《代码的未来》

图书基本信息

书名：《代码的未来》

13位ISBN编号：9787115317513

10位ISBN编号：7115317518

出版时间：2013-6

出版社：人民邮电出版社

作者：[日] 松本行弘

页数：356

译者：周自恒

《代码的未来》

内容概要

《代码的未来》是Ruby之父松本行弘的又一力作。作者对云计算、大数据时代下的各种编程语言以及相关技术进行了剖析，并对编程语言的未来发展趋势做出预测，内容涉及Go、VoltDB、node.js、CoffeeScript、Dart、MongoDB、摩尔定律、编程语言、多核、NoSQL等当今备受关注的话题。

《代码的未来》面向各层次程序设计人员和编程爱好者，也可供相关技术人员参考。

《代码的未来》

作者简介

松本行弘（Yukihiro Matsumoto）
Ruby语言发明者，亦是亚洲首屈一指的编程语言发明者。现兼任网络应用通信研究所（NaCl）研究员、乐天技术研究所研究员、Heroku首席架构师等。昵称“ Matz ”。讨厌东京，喜欢温泉。
译者简介：
周自恒
IT、编程爱好者，技术宅，初中时曾在NOI（国家信息学奥赛）天津赛区获一等奖，大学毕业后曾任IT咨询顾问，精通英语和日语，译著有《30天自制操作系统》、《大数据的冲击》、《Android应用开发入门》。

《代码的未来》

书籍目录

《代码的未来》

编辑推荐

《代码的未来》编辑推荐：20年后、100年后的编程语言会是什么样？《代码的未来》中Ruby之父松本行弘剖析云计算、大数据时代下的技术：Lisp会是未来的发展趋势吗？ Go和Dart能取代C和JavaScript吗？关系型数据库已经走到穷途末路了吗？Go、VoltDB、node.js、CoffeeScript、Dart、MongoDB…… 云计算、大数据时代下谁主沉浮？作者在《代码的未来》中一一剖析。

《代码的未来》

精彩短评

《代码的未来》

精彩书评

1、可以当闲书看. 有两个意思, 一是行文比较流畅, 也没什么太高深的东西, 但可能有你不知道的东西; 二是不要期望读这个就可以不看任何方面的书籍了, 这只是闲书. 代码的未来就是CPU多核和并行分布式处理.
2、这本书其实是连载《松本行弘：技术的剖析》的合集，与其《代码的未来》，我觉得原名更符合这本书的内容。刚开始看到书，翻开目录，发现这是什么啊，整个一个流行技术的合集，从摩尔定律讲到DSL，从C10K讲到nosql。但是仔细看下来，收获很大，有些地方又茅塞顿开的感觉。一般的技术类书籍或者文章，都是讲"How"，怎么使用，结构如何，少有讲的清"Why"的，而Matz这本书虽然讲了很多技术，但是大部分都能从复杂的细节中脱离出来，只讲原理，并说清了原因，难能可贵的是，大部分都有Matz自己的见解，非常有参考价值。可能是东方人的思维比较相近吧，整本书看得也不枯燥，花了一个周末看完了。适合人群：对各项技术都有一定了解，但是尚未形成自身的知识脉络的中级开发者，看这本书会非常有收获。
3、虽然我没有用过Ruby，但是我想他应该很美，有空去学习下。作者也善于去拉家常，很多技术都说的很入味。不过这次我没有细看，只是大概翻看了下，下次我想学习ruby时再细看下。作者是一个老编程家，是一个lisp 和 ruby的老玩家，他的预见是有一定道理的。
4、虽然本书关于代码的未来讨论的不多，而且有点头重脚轻。但是Matz的知识面还是很广的，而且语言也比较幽默。作为一本拓展知识面和回顾已学知识的书籍还不错。1，语言本身是一种dsl，Matz吐槽Pual关于语言应该精简化的预测。我自己觉得是随着需求越来越多越来越复杂，语言本身的抽象能力应该是越来越强。Matz提到api和库涉及也是语言设计这点是很认同的。例如Erlang的Actor模式,各个语言中内置的接口其实都是一种抽象的dsl。表达能力和抽象层次是一种折中。内部dsl(fluentinterface)及外部dsl的比较对于dsl的设计业很有参考作用。代码只是一种工具，有些可能适用于解决特定的问题，当前如果要开发一个适应面很广的语言的话，不论从语法到工具再到杀手级应用都需要考量了。2，GC大统一理论：扫描法(多了慢)和标记法(循环引用)的结合，前者引入了分代的方式。对于实时性较高的系统如果回收时加锁导致系统停顿是不能容忍的从而引入和增量回收。3，闭包就是在过程中包含数据而对象是在数据中包含过程。动态语言VS静态语言：前者通过jit,特殊化来提高运行效率，后者通过类型推到和反射来扩张开发效率。Lisp:list process,all is list,模式匹配4，云计算应该是说将原来在一个机器实现的各个功能分解抽象到了可扩张的集群上面，这样在设计上面就需要为scale考虑，互联网企业为了高可用舍弃了部分一致性，而对于传统软件只能舍弃性能。传统关系数据慢的问题在于日志,事务锁,内存锁,缓存管理等方面。5，从应用服务器到数据服务器现在都提供了越来越多的scaleout的方案，例如使用blance和fork实现应用集群，使用hash链实现数据集群。6，所谓水平分割，就是将一张表中的各行数据直接分割到多个表中。例如，对于像mixi这样的社交化媒体（SNS）网站，如果将用户编号为奇数的用户信息和编号为偶数的用户信息分别放在两张表中，应该会比较有效。 相对地，所谓垂直分割就是将一张表中的某些字段（列）分离到其他的表中。用SNS网站举例的话，相当于按照“ 日记 ”、“ 社区 ”等功能来对数据库进行分割。7，我从软件开发中学会了如何提高效率，作为应用，总结出了下面几个方法： 减负(算法：更高效的思考方式，开销：自动化，用空间换时间：多笔记,花钱) 拖延(分清象限) 委派 总体来说，一方面开发所需的学习材料,ide,工具包越来越强从而开发特定功能越来越容易，但是需求的复杂性导致开发所需要掌握的知识面和整合能力也需要的越来越强。就像以前是几个人盖一个平房而现在是需要现代化的工具建造一个复杂的大楼。
5、冲着作者的名气购买的书。和之前的那本一样，是杂志专栏的大杂烩，而不是深入介绍某一个特定领域的书。书中介绍了很多计算机和编程相关的背景知识，闲暇时候读一读，还是很有好处的。最后吐槽一下，字数你妹。
6、这是本人入手的第二本松本著作，由《松本行宏的程序世界》引入门，每一本著作都在我的面前展露过大师的思想光芒。松本行宏的写作习惯是在技术的文章中穿插传递矛盾性和针对冲突的论述 —— 一门技术为何会是这样子，而不是另一个样子？相较于其他将要介绍一门技术的书籍在开篇时对主体的极致夸赞，松本的著作站在一个更冷静的方位，去观察世界，观察世界的进步。他引导读者去思考，技术被创造出来的时候，创造者的智慧，困难，和精巧的折衷方案。
7、松本行弘，ruby之父，看完他的两本书，你会明白，只有不断的学习，不断的比较，不断的纠正错

误，弥补不足，引进新思想，才会有进步。叶郎自大，坐井观天，夸夸其谈，眼高手低，周围比比皆是，多学习一下别人，我们的书原创少，顶尖软件少，跟在人后面，而永远不知道超越，直至不知道如何超越。
8、《代码的未来》是Ruby之父松本行弘的又一力作。作者对云计算、大数据时代下的各种编程语言以及相关技术进行了剖析，并对编程语言的未来发展趋势做出预测，内容涉及Go、VoltDB、node.js、CoffeeScript、Dart、MongoDB、摩尔定律、编程语言、多核、NoSQL等当今备受关注的话题。《代码的未来》面向各层次程序设计人员和编程爱好者，也可供相关技术人员参考。
9、哈哈这本书看起来不错啊，想买来看一下啊，什么豆瓣居然说我的回复太短了啊，尼玛。。。。哈哈这本书看起来不错啊，想买来看一下啊，什么豆瓣居然说我的回复太短了啊，尼玛。。。。哈哈这本书看起来不错啊，想买来看一下啊，什么豆瓣居然说我的回复太短了啊，尼玛。。。。哈哈这本书看起来不错啊，想买来看一下啊，什么豆瓣居然说我的回复太短了啊，尼玛。。。。
10、代码的未来》是Ruby之父松本行弘的又一力作。作者对云计算、大数据时代下的各种编程语言以及相关技术进行了剖析，并对编程语言的未来发展趋势做出预测，内容涉及Go、VoltDB、node.js、CoffeeScript、Dart、MongoDB、摩尔定律、编程语言、多核、NoSQL等当今备受关注的话题。《代码的未来》面向各层次程序设计人员和编程爱好者，也可供相关技术人员参考。
11、本书是Matz在《日经Linux》上连载的各期内容的合集，虽然内容有些部分重复，但是内容还是很丰富的，主题也比较鲜明，与国内的某些合集甩开太远。通书下来，个人觉得最精华的还是第2章：编程语言的过去、现在和未来。Matz通过简单回顾编程语言的过去，着重分析现在和未来的发展。主要分为以下几类：DSL，meta-programming，内存管理，异常处理，闭包。DSL，即特定领域语言，将几乎只有程序员的编程语言进化成符合具体领域业务的特定语言，而且该语言与自然语言类似，方便非编程人员也可用其进行编程，可以解决开发人员对业务透视程度不够，业务人员无能力进行编码的矛盾。回想还在前东家码程序时，由于是游戏工作室，经常出现小工具方便策划实现自己的想法，这几乎是与DSL方向不谋而合。做为攻城师，如何把自己从代码中解放出来，如何让非编程人员方便的进行编程工作也是一种能力啊。元编程，即可以自己写程序的程序。身为苦逼码农，还记得需求一日一变的痛若吗？还记得数据库迁移时，代码大动的尴尬吗？请用元编程利器，让你一劳永逸，面对千变万化，不动任何代码就是完善支持。善哉，这才是程序员的理想生活。当然，元编程并不是所有语言都支持，当然上古语言Lisp可以说是其始祖，但是便于理解，还是举个Ruby的例子，如下：require 'builder'builder = Builder::XmlMarkup.newxml = builder.person {
|b|b.name("Jay")b.phone("123-123321")}#=>
<person><name>Jay</name><phone>123-123321</phone></person>代码中对person, name, phone标签是用方法调用实现的，但这些方法并不是Builder库所定义的。因为XML中的标签是任意定义的，不可能在Builder库中事先全部准备好所有方法，所以这就是元编程的力量。若要增加home标签，无须动Builder库的任何代码，只需直接调用home方法即可。完美生活啊。内存管理。说到这个，不得不提业务说c/c++是如何难用，很大一部分是由于要开发者进行内存管理。虽然现代c++语言经过一定的封装可以做到不用自己进行内存管理，但是曾经坑害多少无知码农的阴影是不会这么轻易散去。所以，垃圾回收，将内存管理从程序员手上释放出来是巨大的福利。异常处理。C代码中对各种异常返回值的判断，往往在程序中占有很大的比例，如下：int main(){FILE* f = open("/path/to/file");if (f == NULL) {puts("file open failed");} else {puts("file open succeeded");}f.close();return 0;}当然，这里只是举个简单的例子，现实情况比这个糟糕太多。写一个并不复杂的业务，假设只需要10行代码，但是对异常情况的判断并且处理，往往占据了大量的代码量，也许从开始的10行渐渐臃肿到了100行。这不仅对编码人员造成了大量的工作量，而且对维护人员进行代码学习也是一种负担。若采用异常机制，很容量让编码人员只关注重要的逻辑，而不用一头淹没在异常处理代码中。上文的例子，同样实现一个Ruby版本：beginopen("path/to/file", "r") do |f| puts"file open succeed"endrescueputs"file open failed"end闭包，含有“包含”的意思。如其名称，闭包就是将数据包含在函数内，与面向对象正好相反，面向对象是将数据的行为包含在数据内。对于闭包，我目前还没有体会到其带来的好处，当然是巨大的好处，值得做为编程语言未来发展的方法的好处，所以就先搁着，等到感受到其巨大力量再来补上。注：文件代码格式较撮，影响您的阅读，请见谅。
12、只是作者站在大师的高度去评论编程语言。反正未来的语言能够更充分的运用硬件资源来满足未

来的需求。大数据，云，多核。都是未来的方向。看了这么厚一本书感觉啥都没学到，但是却跟随大师去更高的山顶眺望编程的未来。去领略编程的世界。让我们写出高好的程序。
13、刚到手，不错啊。Matz大神讲解的随笔，虽是随笔，神似专著。随着摩尔定律的终结，编程语言的发展也与时俱进，会到一个新高度：在BIG DATA的网络环境下，如何实现多核时代的，多核编程方式。最大程度的发挥硬件的功效，提高代码的质量和速度。很值得阅读一番，有感悟，有思考，利于实际编程工作。
14、有幸得到出版社赠阅，今天终于读完了。本书是期刊文章的汇编，有点拼凑感，零散而不成系统，读起来略觉啰嗦。很多章节我都只是扫了一眼，因为我自信不会去发明一种新的编程语言，对语言的细节特性无须深入理解。看看“小结”、“后记”足以。但这本书确实开拓了我的视野，让我了解了一些大数据时代、云计算时代、多核时代的关键概念以及技术趋势。不要期望这本书有醍醐灌顶的功效，也不要作者给你指明未来，闲来随意翻翻，甚好~
15、周：松本先生今年出版了新书《代码的未来》，这本书的中文版正在由我进行翻译，预计明年会在中国出版。您的上一本书《松本行弘的程序世界》在中国受到了读者的好评，这次的新书和前作相比有哪些不同，又有哪些看点呢？Matz：《松本行弘的程序世界》一共涉及了14个话题，每个话题都是浅尝辄止，内容比较广泛但不是很深入，而这次的新书则是设定了一个大的主题——即对未来即将到来新技术的思考，因此内容比《程序世界》所涉及的范围要窄一些。此外，这本书还在时间尺度上进行了探讨，例如从计算机出现以来，到现在为止经历了怎样的变化，并由此来思考未来可能会发生的变化，也就是对过去和未来两方面都进行了思考。计算机的世界变化非常快，而这本书的目的在于探讨其未来变化的方向。周：说起计算机的发展，您在书中还提到了关于摩尔定律的一些话题呢。Matz：摩尔定律是描述计算机将如何发生变化的一个定律，书中所探讨的不仅包括计算机本身的变化，还包括计算机为周围的环境所带来的变化。周：关于编程语言进化的方向，保罗·格雷厄姆在一篇名叫“一百年后的编程语言”的文章（参见图灵图书《黑客与画家》P156）中，主张“拥有最简洁最小核心的编程语言”将是未来发展的趋势。对于这一观点，您在书中表示“不同意”，这是为什么呢？您对编程语言发展方向的看法又是怎样的呢？Matz：保罗是一个很喜欢Lisp的人，而Lisp所具备的特性正好符合他所说的“一百年后的编程语言”的样子，因此保罗认为一百年后的编程语言就应该变成Lisp这个样子。但实际上，Lisp这个语言的历史已经有50多年了，说实话，Lisp现在并没有成为一种有很多人在用的主流语言。我觉得这也许是因为Lisp对于大多数程序员来说不具备那么大的魅力，也就是说，作为一种“拥有最小核心”的语言，或者从某种意义上说是一种很“美丽”的语言，和程序员们所期望的语言之间，存在着一定的差距。如果一两年的时间里，Lisp的魅力没有被大家所接受，那还可以理解，但已经过了50年还没有被广泛接受的话，是不是它在本质上就不太符合大家的期望呢？“对人类来说好用的语言”和“拥有最小核心的语言”之间的这个差距可能是很大的，我觉得可能将来100年也没办法消除。至于未来的编程语言应该是怎样的，我觉得应该是兼具接近Lisp的运行模型，以及人类容易理解的语法这两方面特征，这么一看Ruby是不是更接近这样一种语言呢？周：松本先生被称为Ruby之父，我们知道在编程语言的设计过程中，可能要做出很多选择，例如动态还是静态、基于原型还是基于类等等。在Ruby的特性中，您认为当初最难做的选择是什么？Matz：在设计Ruby之前，我在上大学的时候还设计过另外一种语言，而那种语言是完全静态的，和Eiffel语言非常相似。而我原本也是特别喜欢静态语言的，不过上大学时设计的那种语言是以学术研究为目的的，多年之后，当我想设计一种编程语言作为自己的工具来用的时候，我就觉得还是动态语言实际用起来比较好用。抱着这样的想法，我设计了Ruby，现在看来这个设计还是正确的。那么当初对于Ruby应该是静态还是动态这个问题，也许算不上是最难的吧，但至少是我在设计中做出的“最大”的一个判断。而在此之后，因为是动态语言，那就借鉴一下Smalltalk和Lisp吧，Perl有一些功能也不错，于是如此这般吸收了这样一些语言的特性，也就显得比较自然而然了。Ruby的特点在于Mixin模块，而这个特点在Ruby诞生当时还算是非常罕见的，因为我不喜欢多继承，总觉得应该有一个更简单的方式，所以就设计了Mixin模块。周：那么现在回过头来看，Ruby当中有哪些地方会让您觉得“如果当初设计成这样就好了”呢？Matz：最开始的时候我的目标只是想实现Perl所具备的功能，因此从Perl借鉴了很多，比如说用美元符号（$）来修饰变量名之类的，现在看来觉得学得有点过了，搞得和Perl太像了。当然，除此之外还有其他一些小地方，但最主要的我觉得就是这个了，也就是跟Perl太像了这一点。刚开始的时候，还没有形成Ruby的语法习惯和文化，因此很多东西都是从Perl“抄”过来的，现在看来好像一股脑拿过来的东西太多了，里面其实有一些是不需要的。而经过一段时间之后，Ruby自己的文化已经

形成，Rails出现之后又形成了Rails的文化，而到了这个时候再看的话，可能就会觉得这些Perl的部分好像没啥必要呢。周：大家都认为“Ruby有现在的人气基本上都是由于Ruby on Rails的贡献”，您在书中也认同这个观点，那么您认为Ruby on Rails获得巨大成功的原因是什么呢？Matz：首先是得益于Web的快速发展，几乎所有的软件开发平台都在瞄准Web这个领域。以往在用CS（客户端-服务器）架构来开发的系统，现在都可以在Web上实现了。在Web上能够开发的应用变多了，这是一个主要的背景。另外，Ruby的优势在于进行软件开发非常容易，也就是开发效率比较高。这两点结合起来，我认为就是Ruby on Rails成功的主要原因。此外，Ruby还有一些比其他语言强大的特性，例如元编程（Metaprogramming）、通过猴子补丁（Monkey patch）所带来的可扩展性等等，通过这些特性，甚至可以对基础的类进行增强。DHH正是运用了Ruby的这些强大之处，开发出了Rails。而对于没有接触过Ruby的人，比如只用过Java这种比较“死板”的语言的人来说，会觉得“唉？居然还可以做到这样吗？”，我觉得这也是Rails成功的原因之一。周：据说DHH曾经是准备用PHP来开发这样一个框架的，但后来却转向了Ruby？Matz：对，因为PHP在元编程方面有很多限制吧。Rails推出之后，又出现了很多（在PHP上实现的）模仿Rails的开发框架，比如Symfony、CakePHP等等，但是Ruby所拥有的强大特性PHP却并非完全具备，即便不考虑它们各自的背景，只是单纯去对比这些开发框架的话，我还是觉得Rails更强大一些，我觉得DHH选择Ruby也正是看重了这一点。顺便，我其实是见过DHH的，在丹麦，那时候他还没开始学习Ruby，说不定那次见面也是对他产生影响的一个原因吧。周：中国读者很关心的一个话题是，Ruby目前最广泛应用的领域就是Web开发，那么在Web开发这个领域之外，Ruby的发展方向又是什么呢？Matz：的确，Ruby在Web开发领域被用得很多，例如Rails、Sinatra等开发框架。但编程的世界并非只有Web而已，我也一直希望Ruby能够从Web中走出去。在不久的将来，我认为Ruby有望被应用的领域，主要有三个。科学计算（Scientific computing），也就是大学科研中所要用到的计算。在这个领域，Python、R、matlab等语言用得非常多。我希望Ruby也能够进入这一领域，为此我们正在开发一个叫做SciRuby的项目，希望借此推动Ruby在大学科研计算领域的应用。高性能计算（High performance computing）。这个和科学计算有点接近，是运用超级计算机来进行计算的领域。和C++比起来Ruby确实要慢很多，所以大家都觉得Ruby不可能被用于高性能计算领域。东京大学一个研究生做了一个研究项目，将Ruby写的代码编译成C语言代码，然后再编译成二进制程序，这个过程中需要用到类型推导等技术，最好的情况下，速度能够达到C语言（指用C语言人工编写的同等程序）的90%。这个项目目前只发表了论文，还没有公开源代码，我希望明年这个项目的成果能够全部公开。嵌入式（Embedded）开发。所谓嵌入式就是指在微型设备，例如手机、医疗器械、机器人，在这些环境下，现在的Ruby其实并不是很适合，内存开销很大，API也不合适，因此才需要开发适合嵌入式开发的，内存开销比较小的，并且具备面向嵌入式开发API的Ruby引擎，这也就是mruby。以这三个领域为首，我希望Ruby能够在Web开发以外的领域有更多的发展。周：Twitter主要是用Rails开发的，最近我看了一则新闻，说美国大选的时候Twitter遇到了前所未有的大访问量，Twitter称为了应付访问量的上升，正在从Ruby转移到其他语言，您对这个问题怎么看呢？Matz：这里面原因很多吧。首先，Twitter刚开始开发的时候，没人知道Twitter会获得今天的成功，当时很多人觉得，这种只能写140个字的博客有什么意思呢？但Twitter却出人意料地获得了巨大的成功。在这个过程中，Twitter增加了很多新功能，在它快速发展的过程中，Ruby的贡献是相当大的。因为一个新功能从构思出来到付诸实现，可以用很短的时间就能够完成。Twitter刚开始开发的时候不可能考虑到会有现在这样大的访问量，也就是遇到了设计上的瓶颈了，因为一般的网站也不可能会有每秒上万的访问量，因此可以说现在的Twitter发展到当初在设计上的极限了。为了解决这个问题，Twitter需要开发一个全新的架构，以应付现在越来越大的访问量。不过，即便要重写架构，我觉得沿用Ruby也是可以做到的吧？（笑）话说，一个网站在遇到设计极限的时候，有很多解决方法，比如重写架构、换其他语言等等，其中重写架构我觉得是最重要的，而实际上Twitter也正是做了这方面的工作。但在这个过程中，他们的工程师想要挑战一些新的东西，那么从编程语言上来说，就提出要改用Scala，因为Scala是编译型语言，性能也不错，正好适合编写新的架构，我觉得这样也不错。在我看来，在网站所提供的服务还没有完全成型的时候，最重要的是能够对需求的变化做出快速的反应，这个时候就需要Ruby这样灵活性比较高的语言；而在网站获得成功之后，遇到了设计瓶颈，用一种新的语言，比如Scala，来编写一个新的架构，以节约一定的资源，我认为这也是很好的一个结果。Twitter转向Scala还只是在其核心部分，而在Web前端和一些内部工具上还有很多地方在用Ruby。其实，上个月我还去拜访了一下Twitter，跟他们的工程师进行了一些交流，Ruby还是用得很多的哦（笑）。周：近年来随着智能手机、平板电脑等移动设

备的普及，移动平台开发也变得非常热门。从编程语言来看，Android上是用Java，而iOS上则是用Objective-C来进行开发的，那么作为脚本语言，不仅限于Ruby，您认为在移动开发上面会有怎样的发挥呢？Matz：目前，提到移动开发，在Android上用Java，在iOS上用Objective-C似乎是板上钉钉的事情，不过这也产生了一定的隔阂，比如某个App是为iOS开发的，如果要移植到Android的话，就得全部用Java重写才行。现在也逐步产生了一种新的尝试，例如PhoneGap、Titanium等框架，通过用JavaScript、Lua等脚本语言，编写出来的App就可以实现在iOS和Android的跨平台移植。作为Ruby来说，也有一种叫Rhodes的框架，通过它就可以用Ruby编写出在iOS、Android以及Blackberry上通用的App。以前移动设备和PC相比性能差距太大，如果App不能全速运行的话，就根本没法用了。但现在移动设备的性能已经得到了大幅度的提升，通过在通用框架的基础上，采用脚本语言来进行开发的方式，性能也逐渐变得可以接受，我想今后通过这种方式，用JavaScript、Lua、Ruby等脚本语言来提升移动开发效率的做法，应该会越来越流行吧。对了，刚才我们说到mruby，其实用mruby来编写iOS和Android应用的项目也已经开始了呢，希望不久的将来这样的App能越来越多吧。周：刚才您提到了mruby，而明天您的演讲题目是Ruby 2.0，可以就mruby和Ruby 2.0的一些亮点，做一下简单的介绍吗？Matz：刚才我们也提到了，mruby是为了在微型设备上运行而设计的一种Ruby，它并非拥有Ruby的所有功能，相应地，能够在微型设备上工作才是它的长处。因此，在以往无法使用Ruby的一些设备上面，例如自动售货机、控制器、机器人等等，今后也可以用Ruby来进行开发了。说不定几年之后，在电视机、汽车等地方也能够见到Ruby开发的软件。Ruby 2.0则包含了两个信息。第一，Ruby从开始开发算起，明年将迎来20周年了。而作为对20周年的纪念，是2.0这个名字所包含的最大的一个信息。如果但从变化来看，从1.8到1.9已经是一个非常大的变化，很多人表示说很不适应，而从1.9到2.0的变化，并不像版本号的变化看起来那么大。实际上，我的目标是保证在1.9上能运行的所有程序，在2.0上面也都能运行，不会因为引擎版本升级导致原来的程序就不能用了。Ruby 2.0确实也增加了一些新的功能。比如说，现在的Ruby中，可以通过猴子补丁，对某个类中的方法进行添加和替换，但这样一来，所有的用户都会受到影响，可能有人觉得还是原来那样的好，不想跟着改，或者说这样的改动和我现在的开发会发生矛盾、冲突等等。为了解决这个问题，Ruby 2.0中可以限定猴子补丁的作用范围，这样就可以在团队开发、库开发中，避免一些改动对开发范围之外的其他人造成不必要的麻烦。以往Ruby可能都是用在小规模的项目中，而当项目逐渐扩大，开始由多个团队合作的时候，这样的机制就显得非常重要了。除此之外，Ruby 2.0中还有一些其他的新功能，也是针对团队开发的需求来设计的。周：目前世界范围内广泛使用的语言大部分都是来自欧美的，作为例外大概只有来自巴西的Lua和来自日本的Ruby，您在书中也说这种情况让人感觉“很寂寞”，那么造成这种情况的原因是什么呢？要改变这种局面，我们应该做出怎样的努力呢？Matz：关于Lua呢，其实如果你要说它是欧美的也可以，巴西是属于“拉丁美洲”嘛（笑）。要说亚洲或者东亚这边的话，那就只有Ruby了，真的是蛮寂寞的一件事。从世界范围来看，（对于编程语言来说）欧洲和美国的影响力应该是最强的，而亚洲虽然有众多的人口，但却没有能够出现很多的编程语言，确实挺寂寞的。如果说对将来的期待，别的国家我不太清楚，至少在日本，其实是有很多人在开发各种各样的编程语言，但除了Ruby以外，其他的语言在日本以外几乎就没人知道了。如果对编程语言感兴趣的人越来越多，所创造出来的编程语言也越来越多的话，这其中应该就会有那么一两个能够取得成功吧。在日本还存在一个问题就是语言障碍，日本人除了母语以外，精通外语的人不多，有趣的是，居然有用日语来编写程序的编程语言呢。（周：说到这个，其实中国也有用汉语写程序的编程语言呢。）中国也有吗？果然。不过这些语言虽然有趣，却只能给日本人用，也就无法走向世界了。说句题外话，我曾经收到过一个美国人发给我的一封邮件，他说你是个日本人，但Ruby看上去却跟英语没什么区别，因为Ruby程序都是用英语写的嘛，难道没有用日语写程序的编程语言吗？为什么没有呢？我只好回答说：有啊，只不过你不知道而已，即便知道你也没办法用嘛。现在在日本对编程语言感兴趣的人不断增加，大概我总是在网上还有书里说编程语言多么有趣，多少也是受了我的这些言论的影响吧，现在有不少人在挑战设计新的编程语言。在这些新的编程语言中，如果能有千分之一的语言能够最终获得成功，我认为就是很好的结果了。我不知道中国、韩国，以及其他一些亚洲国家中，有多少人想要挑战这一领域，不过如果大家以我的这本书为契机，能够改变“编程语言是别人给我们的，我们只能被动接受”这种看法，继而抱有“自己创造一种编程语言也不错嘛”这样想法的人能够越来越多的话，这其中一定会有人获得成功。说到开源，无论是日本，还是中国、韩国，在世界范围内发表的项目还很少，这也算是一个可以去努力的切入点。这里面可能有很多原因，比如英语很难学，（周：比如GitHub也很难用？）哈哈，GitHub

在中国能用吗？（周：能用能用……）唔，还好还好。不过，这个（哔——）的影响还是很大的，有很多资料不太容易获得吧。（周：是啊，比如Go语言的官网就上不去呢。）啊！Go的官网上不去吗！果然是因为它是Google的吧（笑）。总之，需要面对的难题还是很多的。另外，在日本也是如此，程序员把大多数时间都用在了工作（挣钱）上，要进行开源项目之类的活动就比较困难了。10年前，开源这个概念在日本也没人接受，而现在大家都逐渐明白了开源的主要性，开源项目也就逐渐增多了，我想未来几年中，在中国应该也会产生类似的变化吧，我很期待。而且，在刚开始做的时候，没人知道到底做什么会成功。我在设计Ruby的时候，也不可能想着Ruby很不错以后一定会在全世界广泛使用。因此，一种编程语言生逢其时可能更重要一些，但这种事情，你不去尝试一下是不会知道结果的。在中国，也一定会有一些编程语言或者软件因为生逢其时，从而走向世界并获得成功。周：在书里讲到Dart的时候，您说过对于编程语言来说，生态环境是很重要的。那么要建立这样的生态环境，需要一些怎样的努力呢？Matz：从用户的角度来看，用这种编程语言，对我有什么益处，这一点很重要。在Rails出现之前，使用Ruby的人，包括我自己在内，大多数都是觉得Ruby写程序很轻松，这可能是选择Ruby的主要目的。当然，Rails出现之后，因为“我要做个网站，Rails最快”这样的理由而使用Ruby的人变多了。因此，如果你设计一种新的语言，如果能够准确地传递给用户使用这个语言所能带来的好处，我想这样的语言成功的可能性会更大一些。周：非常感谢，在采访的最后，请您谈谈对中国程序员的寄语吧。Matz：在《程序世界》中我也提到了，我觉得编程的未来应该会以开源的形式发展下去，未来的创新性软件或者编程语言，可能都会以开源的形式出现。作为开源软件来说，别人做出来我可以免费使用，这一点大家都很开心。在经历了这个阶段之后，如果能够更进一步，将自己做的软件开源，使其对全世界产生影响，如果能做到这一步的话，你可能就会成为一名一流的软件工程师。中国的软件工程师，就我接触的这些人来看，大家都非常认真和努力地学习技术，我希望这些人之中，能够有更多的人迈出这一步，从而成为可以影响世界的一流程序员。
16、本书的内容是Ruby之父松本行弘在《日经Linux》上连载的技术文章，内容宽泛但不深入，主要是对现在比较流行的技术的一个简要介绍，以及希望在此基础上预测未来的编程方向。通篇读下来，介绍了许许多多现在比较重要的技术，比如DSL（领域特定语言）、元编程、各种编程语言、键值存储技术和NoSQL等等，对于中等水平的开发人员可以很好的开阔自己的视野，但对于初级的开发人员可能读起来有些吃力。总的来说，本书更多的是介绍现在，兼有对未来编程语言的一些预测。
17、对Matz大神的这本书还是蛮期待的 现在书以到手 刚读完前3章 书和我预想的也差不多 比起前一本程序世界来说好很多 程序世界里口水太多还重复 关于DSL那部分觉得很好 把外部DSL 和 内部DSL 还有DSL的优势定义都介绍了一下 简短明晰 再之又有结合一些语言来谈DSL 确实总的来说还是内部DSL的设计更优 当初学jee的SSH的时候 就被一堆XML恶心够了Matz对未来预测这部分给出的是一些主管预测 这部分其实不大明晰 有些散乱 毕竟是从杂志文章中整理得 对于新潮语言这部分 不是太感兴趣 介绍也相对简略 一门语言能发展起来确实需要很长时间检验 Go倒是值得关注 再来说下书的质量吧 纸张什么的都是无话可说 但是代码的排版或者说印刷有点小问题 在试读的pdf里 代码块是有颜色的 但是书里的代码块颜色浅的几乎没有 去看了一下 不是我一个人的问题 是印刷的问题、、、有几个无伤大雅的错误：P88倒数第3行 ：悔==>会P123图15代码最后一行：应该是 v := <- c (c后面的h多余)
18、这本书是Ruby之父松本行弘的专栏集，一方面作业回顾了各种编程语言的发展历程及其当下面临的困境，另一方面也介绍了编程中遇到的各种问题，如GC（垃圾回收）、多线程、并行处理、进程间通信等问题。在书的最后，作者站在云计算与大数据时代，重新思考了编程语言以及应对之道，如GC、NoSQL（随着这比较偏系统架构）等。这本书非常适合对于编程具有一定基础、而且对于编程感兴趣的专业人士，对于小白用户可能并不是很适合。总体来说，个人还是蛮推荐这本书的。
19、我是慕名买下这本书的。读完感觉是有价值， 但是不喜欢。这本书本来就是将杂志上的连载收集起来出的一个合集。因为是杂志文章，普及性比较多，文章写得很松散很随意，浅出但不深入。 而且54万字超过一半是无意义的杂谈或卖萌式的评论。我不知道是因为日本人本来就习惯了这样卖萌的语气， 还是翻译故意要加入些有趣的元素， 但是作为一个一丝不苟的程序员对这样的卖萌非常不喜欢。除了没用的东西， 剩下的东西都却都非常有用。作者提供了很多关于程序运行效率和开发效率方面的观点， 以及很多课本以外的，可以贯通计算机科学里各种领域的一些知识。程序语言方面介绍了一些每个人都需要知道的历史，垃圾回收和异常处理等编程必备知识的一些常见设计。性能方面介绍了超大规模（高性能）程序的设计要素，包括大规模储存（从hash 到 Distributed Hash Ring 到 NoSQL 数据库），大规模计算 （Multi-threading 到 Multi-processing 到 IPC/RPC， Async I/O， Event Driven 等

）。围绕分布式（云）技术还讨论了周边的诸如 CAP Theorem，SQL vs NoSQL 等理解云技术必知的花絮和背景。读完的感觉是学习了编程 2，3年的学生非常适合看这本书，世界观会一下子完整很多，特别是对云技术感兴趣的同学。对于基础的要点（上一段提到的）书里都涵盖得差不多，以每一个主题出发结合 Wikipedia 再用 Google 深入一下差不多就算是被普及了。可惜书里选用来介绍的一些技术框架我都没有听过，感觉赞誉并不是很高（也可能因为我是用 Python 的关系），而且还花了不少篇幅教你安装那些框架和调用她们的API，在互联网时代书真的不用这么写的。
20、先介绍了计算机硬件/语言的发展 之后介绍了各式各样的语言的特点（注意，不是教你怎么用，而是为什么会诞生这样的语言）此书提到的未来的方向是 多核，并行 并且把多线程这种繁琐与不安全的操作隐藏到编译器中去 不得不说激动不已，thread thread try catch的烦都烦死了 但如果说 线程、分布式计算 都可以像openfile一样操作的话，未来的射击狮就可吞掉不少攻城狮而升级为射击攻城狮，而反向兼并就比较困难 这篇博客可以很好的说明这个问题：[右脑革命：别学编程了，学艺术吧](http://www.36kr.com/p/201646.html) >在未来 50 年内，擅长创意的人群，如宏观思想家、艺术家、发明家、设计师等将会崛起。实际上，劳动力市场变迁的历史也证明了这种典范转移：工具的出现让体力无足轻重，装配线的出现替代了家庭手工作坊。而在未来变得无足轻重的，不是那些不懂读写代码的人，而是那些无法将一个个点与对星座的想象联系在一起的人。鉴于那个不确定何时爆发的**奇点**（PS：何为奇点，参照《时间简史》，乃是宇宙和时空的开端），对之前的预测是毫无疑义的！以至于都无法用概率来站队 那尝试下逻辑呢？这里参照该不该信上帝的那个逻辑来画瓢下：if 奇点爆发了:学艺术了？射击狮：失业else :学艺术了？射击狮：攻城狮这样来看，学艺术是有赚无赔的。此书也提到了类似观点：编程可能会消失，属于一种对于程序员来说比较悲观的未来 但其前提是：计算机懂得自然语言《数学之美》中提到自然语言相互之间的翻译都可能会引起信息的丢失，何况自然语言翻译成机器语言 另外，** “ 懂得 ” ** 是需要通过完备图灵测试的 所以在有生之年攻城狮们应该还不至于坚持码代码码到失业的地步此书还有一点受启发是，之前看python/ruby的代码，只是感觉简洁了许多，具体为什么说不清楚 原来其核心的不同之处是：**与描述算法的代码减少了** 仔细想想确实如此 比如描述一个二分，C 的话要 int 还要 main，编译时这少个标点，那少个括号，光改这些边边角角的东西去了；java就更恐怖了，eclipse打开半天，还得先弄个包名，定义个类，写核心代码的时候咖啡都变冰了
21、松本行弘的书读起来轻松有趣，平易近人，从无艰难晦涩之感。在这本书中，作者回顾并展望了编程语言的发展，认为编程语言在摩尔定律终结之后会向着分布式和支持多核的方向发展。包括Go, Dart, Lua等 “ 现代 ” 语言的关键特性也在书中做了介绍，目前还很难预料未来的流行编程语言是否会从这几者当中脱颖而出。本书的不足之处在于深度不够，对于每种技术都只是点到为止，并无深入的剖析，但读者可以根据自己的兴趣选择其中某个点继续深入研究。
22、《代码的未来》是Ruby之父松本行弘的又一力作。作者对云计算、大数据时代下的各种编程语言以及相关技术进行了剖析，并对编程语言的未来发展趋势做出预测，内容涉及Go、VoltDB、node.js、CoffeeScript、Dart、MongoDB、摩尔定律、编程语言、多核、NoSQL等当今备受关注的话题。《代码的未来》面向各层次程序设计人员和编程爱好者，也可供相关技术人员参考。
23、书中的内容主要的是思想阐述，以及各种语言宏观的概念，如果想学实在的技术这本书不适合！不过可以根据书上的所提到的技术语言进行补习！摩尔定律在硬件行业将不在遵守？发展到了极限？软件呢？作者是ruby 之父，当然也提到了ruby的特性，ruby 语言设计还是很完美的。

《代码的未来》

章节试读

1、《代码的未来》的笔记-Tuning - 图灵目录

http://www.ituring.com.cn/book/1073

2、《代码的未来》的笔记-第83页

所谓高阶函数就是用函数作为参数的函数。

3、《代码的未来》的笔记-第327页

针对客服端分配的buffer内存未释放

4、《代码的未来》的笔记-第100页

有趣的是,Ruby 的 case 结构也可以写成同样的形式,用 Ruby 编写出来的程序如图 7 所 示。那个,我并不是说 Go 是抄袭 Ruby 哦,没有证据证明这一点,而且我也觉得不大可能 会抄 Ruby,不过说实话,我还真小小地动过一点这个念头,万一是真的呢?(笑)
松本简直激萌，哈哈哈哈~

5、《代码的未来》的笔记-第183页

C10K问题的本质其实是 " 明明硬件性能已经足够，但因来自客户端的并发连接数过多导致处理产生破绽 " 。

6、《代码的未来》的笔记-第103页

有一种说法称，以硅谷为中心的Web系创业公司中，超过一半都采用了Ruby。

7、《代码的未来》的笔记-第104页

函数名(Move)前面用括号括起来的部分 “ p *Point ” 就是接收器。接收器的名称也必须逐一指定,这一点挺麻烦的,不由得让人想到 Python。 func (p *Point) Move(x, y float) { ... }

松本你这样真的好么，噗……
其实并不知道你说的这个是Python的哪个语法……

8、《代码的未来》的笔记-第89页

" 过程与数据的结合 " 是形容面向对象中的 " 对象 " 时经常使用的表达。对象是数据中以方法的形式内包含了过程，而闭包则是在过程中以环境的形式内含了数据。
外部环境不能访问（即直接读写）闭包环境里面的一般变量。

9、《代码的未来》的笔记-GC

垃圾回收讲得很简单清楚。三种基本形式：标记式，拷贝式，计数器式；三种高级形式：分代回收；增量回收；并行回收；高级形式的底层都是运用三种基本形式。

10、《代码的未来》的笔记-第54页

编程就是为自己的应用程序设计DSL的过程
在设计API时，如果能像“设计一种新的DSL”一样进行设计的话，感触应该会变得不同吧。

11、《代码的未来》的笔记-第221页

Unicorn最大限度利用了UNIX的有点，同时实现了高性能和易管理性。

12、《代码的未来》的笔记-第165页

从计算量的角度来看，理想的数据结构就是散列表。散列表是表达一个对象到另一个对象的映射的数据结构。Ruby中有一种名为Hash的内建数据结构，它就是散列表。

13、《代码的未来》的笔记-第48页

小说中的角色如果知道自己所身处的故事是虚构的，这样的小说就被称为元小说(Metaaficion)

14、《代码的未来》的笔记-第51页

在Ruby中有一个名叫Class的类，所有的类都可以看做是这个Class类的实例。Class类的类是Class本身。

15、《代码的未来》的笔记-第1页

互联网和开源降低了参与创新的门槛。即使没有高学历，即便不属于任何一家企业，只要有技术和点子就有机会。

16、《代码的未来》的笔记-闭包

把闭包的本质和名字的来源说得清楚明白。函数对象不等于闭包；闭包的本质在于闭包内外环境的：变量作用域和变量的生命期。

17、《代码的未来》的笔记-第二章

什么样的语言适合用作内部DSL的宿主语言？
简洁
灵活性

18、《代码的未来》的笔记-第86页

“对外部（局部）变量的访问”是C语言函数指针的最大弱点。

19、《代码的未来》的笔记-从性能看未来

我认为“未来的编程语言”之间，应该在如何充分利用 CPU 资源这个方面进行争夺。即便是现在，也已经有很多语言提供了并行处理的功能，而今后并行处理则会变得愈发重要。如果能将多个核心

的性能充分利用起来，说不定每个单独核心的性能就变得没有name重要的。

《代码的未来》

版权说明