the repository, issue a single command, and have a running system on their machine.

因此，自动化构建的首要前提是有一个支持自动化构建的命令行工具，可以让开发人员可以通过一个简单的命令运行当前项目。

### 命令行工具

自动化构建的命令行工具比持续集成的概念要诞生得早很多，几十年前，Unix世界就已经有了Make，而Java世界有Ant, Maven，以及当前最流行的Gradle,.Net世界则有Nant和MSBuild。作为以GUI和命令行操作结合的完美性著称的苹果公司来说，当然也不会忘记为自己的封闭的iOS系统提供开发环境下命令行编译工具：xcodebuild 【3】

### xcodebuild

在介绍xcodebuild之前，需要先弄清楚一些在XCode环境下的一些概念【4】：

- **Workspace**: 简单来说，Workspace就是一个容器，在该容器中可以存放多个你创建的Xcode Project，以及其他项目中需要使用到的文件。使用Workspace的好处有，1),扩展项目的可视域，即可以在多个项目之间跳转，重构，一个项目可以使用另一个项目的输出。Workspace会负责各个Project之间提供各种相互依赖的关系;2),多个项目之间共享Build目录。
- **Project**: 指一个项目，该项目会负责管理生成一个或者多个软件产品的全部文件和配置，一个Project可以包含多个Target。
- **Target**: 一个Target是指在一个Project中构建的一个产品，它包含了构建该产品的所有文件，以及如何构建该产品的配置。
- **Scheme**: 一个定义好构建过程的Target成为一个Scheme。可在Scheme中定义的Target的构建过程有：Build/Run/Test/Profile/Analyze/Archive
- **BuildSetting**: 配置产品的Build设置，比方说，使用哪个Architectures？使用哪个版本的SDK？。在Xcode Project中，有Project级别的Build Setting，也有Target级别的Build Setting。Build一个产品时一定是针对某个Target的，因此，XCode中总是优先选择Target的Build Setting，如果Target没有配置，则会使用Project的Build Setting。

弄清楚上面的这些概念之后，xcodebuild就很好理解了，官网上对其作用的描述如下：

```
xcodebuild builds one or more targets contained in an Xcode project, or builds a scheme contained in an Xcode workspace or Xcode project.
```

xcodebuild就是用了构建产品的命令行工具，其用法可以归结为3个部分：

- 可构建的对象
- 构建行为
- 一些其他的辅助命令

可以构建的对象有，默认情况下会运行project下的第一个target：

- workspace: 必须和“-scheme”一起使用，构建该workspace下的一个scheme。
- project: 当根目录下有多个Project的时候，必须使用“-project”指定project，然后会运行
- target: 构建某个Target
- scheme: 和“-workspace”一起使用，指定构建的scheme。
- .....

## 构建行为包括：

- clean: 清除build目录下的
- build: 构建
- test: 测试某个scheme, 必须和"-scheme"一起使用
- archive: 打包, 必须和"-scheme"一起使用
- .....

## 辅助命令包括：

- -sdk: 指定构建使用的SDK
- -list: 列出当前项目下所有的Target和scheme。
- -version: 版本信息
- .....

关于xcodebuild更多详细的命令行请参见：<https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man1/xcodebuild.1.html>

下图是使用XcodeBuild运行一个scheme的build的结果：

```

setenv USER_LIBRARY_DIR ~/Users/twer/Library
setenv USE_DYNAMIC_NO_PIC YES
setenv USE_HEADERMAP YES
setenv USE_HEADER_SYMLINKS NO
setenv VALIDATE_PRODUCT NO
setenv VALID_ARCHS i386
setenv VERBOSE_PBXP NO
setenv VERSIONPLIST_PATH LighterViewControllerDemoTest2.octest/version.plist
setenv VERSION_INFO_BUILDER twen
setenv VERSION_INFO_FILE LighterViewControllerDemoTest2_vers.c
setenv VERSION_INFO_STRING "\"@(#)PROGRAM:LighterViewControllerDemoTest2 PROJECT:LighterViewControllerDemo-\""
setenv WRAPPER_EXTENSION octest
setenv WRAPPER_NAME LighterViewControllerDemoTest2.octest
setenv WRAPPER_SUFFIX .ocstest
setenv XCODE_APP_SUPPORT_DIR /Applications/Xcode.app/Contents/Developer/Library/Xcode
setenv XCODE_PRODUCT_BUILD_VERSION 4H1503
setenv XCODE_VERSION_ACTUAL 0463
setenv XCODE_VERSION_MAJOR 0400
setenv XCODE_VERSION_MINOR 0460
setenv YACC yacc
/bin/sh -c '/Users/twer/Library/Developer/Xcode/DerivedData/LighterViewControllerDemo-ajsfpngzwmnnobvarekhgixswf/Build/Intermediates/LighterViewControllerDemo.build/Debug-iphonesimulator/LighterViewControllerDemoTest2.build/Script-49EC8F5F179BE7AA008072EA.sh'
/Applications/Xcode.app/Contents/Developer/Tools/RunUnitTests:68: note: RunUnitTests exited without running tests because TEST_AFTER_BUILD was set to NO.

** BUILD SUCCEEDED **

```

了解了xcodebuild的用法之后，接下来分析一下xcodebuild的主要缺陷：

- 从上图直接可以得到的感觉，其脚本输出的可读性极差，
- 只能要么完整的运行一个target或者scheme，要么全部不运行。不能指定运行Target中特定的测试。
- 最令人发指的是，XCode 4中的xcodebuild居然不支持iOSUnitTest的Target【5】，当我尝试运行一个iOS App的测试target时，得到如下的错误：

```

Lincoln:LighterViewControllerDemo twer$ xcodebuild -scheme LighterViewControllerDemo -sdk iphonesimulator test
Build settings from command line:
SDKROOT = iphonesimulator6.1

xcodebuild: error: Failed to build project LighterViewControllerDemo with scheme LighterViewControllerDemo.
Reason: The run destination My Mac 64-bit is not valid for Testing the scheme 'LighterViewControllerDemo'.

```

对于上面提到的缺陷，Facebook给出了他们的解决方案：xctool【6】

# xctool

xctool在其主页直接表明了其目的：

*xctool is a replacement for Apple's xcbuild that makes it easier to build and test iOS and Mac products. It's especially helpful for continuous integration.*

其作用是替代xcbuild，目的是让构建和测试更加容易，更好的支持持续集成。从个人感受来看，它的确成功取代了xcbuild。但是xctool说到底只是对xcbuild的一个封装，只是提供了更加丰富的build指令，因此，使用xctool的前提是xcbuild已经存在，且能正常工作。

## 安装

xctool的安装非常简单，只需要clone xctool的repository到项目根目录就可以使用，如果你的机器上安装有Homebrew，可以通过“brew install xctool”命令直接安装。（注意：使用**xctool**前一定要首先确认**xcbuild**已安装且能正确工作）。

## 用法

关于xctool的用法就更加人性化了，几乎可以重用所有的xcbuild的指令，配置。只需要注意一下几点：

- xctool不支持target构建，只能使用scheme构建。
- 支持“-only”指令运行指定的测试。
- 支持多种格式的build报告。

例子：

```
path/to/xctool.sh
  -workspaceYourWorkspace.xcworkspace
  -schemeYourScheme
  test -only SomeTestTarget:SomeTestClass/testSomeMethod
```

下图是我使用xctool运行test的效果：

```

Lincoln:LighterViewControllerDemo twer$ xctool/xctool.sh -project LighterViewControllerDemo.xcodeproj -scheme LighterViewControllerDemo -sdk iphonesimulator test

** TEST **

[Info] Collecting build settings ... (612 ms)
xcodebuild build build
LighterViewControllerDemo / LighterViewControllerDemoTest2 (Debug)
  ✓ Check dependencies (44 ms)
  ✓ Run custom shell script 'Run Script' (10 ms)

/Applications/Xcode.app/Contents/Developer/Tools/RunUnitTests:68: note: RunUnitTests exited without running tests because TEST_AFTER_BUILD was set to NO.

LighterViewControllerDemo / LighterViewControllerDemo (Debug)

run-test LighterViewControllerDemoTest2.octest (iphonesimulator6.1, logic-test, GC OFF)
suite BasicTest
  ✓ -[BasicTest testSimpleMockPass] (0 ms)
  ✓ -[BasicTest testSimpleHomecrestMatcher] (0 ms)
  2 of 2 tests passed (11 ms)

suite CalculatorTest
  ✓ -[CalculatorTest testAdd] (0 ms)
  1 of 1 tests passed (0 ms)

** TEST SUCCEEDED: 3 of 3 tests passed ** (3966 ms)

```

## 常见问题：

No architectures to compile for (ONLY\_ACTIVE\_ARCH=YES, active arch=x86\_64, VALID\_ARCHS=armv7 armv7s).

解决方法：到Project Setting中，把"Build Active Architecture Only"设置为NO

Code Sign error: A valid provisioning profile matching the application's Identifier 'dk.muncken.MyApp' could not be found

解决方法：通过"-sdkiphonesimulator"指定SDK，从而能够使用符合iOS约定的application Identifier。

## 依赖管理

选定了命令行工具之后，接下来可以考虑下依赖管理的问题了。我到现在还记得几年前，刚从Ant转到使用Maven的那种爽快的感觉。后来，进入Ruby的世界，其与生俱来的Gem管理系统，也让其依赖管理变得极其简单。对于iOS平台来说，在做项目时，经常需要使用到各种各样的第三方Framework，这同样需要一个爽快的依赖管理系统，不然的话，各位可以想象一下重复的下载Framework文件，拖入各个Target的Build Phase的Link Binary With Libraries中的场景。这种重复的劳动对于“懒惰”的程序员来说，是很难接受的，于是，活跃的社区开发者们提供了这样的一个工具：Cocoapods【7】

Cocoapods开始于2011年8月12日，经过2年多的发展，现在已经超过2500次提交，并且持续保持活跃更新，目前已成为iOS领域最流行第三方依赖管理工具。从技术层面来说，其是一个Ruby Gem，从功能层面来说，其是一个iOS平台下的依赖管理工具，为iOS项目开发提供了类似于在Ruby世界使用Gem的依赖管理体验。

## 安装

前面提到cocoapods本质上是一个Ruby Gem，因此，其使用前提首先是Ruby开发环境。庆幸的是，Mac下都自带Ruby。这样，只需要简单的2条命令，就可以把cocoapods安装好：

```
$ [sudo] gem install cocoapods  
$ pod setup
```

## 用法

cocoapods的使用方式和使用Ruby Gem非常相似，首先需要在项目根目录下创建文件Podfile，在Podfile中，开发人员只需要按照规则配置好如下内容就好：

- 项目支持的平台，版本（iOS/OSX）
- 每个target的第三方依赖

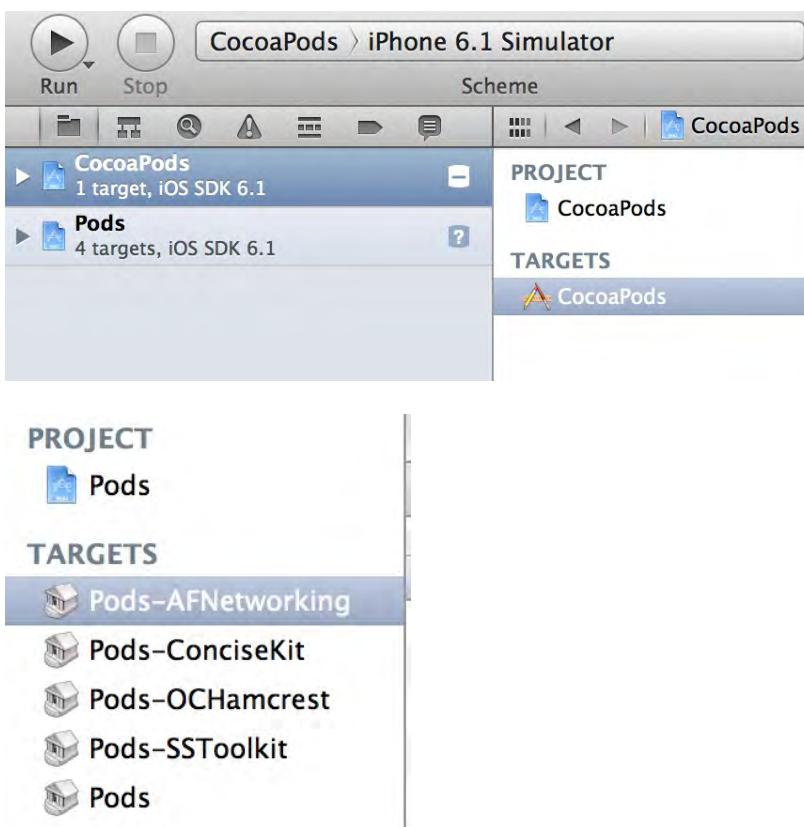
例子：

```
platform :ios, '6.0'  
inhibit_all_warnings!  
xcodeproj `MyProject`  
pod 'ObjectiveSugar', '~> 0.5'  
target :test do  
pod 'OCMock', '~> 2.0.1'  
end  
post_install do |installer|  
installer.project.targets.each do |target|  
puts "#{target.name}"  
end  
end
```

修改好配置文件之后，只需要简单的使用“pod install”即可安装好所有的依赖，执行该命令之后，在项目跟目录下会出现“.xcworkspace”和“Pods”两个目录：

```
lincoln:CocoaPods twer$ ll
total 56
drwxr-xr-x 12 twer staff 408 Jul 31 16:30 .
drwxr-xr-x 13 twer staff 442 Jul 22 21:22 ..
-rw-r--r--@ 1 twer staff 6148 Jul 23 09:42 .DS_Store
drwxr-xr-x 13 twer staff 442 Jul 23 19:16 .git
-rw-r--r-- 1 twer staff 125 Jul 22 21:22 .gitignore
drwxr-xr-x 15 twer staff 510 Jul 23 18:18 CocoaPods
drwxr-xr-x 5 twer staff 170 Jul 23 09:42 CocoaPods.xcodeproj
drwxr-xr-x 4 twer staff 136 Jul 23 09:34 CocoaPods.xcworkspace
-rw-r--r-- 1 twer staff 213 Jul 31 11:04 Podfile
-rw-r--r-- 1 twer staff 349 Jul 31 11:04 Podfile.lock
drwxr-xr-x 28 twer staff 952 Jul 31 11:04 Pods
-rw-r--r--@ 1 twer staff 7650 Jul 22 21:17 placeholder.png
```

接下来，开发者需要使用xcworkspace打开项目而不是使用xcodeproject，打开项目之后，在项目目录下除了自己的project以外，还可以看到一个叫做Pods的项目，该项目会为每一个依赖创建一个target：



在Podfile中，还可以指定依赖专属于某个Target，

```
target :CocoaPodsTest do
pod 'OCMock', '~> 2.0.1'
pod 'OCHamcrest'
end
```

如果你记不清楚某个依赖库的名称，可以使用“pod search <name>”模糊搜索依赖库中的相似库，另外，如果你想使用的库在cocoapods的中央库中找不到，那么，你可以考虑为开源社区做贡献，把你觉得好用的库添加到中央库中，Coc

cocoapods的官网上有具体的步骤【8】

## 原理

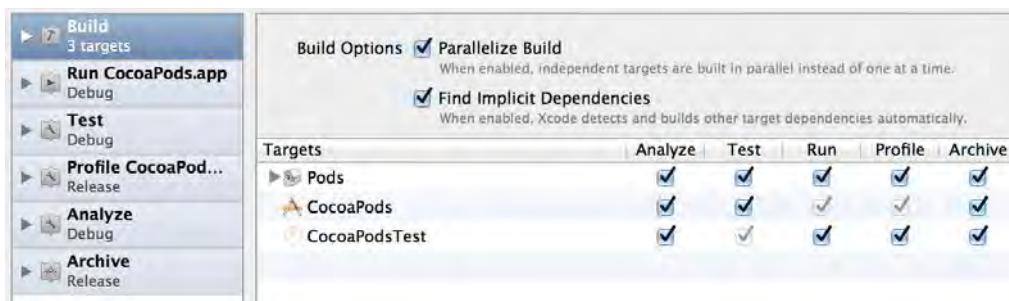
CocoaPods的原理思想基本上来自于Jonah Williams的博客“Using Open Source Static Libraries in Xcode 4”【9】，当使用“pod install”安装文件时，cocoapods做了如下这些事：

- 创建或者更新当前的workspace
- 创建一个新的项目来存放静态库
- 把静态库会编译生成的libpods.a文件配置到target的build phase的link with libraries中
- 在依赖项目中创建\*.xcconfig文件,指定在编译时的一些参数和依赖
- 添加一个新的名为“Copy Pods Resource”的Build Phase，该build phase会使用"\${SRCROOT}/Pods/Pods-CocoaPodsTest-resources.sh"把Pods下的资源文件拷贝到app bundle下。

## 注意事项

当使用xctool作为命令行工具构建项目时，使用cocoapods管理依赖时，需要做一些额外的配置：

- 编辑Scheme，把pods静态库项目作为显式的依赖添加到项目的build中，
- 把pods依赖项目拖动到本来的项目之上，表示先编译pods静态库项目，再编译自己的项目。



感谢张凯峰对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至[editors@cn.infoq.com](mailto:editors@cn.infoq.com)。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

原文链接：<http://www.infoq.com/cn/articles/build-ios-continuous-integration-platform-part1>

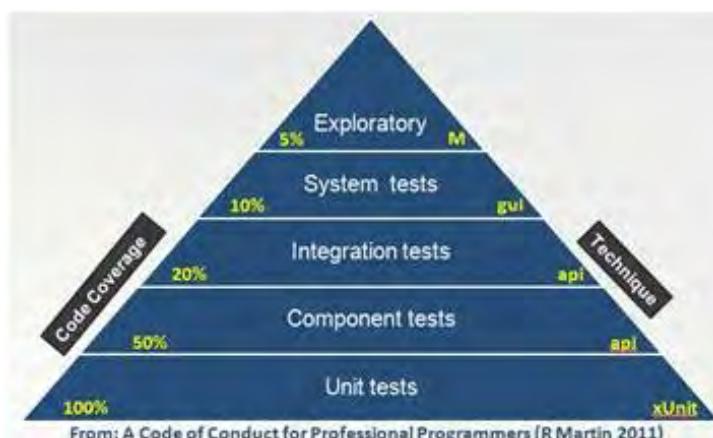
# 本期专题：构建iOS持续集成平台 | Topic

## 构建iOS持续集成平台（二）——测试框架

### 测试框架

作者 [刘先宁](#)

有了自动化构建和依赖管理之后，开发者可以很轻松的在命令行构建整个项目，但是，作为持续集成平台来说，最重要的还是测试，持续集成最大的好处在于能够尽早发现问题，降低解决问题的成本。而发现问题的手段主要就是测试。在Martin Fowler的Test Pyramid【10】一文中论述了测试金字塔的概念，测试金字塔的概念来自[Mike Cohn](#)，在他的书[Succeeding With Agile](#)中有详细描述：测试金字塔最底层是单元测试，然后是业务逻辑测试，如果更细化一点的话，可以分为把完整的测试策略分为如下的层级：



作为持续集成平台，能自动化的测试层级越多，平台就能产生越大的价值。

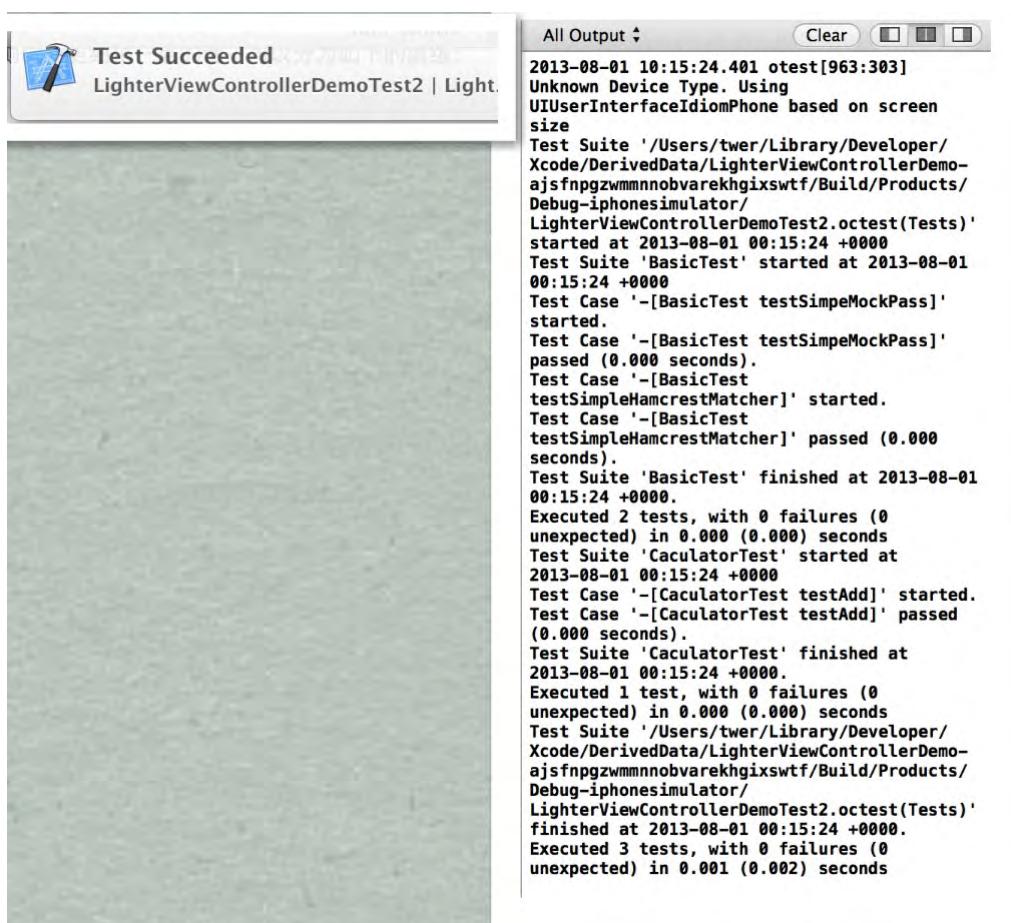
### Unit Test

目前，在iOS领域，最流行的Unit测试框架有2个：OCUnit【11】和GHUnit【12】，这两个框架各有其优缺点：

	优点	缺点
OCUnit	与Xcode无缝集成，快捷键，Scheme配置都非常方便	1. 只能一次运行整个测试，不能灵活的运行某个测试集；2. 测试结果输出的可读性不好，不容易找到失败的测试
GHUnit	1.自带GUI，测试结果清晰可见；2.可以灵活的运行指定的测试；3.开源项目	1.需开发者安装，配置略显复杂；2. 对命令行运行测试的支持不是很好，

OCUnit的运行结果会通过弹窗直接告诉开发者，运行的细节信息则会打印在Xc

ode的输出窗口中：



The screenshot shows the Xcode interface with the "Test Succeeded" status bar at the top. Below it is a large gray area representing the application's view. To the right is the "All Output" terminal window. The terminal output displays the log of the test run, including the start time (2013-08-01 10:15:24.401), the test suite path, and a detailed list of test cases and their outcomes. The log ends with "Executed 3 tests, with 0 failures (0 unexpected) in 0.001 (0.002) seconds".

```
All Output ▾ Clear [ ] [ ] [ ] [ ]  
2013-08-01 10:15:24.401 otest[963:303]  
Unknown Device Type. Using  
UIUserInterfaceIdiomPhone based on screen  
size  
Test Suite '/Users/twer/Library/Developer/  
Xcode/DerivedData/LighterViewControllerDemo-  
ajsfnpgzwmmnnobvarekhgixswtf/Build/Products/  
Debug-iphonesimulator/  
LighterViewControllerDemoTest2.octest(Tests)'  
started at 2013-08-01 00:15:24 +0000  
Test Suite 'BasicTest' started at 2013-08-01  
00:15:24 +0000  
Test Case '-[BasicTest testSimpleMockPass]'  
started.  
Test Case '-[BasicTest testSimpleMockPass]'  
passed (0.000 seconds).  
Test Case '-[BasicTest  
testSimpleHamcrestMatcher]' started.  
Test Case '-[BasicTest  
testSimpleHamcrestMatcher]' passed (0.000  
seconds).  
Test Suite 'BasicTest' finished at 2013-08-01  
00:15:24 +0000.  
Executed 2 tests, with 0 failures (0  
unexpected) in 0.000 (0.000) seconds  
Test Suite 'CalculatorTest' started at  
2013-08-01 00:15:24 +0000  
Test Case '-[CalculatorTest testAdd]' started.  
Test Case '-[CalculatorTest testAdd]' passed  
(0.000 seconds).  
Test Suite 'CalculatorTest' finished at  
2013-08-01 00:15:24 +0000.  
Executed 1 test, with 0 failures (0  
unexpected) in 0.000 (0.000) seconds  
Test Suite '/Users/twer/Library/Developer/  
Xcode/DerivedData/LighterViewControllerDemo-  
ajsfnpgzwmmnnobvarekhgixswtf/Build/Products/  
Debug-iphonesimulator/  
LighterViewControllerDemoTest2.octest(Tests)'  
finished at 2013-08-01 00:15:24 +0000.  
Executed 3 tests, with 0 failures (0  
unexpected) in 0.001 (0.002) seconds
```

GHUnit的运行结果则全部显示在自己的应用界面中，开发者可以在应用中查看所有的信息，以及做运行测试等各种各样的操作。



关于如何使用OCUnit和GHUnit，InfoQ上有高嘉峻的文章《iOS开发中的单元测试》（<http://www.infoq.com/cn/articles/ios-unit-test-1>）有详细的介绍，我就不再这儿重复叙述了。

如果单从单元测试框架来看，个人更喜欢GHUnit测试结果的可读性和运行测试的灵活性，但是，随着Facebook的xctool的发布，OCUnit华丽丽的逆袭了，因为xctool帮助OCUnit把运行测试的灵活性和测试结果的可读性这两块短板给补齐了，再加上其和Xcode的集成优势以及通过命令行运行的便捷性，让其成为持续集成平台的Unit测试框架的首选。

在Java程序员的心中，Junit和Hamcrest永远是一体的，Hamcrest为junit提供了非常丰富的断言机制，极大的增强了测试的可读性。越来越活跃的iOS开发社区，当然不会让Object-C的世界缺失这样一个优秀的框架，于是OCHamcrest【13】诞生了。

在测试项目中使用OCHamcrest非常简单，尤其是使用了cocoapods管理依赖的项目。只需要在Podfile文件中加上：

```
target :<TestTargetName> do
  ...
  pod 'OCHamcrest'
end
```

然后，运行“pod install”命令安装Hamcrest到测试Target，安装好之后，为了在测试类中使用OCHamcrest的断言。还需要在测试类的头文件中加入如下代码：

```
#define HC_SHORTHAND
#import<OCHamcrest/OCHamcrest.h>
```

开发者可以把这段代码加入<TestTargetName>-prefix.pch中，这样所有的测试类就都可以使用OCHamcrest的断言了。在前面提到的高嘉峻的文章中的第二部分更加详细的讲解了OCHamcrest的断言，以及其和另一个断言框架Expecta的对比，感兴趣的同学可以跳过去看看（[http://www.infoq.com/cn/articles/Matching-Engine-Enliven-Assertion-2?utm\\_source=infoq&utm\\_medium=related\\_content\\_link&utm\\_campaign=relatedContent\\_articles\\_clk](http://www.infoq.com/cn/articles/Matching-Engine-Enliven-Assertion-2?utm_source=infoq&utm_medium=related_content_link&utm_campaign=relatedContent_articles_clk)）。

## Component Test & Integration Test

在开发手机应用时，总难免会和其他的系统集成，尤其当开发的应用是某个系统的手机客户端时，这样就涉及到很多第三方API的集成点需要测试，在成熟的Java世界中，诞生了EasyMock，Mockito，moco等针对这种集成点的测试工具。同样的，活跃的社区力量正一步一步的在让Object-C世界成熟，OCMock【14】诞生。

## OCMock

有了cocoaPods，新加框架变得非常容易，基本上就是“哪里没有加哪里”的节奏，添加OCMock框架，只需要在Podfile文件中加上：

```
target :<TestTargetName> do
  ...
  pod 'OCMock', '~> 2.0.1'
end
```

然后，运行“pod install”命令安装OCMock到测试Target，同样的，需要把OCmock的头文件添加<TestTargetName>-prefix.pch文件中

```
#import<OCMock/OCMock.h>
```

下面是一个简单的使用OCMock的例子，更多的用法请参考OCMock的官网：<http://ocmock.org/features/>：

```
- (void)testSimpleMockPass{
    id mockObject = [OCMockObject mockForClass:[NSString class]];
    [[[mockObject stub] andReturn:@[@"test"]] lowercaseString];

    NSString * returnValue = [mockObject lowercaseString];
    assertThat(returnValue, equalTo(@"test"));
}
```

## moco

moco【15】以其对系统集成点测试的贡献荣获了2013年的“Duke's Choice Award”奖，虽然其以Java写成，主要的生态系统也是围绕Java打造，但是，其Standalone模式可以非常方便的构造一个开发者期望的服务器，这对于Mobile领域的集成点测试来说，本就是一个非常好的Mock服务器的工具。Moco有如下的特点：

- 易于配置，使用：一个Json配置文件，然后“java -jar moco-runner--standalone.jar -p 8080 \*\*\*.json”就可以启动一个Mock服务器。该服务器的所有行为都在配置文件里。如果你想添加，修改服务器行为，只需要修改一下配置文件，然后重新启动该服务器就行了。
- 配置文件可读性好，使用Json格式的配置文件，对绝大多数开发者来说都可以很容易理解。
- 支持模拟客户端需要的所有http操作，moco实现了针对请求Content、URI、Query Parameter、Http Method、Header、Xpath的模拟。对响应的格式支持有Content、Status Code、Header、URL、甚至支持Sequence请求，即根据对同一请求的调用次数返回不同的结果。
- 完全开源，代码不多也比较易懂，如果没有覆盖到我们的场景，完全可以在该项目基础上实现一个自己的Mock服务器。

熊节在infoQ上发表的《企业系统集成点测试策略》【16】一文中，详细的论述了在企业系统中，moco对测试系统集成点的帮助。这些论点在Mobile开发领域同样适用，因此合理的使用moco可以帮助iOS开发者更加容易的构建一个稳固的持续集成平台。

## System Test

对于iOS的系统（UI）测试来说，比较知名的工具有UIAutomation【17】和FonMonkey【18】。

### UIAutomation

UIAutomation是随着iOS SDK 4.0引入，帮助开发者在真实设备和模拟器上执行自动化的UI测试。其本质上是一个Javascript的类库，通过界面上的标签和值的访问性来获得UI元素，完成相应的交互操作，从而达到测试的目的，类似于Web世界的Selenium。

通过上面的描述，可以得知，使用UIAutomation做测试时，开发者必须掌握两件事：

- 如何找到界面上的一个UI元素
- 如何指定针对一个UI元素的操作

在UIAutomation中，界面就是由一堆UI元素构建的层级结构，所有UI元素都继承对象UIAElement，该对象提供了每个UI元素必须具备的一些属性：

- name
- value
- elements

- parent
- ...

而整个界面的层级结构如下：

```

Target (设备级别的UI, 用于支持晃动, 屏幕方向变动等操作)
Application (设备上的应用, 比方说Status Bar, keyboard等)
Main window (应用的界面, 比方说导航条)
View (界面下的View, 比方说UITableView)
Element (View下的一个元素)
Child element (元素下的一个子元素)

```

下面是一个访问到Child element的例子：

```
UIATarget.localTarget().HamcrestDemo().tableViews()[0].cells()[0].elements()
```

开发者还可以通过“UIATarget.localTarget().logElementTree()”在控制台打印出该target下所有的elements。

找到UI元素之后，开发者可以基于该UI元素做期望的操作，UIAutomation作为原生的UI测试框架，基本上支持iOS上的所有UI元素和操作，比方说：

- 点击按钮，例: \*\*\*.buttons[“add”].tap()
- 输入文本，例: \*\*\*.textfields[0].setValue(“new”)
- 滚动屏幕，例: \*\*\*.scrollToElementWithPredicate(“name begin with ‘test’”)
- .....

关于使用UIAutomation做UI测试，推荐大家一定要看一下2010的WWDC的Session 306：[Automating User Interface Testing with Instruments 【19】](#)。另外，这儿还有一篇很好的博客，详细的讲解了如何使用UIAutomation做UI自动化测试：<http://blog.manbolo.com/2012/04/08/ios-automated-tests-with-uiautomation>

Apple通过Instruments为UIAutomation测试用例的命令行运行提供了支持，这样就为UIAutomation和CI服务器的集成提供了便利。开发者可以通过如下的步骤在命令行中运行UIAutomation测试脚本

1. 指定目标设备，构建被测应用，该应用会被安装到指定的DSTROOT目录下

```
xcodebuild
-project "/Users/twer/Documents/xcodeworkspace/AudioDemo/AudioDemo.xcodeproj"
-scheme AudioDemo
-sdk iphonesimulator6.1
-configuration Release SYMR0OT="/Users/twer/Documents/xcodeworkspace/
AudioDemo/build" DSTROOT="/Users/twer/Documents/xcodeworkspace/AudioDemo/
build" TARGETED_DEVICE_FAMILY="1"
install
```

## 2. 启动Instruments，基于第一步生成的应用运行UIAutomation测试

```
instruments
-t "/Applications/Xcode.app/Contents/Applications/Instruments.app/
Contents/PlugIns/AutomationInstrument.bundle/Contents/Resources/
Automation.tracetemplate" "/Users/twer/Documents/xcodeworkspace/AudioDemo/
/build/Applications/TestExample.app"
-e UIASCRIPT <absolute_path_to_the_test_file>
```

为了更好的展示测试效果以及与CI服务器集成，活跃的社区开发者们还尝试把UI Automation和jasmine集成：<https://github.com/shaune/jasmine-ios-access-tests>

UIAutomation因其原生支持，并且通过和Instruments的绝佳配合，开发者可以非常方便的使用录制操作自动生成测试脚本，赢得了很多开发者的支持，但是因苹果公司的基因，其系统非常封闭，导致开发者难以扩展，于是活跃的社区开发者们开始制造自己的轮子，Fone Monkey就是其中的一个优秀成果。

## Fone Monkey

Fone Monkey是由Gorilla Logic 公司创建并维护的一个iOS自动化测试工具，其功能和UIAutomation差不多，但是由于其开源特性，极大的解放了活跃开发者的生产力，开发者可以很容易的根据自身需要扩展其功能。

Fone Monkey的安装虽然简单，但是比UIAutomation的原生支持来说，也算是的一大劣势了，具体的安装过程可以参考：[http://www.gorillalogic.com/fone\\_monkey-ios/fonemonkey-setup-guide/add-fonemonkey-your-xcode-project](http://www.gorillalogic.com/fone_monkey-ios/fonemonkey-setup-guide/add-fonemonkey-your-xcode-project)

Fone Monkey的使用方式主要就是录制/回放，也可以把录制好的测试用例保存为脚本以其他方式运行。安装好Fone Monkey启动测试以后，应用界面会有点变化：



开发者通过点击Record按钮录制操作，点击Play按钮播放之前录制的操作，点击More按钮可以添加一些针对元素的验证，这样就形成了一个测试用例。



在Fone Monkey中录制的测试用例可以保存为3种格式，以支持多种运行方式：

- `scriptName.fm`: 用于支持在Fone Monkey的窗口中运行测试
- `scriptName.m`: 用于和Xcode的OCUnit test 集成，可以以OCUnit的测试用例的形式运行UI测试，这就让UI具备了命令行运行和与CI集成的能力。
- `scriptName.js`: UIAutomation的格式，用于支持在Instruments中，以UIAutomation的形式运行测试。

原文链接：<http://www.infoq.com/cn/articles/build-ios-continuous-integration-platform-part2>

## 相关内容

- [构建iOS持续集成平台（三）——CI服务器与自动化部署](#)
- [构建iOS持续集成平台（一）——自动化构建和依赖管理](#)
- [云计算与持续集成——七牛的实践](#)
- [爱立信软件开发高级专家蔡煜：自动化测试和持续集成如何保持激情？](#)
- [敏捷自动化测试（4）——围绕自动化测试开展持续集成](#)

# 本期专题：构建iOS持续集成平台 | Topic

## 构建iOS持续集成平台（三）——CI服务器与自动化部署

### CI服务器

作者 [刘先宁](#)

写到这儿，对于iOS开发者来说，需要准备好：

- 一个比较容易获取的源代码仓库(包含源代码)
- 一套自动化构建脚本
- 一系列围绕构建的可执行测试

接下来就需要一个CI服务器来根据源代码的变更触发构建，监控测试结果。目前，业界比较流行的，支持iOS构建的CI服务器有Travis CI和Jenkins

### Travis CI

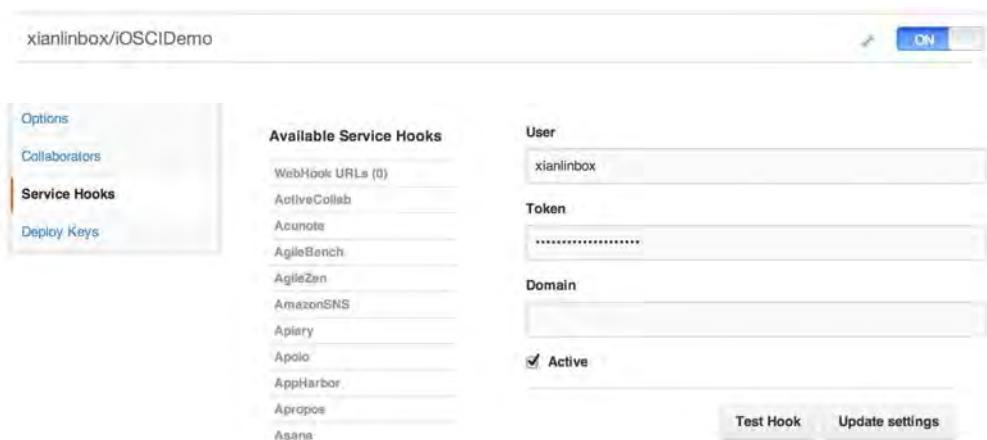
Travis CI 【20】是一个免费的云服务平台，主要功能就是为开源社区提供免费的CI服务，对于商业用户可以使用Travis Pro版本，其基本上支持所有目前主流的语言，Object-C自然也在其中。但是，Travis CI只支持github极大的限制了其应用场景。但是也由于其只支持github，其把和github的集成做到了极致的平滑，易用，因此，对于本就把github作为代码托管平台的项目来说，Travis CI可以做为第一选择。

使用Travis CI只需要简单的2步即可为你托管在github的项目增加一个CI服务器

第一步：在项目的根目录下创建travis CI配置文件“.travis.yml”，在配置文件指定语言，环境要求，构建脚本等

```
language: objective-c
before_install:
  - brew update
  - brew install xctool
script: xctool -project LighterViewControllerDemo.xcodeproj -scheme LighterViewControllerDemo -sdkiphonesimulator test
```

第二步：使用github账号登陆Travis CI，在账户的repositories开启该项目的自动构建功能，该设置会在github上该项目repository中开启对Travis CI的Service Hook



下图就是我的一个iOS项目在Travis CI的构建记录：

除此之外，Travis CI还非常贴心的提供了一个指示灯，可以让开发者在自己的repository展示构建状态。使用指示灯的方法很简单，只需要在repository的README.md中添加下面这行代码就行了。



效果如下：



## Jenkins

Jenkins【21】经过多年的发展，其活跃的社区和丰富的插件让其成为了业界最受欢迎的CI服务器。通过使用Xcode插件，可以非常方便在Jenkins中运行iOS项目的构建脚本。

## 安装

Jenkins的安装非常简单，尤其是在Mac环境下使用Homebrew安装，只需要简单的使用“brew install jenkins”，即可成功安装，这个安装过程做的事情非常简单，就是把对应版本的jenkins.war文件下载到对应的目录“/usr/local/Cellar/jenkins/1.524/libexec/”。因此，如果没有Homebrew，可以直接到官网下载安装文件，自己安装。

配置安装完之后，只需要使用“java -jar \*/jenkins.war”即可启动Jenkins，开发者可以自己创建一个bash alias简化输入，在用户目录下的.bashrc文件中添加如下代码

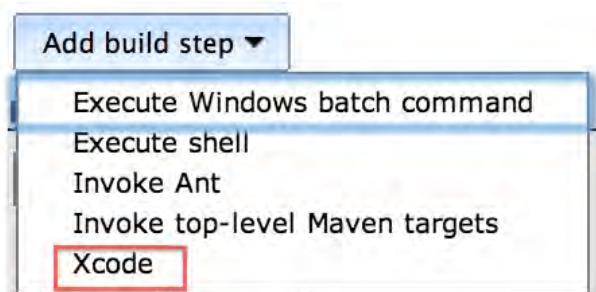
```
alias jenkins='java -jar /Applications/Jenkins/jenkins.war'
```

启动Jenkins之后，可以通过浏览器访问http://localhost:8080查看Jenkins界面。

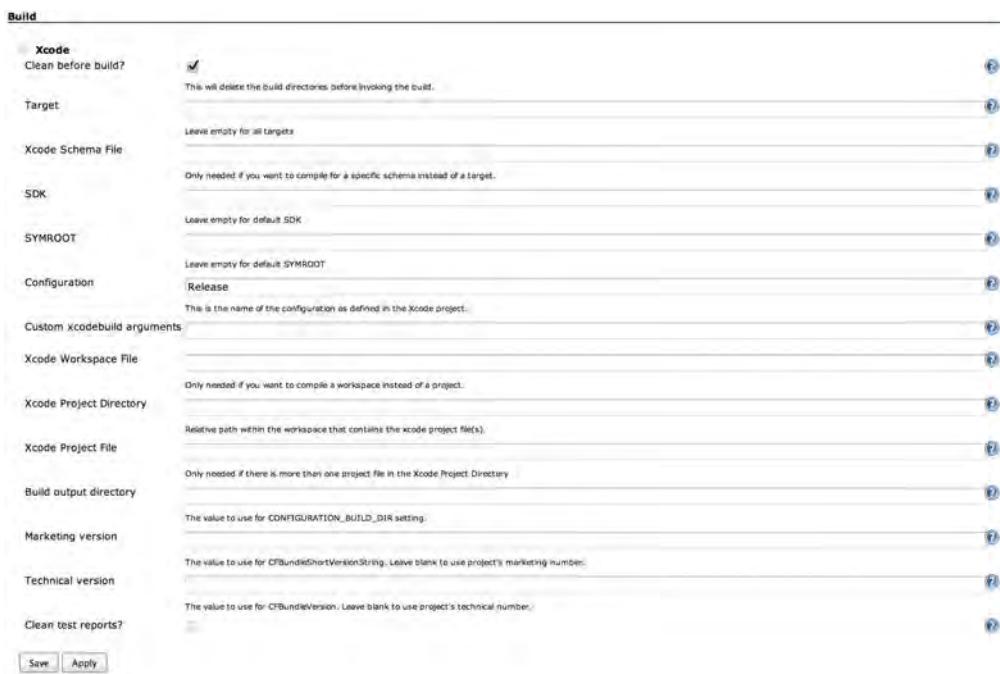
然后，开发者可以到Manage Plugins界面，安装需要的插件：Xcode Plugin，Git Plugin，Github Plugin（我使用git做源代码管理），Cocoapods Plugin。安装好插件之后，需要重启Jenkins，一切就绪之后，开始配置自己的iOS构建过程。

在Jenkins中配置一个iOS Job的步骤如下：

1. 新建Job，Job类型选择“Build a free-style software project”。
2. 配置代码仓库，
3. 点击“Add build step”添加构建步骤，如果已安装Xcode插件，则可以在Step类型中看到Xcode选项：



选择Xcode，可以看到Xcode构建step的所有配置选项：



4. 点击“Add build step”添加测试步骤,选择“Execute shell”选项,然后,添加脚本,执行测试并生成期望的Report,可以重复本步骤添加“Acceptaince Test”等构建步骤。

```
path-to/xctool.sh
-workspaceAudioDemo.xcworkspace -scheme AudioDemo -sdkiphonesimulator
-reporter junit:test-reports/junit-report.xml clean test
```

5. 点击“Add post-build action”添加一个新的步骤,选择“publish Junit test result report”,把测试报告展示在Jenkins中。



配置好之后,可以点击Build Now运行Job,下图是在Jenkins中运行iOS构建Job的结果图:

The screenshot shows the Jenkins Project AudioDemo dashboard. On the left, there's a sidebar with links like Status, Changes, Workspace, Build Now, Delete Project, Configure, GitHub, GitHub Hook Log, and Build History (trend). The main area displays a "Test Result Trend" chart showing a count of failures from build #3 to #6. Below the chart is a list of recent builds. A "Permalinks" section lists links for each build.

开发者可以点击每次的构建，查看具体的构建信息及测试结果：

This screenshot shows the "Test Result : AudioDemoTests" page for build #6. It displays 0 failures and 3 tests. The "All Tests" table shows three successful test cases: "testExample", "testExample2", and "testExample3", all with 0 ms duration and Passed status.

Test name	Duration	Status
testExample	3 ms	Passed
testExample2	0 ms	Passed
testExample3	0 ms	Passed

## 常见问题

启动Jenkins提示运行“java”命令需要X11支持，

解决方法：这是因为在OS X Mountain Lion系统中不再内置X11，当需要使用X11时，可通过安装XQuartz project得到支持，具体信息：<http://support.apple.com/kb/HT5293>

## 自动化部署

这儿的想谈的“部署”不是传统意义上的直接部署到产品环境的部署，而是指如何把最新版本的应用快速的部署到测试用户的机器上以收集反馈，或者做一些探索性的测试。

在我写第一个iOS应用的时候，我想把应用安装到多个机器上测试的时候，需要非常繁琐的步骤：

1. 需要申请到苹果开发者账号，获得开发者证书。

2. 需要在苹果的开发者网站上注册我想使用的设备。
3. 使用开发者证书打包应用，使用Ad-HOC部署模式，生成ipa文件。
4. 通过ipa文件把应用安装到iTunes上。
5. 通过iTunes把应用同步到多台测试机器上。

如果是测试机器在多个地理位置的时候，还需要把ipa文件发送到对应的地点，每个地点都需要重复的做第4，5步。这样一个繁琐，且低效的过程让开发者非常痛苦，直到TestFlight的出现。

## TestFlight

TestFlight【22】就是一个专门解决上面提到的痛点的云服务方案，它可以帮助开发者：

- 轻松采集测试用户的UDID和iOS 版本、硬件版本，并发送给开发者。
- 实时反馈应用是否成功安装到测试机器
- 轻松部署最新版本应用到测试用机上。
- 开发者可以灵活选择部署哪个版本到哪部分测试机器上。

使用使用Test Flight服务非常简单，只需要到Test Flight注册一个账号。然后把链接发送给测试设备，测试设备只要打开该链接，并授权给Test Flight，在Test Flight的设备中心就可以看到这些设备。

The screenshot shows the 'User Details' section of the TestFlight website. On the left, there's a sidebar with user information (Xianning Liu, xianning.liu@gmail.com) and roles (Leader, Developer, Tester). The main area is titled 'Registered Devices' and lists two devices:

Platform	Hardware	OS	UDID	Actions
iOS	iPod Touch 4th Gen	iOS 6.1.3	b3210de600a250...79db641070	<a href="#">Export</a>
iOS	iPad 4 Wi-Fi	iOS 6.0.1	d9723a43e0a5a61...e55f3cf1c	<a href="#">Export</a>

而测试设备上，则会多一个Test Flight的应用。



当应用有了新的测试包之后，只需要将IPA上传到TestFlight网站，然后勾选合适的测试用户或者合适的设备，点击Update & Notify。

The screenshot shows the TestFlight dashboard interface. At the top, there are sections for 'User' and 'Devices'. Under 'User', it lists 'Xianning Liu' with a checkmark. Under 'Devices', it shows 1 device. Below these are two main sections: 'TestFlight Users In The Provisioning Profile' and 'Anonymous Devices In The Provisioning Profile'. The first section lists users: 'User' (Cyril Wei, Han Qin, Jack Wang, xuehai zeng), each with 1 device. The second section lists UDIDs: d58c71fa8d06218b..., 8a7b86fcf61d8cf..., 91295cc6973937b..., 84f0078e56b43f2..., and 225a238f0626095..., each also with 1 device. At the bottom, there is a note: 'NOTE: Update & Notify will send a message to all freshly permitted users, as well as permitted users who haven't installed your build yet.' followed by two buttons: 'Just Update' and 'Update & Notify'.

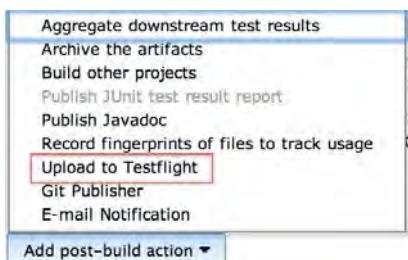
TestFlight会向对应的测试设备发送更新通知，测试用户只需在测试设备上打开TestFlight应用，点击Install，TestFlight就会自动将新版本的IPA文件安装到测试设备上。



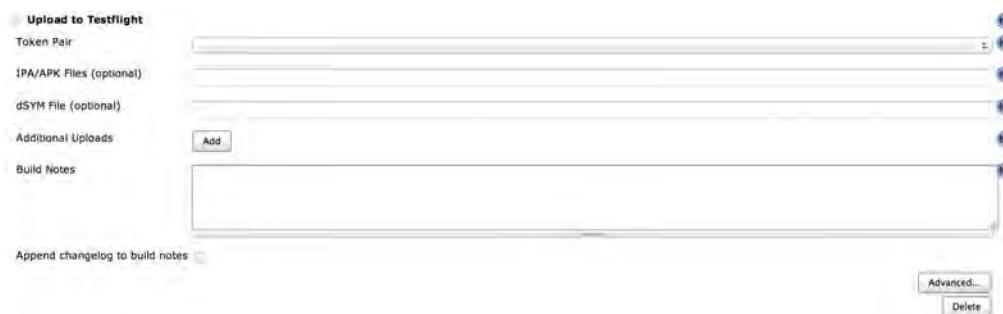
另外，TestFlight还提供了Upload API，这样就等于提供了和CI服务器集成的能力。其Upload API非常简单，就是一个简单的，带有指定参数的HTTP POST请求，具体细节可参考官网：<https://www.testflightapp.com/api/doc/>。

在CI服务器中，开发者可以在构建过程中，添加步骤通过upload API把通过测试的ipa文件自动上传到TestFlight上，从而达到持续部署的目的。而像Jenkins这样有丰富插件机制的CI服务器，活跃的社区开发者们早为其开发了十分便于使用的TestFlight的插件。

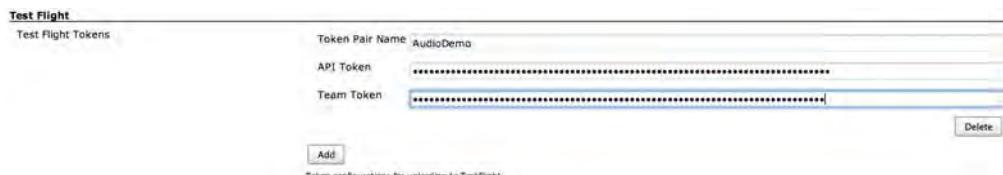
在Jenkins中使用TestFlight插件也非常简单，安装好插件，重启Jenkins，然后在项目的构建过程配置页面的Post-build Actions中，点击add post-build action，可以看到Upload to Testflight选项：



选择之后，可以在页面上看到TestFlight的配置项：



为了增强安全性，该插件把Token的设置移到了Jenkins的Global Setting界面  
：



配置好Token Pair之后，在TestFlight的配置项 上选择相应的pair即可，设置想要的参数，保存即可。

你如果不使用插件或者说使用的其它CI服务器的话，可以通过添加一个Execute shell步骤，直接通过代码上传构建好的ipa文件：

```
curl http://testflightapp.com/api/builds.json
-F file=@testflightapp.ipa
-F dsym=@testflightapp.app.dSYM.zip
-F api_token='your_api_token'
-F team_token='your_team_token'
-F notes='This build was uploaded via the upload API'
-F notify=True
-F distribution_lists='Internal, QA'
```

## 结语

《持续集成》一书中引用了Javaranch.com的创始人Kathy Sierra的一句话：

There's a big difference between saying, "Eat an apple a day" and actually eating the apple

正所谓知易行难，您几乎很难听到开发者说：“持续集成毫无价值”，但是，构建一个持续集成平台并非易事，希望本文中介绍的iOS应用的构建过程，以及在构建过程的各个阶段可以使用的一些优秀的类库，服务，能够让iOS开发者们在想搭建一个持续集成平台时有所参考，从而能够更加坚定，且容易的为自己的项目搭建一个持续集成平台。

## 参考文献

1. Matin Fowler's Blog: Continuous Integration (<http://martinfowler.com/articles/continuousIntegration.html>)
2. Continuous Integration: Improving Software Quality and Reducing Risk (<http://www.amazon.com/gp/product/0321336380?ie=UTF8&tag=martinfowlerc-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=0321336380>)
3. XCodeBuildMannualPage(<https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man1/xcodebuild.1.html>)
4. XCodeConcepts([http://developer.apple.com/library/ios/#featuredarticles/XcodeConcepts/Concept-Schemes.html#/apple\\_ref/doc/uid/TP40009328-CH8-SW1](http://developer.apple.com/library/ios/#featuredarticles/XcodeConcepts/Concept-Schemes.html#/apple_ref/doc/uid/TP40009328-CH8-SW1))
5. Running OCUnit (or Specta) Tests from Command Line (<http://www.raingrove.com/2012/03/28/running-ocunit-and-specta-tests-from-command-line.html>)
6. xctool(<https://github.com/facebook/xctool>)
7. cocoapods(<http://cocoapods.org/>)
8. cocoapods: contributing \_to\_the\_master\_repo([http://docs.cocoapods.org/guides/contributing\\_to\\_the\\_master\\_repo.html](http://docs.cocoapods.org/guides/contributing_to_the_master_repo.html))
9. Using Open Source Static Libraries in Xcode 4([http://blog.carbonfive.com/2011/04/04/using-open-source-static-libraries-in-xcode-4/#creating\\_a\\_workspace](http://blog.carbonfive.com/2011/04/04/using-open-source-static-libraries-in-xcode-4/#creating_a_workspace))
10. Test Pyramid(<http://martinfowler.com/bliki/TestPyramid.html>)
11. OCUnit (<http://cocoadev.com/OCUnit>)
12. GHUnit (<http://gabriel.github.io/gh-unit/>)
13. OCHamcrest (<https://github.com/hamcrest/OCHamcrest>)
14. OCMock (<http://ocmock.org/>)
15. moco (<https://github.com/dreamhead/moco>)
16. 企业系统集成点测试策略 (<http://www.infoq.com/cn/articles/enterprise-systems-integration-points>)

17. UIAutomation ( [http://developer.apple.com/library/ios/#documentation/DeveloperTools/Reference/UIAutomationRef/\\_index.html](http://developer.apple.com/library/ios/#documentation/DeveloperTools/Reference/UIAutomationRef/_index.html) )
18. Fone Monkey ( <http://www.gorillalogic.com/fonemonkey-ios> )
19. Travis CI ( <http://about.travis-ci.org/docs/> )
20. WWDC 2010 Session 306: [Automating User Interface Testing with Instruments](http://developer.apple.com/devcenter/download.action?path=/wwdc_2010/wwdc_2010_video_assets_pdfs/306_automating_user_interface_testing_with_instruments.pdf) ([http://developer.apple.com/devcenter/download.action?path=/wwdc\\_2010/wwdc\\_2010\\_video\\_assets\\_pdfs/306\\_automating\\_user\\_interface\\_testing\\_with\\_instruments.pdf](http://developer.apple.com/devcenter/download.action?path=/wwdc_2010/wwdc_2010_video_assets_pdfs/306_automating_user_interface_testing_with_instruments.pdf))
21. Jenkins(<http://jenkins-ci.org/>)
22. TestFlight(<https://testflightapp.com/>)

原文链接：<http://www.infoq.com/cn/articles/build-ios-continuous-integration-platform-part3>

## 相关内容

- [构建iOS持续集成平台（二）——测试框架](#)
- [构建iOS持续集成平台（一）——自动化构建和依赖管理](#)
- [云计算与持续集成——七牛的实践](#)
- [爱立信软件开发高级专家蔡煜：自动化测试和持续集成如何保持激情？](#)
- [敏捷自动化测试（4）——围绕自动化测试开展持续集成](#)

# 百度开发者中心

Baidu Developer Center

应用万象 云创未来



百度秉持“平等地成就每一个开发者”的理念，围绕百度云平台能力整合开放给开发者四大服务体系：开发支持、运营支撑、渠道推广、商业变现。

我们希望藉由这些服务的开放，让开发者在开发、运营、推广、变现各环节有所受益，最大程度地保证开发者的利益，共建开放共赢的局面！



开发支持



运营支撑



渠道推广



商业变现

## 推荐文章 | Article

### 中国IT高管谈创新型组织

作者 熊节

自从彼得·德鲁克提出“创新型组织”的概念，一个充满创新活力的企业就一直是老板和员工共同向往的理想工作环境。作为处于整个社会创新前沿的IT行业，企业的领导者们是如何看待创新与创新型组织，他们又如何打造各自组织的创新力？带着这些问题，InfoQ采访了几位IT企业的领导者。他们是：

- 谢恩伟，微软大中华区开发工具及平台事业部总经理。
- 冯晓焰，来自于英特尔亚太研发有限公司，隶属于开源软件技术中心中国团队，现为英特尔中国的首席开源科学家。
- 汤鹏，易到用车CTO。
- 陈磊，腾讯云平台总经理，开放平台副总经理。

### 定义创新力

对于创新力给组织带来的价值，这几位IT高管都很重视。谢恩伟认为“创新是微软的DNA”，三十多年来微软一直以技术变革为用户创造新的价值与体验。冯晓焰指出：作为以开源软件为主要工作目标的团队，英特尔开源软件技术中心的成绩离不开内在的创新能力，以创新能力为基础的技术领导力和影响力是至关重要的。汤鹏则看重创新对于企业内部的正面影响，他指出创新能促进产品经理与项目团队之间、主管与部门员工之间的协作，并能催生企业内部融洽的文化。

然而对于“什么是创新”，IT高管们的看法略有不同。一种观点认为，“创新”意味着用更好的方式处理现有的工作。汤鹏就认为，只要有助于提高效率、促进沟通、帮助成长的事物，都可以被视为创新。另一种观点则认为，创新意味着针对全新的问题、采用全新的方式方法、创造全新的使用模式。冯晓焰这样说道：

“以现代计算机为例，它的存储系统，经历了从磁带，磁盘再到SSD的演变过程，在这个过程中，要解决的都是一个数据的存储问题，创新在于不同的技术引入的不同方式方法。而现代计算机系统的从无到有是一个从问题到方式方法进而使用模式的彻底的创新。”

不管持哪种观点，有一点是大家公认的：在创新的过程中，人的创造性思维是不可或缺的。

这种对创新力的重视并非随口说说。受访的这几家企业在最近一年中分别都有令人瞩目的创新成果。微软在2012年发布了Windows 8和平板移动设备Surface这两个重量级产品，并研发出了众多创新的云计算产品。英特尔开源软件技术中心与三星合作推出了基于Linux的Tizen操作系统，并推出了能极大提高HTML 5并行性的Parallel JS方案。

国内企业在创新成果上也不遑多让。易到用车从公司内部工程师实验室诞生的“打车小秘”是国内第一款打车类App，并率先开发了通过微信实现打车功能的“微打车”产品。腾讯云通过全国部署的200多个网络节点实现移动加速，移动腾讯分析产品通过标签来识别各种重要的用户特征，可以指导游戏和社交应用的开发者进行有针对性的运营活动。陈磊表示：

“这些都还只是我所负责的业务中的一小部分创新。腾讯公司作为一个整体，每天都在创造出类似的创新。”

## 企业环境对于创新至关重要

谈到如何打造创新型组织，几位高管不约而同地谈到了企业环境的重要性。谢恩伟这样介绍微软的环境：

“微软努力为员工营造一个宽松而又自由的工作环境，鼓励每个人最大程度的发挥智慧，使微软成为最佳的创新孵化器。对于员工而言，需要鼓励他们不仅思考当下的工作，更要思考他们未来所要传递的理念。这包括对未来三年、五年，甚至十年的发展趋势做出展望和规划，要让所有的员工不仅考虑产品的今天，也要考虑产品的未来。”

汤鹏认为，一个鼓励创新的环境需要从上到下三个层面共同作用：

- 首先是鼓励发言的文化，倡导员工积极表达自己的观点；
- 然后是团队的理解和支持，让团队主动迎接变化，而不是自上而下干硬地实施改变；
- 最后必须有说到做到的执行作为保障。

冯晓焰也强调了开放的工作环境和气氛对于鼓励创新的价值。另外他指出评价机制对于创新的重要意义：鼓励员工在工作中进行可控的冒险并容忍失败也是创新的必要条件，因为创新本身就是在走一条前人没有走过的路，风险和失败是很难避免的，在评价机制上也要针对员工的创新精神和尝试有所肯定，以便打消员工

的后顾之忧。

陈磊则指出了业务压力对创新的促进作用：“创新往往是被逼出来的”。能够感受到最大压力和挑战的环境往往有最多的创新：

“比如微信、手Q面对着海量的移动用户，遍布全国各地每一个角落和各种复杂的网络环境，为了让用户能够很好的使用微信和手Q，腾讯不得不在移动网络技术上有所创新，这也是腾讯云移动网络加速技术的来源。”

## 打造创新型组织的指导原则与实践

在谈到如何打造创新型组织时，汤鹏提出了三条指导原则：首先是方向引导，要鼓励员工大胆尝试新事物、新方法；其次是提供资源保障，要给予员工开展创新活动的时间；最后，在有一定的创新成果出现时要适时加以激励。

各家企业在打造创新型组织时的具体方法可谓八仙过海各显神通。微软致力于建立有效的员工创新平台，鼓励员工围绕新技术做一些动手实践的工作。谢恩伟介绍说：

“在微软上海科技园园区内，有一间叫做创新空间（TheSpace）的小屋，无论谁有好的创意都可以来这里寻找志同道合的人一起研究，如今它已成为许多微软员工进行内部创业的起点。与一些公司内部孵化项目不同，员工的这些想法不需要经过上级批准，创新空间里也没有管理层对你指手画脚，任何员工都可以提出自己的创意。”

英特尔也有类似的创新孵化机制。冯晓焰介绍说：

“我们有一些常年的创新孵化项目，鼓励员工提交创新性想法，组织专家针对这些想法进行评估，为好的想法提供资源，支持想法的提出者领导验证原型的开发。一旦想法的可行性得到证明，相应的产品部门就有可能将其列入实际产品计划。为鼓励员工积极提交创新性想法，即使该想法没获准进入原型开发阶段，员工也会得到一定的物质奖励。”

陈磊则以互联网企业的视角指出，促进创新需要降低试错成本，因为在一个试错成本低的环境里，大家才能打开思路、勇敢尝试新的方法、并快速找到正确的结论。他具体介绍腾讯的实践说：

“为了降低产品设计上试错的成本，我们建设实验系统，让ABTest变得更容易。为了降低研发的试错成本，我们提倡小步快跑，尽量缩短开发周期。要求一个系统的上线运营的周期不超过两个月，把问题尽快地暴露出来、尽快地解决。这些对创新都有很大的帮助。”

谢恩伟认为，企业中的创新并非员工的个人行为，而是一个相互联系、相互作用的系统，需要有效而系统化的管理机制能够推动和促进创新活动的前进。但同时这几位IT高管都感到，对创新的管理还是以人和环境为主，着重鼓励员工的创新意识、创造适宜创新的环境，创新本身往往还是没有固定模式，难以被系统地管理。

组织的领导者对于增强组织创新力需要承担更多的责任。冯晓焰这样描述领导者的职责：

“为了增强组织的创新力，作为一个组织的领导者，首先要明确将创新性开发作为组织的一个目标，鼓励员工的创新性思维和活动，从而创造出一个适合创新活动的大环境；其次，还要为提升员工的创新能力做一些切实的工作，这包括：提供相应的创新性思维培训，提高员工的技术能力从而为创新性开发打下基础，建立创新性项目以对员工的创新性活动提供资源上的支持和个人利益上的激励等；最后还需要帮助员工找到有效的渠道将创新转化为生产力。”

的确，一家企业的大环境很大程度上取决于企业领导者，不论创新平台还是创新机制，都需要领导者有意识地建设和培育。看来，在IT高管们的工作列表上，“打造创新型组织”应该占据一席之地了。

**作者简介：**熊节是ThoughtWorks成都分公司的负责人。在IT行业工作了十三年，拥有技术、商务、传媒、管理等多方面的经历，熊节现在致力于IT组织竞争力和企业文化的塑造，力求通过打造组织的学习力、创新力来提升企业竞争力，并创造更有利于IT工作者的企业环境。

## 小调查

看了这些IT高管关于创新型组织的观点，你对于企业创新力有何想法？你所在的企业是一家具备创新力的组织吗？请花五分钟时间填写一份简单的调查表单，帮助我们了解中国IT行业的创新力现状。

表单地址：<https://jinshuju.net/f/J1XiYq>

或扫描二维码：



---

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至[editors@cn.infoq.com](mailto:editors@cn.infoq.com)。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

原文链接：<http://www.infoq.com/cn/articles/chinese-IT-executives-talk-about-innovative-organization>

#### 相关内容

- [SAP的Jonathan Becher声称头脑风暴是失败的](#)
- [双重检查锁定与延迟初始化](#)
- [云计算技术的实践](#)
- [有道云笔记的特色云架构](#)
- [Node.js的企业实践之路](#)

# 推荐文章 | Article

## 从Groovy到Java 8

作者 [Dan Woods](#)，译者 夏雪

Groovy开发人员早已熟知Java 8中新引入的概念和新的语言结构了。在Java新版本即将推出的增强特性中，有很多是Groovy在几年前就已经提供了的。从用于函数式编程风格的新语法，到lambdas表达式、collection streaming和要把方法引用作为一等公民，Groovy开发人员在未来编写Java代码时具有先天性优势。本文将重点关注Groovy和Java 8的共同点，并阐述了Java 8如何解读Groovy中那些熟悉的概念。

我们先来讨论一下函数式编程风格，目前在Groovy中如何使用函数式编程，Java 8的概念如何提供更好的函数式编程风格。

闭包（Closures）也许是Groovy中最好的函数式编程实例了。从内部结构来看，Groovy中的closure只是一个函数式接口实现。函数式接口是指任意只需要实现一个方法的接口。默认情况下，Groovy的closure实现了一个名为“Callable”的函数式接口，实现了这个接口的“call”方法。

```
def closure = {
    "called"
}
assert closure instanceof java.util.concurrent.Callable
assert closure() == "called"
```

通过转换closure的类型，我们可以让Groovy实现其他函数式接口。

```
public interface Function {
def apply();
}

def closure = {
    "applied"
} as Function

assert closure instanceof Function
assert closure.apply() == "applied"
```

在Java 8中很好地引入了闭包和函数式编程的思想。在Java即将发布的版本中函

数式接口极为重要，因为在Java 8中针对新引入的Lambda函数式接口提供了隐含的实现。

我们可以把Lambda函数当成Groovy中的闭包那样去理解和使用。在Java 8中实现callable接口像Groovy中的闭包一样简单。

```
Callable callable = () -> "called";
assert callable.call() == "called";
```

你需要特别注意是，Java 8为单行的lambda函数提供了隐含的返回语句，后来Groovy也借鉴了这个概念。将来，Groovy也会为单个抽象方法提供隐含实现（类似于Java 8提供的那些实现）。这个特性使你不必完全派生出closures的具体子类对象就可以使用实例的属性和方法。

```
abstract class WebFlowScope {
    private static final Map scopeMap = [:]

    abstractdefgetAttribute(def name);

    publicdef put(key, val) {
        scopeMap[key] = val
        getAttribute(key)
    }

    protected Map getScope() {
        scopeMap
    }
}

WebFlowScope closure = { name ->
    "edited_${scope[name]}"
}

assert closure instanceofWebFlowScope
assert closure.put("attribute", "val") == "edited_val"
```

Java 8针对带有接口默认方法的函数式接口提出了一个类似的概念，即Java的新概念“接口默认方法”。他们希望借此概念在不违反接口实现规约（在Java之前的版本中建立的实现规约）的前提下改进核心的API。

当把Lambda函数强制转型为接口时，它们也可以使用接口的默认方法。也就是说在接口中可以内置健壮的API，使开发人员不必改变类型的种类或规约就可以使用这些API。

```

public interface WebFlowScope {
    static final Map scopeMap = new HashMap();

    Object getAttribute(Object key);

    default public Object put(Object key, Object val) {
        scopeMap.put(key, val);
        return getAttribute(key);
    }

    default Map getScope() {
        return scopeMap;
    }
}

static final WebFlowScope scope = (Object key) ->
"edited_" + scope.getScope().get(key);
assert scope.put("attribute", "val") == "val";

```

Java 8中的接口默认方法还可以帮我们实现像memoization和trampolining这样的Groovy特性。你可以很简单就实现memoization特性，只需要创建一个带有接口默认方法的函数式接口，并实现这个默认方法让它从缓存中确定估算结果或返回结果就可以了。

```

public interface MemoizedFunction<T, R> {
    static final Map cache = new HashMap();

    R calc(T t);

    public default R apply(T t) {
        if (!cache.containsKey(t)) {
            cache.put(t, calc(t));
        }
        return (R)cache.get(t);
    }
}

static final MemoizedFunction<Integer, Integer> fib
= (Integer n) -> {
    if (n == 0 || n == 1) return n;
    return fib.apply(n - 1)+fib.apply(n-2);
};
assert fib.apply(20) == 6765;

```

同样，我们还可以使用Java 8的接口默认方法开发Trampoline的实现。Trampoline是Groovy的一种递归策略，这个特性非常适用于深度递归，而不可能取代J

ava的调用栈。

```
interfaceTrampolineFunction<T, R> {
    R apply(T...obj);

    public default Object trampoline(T...objs) {
        Object result = apply(objs);
        if (!(result instanceof TrampolineFunction)) {
            return result;
        } else {
            return this;
        }
    }
}

// Wrap the call in a TrampolineFunction so that
we can avoid StackOverflowError
static TrampolineFunction<Integer, Object>
fibTrampoline = (Integer...objs) -> {
    Integer n = objs[0];
    Integer a = objs.length >= 2 ? objs[1] : 0;
    Integer b = objs.length >= 3 ? objs[2] : 1;

    if (n == 0) return a;
    else return fibTrampoline.trampoline(n-1, b, a+b);
};
```

除了closures的基本特性以及那些Memoization和Trampolining的高级特性，Groovy还为Collections API提供了一些有巨大实用价值的语言扩展。我们在使用Groovy时可以充分利用这些扩展点，比如用list 的“each”方法非常简捷地完成写操作。

```
def list = [1, 2, 3, 4]
list.each { item ->
    println item
}
```

Java 8针对集合的迭代引入了一种与Groovy类似的概念，提供了一个与“each”相似的“forEach”方法，可以用它取代list传统的迭代方式。

```
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(3);
```

```
list.add(4);
list.forEach( (Integer item) ->System.out.println(item); );
```

除了简化list的迭代，Groovy还为应用开发人员提供了各种快捷写法以简化各类list操作。比如“collect”方法，你用这个方法可以将list元素快速映射为新的类型（或新的值），然后把结果放入新的list里。

```
def list = [1, 2, 3, 4]
def newList = list.collect { n -> n * 5 }
assert newList == [5, 10, 15, 20]
```

在Groovy中“collect”的实现比较简单，你只需要把映射当作一个参数传递给“collect”方法。但是，Java 8的实现就稍微有点复杂了，开发人员可以使用Java 8的Stream API实现同样的映射和收集策略，实现时要调用“list”的“stream”组件的“map”方法，然后再调用“map”方法返回的“stream”的“collect”方法。开发人员可以这样连续使用Stream API完成list一连串的操作。

```
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(3);
list.add(4);
List<Integer> newList = list.stream().map((Integer n) -> n * 5).collect(
Collectors
.toList());
assert newList.get(0) == 5 &&newList.get(1) == 10
&&newList.get(2) == 15 &&newList.get(3) == 20;
```

Groovy还能让开发人员使用“findAll”方法简捷地筛选list。

```
def emails = ['danielpwoods@gmail.com', 'nemnesic@gmail.com',
'daniel.woods@objectpartners.com', 'nemnesic@nemnesic.com']
def gmaills = emails.findAll { it.endsWith('@gmail.com') }
assert gmaills = ['danielpwoods@gmail.com', 'nemnesic@gmail.com']
```

同样地，Java 8开发人员可以使用Stream API筛选list。

```
List<String> emails = new ArrayList<>();
emails.add("danielpwoods@gmail.com");
emails.add("nemnesic@gmail.com");
emails.add("daniel.woods@objectpartners.com");
```

```

emails.add("nemnesic@nemnesic.com");
List<String>gmails = emails.stream().filter(
(String email) ->email.endsWith("@gmail.com") ).collect(Collectors.toList());
assert gmails.get(0) == "danielpwoods@gmail.com"
&&gmails.get(1) == "nemnesic@gmail.com";

```

Groovy Collections API扩展还提供了一个“sort”方法，你使用这个方法可以简单地完成对list的排序。“sort”方法还可以接受闭包参数，你可以在闭包中实现所需的特定排序逻辑，闭包会被转为比较器后完成对list的排序。另外，如果只需要对list进行简单地逆序排序，可以调用“reverse”方法反转list的顺序。

```

def list = [2, 3, 4, 1]
assert list.sort() == [1, 2, 3, 4]
assert list.sort { a, b -> a-b <=> b } == [1, 4, 3, 2]
assert list.reverse() == [2, 3, 4, 1]

```

再来看Java 8的Stream API，我们可以使用“sorted”方法对list排序，然后用“toList”方法收集排序结果。“sorted”方法也支持自定义的比较器，它有一个可选的函数式参数（比如Lambda函数），你可以将自定义的比较器作为参数传给方法，就可以很容易地实现特定的排序逻辑和反转list条目的操作了。

```

List<Integer> list = new ArrayList<>();
list.add(2);
list.add(3);
list.add(4);
list.add(1);

list = list.stream().sorted().collect(Collectors.toList());
assert list.get(0) == 1 &&list.get(3) == 4;
list = list.stream().sorted((Integer a, Integer b) <br/>->Integer.valueOf(a-
b).compareTo(b)).collect(Collectors.toList());
assert list.get(0) == 1 &&list.get(1) == 4 &&list.<br/>.get(2) == 3 &&li-
st.get(3) == 2;
list = list.stream().sorted((Integer a, Integer b) <br/>->b.compareTo(a
)).collect
(Collectors.toList());
assert list.get(0) == 2 &&list.get(3) == 1;

```

如果你试图在一个闭包或Lambda函数内完成所有的处理而连续调用API（比如list streaming），那么很快就会使代码难以维护。换一个角度来看，如果你要委托相应工作单元特定的方法完成特定的处理，那么这种用法就是一个不错的选择了。

我们使用Groovy时，把方法引用传给函数也可以实现上面所说的目标。你只要使用“.&”操作符去引用方法，就可以把该方法强制转型为闭包传给另一个方法了。由于可以从外部源码引入过程代码，就从本质上提高了实现的灵活性。这样，开发人员就可以在逻辑上组织处理方法，完成更易维护、可持续演进的应用架构了。

```
def modifier(String item) {  
    "edited_${item}"  
}  
  
def list = ['item1', 'item2', 'item3']  
assert list.collect(this.&modifier) == ['edited_item1'  
, 'edited_item2', 'edited_item3']
```

Java 8也为开发人员提供了同样的灵活性，使开发人员可以使用“::”操作符获得方法的引用。

```
List<String> strings = new ArrayList<>();  
strings.add("item1");  
strings.add("item2");  
strings.add("item3");  
  
strings = strings.stream().map(Helpers::modifier).  
collect(Collectors.toList());  
assert "edited_item1".equals(strings.get(0));  
assert "edited_item2".equals(strings.get(1));  
assert "edited_item3".equals(strings.get(2));
```

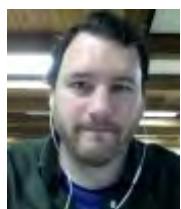
你可以把方法引用传给任意以函数式接口为形参的方法。那么，这个方法就会被转型为函数式接口，作为函数式接口执行。

```
public interface MyFunctionalInterface {  
    boolean apply();  
}  
void caller(MyFunctionalInterfacefunctionalInterface) {  
    assert functionalInterface.apply();  
}  
booleanmyTrueMethod() {  
    return true;  
}  
caller(Streaming::myTrueMethod);
```

在Java 8里，如果类库开发人员修改了接口规约，那么这些接口的使用者不必为了这些变更去修改那些使用了这个类库的接口。

这些概念和编程风格的无缝转化是从Groovy到Java 8的一次具有重要意义的过渡。Groovy为了提高内部灵活性和改进Java原有的API，使用了大量的JVM空间。随着这些改进在Java 8里生根发芽，意味着两种语言将有更多的相同点、而不同点会越来越少，事实上这正是本文要介绍的主要内容。当学习和使用这些新API、新特性和新概念时（从Java 8引入到Java生态系统中），熟练的Groovy开发人员只需要更短的学习曲线。

## 本文作者



**Daniel Woods**是Object Partners有限公司的一名高级顾问。他专门从事于Groovy和Grails应用架构的研究，对Java和其他基于JVM的语言一直抱有浓厚的兴趣。它是一名开源贡献者，并出席了Gr8 Conf和SpringOne 2GX的本年度年会。可以通过电子邮件（[danielwoods@gmail.com](mailto:danielwoods@gmail.com)）或Twitter（@danveloper）与Daniel取得联系。

查看英文原文：[From Groovy to Java 8](#)

---

感谢[侯伯薇](#)对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至[editors@cn.infoq.com](mailto:editors@cn.infoq.com)。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

原文链接：<http://www.infoq.com/cn/articles/groovy-to-java-8>

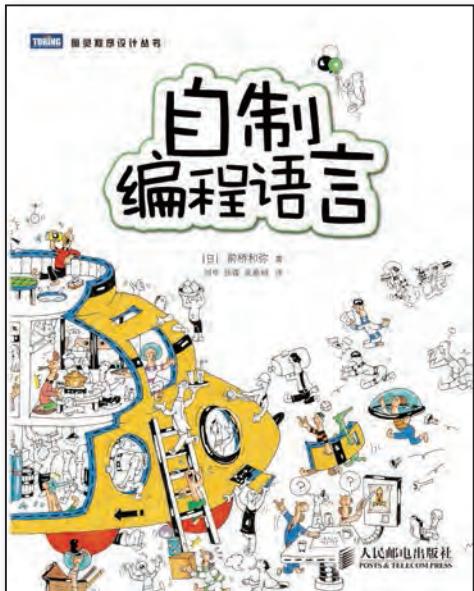
## 相关内容

- [Atmosphere 1.0：支持Java/JavaScript的异步通信框架](#)
- [Travis CI宣布支持Java，并计划推出Travis Pro](#)
- [Spring Framework 4.0相关计划公布---包括对于Java SE 8 和Groovy2的支持](#)
- [Groovy 2.0新特性](#)
- [Java 8开发者预览版发布](#)

TURING



# 图灵教育推荐



书名：自制编程语言

作者：前桥和弥

译者：刘卓 徐谦 吴雅明

- 只需编程基础
- 从零开始自制编程语言
- 支持面向对象、异常处理等高级机制
- 自己动手，深入理解编程语言

本书手把手地教读者用 C 语言制作两种编程语言：crowbar 与 Diksam。crowbar 是运行分析树的无类型语言，Diksam 是运行字节码的静态类型语言。这两种语言都具备四则运算、变量、条件分支、循环、函数定义、垃圾回收等功能，最终版则可以支持面向对象、异常处理等高级机制。所有源代码都提供下载，读者可以一边对照书中的说明一边调试源代码。这个过程对理解程序的运行机制十分有帮助。

详情请点击：<http://www.ituring.com.cn/book/1159>



- 国内第一本 jQuery 权威教程，一版再版，累计重印 14 次，不可错过的实战类经典技术著作！

本书是 jQuery 经典技术教程的最新升级版，涵盖 jQuery 1.10.x 和 jQuery 2.0.x。本书前 6 章以通俗易懂的方式讲解了 jQuery 的核心组件，包括 jQuery 的选择符、事件、动画、DOM 操作、Ajax 支持等。第 7 章和第 8 章介绍了 jQuery UI、jQuery Mobile 及利用 jQuery 强大的扩展能力开发自定义插件。随后的几章更加深入地探讨了 jQuery 的各种特性及一些高级技术。附录 A 特别讲解了 JavaScript 中闭包的概念，以及如何在 jQuery 中有效地使用闭包。附录 B 讲解了使用 QUnit 测试 JavaScript 代码的必备知识。附录 C 给出了 jQuery API 的快速参考。

详情请点击：<http://www.ituring.com.cn/book/1169>



TURING

北京图灵文化发展有限公司

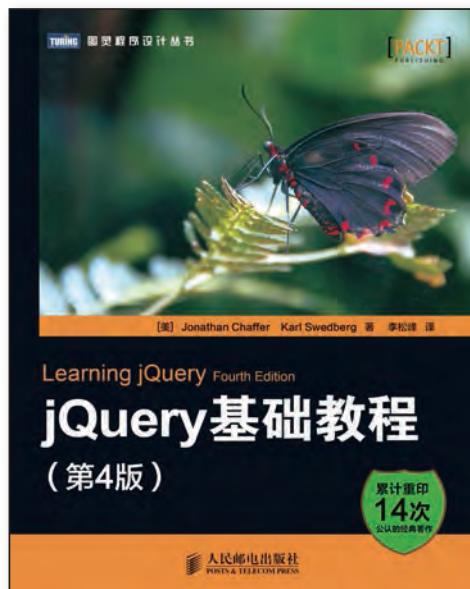
联系我们：

官方微博：@图灵教育

联系人：@图灵郭志敏

邮箱：guozm@turingbook.com

Q Q：52398720

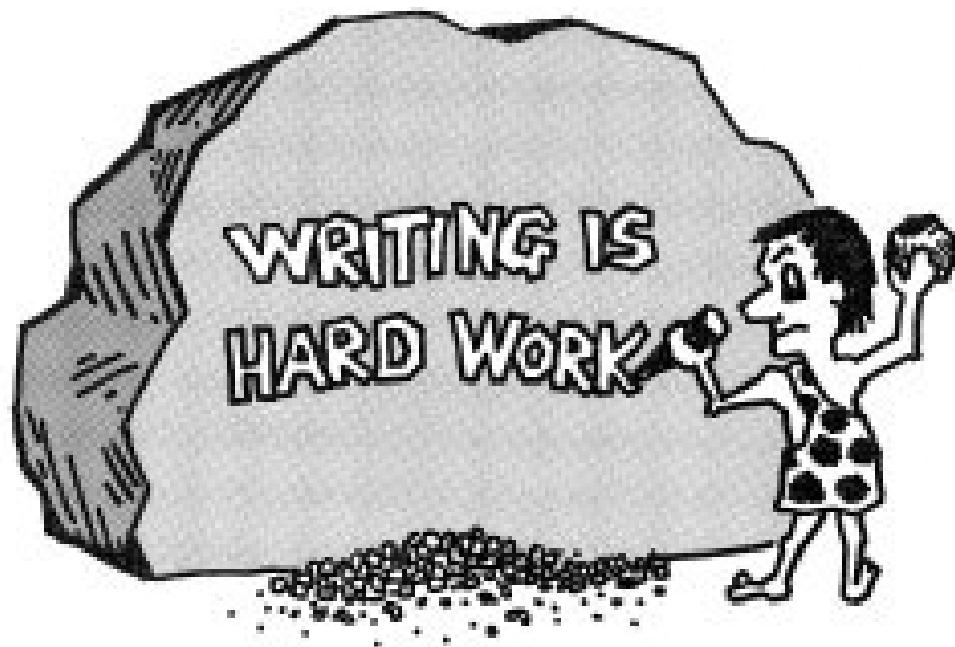


书名：jQuery 基础教程（第 4 版）  
作者：Jonathan Chaffer Karl Swedberg  
译者：李松峰  
书号：978-7-115-33055-0

## 推荐语

### 专栏主编有话说

技术改变未来，技术人的特点是闷骚，很抱歉我用这样的话来形容技术人，但是实际的场景可能也是这样的。但是技术人的闷骚 代表着他们的满足点真到的单纯，写出了一个新的类库、算法得到了提升亦或是新的架构在系统中的表现很好，这些都让他们不亦乐乎。同时，技术人乐于分享，乐于把自己的思考奉献出去，这和InfoQ的Slogan：“促进软件开发领域只是与创新的传播”如出一辙，回顾InfoQ的内容历程，我们发现借助这些技术人的力量，InfoQ也一直努力为大家带来精彩纷呈的内容，也因此我们也希望有越来越多的朋友能够加入到这个行列中。在这几期的专栏中我们将会介绍目前的专栏和内容规划，如果在这些领域有话想说，不要犹豫，给我们发邮件（[editors@cn.infoq.com](mailto:editors@cn.infoq.com)）来，参与到这个知识传播的过程中吧！



# 特别专栏 | Column

## 测试专栏主编：侯伯薇

作者 [侯伯薇](#)

**InfoQ：**首先，做一个简单的自我介绍吧，着重介绍一下您在您所负责专栏领域的一些经验和工作背景。

侯伯薇：大家好，我是InfoQ中文站测试专栏的主编侯伯薇，加入到InfoQ中文站的编辑团队快有四年时间了，在此期间参与了各种各样的活动，学习到了很多，所以首先要感谢InfoQ中文站这个优秀的平台，也正因为如此，我在平日里也尽量抽出时间来做些力所能及的工作，回馈InfoQ中文站，也为广大读者朋友们提供服务。尽管目前我负责的是测试专栏，但自己却并非从事专门的测试工作，在我日常的工作中，形式比较敏捷，设计、开发、测试都是由很少甚至是一个人完成的。但在之前的工作中，曾经在团队中负责检查团队中其他成员编写的程序，以确保能够更好地满足客户所提出的需求，所以一直对于测试领域或者说如何保证程序能够达到最终用户的要求非常感兴趣。可以说，负责这个专栏，也是和诸位测试方面的专家学习的一个过程。目前我在一家保险公司做程序员的工作，还一直冲在生产第一线，每天都会和代码打交道，正是因为是在一家业务主导的企业中，所以对如何保证编写出来的程序的质量，也就是如何真正能够帮助到业务用户非常注意，而我理解，测试最终的目的，不仅仅是要和各种开发文档一致，更应该是如何最终帮助到使用系统的人。因此，测试和设计、开发都是密不可分的，而即便是专门的测试人员，我也认为需要了解业务，知晓设计和开发的基本原理，那样才能够更好地发挥出更大的作用。

**InfoQ：**介绍一下您所负责的专栏。您如何定义这个领域？这个领域当前有哪些值得关注的话题和趋势？

侯伯薇：测试领域近些年来在广度和深度上都有了很大的发展。首先，测试的类型有了非常大的扩展，不仅仅有单元测试、集成测试、用户验收测试，还有性能测试、负载测试，以及与架构相关的安全性测试、可用性测试、易用性测试等等。之前参加的一个培训过程中，大家列举出来的各种各样的测试，种类甚至超过了30种，所以看来，测试的任务并非像我们想象的那么简单。另外，随着软件工程领域的发展，测试也不再仅仅是传统软件工程中下游的一项

活动，在敏捷团队中，测试已经深入到整个开发过程之中，因此出现了自动化的单元测试，测试驱动开发等做法。测试人员和开发人员角色之间的边界也越来越模糊。这样，不仅仅是专职的测试人员要了解测试的原理和做法，团队中的其他角色——比方说需求分析人员、开发人员、项目Leader等——都需要了解，那样才能够更好地完成团队的任务。随着不同思想和做法的出现，测试人员也会产生一些困惑。很多专门的测试人员都会去考虑自己在团队中的位置，是越来越重要呢，还是越来越可有可无。而真正优秀的测试人才，更是可遇而不可求。

**InfoQ：**在这个领域，您有什么相关的资源推荐？比如博客，微博，文档，相关文章系列等，包括线上线下活动等。

侯伯薇：在我读过的书中，有两本比较值得推荐，一本是《敏捷软件测试：测试人员与敏捷团队的实践指南》，另一本是《测试驱动开发》，但这两本书都是从敏捷的角度出发的。

国内比较专业的大会是ChinaTest中国软件测试大会，其中的内容不管从深度还是广度上，都非常不错。

**InfoQ：**您的专栏今年经手过的内容，您最推荐哪些？最好列出三到五篇，并对这几篇文章进行简单的点评。

侯伯薇：我想推荐的文章是两个系列的文章：首先是殷坤的《敏捷自动化测试》系列，他在实际的工作中发现，对于专门的测试人员来说，想要让测试真正自动化，特别是web页面测试自动化，是件不容易完成的工作。因此，他在系列文章中详细叙述了他所带领的团队的做法，非常注重实践，对于想要实施自动化测试的团队来说有很大的借鉴意义。另一个系列是邵晓梅《黑天鹅》，这个系列文章更专注于理念而不是实际的做法，这对于测试人员来说同样重要，这就像是武功中的内功一样，练好了之后，即便是最简单的招式，施展开来也是威力无穷。

**InfoQ：**如果一位新的作者来投稿、沟通选题，而他可能在领域方面很熟悉，却在写作方面经验不多。您一般采取怎样的方式跟作者沟通，培养他？

侯伯薇：首先我会请作者列出文章的大纲，然后和他一起商量并确定，文章对于读者来说是否有实际意义、是否易于理解和操作、是否可以很容易地移植到

他们的场景之中，这样就可以确定文章的具体方向，防止浪费作者的时间和精力。对于内容方面，针对InfoQ中文站的风格，我会建议作者不要过于口语化，博客式的文章可能需要斟酌一下才能够适用于InfoQ中文站，因为读者们更希望看到的是干货，一些口语化的文字会干扰大家的注意力。另外，对于配图、文字等细节，我也会在作者定稿之后，详细商量，争取把内容以更好地形式呈献给读者朋友。

**InfoQ：**专栏目前希望招募哪些方面，哪些类型的稿件？

侯伯薇：凡是与测试相关的内容我都表示欢迎，最好要有一定深度，并且有实践意义，那样才能够让读者们有更大的收获。

原文链接：<http://www.infoq.com/cn/news/2013/09/architec-ceshi-column>

#### 相关内容

- [博文共赏：SAP自动化准备测试数据：一个基于AutoIt VBS XML的实现思路](#)
- [iOS开发中的单元测试（三）——URLManager中的测试用例解析](#)
- [天猫王德山谈性能测试](#)
- [软件测试中的黑天鹅（二）：黑天鹅发生的前后](#)
- [自动化测试基础设施（一）——为功能测试构建通用mock server系统](#)

# 特别专栏 | Column

## 软件测试中的黑天鹅（三）——测试的平均斯坦与极端斯坦

### 1. 突破性与非突破性

作者 [邹晓梅](#)

《黑天鹅》里谈到了突破性与非突破性的概念。

这世界上有些职业的收入是不具突破性的，比如面包师、咨询师、按摩师、牙医等等，其收入受到既定时间内所服务的客人的数量的限制，这种工作在很大程度上是可以预测的，面包师必须为每一位客户烤出新面包，不论他出售的面包单价有多贵，其收入总是受到限制的。

而另外一些职业如录音师、电影演员、作家、投机师等等，他们只需花费单次的投入而不必过多劳动就有可能使得收入后面增加几个零，《哈利·波特》的作者不必每次有读者想读这本书的时候就得为其重写一遍，这种职业属于收入具有突破性的职业。

仔细想想，就会发现，突破性的工作往往是那些更擅长运用和组织创造性思维和创新技术的工作，而不那么具有突破性的部分往往可以被剥离出去，交给那些按照计件取酬、或按时取酬的人去做。

测试是否是具有突破性的工作？这个不能一概而论，测试的范畴太大了，包含了太多的内容，我们在后面的部分会持续讨论这个话题。

### 2. 平均斯坦与极端斯坦

这种突破性与非突破性的差异与另外两个概念有关：平均斯坦与极端斯坦。

人的身高、体重等物理属性是不具有突破性的，体现的是一种叫做“平均斯坦（Mediocristan）”的现象。拿《黑天鹅》里的例子为例：假如从普通人群里随机挑选1000人并排站着，然后把你想到的最重的人加入样本，假设他的体重是平均体重的5倍，在总体重中也不过占了0.5%而已，假设你挑选了10000人，他的体重所占比例几乎可以忽略不计。“在理想的平均斯坦，特定事物的单独影响很小，只有群体影响才大。”实际上，对于平均斯坦而言，“当样本量足够大时，任何个例都不会对整体产生重大影响。”

而人的收入是具有突破性的，体现的是一种叫做“极端斯坦（Extremistan）”的

现象。假如考虑的是这1000个人的收入，如果把比尔·盖茨放入样本，那么他的资产占总资产的比例是多少？可能所有其他1000人资产总和也不过是比尔·盖茨的资产的零头。在极端斯坦，个体对整体产生的作用是具有突破性的，可以产生不成比例的影响。值得注意的是，“极端斯坦能够制造黑天鹅现象”。

对软件测试而言，由于黑天鹅的影响巨大，我们需要时刻注意区分，哪些是极端斯坦的问题，哪些是平均斯坦的问题，并利用我们收集到的信息，帮助判断如何避免黑天鹅现象的发生、或者是促成黑天鹅现象的发生。

### 3. 测试里的平均斯坦

关键字：平均斯坦，大数定律，N的平方根定律

假如你是一名测试经理，每天有无数的数据涌入你的大脑，你需要学会如何判断和使用这些数据，来帮助你更好的管理你的测试项目。

假如你需要为一个新的测试项目制定一个测试计划，你需要先做测试估计，这个项目需要投入多少测试人月？大概能执行多少个手工测试用例？测试计划、测试设计、测试执行的时间大概各占多少？等等。有一种做法是参考这个产品的各历史版本的产品规模和测试投入情况，从而估算出当前版本的测试投入。

当你这么做的时候，你要知道：

- 你面对的是一个平均斯坦的问题；
- 也许某个历史版本的数据显得异常突出（偏大或偏小），但你不必特别在意，甚至估算的时候会忽略掉这些畸点数据；
- 你估算的准确程度受所选择的历史样本数据量的影响，如果你只有一个历史版本可供参考，你估算的偏差可能会很大；
- 你关心的是这些数据总体上对你要做的决策的参考价值，而不是在追求一个绝对准确的估计值，因为你知道，不论你多么努力，都不可能制定一个绝对准确的测试计划。

类似的平均斯坦的例子有，如果你想了解你的测试团队平均生产力，比如“日执行用例数”、“日设计用例数”、“日提交问题单数”等等，你不妨观察一段时间，记录数据，算个平均值。当然，你这么做是因为你此时更关注集体事件、平均事件、常规事件和已预测到的事件对你的影响，并且容许有一定的误差存在。

Gerald M. Weinberg在《An Introduction to General Systems Thinking》里谈到的“大数定律”与平均斯坦有关。19世纪的物理学家希望研究瓶子里空气分子的运动，他们没有办法搞明白每一个分子的运动情况，因为分子的数量实在是太大了，而不得不假设他们所感兴趣的观察特性是大量分子的一些平均特性，

而不是其中任何个体的特性。由于分子的数量特别大，他们的研究也满足所谓的“大数定律”：观测样本的数目越多，观测值越接近于预测的平均值。而SchrÖdinger更是给出了预测准确程度的“N的平方根定律”：

如果假定由n个分子组成的某种气体在一定压力和温度下具有一定的密度，如果在某个特殊时刻检验它，结果表明上面的表述是不准确的，其偏离的数量级大约为n的平方根。如果n为100，则偏离值约为10，相对误差为10%；如果n为1000000，则偏离值约为1000，相对误差为0.1%。

所以，当我们需要关注整体的特性，而不是某个个体的特性时，我们所基于的这个样本数量必须足够大，否则不足以建立据此判断的信心。记得有一个团队开始尝试敏捷开发模式时，一个迭代下来，只发现了少量的bug，这样的情况持续了数个迭代。这与他们之前使用瀑布模式开发时总是有大量的bug冒出来差别较大。无论是开发人员还是测试人员，无论是项目经理还是测试经理，都不确定这个现象到底是好还是不好：是产品质量确实提高了所以bug更少了？是敏捷确实使得大量bug在过程中被提前发现并预防了？还是紧凑的迭代过程使得测试人员没有足够的精力和时间做更深入的测试？确实，如果仅通过有限的bug数、有限的测试时间内执行的有限的测试用例数，来评估整个被测系统的质量，确实难以建立起评估的信心，因为所参考数据的基数n不够大。

这让我想起那个“测试何时可以结束”的问题，答案可能是“No ending”，只要还有时间，只要条件还允许，我们应该继续测试，增加n的测试样本量，扩大n的覆盖范围，多一些对系统未知领域的了解，那么，测试为利益相关者提供的质量相关的信息就更为准确。

这也提醒我们，如果你希望通过快速而少量的测试活动来获得对整个系统质量的初步认识，那么尽可能的增大n的覆盖范围，让其更具有随机性和普遍意义。比如要做一个1天的针对被测系统的验收测试，那么验收测试用例的选取要尽可能覆盖的特性更多一些、验收测试用例的数量尽可能多一些。同样地，应用“大数定律”的原理，不论你选取的验收测试用例多么多，1天里能够执行的验收用例的个数是受限的，相对于要很好地了解被测系统的质量应该执行的用例数而言，这个n还是小的可怜，这是个平均斯坦的问题，在这里，我们更关注的是整个系统是否可以被“验收”通过，而这1天的验收测试的结果和被测系统的真实质量关系并不大。

在软件测试中，当你更关注数据的整体趋势，而不关注单个数据的影响时，你面临的是平均斯坦的问题，此时单个数据是不具有突破性的影响的。那么，至少有两点需要知晓：

- 数据样本量的多少会影响你做判断的准确程度；
- 不论样本量多么大，最终的结论都不可能成为绝对的基准，而仅能作为相对的参考。

## 4. 测试里的极端斯坦

我一直认为，不论是在软件测试领域还是在其他某个学科领域，数字仅能衡量一部分信息，甚至还不一定是最重要的部分，定量的方法只能作为人们决策中的辅助手段，不应成为主导甚或是唯一的手段。

如果你是一名测试经理，你可能对很多数字很感兴趣，比如：

- 每日发现的缺陷数
- 每日解决的缺陷数
- 每个版本挂起的缺陷数
- 每个迭代或版本发现的致命缺陷数
- 每个迭代或版本发现的严重缺陷数
- 每个迭代或版本发现的一般缺陷数
- 每个迭代或版本发现的提示缺陷数
- 每个迭代或版本发现的资料问题的缺陷数
- 每个迭代或版本非用例发现的缺陷数
- 每个版本探索性测试发现的缺陷数
- 每个版本执行的探索性测试的session数
- 每次探索性测试session的时长
- 每次探索性测试session花在测试准备上的时间
- 每次探索性测试session花在测试设计和执行上的时间
- 每次探索性测试session花在测试问题调查和测试记录上的时间
- 每日执行用例数
- 每日测试用例pass数
- 每日测试用例fail数
- 每日测试用例block数
- 每日设计用例数
- 每个版本或迭代测试期间新增的测试用例数
- 每个版本或迭代测试期间修改的测试用例数
- 每个版本或迭代测试期间删除的测试用例数
- 每个版本或迭代由于测试本身的原因导致的以非问题关闭的缺陷数
- 每个版本测试与开发的成本投入比
- 每个版本或迭代千行代码缺陷数
- 每个版本或迭代千行代码用例数
- 每个测试人员发现的bug数
- 每日处于open状态的bug数

- 每日处于new状态的bug数
- 每日处于retest状态的bug数
- 每日自动化用例的个数
- 每天每个测试人员用于测试执行的时长
- 每个测试用例的设计时长
- 每个测试用例的执行时长
- 每个版本投入的测试人员数
- 每个迭代的测试时长
- 每天测试执行用于搭建测试环境的时长
- 每个版本的缺陷数
- 每个版本用户报告的缺陷数
- 提交一个缺陷报告的时长
- 回归一个缺陷的时长
- 每个阶段的DDP
- 每个特性的测试设计与测试执行的时长比
- 每个特性的用例发现缺陷的比率
- 每个特性的代码覆盖率
- 每个版本的代码覆盖率
- 每个特性的代码更新率
- 每个版本的代码更新率
- 每个特性的代码圈复杂度
- 每个版本的代码圈复杂度
- 每个版本的测试人员更新率
- 每个版本的开发人员更新率
- 开发阶段版本新增需求比率
- 开发阶段版本修改需求比率
- 开发阶段版本删除需求比率
- 每个迭代的延迟交付时长
- 每个版本的迭代按时交付率
- .....

是的，这个关于测试中各种数字的表单可以很长。当这些数字摆到你的面前，该如何分析和对待呢？

例如你一直在观察这个指标“每日发现缺陷数”，随着测试天数的增加，这个值呈现平稳增长然后又平稳下降的态势，临近版本对外交付前一周内的每日发现缺陷数均小于等于1，整个版本的缺陷总数为99，除了少数几个严重缺陷早已解决外，其他缺陷均为一般或提示性问题且已得到修复。也许你的结论是：版本可以如期对外交付了。可是在版本发布前一天，突然发现了一个致命问题，这个问题只占所有缺陷的1%，但是它产生的影响不可小觑，你也许要因此推迟版本发布日期、重新制定测试计划、甚至影响到需求的调整。。。

这一个致命问题无疑是个黑天鹅，它对你来讲是单个事件、意外事件、未知事件、和未预测到的事件，这个个体事件以不成比例的方式影响着整体的决策，你面临的是极端斯坦的问题。极端斯坦问题意味着：很难从过去的信息中做出预测、具有突破性、需要花很长时间了解情况、整体取决于少数极端事件、易于制造出黑天鹅事件。

这让我联想到一个关于火鸡的故事。想象一下火鸡每天都有主人喂食，这样持续了很多天，比如说1000天，于是火鸡得到了一个结论，生活就是这样安全而幸福的，随着喂食次数的增加，它的这个信念也增强了。可是到了感恩节前的一天，一件意想不到的事将会发生，从此改变了火鸡的命运，也改变了它坚守很长时间的信念。

测试中类似的极端斯坦现象还有很多。比如你观察某个测试人员“每日执行用例数”来判断每天测试的实际投入情况和测试的生产力，那些异常大的数据不一定就代表着这一天测试工作很饱满、成果很丰硕；那些每日执行用例数为0或1的日子，也不一定就代表着测试人员投入很少、没有什么收获；有可能测试人员花了整整一天的时间执行了1个用例，连午饭都来不及吃，但他发现了软件的一个很大的漏洞、发现了搭建测试环境的秘诀、解决了一个系统性能瓶颈问题。缺少这一天，这个测试人员的成绩平平；有了这一天的这一个用例，这个测试人员成了人们心中的英雄！

您可能已经看出来了，针对同一个指标，如果你希望通过它的变化趋势来获得一个平均值，估计团队的执行能力，那么这是个平均斯坦的问题；如果你希望通过它来做质量相关的决策，比如版本质量是否稳定、测试是否可以结束、测试效率如何、测试效果好不好等，那么这是个极端斯坦的问题。对于极端斯坦问题，仅看表面的数字可能让你觉得很困惑，可能让你做出错误的判断，你得考虑每一个数字背后隐藏的故事，看是否有可能存在某个个体会以不成比例的方式影响整体的决策和判断。当然，极端斯坦有可能制造黑天鹅，但不一定总会制造黑天鹅，对于决策者而言，你需要做好黑天鹅发生的准备。

## 5. 结论

在一个软件测试项目里，当你需要做一个决策或解决一个问题时，需要：

- 判断这个问题是平均斯坦问题还是极端斯坦问题：即是否存在某个个体会以不成比例的方式影响整体；
- 若是平均斯坦问题，关注集体事件、平均事件、常规事件对你的影响，所选取的样本量尽可能大，且容许存在一定的误差；
- 若是极端斯坦问题，关注个体事件、极端事件、未知事件对你的影响，不是每个

个体都同等重要，要找寻那些有重大影响的个体事件重点关注，预测和准备迎接黑天鹅事件的发生。

## 参考文献

Nassim Nicholas Taleb, The Black Swan: Second Edition: The Impact of the Highly Improbable, 2008

Gerald M. Weinberg, An Introduction to General Systems Thinking, Silver Anniversary Edition

## 作者简介

邵晓梅：软件测试独立顾问

网址：[www.sharetesting.com](http://www.sharetesting.com)

博客：[www.taixiaomei.com](http://www.taixiaomei.com)

原文链接：<http://www.infoq.com/cn/articles/average-stein-and-extreme-stein-in-test>

### 相关内容

- [软件测试宣言](#)
- [测试工具如何创新](#)
- [Web软件测试中数据输入的检查清单](#)
- [新一代服务器性能测试工具Gatling](#)
- [虚拟座谈会：专家眼中的QA、敏捷测试、探索式测试及测试的开放性](#)

## 特别专栏 | Column

# 敏捷自动化测试（4）——围绕自动化测试开展持续集成

作者 殷坤

本文不会介绍持续集成的概念、理论以及相关工具的用法，而是基于实际的项目案例，分享如何利用自动化测试保障持续集成的有效性，同时也借助持续集成提升自动化测试用例的价值。

在本系列的前几篇文章中，首先分析了[测试不够敏捷的原因](#)，然后从“[降低测试脚本维护成本](#)”和“[优化断言机制](#)”两方面分享了Web自动化测试的改善实践。

前面几篇文章其实都在讲“如何成功实施Web自动化测试”，本文将要涉及一个很敏感，但不应该回避的话题“自动化测试的意义到底有多大？”。

做过自动化测试的同仁都应该对“自动化测试用例能发现多少缺陷”这话题不陌生，业界也有很多支持“自动化测试用例主要不是用来发现缺陷，而是提高回归测试效率”这个观点的文章。即便这样，“发现缺陷”也像是自动化测试人员“肉中的一根刺”，很多时候大家不愿意去触碰它（不碰就不痛）。这样时间久了，很可能会让自动化测试人员丧失发现缺陷的斗志。笔者在这里就尝试拨动一下这根“刺”，希望可以再次引发大家的思考。

“自动化测试主要不是用来发现缺陷”这一观点的主要依据是“自动化测试是严格按照已有用例进行的自动回归测试”。说白了就是“它每次做的事情相同的”，所以不能像手工测试那样依靠人的主观能动性来发现新的缺陷。

“它每次做的事情相同的”，这一点我们改变不了。

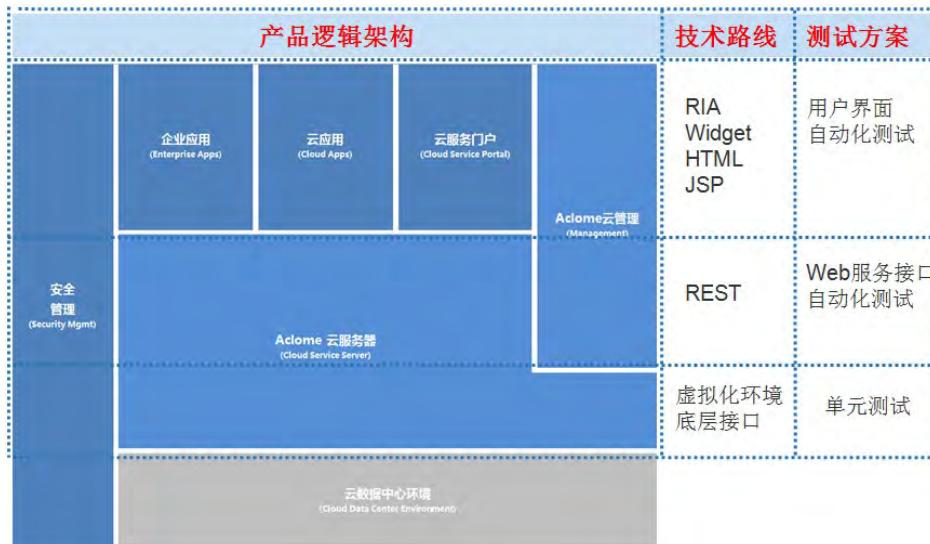
但我们能做的是在不同的场景中做“相同的事情”！从而增加发现缺陷的几率，降低软件释放后风险。

下面就通过实际案例分享笔者在提升自动化测试价值方面的经验：“在持续集成中对每日构建的版本进行回归测试”，并对一些里程碑版本进行“在各种环境组合下（应用服务器、数据库、浏览器、操作系统）的兼容性测试”和“7\*24小时的稳定性测试”。

## 对每日构建版本的回归测试

本案例中的产品是面向云计算领域的通用云环境管理软件，在云数据中心构建及运维过程中提供全方位、多层次的管理能力，基于云环境实现应用的快速部署及其资源的弹性供应，通过简化管理极大地降低成本、提高效益。

产品的逻辑架构、主要技术路线及对应的自动化测试方案如下图所示：



该产品大致可以分为三层：

- 最底层负责与各类虚拟化资源或物理资源的交互，比如，创建虚拟机、获取CPU利用率；
- 中间层负责对底层返回的数据进行持久化及业务处理，并向外提供大量REST接口；
- 最顶层通过多个应用门户向不同类型的用户提供体验一致的交互界面，它本身没有太多业务逻辑和持久化操作，主要靠调用中间层的REST接口来实现。

持续集成方案选择的CI工具是[Jenkins](#)，具体步骤如下图：



对每个步骤的要点简述如下：

## 1. 更新编译代码

项目采用分布式开发，需要从不同的配置管理库中更新各个模块的功能代码和自动化测试用例。

## 2. 代码质量检查

项目要求开发人员使用Findbugs和PMD对自己的代码进行质量检查并修正相关问题。同样也在持续集成过程中通过Sonar（集成Findbugs和PMD）使用相同的策略进行代码检查，以确保该要求的执行效果。

## 3. 更新数据库

在更新功能代码和测试脚本之后，自动化测试还是可能出现大面积无法通过的情况。其中一个很常见的原因就是“当天开发库中的数据库结构发生了变化”，而持续集成所使用的数据库并没有更新。

所以我们执行自动化测试用例之前增加一个“检查开发库和持续集成库的表结构是否一致，如果不一致就做增量同步”的任务。

## 4. 执行单元测试

通过单元测试确保系统与虚拟化资源或物理资源的交互是没问题的。如果这个环节出现严重问题，后面的持续集成任务就没必要执行了。

## 5. 部署后台服务

单元测试通过之后，把后台服务（即，上面说的中间层）发布到目标服务器并启动服务。

## 6. 执行REST接口测试

基于上面发布的后台服务，对其提供的大量REST接口进行回归测试。如果这个环节出现严重问题，就没有必要再部署前台应用并进行后续测试了。

## 7. 部署前台应用

REST接口测试通过之后，才把各个前台门户应用发布到目标服务器并启动服务。

## 8. 执行Web UI测试

基于上面发布的前台应用，执行各自Web UI层面的自动化测试用例（采用本系列前面几篇文章中介绍的方案）。

## 9. 发布构建结果

通报每日构建结果（如下图所示），测试用例按照模块进行分类，对不通过的用例进行分析然后提交相关缺陷。

模块	用例	失败	异常分析	备注
mc_alert	13	0		告警查询及确认
mc_dashboard	28	0		MC仪表盘
mc_database	108	0		MySQL、Oracle、Sql server、DB2数据库监控
mc_globalsearch	47	0		全局搜索
mc_infrastructure	33	0		施耐德空调监控
mc_middleware	76	0		Tomcat、Weblogic、Websphere、JBoss监控
mc_monitor	349	0		虚拟化、非虚拟化资源监控数据
mc_monitor_chart	14	0		统计图表相关
mc_network_topo	4	0		网络设备探测
mc_networkdevice	32	0		网络设备相关(路由器、交换机、防火墙、F5)
mc_physicalHost	51	0		物理主机监控
mc_policy	17	2	Adem00011180	策略配置相关
mc_rhev	50	0		Rhev虚拟化环境相关操作
mc_service	36	0		ftp、http、telnet常规服务监控
mc_storagedevice	22	0		存储设备相关
mc_swift	8	0		Swift监控
mc_tasklog	3	0		系统操作日志相关
mc_topconfig	4	0		网络拓扑配置相关操作
mc_vmware	60	0		VMware虚拟化环境相关操作
mc_xenserver	65	0		Xenserver虚拟化环境相关操作
vdc_bizTemplate_Management	25	0		虚拟机模板业务
vdc_cloudApp	273	33	Adem00010532	云应用
vdc_globalQuery	24	0		全局搜索
vdc_ip_management	117	0		IP池管理
vdc_monitor_chart	28	0		管理员统计分析
vdc_sla	5	0		SLA
vdc_virtualNetwork	16	0		租户创建虚拟网络
<b>总计</b>	<b>1508</b>	<b>35</b>		

上述持续集成案例的关键是“自动化测试”，而无论从技术还是从管理的角度，其难点也是“自动化测试”。

从技术角度来说，在上面三类自动化测试中最困难的是UI层面的自动化测试。笔者在本系列前面几篇文章中主要分享的就是这方面的改善实践。

从管理角度来说，要对自动化测试相关数据（比如，用例通过率、用例增长率、代码覆盖率等）实施公开、准确的度量，如下图所示。



度量的最终目标不是为了考核，而是为了改善。

从度量结果中团队能看出问题，才能有的放矢的改善；

从度量结果中团队能看出进步，才能得到激励和信心。

众所周知，没有自动化测试的持续集成是“伪”持续集成。但笔者了解的很多做持续集成的项目都存在着“自动化测试用例很不充分”的问题。

上述持续集成案例中是如何做到充分自动化测试的呢？笔者认为有如下几个要点，希望可以引发大家的思考：

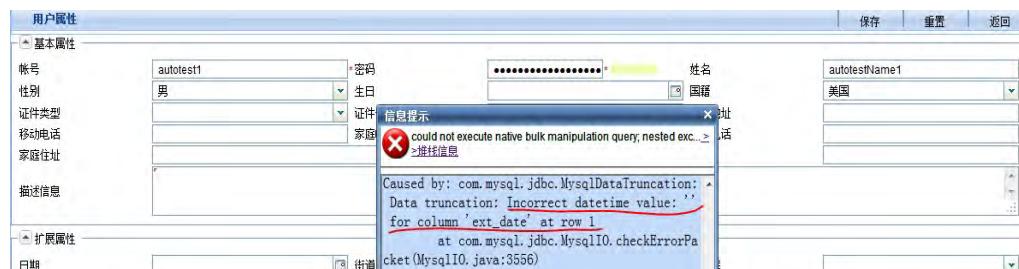
1. 测试团队负责产品的持续集成；
2. 先推行自动化测试后实施持续集成；
3. 把自动化测试用例通过率和增长率作为迭代的关键度量指标；

## 兼容性测试

现在软件需要兼容的环境越来越多，包括操作系统、应用服务器、数据库、浏览器，像上面案例中的产品还要兼容主流虚拟化服务（比如，VMware、XenServer、KVM）的各种版本。如果全靠人工测试来保证兼容性，是不太现实的事情。

兼容性测试有个特点，就是如果不兼容，往往在一些主要流程上就能明确的体现出来，一般不用深入到每个业务细节，也不用靠人来主观分析。因此也非常适合通过自动化测试来完成。

下面分享一个笔者在撰文的前一天刚刚遇到的实际案例（如下图所示）。



这个一个最基本的用户维护界面，里面包括必填信息和可选信息（比如，生日）。在一次自动化测试中，Web UI层面的“用户信息修改”用例出现了如下异常：

Caused by: com.mysql.jdbc.MySQLDataTruncation: Data truncation: Incorrect datetime value: '' for column 'ext\_date' at row 1

这个功能后台业务接口是有单元测试用例对应的，并且也做了一些边界值测试（比如，“生日”属性为空），但遗漏了另外一个边界条件（空字符串），而恰恰如果界面不选择“生日”，传到后台的就是空字符串。Web UI层面最基本的测试用例恰恰弥补了这个缺憾。

有些人可能会质疑：如果界面不选择“生日”，传到后台的也是空（null），那么这个Web UI测试用例不也一样发现不了这个Bug嘛。

确实如此，但换个角度想一下，如果这样的话，这个“Bug”即使遗漏出去，也永远会被用户发现。

这正体现了Web UI层面自动化测试的意义——比其它层面的自动化测试更代表用户！

写到这里，可能会有敏锐且尖锐的读者继续提出质疑：“这么表面的缺陷在手工测试阶段一定会发现的，怎么会轮到通过Web UI自动化回归测试来发现呢？”

我们来回顾一下上面的异常信息“Caused by: com.mysql.jdbc.MySQLDataTruncation...”。是的，这个缺陷只有在使用MySQL数据库时才出现，而日常开发和手工测试使用的都是Oracle数据库。

我们把在各种组合环境下的兼容性测试交给自动化用例来完成。因此，如果有兼容性方面的Bug，也是“得来全不费工夫”！

## 稳定性测试

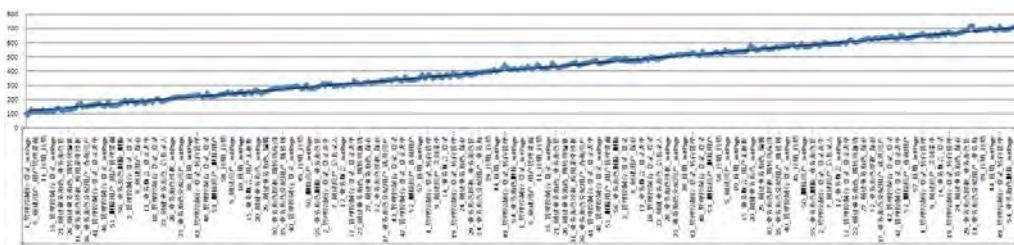
稳定性测试可以分为“服务端稳定性测试”和“客户端稳定性测试”。其中服务端（比如，应服务器、数据库等）的稳定性测试是最常见的，笔者这里不做赘述。

RIA ( Rich Internet Applications ) 技术的广泛应用为Web用户提供越来越赞的使用体验。与此同时，也比传统应用占用更多的客户端（浏览器）资源。所以对此类应用的客户端稳定性测试也是非常关键的。

验证客户端的稳定性需要两个条件：长时间的不间断操作、监控浏览器资源占用。

我们通过对Web UI自动化测试用例的循环执行可以模拟长时间的不间断操作。另外，在自动执行的同时，自动获取并记录浏览器资源占用，就可以达到验证客户端稳定性的目的。

如果系统前端代码出现内存泄露（比如，弹出页面关闭后未销毁生成的DOM对象），Web UI自动化用例长时间运行后生成的浏览器内存占用报告就会如下图所示：



而正常应该是整体呈水平趋势的锯齿状图形。如果出现上述情况，其表现出来的现象是用户感觉操作响应越来越慢，严重的情况下会导致浏览器宕掉。

至此，本系列文章就结束了。下面用三句短话总结一下本系列的主要内容：

- Web自动化测试困难的根本原因是什么；
- 如何才能更容易的成功实施Web自动化测试；
- 要么不断提升、持续证明自动化测试的价值，要么就变成鸡肋！

最后分享笔者这几年在实施自动化测试和持续集成工作中的一些心得体会：

- 在设计某个功能“自动化测试应该怎么实现”之前，一定要先全面、仔细的分析“手工测试时是怎么做的”；
- 当提到自动化测试的“测试设计”时，不光要考虑“测试用例设计”，一定还要考虑“自动化测试方案或框架的设计”
- 敢于面对各种质疑和挑战，并用持续的改善作为回应；
- 对于开发团队来说“不改善，尤可活”，对自动化测试来说“不改善，就等死”；
- 不断发掘自动化测试对各个团队的附加价值，只有这样才能得到来自四面八方的支持；
- 自动化测试的目的是节省成本，所以如果发现某个项目、模块或功能在进行自动化测试时的投入产出比还不如人工测试，就应该果断暂停；
- 当有人用“自动化测试的返工成本”来刻意刁难测试团队时，不妨请他考虑一下研发阶段的返工吧；
- 对测试用例失败的原因进行归类（比如，缺陷、环境错误、用例未更新等），并据此生成测试报告，非常有助于提高测试结果分析的效率；
- 在持续集成过程中，最初觉得技术是个坎儿，迈过之后发现管理是个更大的坎儿。要想持续集成获得成功必须让研发、测试、实施等各个团队都认识到它对自己的价值，然后大家才会主动的、同心协力将此事做好；
- 再忙也必须每日修正完持续集成测试发现的缺陷，唯有此持续集成发现的缺陷比才会越来越少。当每日构建都能成功的时候，团队信心和士气会得到很大的提振；
- 每次集成后测试用例全部都通过和基本都不通过都是不正常的现象，“全部都通过”往往代表测试用例覆盖率不够或缺乏有效断言，“基本都不通过”则可能因为研发提交代码质量太差或持续集成环境有问题；
- 持续集成的过程很不易，涉及技术改进、资源投入、团队协作、文化养成等方面，但做成之后觉得每一分投入都非常值得；

- 水滴石穿、贵在坚持！

## 作者简介

殷坤，东软集团资深测试经理、技术讲师，10年软件研发、实施、测试及项目管理工作经验。

目前专注于敏捷项目管理及质量控制、过程改善、自动化测试、持续集成、用户体验提升等方面。

---

感谢[侯伯薇](#)对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至[editors@cn.infoq.com](mailto:editors@cn.infoq.com)。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

原文链接：<http://www.infoq.com/cn/articles/develop-continuous-integration-around-automation-test>

### 相关内容

- [爱立信软件开发高级专家蔡煜：自动化测试和持续集成如何保持激情？](#)
- [敏捷自动化测试（2）——像用户使用软件一样享受自动化测试](#)
- [敏捷自动化测试\(1\)——我们的测试为什么不够敏捷？](#)
- [敏捷自动化测试（3）——让断言不再成为自动化测试的负担](#)
- [Poang，基于Node.js的自动化测试范例](#)

# 避开那些坑 | Void

## 前端工程与性能优化（上）：静态资源版本更新与缓存

作者 [张云龙](#)



每个参与过开发企业级web应用的前端工程师或许都曾思考过前端性能优化方面的问题。我们有雅虎14条性能优化原则，还有两本很经典的性能优化指导书：《高性能网站建设指南》、《高性能网站建设进阶指南》。经验丰富的工程师对于前端性能优化方法耳濡目染，基本都能一一列举出来。这些性能优化原则大概是在7年前提出的，对于web性能优化至今都有非常重要的指导意义。

然而，对于构建大型web应用的团队来说，要坚持贯彻这些优化原则并不是一件十分容易的事。因为优化原则中很多要求是与工程管理相违背的，比如“把css放在头部”和“把js放在尾部”这两条原则，我们不能让团队的工程师在写样式和脚本引用的时候都去修改一个相同的页面文件。这样做会严重影响团队成员间并行开发的效率，尤其是在团队有版本管理的情况下，每天要花大量的时间进行代码修改合并，这项成本是难以接受的。因此在前端工程界，总会看到周期性的性能优化工作，辛勤的前端工程师们每到月圆之夜就会倾巢出动根据优化原则做一次性能优化。

本文从一个全新的视角来思考web性能优化与前端工程之间的关系，通过解读百度前端集成解决方案小组（F.I.S）在打造高性能前端架构并统一百度40多条前端产品线的过程中所经历的技术尝试，揭示前端性能优化在前端架构及开发工具设计层面的实现思路。

### 性能优化原则及分类

笔者先假设本文的读者是有前端开发经验的工程师，并对企业级web应用开发及性能优化有一定的思考，因此我不会重复介绍雅虎14条性能优化原则。如果您没有这些前续知识，请移步[这里](#)来学习。

首先，我们把雅虎14条优化原则，《高性能网站建设指南》以及《高性能网站建设进阶指南》中提到的优化点做一次梳理，按照优化方向分类，可以得到这样一

张表格：

优化方向	优化手段
请求数量	合并脚本和样式表, CSS Sprites, 拆分初始化负载, 划分主域
请求带宽	开启GZip, 精简JavaScript, 移除重复脚本, 图像优化
缓存利用	使用CDN, 使用外部JavaScript和CSS, 添加Expires头, 减少DNS查找, 配置ETag, 使Ajax可缓存
页面结构	将样式表放在顶部, 将脚本放在底部, 尽早刷新文档的输出
代码校验	避免CSS表达式, 避免重定向

表格1 性能优化原则分类

目前大多数前端团队可以利用[yui compressor](#)或者[google closure compiler](#)等压缩工具很容易做到“精简Javascript”这条原则；同样的，也可以使用图片压缩工具对图像进行压缩，实现“图像优化”原则。这两条原则是对单个资源的处理，因此不会引起任何工程方面的问题。很多团队也通过引入代码校验流程来确保实现“避免css表达式”和“避免重定向”原则。目前绝大多数互联网公司也已经开启了服务端的Gzip压缩，并使用CDN实现静态资源的缓存和快速访问；一些技术实力雄厚的前端团队甚至研发出了自动CSS Sprites工具，解决了CSS Sprites在工程维护方面的难题。使用“查找-替换”思路，我们似乎也可以很好的实现“划分主域”原则。

我们把以上这些已经成熟应用到实际生产中的优化手段去除掉，留下那些还没有很好实现的优化原则。再来看看一下之前的性能优化分类：

优化方向	优化手段
请求数量	合并脚本和样式表, 拆分初始化负载
请求带宽	移除重复脚本
缓存利用	添加Expires头, 配置ETag, 使Ajax可缓存
页面结构	将样式表放在顶部, 将脚本放在底部, 尽早刷新文档的输出

表格2 较难实现的优化原则

现在有很多顶尖的前端团队可以将上述还剩下的优化原则也都一一解决，但业界大多数团队都还没能很好的解决这些问题。因此，本文将就这些原则的解决方案做进一步的分析与讲解，从而为那些还没有进入前端工业化开发的团队提供一些基础技术建设意见，也借此机会与业界顶尖的前端团队在工业化工程化方向上交流一下彼此的心得。

## 静态资源版本更新与缓存

如表格2所示，“缓存利用”分类中保留了“添加Expires头”和“配置ETag”两项。

或许有些人会质疑，明明这两项只要配置了服务器的相关选项就可以实现，为什么说它们难以解决呢？确实，开启这两项很容易，但开启了缓存后，我们的项目就开始面临另一个挑战：如何更新这些缓存。

相信大多数团队也找到了类似的答案，它和《高性能网站建设指南》关于“添加Expires头”所说的原则一样——修订文件名。即：

最有效的解决方案是修改其所有链接，这样，全新的请求将从原始服务器下载最新的内容。

思路没错，但要怎么改变链接呢？变成什么样的链接才能有效更新缓存，又能最大限度避免那些没有修改过的文件缓存不生效呢？

先来看看现在一般前端团队的做法：

```
1 <script type="text/javascript" src="a.js?t=20130825"></script>
```

或者

```
1 <script type="text/javascript" src="a.js?v=1.0.0"></script>
```

大家会采用添加query的形式修改链接。这样做是比较直观的解决方案，但在访问量较大的网站，这么做可能将面临一些新的问题。

通常一个大型的web应用几乎每天都会有迭代和更新，发布新版本也就是发布新的静态资源和页面的过程。以上述代码为例，假设现在线上运行着index.html文件，并且使用了线上的a.js资源。index.html的内容为：

```
1 <script type="text/javascript" src="a.js?v=1.0.0"></script>
```

这次我们更新了页面中的一些内容，得到一个index.html文件，并开发了新的与之匹配的a.js资源来完成页面交互，新的index.html文件的内容因此而变成了：

```
1 <script type="text/javascript" src="a.js?v=1.0.1"></script>
```

好了，现在要开始将两份新的文件发布到线上去。可以看到，index.html和a.js的资源实际上是要覆盖线上的同名文件的。不管怎样，在发布的过程中，index.html和a.js总有一个先后的顺序，从而中间出现一段或大或小的时间间隔。对于一个大型互联网应用来说即使在一个很小的时间间隔内，都有可能出现新用户访问。在这个时间间隔中，访问了网站的用户会发生什么情况呢？

1. 如果先覆盖index.html，后覆盖a.js，用户在这个时间间隙访问，会得到新的index.htm配合旧的a.js的情况，从而出现错误的页面。
2. 如果先覆盖a.js，后覆盖index.html，用户在这个间隙访问，会得到旧的index.html配合新的a.js的情况，从而也出现了错误的页面。

这就是为什么大型web应用在版本上线的过程中经常会较集中的出现前端报错日志的原因，也是一些互联网公司选择加班到半夜等待访问低峰期再上线的原因之一。此外，由于静态资源文件版本更新是“覆盖式”的，而页面需要通过修改query来更新，对于使用CDN缓存的web产品来说，还可能面临CDN缓存攻击的问题。我们再来观察一下前面说的版本更新手段：

```
1 <script type="text/javascript" src="a.js?v=1.0.0"></script>
```

我们不难预测，a.js的下一个版本是“1.0.1”，那么就可以刻意构造一串这样的请求“a.js?v=1.0.1”、“a.js?v=1.0.2”、……让CDN将当前的资源缓存为“未来的版本”。这样当这个页面所用的资源有更新时，即使更改了链接地址，也会因为CDN的原因返回给用户旧版本的静态资源，从而造成页面错误。即便不是刻意制造的攻击，在上线间隙出现访问也可能导致区域性的CDN缓存错误。

此外，当版本有更新时，修改所有引用链接也是一件与工程管理相悖的事，至少我们需要一个可以“查找-替换”的工具来自动化的解决版本号修改的问题。

对付这个问题，目前来说最优方案就是基于文件内容的**hash**版本冗余机制了。也就是说，我们希望工程师源码是这么写的：

```
1 <script type="text/javascript" src="a.js"></script>
```

但是线上代码是这样的：

```
1 <script type="text/javascript" src="a_82244e91.js"></script>
```

其中“\_82244e91”这串字符是根据a.js的文件内容进行hash运算得到的，只有文件内容发生变化了才会有更改。由于版本序列是与文件名写在一起的，而不是同名文件覆盖，因此不会出现上述说的那些问题。同时，这么做还有其他的好处：

1. 线上的a.js不是同名文件覆盖，而是文件名+hash的冗余，所以可以先上线静态资源，再上线html页面，不存在间隙问题；
2. 遇到问题回滚版本的时候，无需回滚a.js，只须回滚页面即可；
3. 由于静态资源版本号是文件内容的hash，因此所有静态资源可以开启永久强缓存，只有更新了内容的文件才会缓存失效，缓存利用率大增；
4. 修改静态资源后会在线上产生新的文件，一个文件对应一个版本，因此不会受到构造CDN缓存形式的攻击

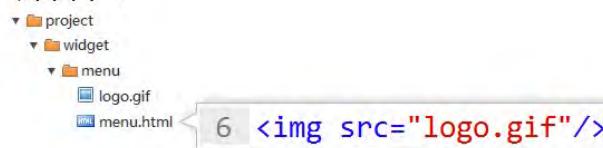
虽然这种方案是相比之下最完美的解决方案，但它无法通过手工的形式来维护，因为要依靠手工的形式来计算和替换hash值，并生成相应的文件。这将是一项非常繁琐且容易出错的工作，因此我们需要借助工具。我们下面来了解一下fis是如何完成这项工作的。

首先，之所以有这种工具需求，完全是由web应用运行的根本机制决定的：web应用所需的资源是以字面的形式通知浏览器下载而聚合在一起运行的。这种资源加载策略使得web应用从本质上区别于传统桌面应用的版本更新方式。为了实现资源定位的字面量替换操作，前端构建工具理论上需要识别所有资源定位的标记，其中包括：

- css中的@import url(path)、background:url(path)、background-image:url(path)、filter中的src
- js中的自定义资源定位函数，在fis中我们将其规定为\_\_uri(path)。
- html中的<script src="path">、<link href="path">、、已经embed、audio、video、object等具有资源加载功能的标签。

为了工程上的维护方便，我们希望工程师在源码中写的是相对路径，而工具可以将其替换为线上的绝对路径，从而避免相对路径定位错误的问题（比如js中需要定位图片路径时不能使用相对路径的情况）。

✓ 开发中：



✓ 上线后：

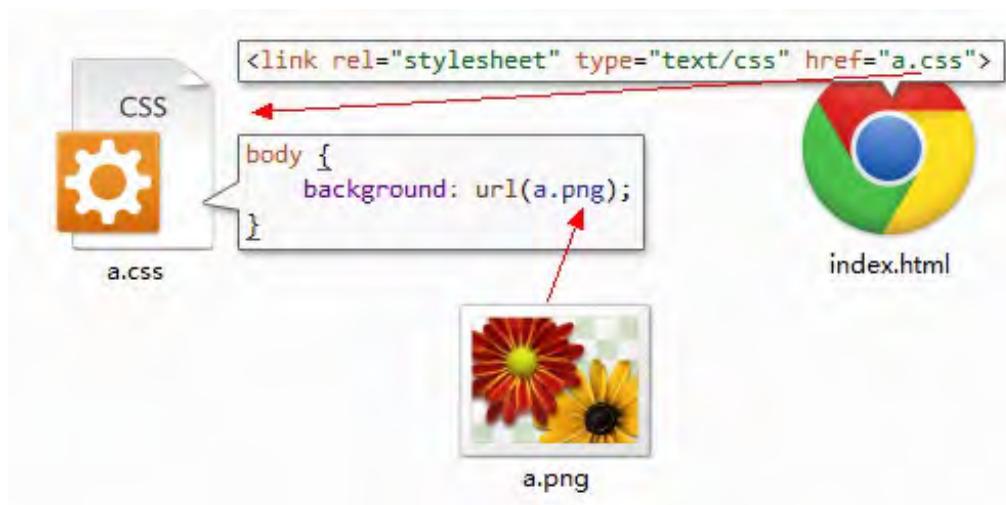


## fis的资源定位设计思想

fis有一个非常棒的资源定位系统，它是根据用户自己的配置来指定资源发布后的地址，然后由fis的资源定位系统识别文件中的定位标记，计算内容hash，并根据配置替换为上线后的绝对url路径。

要想实现具备hash版本生成功能的构建工具不是“查找-替换”这么简单的。我们

考虑这样一种情况：



### 资源引用关系

由于我们的资源版本号是通过对文件内容进行hash运算得到，如上图所示，`index.html`中引用的`a.css`文件的内容其实也包含了`a.png`的hash运算结果，因此我们在修改`index.html`中`a.css`的引用时，不能直接计算`a.css`的内容hash，而是要先计算出`a.png`的内容hash，替换`a.css`中的引用，得到了`a.css`的最终内容，再做hash运算，最后替换`index.html`中的引用。

这意味着构建工具需要具备“递归编译”的能力，这也是为什么fis团队不得不放弃gruntjs等task-based系统根本原因。针对前端项目的构建工具必须是具备递归处理能力的。此外，由于文件之间的交叉引用等原因，fis构建工具还实现了构建缓存等机制，以提升构建速度。

在解决了基于内容hash的版本更新问题之后，我们可以将所有前端静态资源开启永久强缓存，每次版本发布都可以首先让静态资源全量上线，再进一步上线模板或者页面文件，再也不用担心各种缓存和时间间隙的问题了！

在本系列的下一部分，我们将介绍静态资源管理与模板框架的思路和用法。

作者简介：张云龙，百度公司Web前端研发部前端集成解决方案小组技术负责人，目前负责F.I.S项目，读者可以关注他的微博：<http://weibo.com/foubert/>。

---

感谢崔康对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至[editors@cn.infoq.com](mailto:editors@cn.infoq.com)。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

# 避开那些坑 | Void

## 前端工程与性能优化（下）：静态资源管理与模板框架

作者 [张云龙](#)

本系列文章从一个全新的视角来思考web性能优化与前端工程之间的关系，通过解读百度前端集成解决方案小组（F.I.S）在打造高性能前端架构并统一百度40多条前端产品线的过程中所经历的技术尝试，揭示前端性能优化在前端架构及开发工具设计层面的实现思路。

在上一部分，我们介绍了[静态资源版本更新与缓存](#)。今天的部分将会介绍静态资源管理与模板框架的用法。

### 静态资源管理与模板框架

让我们再来看看前面的优化原则表还剩些什么：

优化方向	优化手段
请求数量	合并脚本和样式表，拆分初始化负载
请求带宽	移除重复脚本
缓存利用	使Ajax可缓存
页面结构	将样式表放在顶部，将脚本放在底部，尽早刷新文档的输出

很不幸，剩下的优化原则都不是使用工具就能很好实现的。或许有人会辩驳：“我用某某工具可以实现脚本和样式表合并”。嗯，必须承认，使用工具进行资源合并并替换引用或许是一个不错的办法，但在大型web应用，这种方式有一些非常严重的缺陷，来看一个很熟悉的例子：

```
<html>
  <link href="A.css">
  <link href="B.css">
  <link href="C.css">
  <div>html of A</div>
  <div>html of B</div>
  <div>html of C</div>
</html>
```

第一天



某个web产品页面有A、B、C三个资源

```
<html>
  <link href="A-B-C.css">
  <div>html of A</div>
  <div>html of B</div>
  <div>html of C</div>
</html>
```

第二天



工程师根据“减少HTTP请求”的优化原则合并了资源

```
<html>
  <link href="A-B-C.css">
  <div>html of A</div>
  <div>html of B</div>
  {if $user_has_C}
    <div>html of C</div>
  {/if}
</html>
```

第三天



差不多一个意思

产品经理要求C模块按需出现，此时C资源已出现多余的可能

```
<html>
  <link href="A-B-C.css">
  <div>html of A</div>
  <div>html of B</div>
  {*if $user_has_C}
    <div>html of C</div>
  {*}if}
</html>
```

# 第四天



C模块不再需要了，注释掉吧！但C资源通常不敢轻易剔除

```
<html>
  <link href="A-B-C-D-E-F-G-H....css">
  {*if $not_used_r}
    <div>html of E</div>
  {*}else}
    <div>html of F</div>
    <div>html of G</div>
  {*}if}
</html>
```

# 后来。。。



不知不觉中，性能优化变成了性能恶化.....

事实上，使用工具在线下进行静态资源合并是无法解决资源按需加载的问题的。如果解决不了按需加载，则势必会导致资源的冗余；此外，线下通过工具实现的资源合并通常会使得资源加载和使用的分离，比如在页面头部或配置文件中写资源引用及合并信息，而用到这些资源的html组件写在了页面其他地方，这种书写方式在工程上非常容易引起维护不同步的问题，导致使用资源的代码删除了，引用资源的代码却还在的情况。因此，在工业上要实现资源合并至少要满足如下需求：

1. 确实能减少HTTP请求，这是基本要求（合并）
2. 在使用资源的地方引用资源（就近依赖），不使用不加载（按需）

3. 虽然资源引用不是集中书写的，但资源引用的代码最终还能出现在页面头部（css）或尾部（js）
4. 能够避免重复加载资源（去重）

将以上要求综合考虑，不难发现，单纯依靠前端技术或者工具处理是很难达到这些理想要求的。现代大型web应用所展示的页面绝大多数都是使用服务端动态语言拼接生成的。有的产品使用模板引擎，比如smarty、velocity，有的则干脆直接使用动态语言，比如php、python。无论使用哪种方式实现，前端工程师开发的html绝大多数最终都不是以静态的html在线上运行的。

接下来我会讲述一种新的模板架构设计，用以实现前面说到那些性能优化原则，同时满足工程开发和维护的需要，这种架构设计的核心思想就是：

### 基于依赖关系表的静态资源管理系统与模板框架设计

考虑一段这样的页面代码：

```

1 <html>
2 <head>
3   <title>hello world</title>
4   <link rel="stylesheet" type="text/css" href="A.css">
5   <link rel="stylesheet" type="text/css" href="B.css">
6   <link rel="stylesheet" type="text/css" href="C.css">
7 </head>
8 <body>
9   <div>html of A</div>
10  <div>html of B</div>
11  <div>html of C</div>
12 </body>
13 </html>
```

根据资源合并需求中的第二项，我们希望资源引用与使用能尽量靠近，这样将来维护起来会更容易一些，因此，理想的源码是：

```

1 <html>
2 <head>
3   <title>hello world</title>
4 </head>
5 <body>
6   <link rel="stylesheet" type="text/css" href="A.css"><div>html of A</div>
7   <link rel="stylesheet" type="text/css" href="B.css"><div>html of B</div>
8   <link rel="stylesheet" type="text/css" href="C.css"><div>html of C</div>
9 </body>
10 </html>
```

当然，把这样的页面直接送达给浏览器用户是会有严重的页面闪烁问题的，所以我们实际上仍然希望最终页面输出的结果还是如最开始的截图一样，将css放在头部输出。这就意味着，页面结构需要有一些调整，并且有能力收集资源加载需

求，那么我们考虑一下这样的源码：

```

1 <html>
2 <head>
3   <title>hello world</title>
4   <!--[CSS LINKS PLACEHOLDER]-->
5 </head>
6 <body>
7   {require name="A.css"}<div>html of A</div>
8   {require name="B.css"}<div>html of B</div>
9   {require name="C.css"}<div>html of C</div>
10 </body>
11 </html>
```

在页面的头部插入一个html注释“<!--[CSS LINKS PLACEHOLDER]-->”作为占位，而将原来字面书写的资源引用改成模板接口（require）调用，该接口负责收集页面所需资源。require接口实现非常简单，就是准备一个数组，收集资源引用，并且可以去重。最后在页面输出的前一刻，我们将require在运行时收集到的“A.css”、“B.css”、“C.css”三个资源拼接成html标签，替换掉注释占位“<!--[CSS LINKS PLACEHOLDER]-->”，从而得到我们需要的页面结构。

经过fis团队的总结，我们发现模板层面只要实现三个开发接口，既可以比较完美的实现目前遗留的大部分性能优化原则，这三个接口分别是：

1. require(String id): 收集资源加载需求的接口，参数是资源id。
2. widget(String template\_id): 加载拆分成小组件模板的接口。你可以叫它为load、component或者pagelet之类的。总之，我们需要一个接口把一个大的页面模板拆分成一个个的小部分来维护，最后在原来的大页面以组件为单位来加载这些小部件。
3. script(String code): 收集写在模板中的js脚本，使之出现的页面底部，从而实现性能优化原则中的“将js放在页面底部”原则。

实现了这些接口之后，一个重构后的模板页面的源代码可能看起来就是这样的了：

```

1 <html>
2 <head>
3     <title>hello world</title>
4     <!--[CSS LINKS PLACEHOLDER]-->
5     {require name="jquery.js"}
6     {require name="bootstrap.css"}
7 </head>
8 <body>
9     {require name="A/A.css"}{widget name="A/A.tpl"}
10    {script} console.log('A loaded');{/script}
11
12    {require name="B/B.css"}{widget name="B/B.tpl"}
13    {require name="C/C.css"}{widget name="C/C.tpl"}
14
15    <!--[SCRIPTS PLACEHOLDER]-->
16 </body>
17 </html>

```

而最终在模板解析的过程中，资源收集与去重、页面script收集、占位符替换操作，最终从服务端发送出来的html代码为：

```

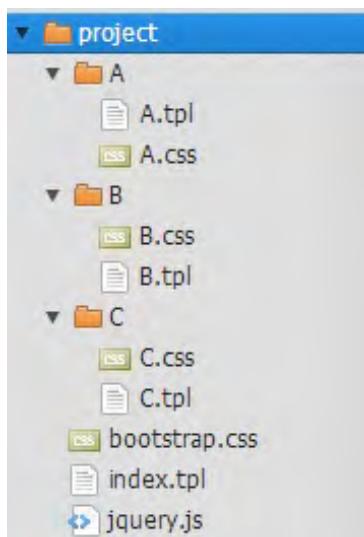
1 <html>
2 <head>
3     <title>hello world</title>
4     <link rel="stylesheet" type="text/css" href="bootstrap.css">
5     <link rel="stylesheet" type="text/css" href="A/A.css">
6     <link rel="stylesheet" type="text/css" href="B/B.css">
7     <link rel="stylesheet" type="text/css" href="C/C.css">
8 </head>
9 <body>
10    <div>html of A/A.tpl</div>
11    <div>html of B/B.tpl</div>
12    <div>html of C/C.tpl</div>
13
14    <script type="text/javascript" src="jquery.js"></script>
15    <script type="text/javascript"> console.log('A loaded'); </script>
16 </body>
17 </html>

```

不难看出，我们目前已经实现了“按需加载”，“将脚本放在底部”，“将样式表放在头部”三项优化原则。

前面讲到静态资源上线后需要添加hash戳作为版本标识，那么这种使用模板语言来收集的静态资源该如何实现这项功能呢？答案是：静态资源依赖关系表。

假设前面讲到的模板源代码所对应的目录结构为下图所示：



那么我们可以使用工具扫描整个project目录，然后创建一张资源表，同时记录每个资源的部署路径，可以得到这样的一张表：

```

1  {
2      "res": {
3          "A/A.css": {
4              "uri": "/A/A_1688c82.css",
5              "type": "css"
6          },
7          "B/B.css": {
8              "uri": "/B/B_52923ed.css",
9              "type": "css"
10         },
11         "C/C.css": {
12             "uri": "/C/C_6dda253.css",
13             "type": "css"
14         },
15         "bootstrap.css": {
16             "uri": "/bootstrap_08f2256.css",
17             "type": "css"
18         },
19         "jquery.js": {
20             "uri": "/jquery_9151577.js",
21             "type": "js"
22         }
23     },
24     "pkg": {}
25 }
```

基于这张表，我们就很容易实现 {require name="id"} 这个模板接口了。只须查表即可。比如执行{require name="jquery.js"}，查表得到它的url是"/jquery\_9151577.js"，声明一个数组收集起来就好了。这样，整个页面执行完毕之后，收集资源加载需求，并替换页面的占位符，即可实现资源的hash定位，得到：

```

1 <html>
2 <head>
3   <title>hello world</title>
4   <link rel="stylesheet" type="text/css" href="/bootstrap_08f2256.css">
5   <link rel="stylesheet" type="text/css" href="/A/A_1688c82.css">
6   <link rel="stylesheet" type="text/css" href="/B/B_52923ed.css">
7   <link rel="stylesheet" type="text/css" href="/C/C_6dda253.css">
8 </head>
9 <body>
10  <div>html of A/A.tpl</div>
11  <div>html of B/B.tpl</div>
12  <div>html of C/C.tpl</div>
13
14  <script type="text/javascript" src="/jquery_9151577.js"></script>
15  <script type="text/javascript"> console.log('A loaded'); </script>
16 </body>
17 </html>

```

接下来，我们讨论如何在基于表的设计思想上是如何实现静态资源合并的。或许有些团队使用过combo服务，也就是我们在最终拼接生成页面资源引用的时候，并不是生成多个独立的link标签，而是将资源地址拼接成一个url路径，请求一种线上的动态资源合并服务，从而实现减少HTTP请求的需求，比如：

```

1 <html>
2 <head>
3   <title>hello world</title>
4   <link rel="stylesheet" type="text/css" href="/combo?files=/bootstrap_08f2256.css,/A/A_1688c82
 .css,/B/B_52923ed.css,/C/C_6dda253.css">
5 </head>
6 <body>
7   <div>html of A/A.tpl</div>
8   <div>html of B/B.tpl</div>
9   <div>html of C/C.tpl</div>
10
11  <script type="text/javascript" src="/jquery_9151577.js"></script>
12  <script type="text/javascript"> console.log('A loaded'); </script>
13 </body>
14 </html>

```

这个“/combo?files=file1,file2,file3,...”的url请求响应就是动态combo服务提供的，它的原理很简单，就是根据get请求的files参数找到对应的多个文件，合并成一个文件来响应请求，并将其缓存，以加快访问速度。

这种方法很巧妙，有些服务器甚至直接集成了这类模块来方便的开启此项服务，这种做法也是大多数大型web应用的资源合并做法。但它也存在一些缺陷：

1. 浏览器有url长度限制，因此不能无限制的合并资源。
2. 如果用户在网站内有公共资源的两个页面间跳转访问，由于两个页面的combo的url不一样导致用户不能利用浏览器缓存来加快对公共资源的访问速度。

对于上述第二条缺陷，可以举个例子来看说明：

- 假设网站有两个页面A和B
- A页面使用了a, b, c, d四个资源
- B页面使用了a, b, e, f四个资源
- 如果使用combo服务，我们会得：

- A页面的资源引用为： /combo?files=a,b,c,d
- B页面的资源引用为： /combo?files=a,b,e,f
- 两个页面引用的资源是不同的url，因此浏览器会请求两个合并后的资源文件，跨页面访问没能很好的利用a、b这两个资源的缓存。

很明显，如果combo服务能聪明的知道A页面使用的资源引用为“/combo?files =a,b”和“/combo?files=c,d”，而B页面使用的资源引用为“/combo?files=a,b”，“/combo?files=e,f”就好了。这样当用户在访问A页面之后再访问B页面时，只需要下载B页面的第二个combo文件即可，第一个文件已经在访问A页面时缓存好了的。

基于这样的思考，fis在资源表上新增了一个字段，取名为“pkg”，就是资源合并生成的新资源，表的结构会变成：

```

1  {
2      "res": {
3          "A/A.css": {
4              "uri": "/A/A_1688c82.css", "type": "css",
5              "pkg": "p0"
6          },
7          "B/B.css": {
8              "uri": "/B/B_52923ed.css", "type": "css",
9              "pkg": "p1"
10         },
11         "C/C.css": {
12             "uri": "/C/C_6dda253.css", "type": "css",
13             "pkg": "p1"
14         },
15         "bootstrap.css": {
16             "uri": "/bootstrap_08f2256.css", "type": "css",
17             "pkg": "p0"
18         },
19         "jquery.js": {
20             "uri": "/jquery_9151577.js",
21             "type": "js"
22         }
23     },
24     "pkg": {
25         "p0": {
26             "uri": "/pkg/utils_b967346.css", "type": "css",
27             "has": [ "bootstrap.css", "A/A.css" ]
28         },
29         "p1": {
30             "uri": "/pkg/others_0d4b52a.css", "type": "css",
31             "has": [ "B/B.css", "C/C.css" ]
32         }
33     }
34 }
```

相比之前的表，可以看到新表中多了一个pkg字段，并且记录了打包后的文件所包含的独立资源。这样，我们重新设计一下{require name="id"}这个模板接

口：在查表的时候，如果一个静态资源有**pkg**字段，那么就去加载**pkg**字段所指向的打包文件，否则加载资源本身。比如执行{require name="bootstrap.css"}，查表得知bootstrap.css被打包在了“p0”中，因此取出p0包的url“/pkg/utils\_b967346.css”，并且记录页面已加载了“bootstrap.css”和“A/A.css”两个资源。这样一来，之前的模板代码执行之后得到的html就变成了：

```

1 <html>
2 <head>
3   <title>hello world</title>
4   <link rel="stylesheet" type="text/css" href="/pkg/utils_b967346.css">
5   <link rel="stylesheet" type="text/css" href="/pkg/others_0d4b52a.css">
6 </head>
7 <body>
8   <div>html of A/A.tpl</div>
9   <div>html of B/B.tpl</div>
10  <div>html of C/C.tpl</div>
11
12  <script type="text/javascript" src="/jquery_9151577.js"></script>
13  <script type="text/javascript"> console.log('A loaded'); </script>
14 </body>
15 </html>
```

css资源请求数由原来的4个减少为2个。

这样的打包结果是怎么来的呢？答案是配置得到的。

我们来看一下带有打包结果的资源表的fis配置：

```

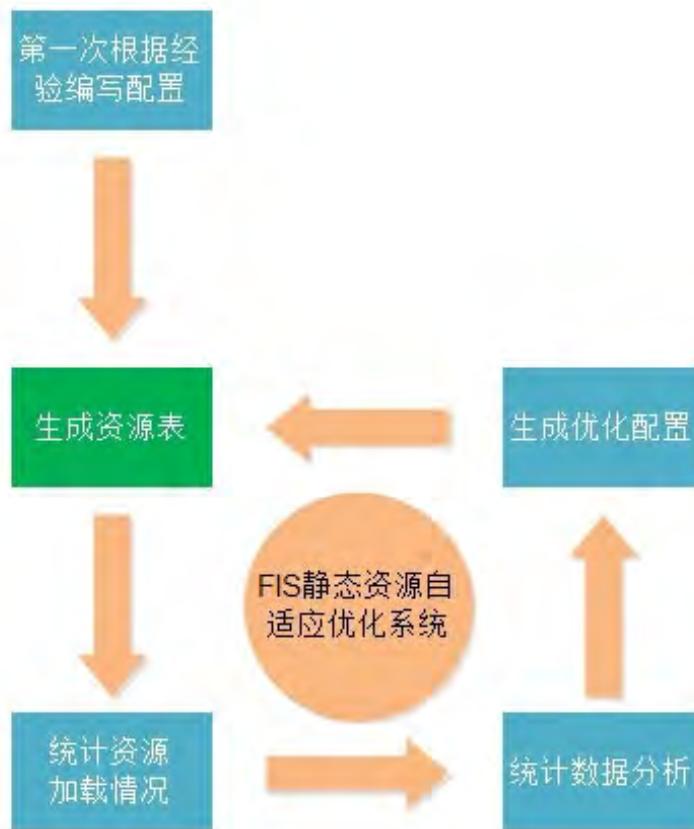
1 fis.config.set('pack', [
2   'pkg/utils.css' : [ 'bootstrap.css', 'A/A.css' ],
3   'pkg/others.css' : '**.css'
4 ]);
```

我们将“bootstrap.css”、“A/A.css”打包在一起，其他css另外打包，从而生成两个打包文件，当页面需要打包文件中的资源时，模块框架就会收集并计算出最优的资源加载结果，从而解决静态资源合并的问题。

这样做的原因是为了弥补combo在前面讲到的两点技术上的不足而设计的。但也不难发现这种打包策略是需要配置的，这就意味着维护成本的增加。但好在它有两个优势可以一定程度上弥补这个问题：

1. 打包的资源只是原来独立资源的备份。打包与否不会导致资源的丢失，最多是没有合并的很好而已。
2. 配置可以由工程师根据经验人工维护，也可以由统计日志生成，这为性能优化自适应网站设计提供了非常好的基础。

关于第二点，fis有这样辅助系统来支持自适应打包算法：



至此，我们通过基于表的静态资源管理系统和三个模板接口实现了几个重要的性能优化原则，现在我们再来看看前面的性能优化原则分类表，剔除掉已经做到了的，看看还剩下哪些没做到的：

优化方向	优化手段
请求数量	拆分初始化负载
缓存利用	使Ajax可缓存
页面结构	尽早刷新文档的输出

“拆分初始化负载”的目标是将页面一开始加载时不需要执行的资源从所有资源中分离出来，等到需要的时候再加载。工程师通常没有耐心去区分资源的分类情况，但我们可以利用组件化框架接口来帮助工程师管理资源的使用。还是从例子开始思考：

```

1 <html>
2 <head>
3   <title>hello world</title>
4   {require name="jquery.js"}
5 </head>
6 <body>
7   <button id="myBtn">Click Me</button>
8   {script}
9     $('#myBtn').click(function(){
10       var dialog = require('dialog/dialog.js');
11       dialog.alert('you catch me!');
12     });
13   {/script}
14
15   <!--[SCRIPTS PLACEHOLDER]-->
16 </body>
17 </html>

```

## 模板源代码

在fis给百度内部团队开发的架构中，如果这样书写代码，页面最终的执行结果会变成：

```

1 <html>
2 <head>
3   <title>hello world</title>
4 </head>
5 <body>
6   <script type="text/javascript" src="/jquery_9151577.js"></script>
7   <script type="text/javascript" src="/dialog/dialog_ae8c228.js"></script>
8   <script type="text/javascript">
9     $('#myBtn').click(function(){
10       var dialog = require('dialog/dialog.js');
11       dialog.alert('you catch me!');
12     });
13   </script>
14 </body>
15 </html>

```

## 模板运行后输出的html代码

fis系统会分析页面中require(id)函数的调用，并将依赖关系记录到资源表对应资源的**deps**字段中，从而在页面渲染查表时可以加载依赖的资源。但此时dialog.js是以script标签的形式同步加载的，这样会在页面初始化时出现资源的浪费。因此，fis团队提供了require.async的接口，用于异步加载一些资源，源码修改为：

```

1 <html>
2 <head>
3   <title>hello world</title>
4   {require name="jquery.js"}
5 </head>
6 <body>
7   <button id="myBtn">Click Me</button>
8   <script>
9     $('#myBtn').click(function(){
10       require.async('dialog/dialog.js', function(dialog){
11         dialog.alert('you catch me!');
12       });
13     });
14   </script>
15
16   <!--[SCRIPTS PLACEHOLDER]-->
17 </body>
18 </html>

```

这样书写之后，fis系统会在表里以**async**字段来标准资源依赖关系是异步的。fis提供的静态资源管理系统会将页面输出的结果修改为：

```

1 <html>
2 <head>
3   <title>hello world</title>
4 </head>
5 <body>
6   <script type="text/javascript" src="/jquery_9151577.js"></script>
7   <script type="text/javascript">
8     require.resourceMap({ "res": ["dialog/dialog.js"] : {"url": "/dialog/dialog_ae8c228.js"} });
9   </script>
10  <script type="text/javascript">
11    $('#myBtn').click(function(){
12      require.async('dialog/dialog.js', function(dialog){
13        dialog.alert('you catch me!');
14      });
15    });
16  </script>
17 </body>
18 </html>

```

dialog.js不会在页面以script src的形式输出，而是变成了资源注册，这样，当页面点击按钮触发require.async执行的时候，async函数才会查表找到资源的url并加载它，加载完毕后触发回调函数。

到目前为止，我们又以架构的形式实现了一项优化原则（拆分初始化负载），回顾我们的优化分类表，现在仅有两项没能做到了：

优化方向	优化手段
缓存利用	使Ajax可缓存
页面结构	尽早刷新文档的输出

剩下的两项优化原则要做到不容易，真正可缓存的Ajax在现实开发中比较少见，而尽早刷新文档的输出的情况facebook在2010年的velocity上提到过，就是BigPipe技术。当时facebook团队还讲到了Quickling和PageCache两项技术，其中的PageCache算是比较彻底的实现Ajax可缓存的优化原则了。fis团队也曾

与某产品线合作基于静态资源表、模板组件化等技术实现了页面的PipeLine输出、以及Quickling和PageCache功能，但最终效果没有达到理想的性能优化预期，因此这两个方向尚在探索中，相信在不久的将来会有新的突破。

## 总结



其实在前端开发工程管理领域还有很多细节值得探索和挖掘，提升前端团队生产力水平并不是一句空话，它需要我们能对前端开发及代码运行有更深刻的认识，对性能优化原则有更细致的分析与研究。fis团队一直致力于从架构而非经验的角度实现性能优化原则，解决前端工程师开发、调试、部署中遇到的工程问题，提供组件化框架，提高代码复用率，提供开发工具集，提升工程师的开发效率。在前端工业化开发的所有环节均有可节省的人力成本，这些成本非常可观，相信现在很多大型互联网公司也都有了这样的共识。

本文只是将这个领域中很小的一部分知识的展开讨论，抛砖引玉，希望能为业界相关领域的工作者提供一些不一样的思路。欢迎关注[fis项目](#)，对本文有任何意见或建议都可以在fis开源项目中进行反馈和讨论。

---

作者简介：张云龙，百度公司Web前端研发部前端集成解决方案小组的技术负责人，目前负责F.I.S项目，读者可以关注他的微博：<http://weibo.com/fouber/>。

---

感谢[崔康](#)对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至[editors@cn.infoq.com](mailto:editors@cn.infoq.com)。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

原文链接：<http://www.infoq.com/cn/articles/front-end-engineering-and-performance-optimization-part2>

### 相关内容

- [前端工程与性能优化（上）：静态资源版本更新与缓存](#)
- [构建iOS持续集成平台（二）——测试框架](#)
- [在线业务数据框架的探讨](#)
- [深入浅出node.js游戏服务器开发——Pomelo框架的设计动机与架构介绍](#)
- [CQRS框架Axon 2提供了对MongoDB的支持及性能提升](#)

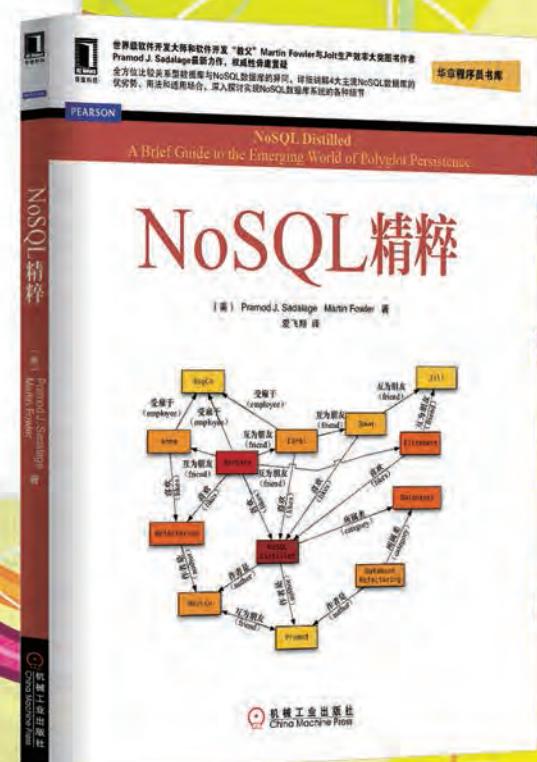
# 华章精品图书推荐



**NoSQL精粹**

世界级软件开发大师和软件开发“教父”Martin Fowler与Jolt生产效率大奖图书作者Pramod J. Sadalage最新力作，权威性毋庸置疑

ISBN: 978-7-111-43303-3  
定价: 49.00元  
作者: (美) Pramod J. Sadalage, Martin Fowler 著  
译者: 爱飞翔 译



## 诚征合作，欢迎投稿

如果您有写作或翻译计算机图书方面的计划，欢迎通过以下方式与我们取得联系。

邮箱: linux1689@gmail.com 电话: 010-88379525

微信: linux1689 联系人: 杨福川 (副总编辑)

## 新品推荐 | Product

### Ceylon整装待发

作者 [Abel Avram](#) , 译者 [吴海星](#)

Ceylon 项目的领导者Gavin King宣布Ceylon已经发布了M6版，该版本也被打上了 Ceylon 1.0 Beta的标签，语言的特性已经完备了。这次发布中包含完整的语言规范、命令行工具集（JVM和JavaScript VM的编译器、文档编译器）、SDK和基于Eclipse的IDE。

原文链接：<http://www.infoq.com/cn/news/2013/09/ceylon-beta>

### 企业软件开发者继续使用.NET 4.0

作者 [Jonathan Allen](#) , 译者 [邵思华](#)

每次一有新版本的CLR发布，例如.NET 2.0和4.0，开发者更新时都显得颇为无奈。CLR的更新为运行时的表现带来了各种微妙的变化，这有可能破坏现有代码的运行。不过像.NET 4.5这种纯类库的升级，就会使问题下降到操作系统的层面。

原文链接：<http://www.infoq.com/cn/news/2013/09/.NET-4>

### RAD Studio XE5支持Android、iOS和REST 客户端

作者 [Anand Narayanaswamy](#) , 译者 [孙镜涛](#)

最近发布的RAD Studio XE5支持Android、iOS和REST客户端。开发者能够通过RAD Studio XE5使用标准的C++或者Delphi编程语言同时为多种设备构建原型和本地应用，不需要多个项目和日程表。

原文链接：<http://www.infoq.com/cn/news/2013/09/rad-studio-xe-5>

### Visual Studio 2013 RC版本已提供下载，并且宣布了正式发布的日期

作者 [Jeff Martin](#) , 译者 [邵思华](#)

微软向开发者们宣布了一些重要的产品发布：Visual Studio 2013 RC已经可以下载，同时Windows 8.1 RTM也已经向订阅了MSDN或TechNet的开发者们开放下载了。在之前，应用开发者必须等到一项产品正式上线才能获得下载，这次的做法颠覆了以往的传统。

原文链接：<http://www.infoq.com/cn/news/2013/09/vs2013RC>

## Node.js 进入移动领域：StrongLoop 推出开源的 mBaaS

作者 [Zef Hemel](#)，译者 [吴海星](#)

之前推出即开即用企业版Node.js的公司StrongLoop今天推出了一款新产品—LoopBack，这是一个开源的移动端后台即服务产品。

原文链接：<http://www.infoq.com/cn/news/2013/09/loopback>

## 用于Visual Studio的免费PowerShell工具

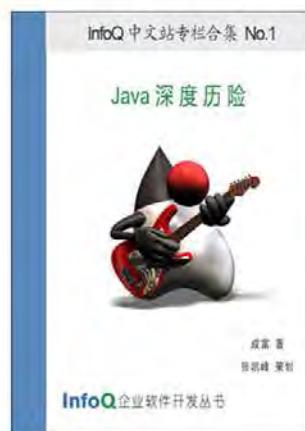
作者 [Anand Narayanaswamy](#)，译者 [李彬](#)

用于Visual Studio的PowerShell工具近期发布了，该工具拥有语法高亮显示、智能感知、代码折叠、函数导航、脚本输出，以及对断点、局部变量、堆栈帧和项目系统的支持。

原文链接：<http://www.infoq.com/cn/news/2013/09/powershell-tools-visual-studio>

# InfoQ 软件开发丛书

欢迎免费下载



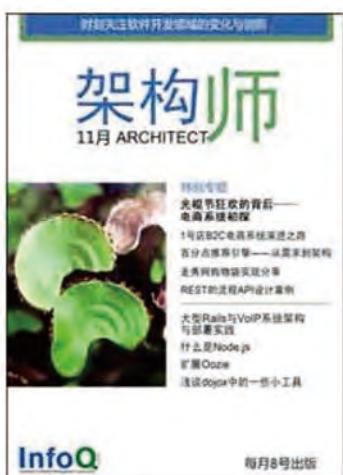
商务合作: [sales@cn.infoq.com](mailto:sales@cn.infoq.com)

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

# 架构师

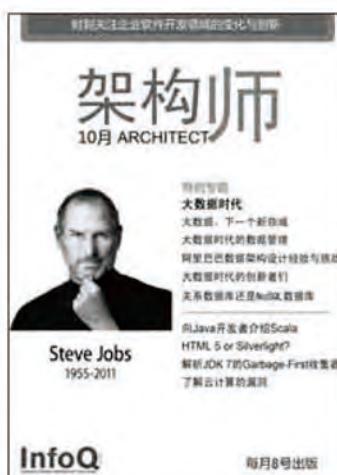
[www.infoq.com/cn/architect](http://www.infoq.com/cn/architect)

每月8号出版



InfoQ

每月8号出版



InfoQ

每月8号出版



InfoQ

每月8号出版



InfoQ

每月8号出版



InfoQ

每月8号出版

# QCon

## 全球软件开发大会 [上海站] 2013

International Software Development Conference

上海光大会展中心国际大酒店

2013年11月1-3日



9折优惠 截止10月10日

5人团购更享超值优惠

### 大会最新精彩内容呈现：

豆瓣网工程副总裁 / 段念

工程师文化中的工具 “情结”

百度IDL首席科学家 / 张潼

大数据及深度机器学习

新浪信息安全高级技术经理新浪 / 陈洋

反机器人行为系统漫谈

阿里巴巴系统运维CDN架构师 / 赵永明

Apache Traffic Server与CDN实践

ThoughtWorks首席咨询师 / 郑晔

不可思议的Moco

腾讯应用运维安全中心技术经理 / 胡珀

企业安全体系建设——理论与实践

百年人寿信息技术总监 / 李杰

百年人寿系统架构演进

腾讯资深前端开发工程师 / 石玉磊

Qzone Touch 跨终端优化

百度LAMP开发基础架构师 / 张东进

百度LAMP基础设施

腾云天下首席架构师 / 黄洋成

Off-heap技术在大数据处理中的应用

微策略高级部门经理 / 苏春园

创新与交付的探索

大众点评平台首席架构师 / 顾庆

打造高效的单机开发环境

汤森路透Elektron中国区业务主管 / 刘相峰

构建新一代高性能金融数据网络

资深游戏技术专家 / 陈超强

游戏并发编程的思想变革

抢购热线 : 010-64738142

会务咨询 : qcon@cn.infoq.com

精彩内容策划中，讲师自荐 / 推荐，线索提供 : speakers@cn.infoq.com 更多经典专题 精彩内容 敬请登录 : qconshanghai.com

# 封面植物

## 山茶花



茶花，又名山茶花，山茶科植物，属常绿灌木和小乔木。古名海石榴。有玉茗花、耐冬或曼陀罗等别名，又被分为华东山茶、川茶花和晚山茶。茶花的品种极多，是中国传统的观赏花卉，“十大名花”中排名第七，亦是世界名贵花木之一。分布于重庆、浙江、四川、江西及山东；日本、朝鲜半岛也有分布。山茶是常绿阔叶灌木或小乔木。枝条黄褐色，小枝呈绿色或绿紫色至紫褐色。叶片革质，互生，椭圆形、长椭圆形、卵形至倒卵形，长4~10cm，先端渐尖或急尖，基部楔形至近半圆形，边缘有锯齿，叶片正面为深绿色，多数有光泽，背面较淡，叶片光滑无毛，叶柄粗短，有柔毛或无毛。花两性，常单生或2~3朵着生于枝梢顶端或叶腋间。花梗极短或不明显，苞萼9~10片，覆瓦状排列，被茸毛。花单瓣，花瓣5~7片，呈1~2轮覆瓦状排列，花朵直径5~6cm，色大红，花瓣先端有凹或缺口，基部连生成一体而呈筒状；雄蕊发达，多达100余枚，花丝白色或有红晕，基部连生成筒状，集聚花心，花药金黄色；雌蕊发育正常，子房光滑无毛，3~4室，花柱单一，柱头3~5裂，结实率高。蒴果圆形，外壳木质化，成熟蒴果能自然从背缝开裂，散出种子。山茶花为常绿花木，开花于冬春之际，花姿绰约，花色鲜艳，郭沫若同志盛赞曰：“茶花一树早桃红，百朵彤云啸傲中。”对云南山茶郭老也曾赋诗赞美：“艳说茶花是省花，今来始见满城霞；人人都道牡丹好，我道牡丹不及茶。”

# QCon

# 全球软件开发大会

SHENZHEN

# 深圳 2014

International Software Development Conference

中国 · 深圳 · 万科国际会议中心

2014年1月10-11日



6折报名中，限额100人  
优惠只限10月10日前

International  
Software Development Conference  
SHENZHEN

抢购热线 : 010-64738142

会务咨询 : [qcon@cn.infoq.com](mailto:qcon@cn.infoq.com)

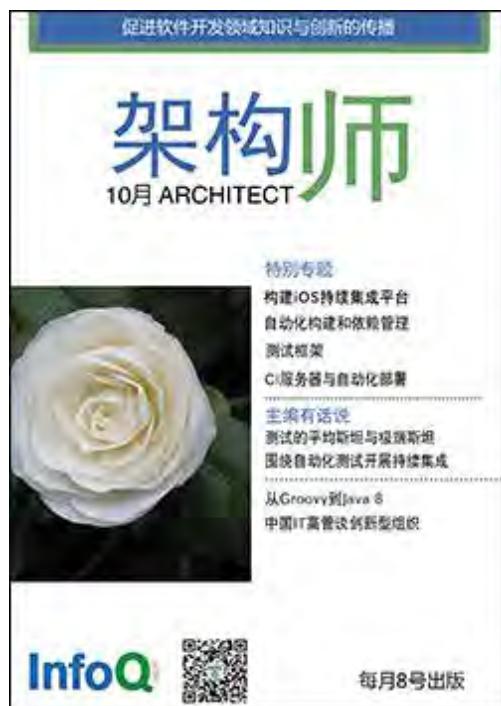
精彩内容策划中，讲师自荐 / 推荐，线索提供：[speakers@cn.infoq.com](mailto:speakers@cn.infoq.com)  
更多精彩专题 精彩内容 敬请登录：[qconshenzhen.com](http://qconshenzhen.com)

1kg.org

多背一公斤

爱自然 | 更爱孩子





## 架构师 10 月刊

每月 8 日出版

本期主编：水羽哲

顾问：杨赛

美术/流程编辑：水羽哲

发行人：霍泰稳

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

投稿：[editors@cn.infoq.com](mailto:editors@cn.infoq.com)

InfoQ 中文站新浪微博：<http://weibo.com/infoqchina>

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



本期主编：水羽哲，InfoQ 中文站内容运营编辑

水羽哲（@麦可思哲），写代码、看书、追美剧：）