在咖啡里加糖论Java世界的Ruby PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/252/2021_2022__E5_9C_A8_E5_92_96_E5_95_A1_E9_c104_252333.htm 利用动态脚本编写你的Java应用程序以及重用你的Java类库 自从计算机诞生以来，软件开发就倾向于使用高级语言进行开发。从汇编，到C，到C，再到Java，每一次升级就会面临来自各界同样的问题：太慢、而且有太多的Bug、开发者不想放弃对这些原有语言的使用。渐渐地，随着硬件的快速发展，新的研究和开发技术大大改进了编译器、解释器、和虚拟机，开发者不得不向高级语言转移，放弃他们使用的低级语言开发以提高生产力（将他们从低级语言的障碍中释放出来以提高他们的生产力）。Java现在在软件开发的很多领域里面占有主导地位，但是在这个发展过程中，动态脚本很有可能无情地取代它的地位。许多年以来，像Python、Perl、Rexx、Groovy、TCL和Ruby这样的语言能够在很多专业领域里面非常出色地工作，例如文件处理、自动测试、软件构建、代码重构、和Web图形页面设计他们有着历史性的名字“ 脚本语言 ”。而且在最近的一些年里，在大多数由Java，C 和其他编译型计算机语言开发的大型工作里面，他们也取得了相应的进展。 去年的时候，Ruby on Rails（RoR）Web框架使Ruby有了更进一步的发展。RoR结构利用简单的Ruby代码定义了一个典型的多层次Web应用程序图形页面层、业务逻辑层和数据持久层，因此减小了冗余文件、样本文件代码、生成的源代码以及配置文件。RoR框架能够更加优化更加容易地使用Ruby语言；而且Ruby，这种完善的脚本语言，相对于RoR框架来说可以在

更多的领域里面使用。 作为一个长期的Java开发者，我很可能坚持在一段时间里一直用Java作开发。但是我仍然保持在我开发的基于Java的系统里面使用其他的语言，而且Ruby最近显示出来是特别好的一种候选语言。在JRuby解释器的帮助下，Ruby和Java一起工作得很好，包括配置、整合、和Java软件的重用。而且在简单学习Ruby的过程中也提高了我Java代码的质量。使用Ruby可以让我很容易地完成像功能程序和元程序一样的技术手法，这些技术手法我在Java里面都是很难实现的。学习这些Ruby里面的技术手法可以帮助我更好鉴别什么时候而且怎样在Java开发中使用它。 这篇文章，我希望能够和你一起分享我在开发Java系统的时候使用Ruby的那种兴奋感。我比较一下Java和Ruby的优点和缺点，而且介绍一下JRuby解释器的支持者和反对者。而且我会向大家显示区分Ruby和Java使用的最佳实践以让它们各自得到最优化的使用。我会使用一些简单的代码来举例说明这个观点，并且介绍一个消息实例来展示在Java系统里面怎样结合使用Ruby，使其能够更好地使用动态元程序语言的弹性、表现方式以及功能。 Ruby vs. Java 这篇文章从一个Java开发者的角度解释了Ruby，主要是集中比较这两种语言。像Java一样，Ruby也是一种完全的面向对象的语言。但是这两种语言有很大的不同。Ruby是动态类型的而且是在源代码解释器里面运行的，这种语言能够像程序和功能范例一样支持元编程。我这里不会介绍Ruby的具体语法，接下来的文章里面会广泛地覆盖其他各个方面。 动态类型 Java有静态类型。你定义每个变量的类型，接下来在编译的过程中，如果你使用了类型错误的变量将会得到一个编译时错误。Ruby却相反，拥有动态类型：

你不用定义函数和变量的类型，而且没有到运行的时候不会使用类型检测，如果你调用一个不存在的方法就会得到错误信息。尽管这样，Ruby不会关心一个对象类型，仅仅看它是否在一个方法里面调用了这个对象的方法。因为这个原因，这种动态方法可以得到这样一个duck类型：“ 如果一个事物走起来像一只鸭子（duck）而且像一只鸭子（duck）呷呷地叫，它就是一只鸭子。 ” Listing1.Duck typing class ADuck def quack() puts "quack A". endendclass BDuck def quack() puts "quack B". endend# quack_it doesn ’ t care about the type of the argument duck, as long# as it has a method called quack. Classes A and B have no# inheritance relationship.def quack_it(duck) duck.quackenda = ADuck.newb = BDuck.newquack_it(a)quack_it(b) Java也可以通过反射让你使用动态类型，但是这种笨拙冗长的工作会导致很多混乱的异常发生，像NoSuchMethodError和InvocationTargetException；在实践中，这些异常倾向于在Java反射的代码中突然出现，而且相对于Ruby而言出现频率更高。 即使在没有使用反射的Java代码中，你会经常丢失掉静态类型的信息。比如，在Command设计模式里面使用execute（）方法必须返回Object胜于在Java代码里面使用的特殊类型，结果会导致很多ClassCastException发生。同样的，当在编译时和运行时修改方法签名的时候，运行时错误就会发生。在实践开发中，不论是Java还是Ruby，这样的错误很少引起严重的程序Bug。一个健壮的单元测试任何时候你都会用到的通常都能够及时捕捉他们。 Ruby的动态类型意思是你不用重复问你自己一个问题：在Java里面你是否经常在一行里面遇到这样冗长的代码

：XMLPersistence XMLPersistence = (XMLPersistence)persistenceManager.getPersistence(). Ruby消除了这种对于类型定义和转换的需要，上边的代码用一个典型的Ruby等价表达为；XMLPersistence = persistence_manager.persistence. Ruby的动态类型意义上不是弱类型Ruby经常需要你传递正确类型的对象。事实上，Java强制类型转换比Ruby要弱。例如，Java里面：”4”2等于”42”，这里会将整数转化为字符串，在Ruby里会抛出一个TypeError，告诉你这个“can’t convert Fixnum into String.”（Fixnum类型是不可以转化为String的）。同样的，Java里，因为作类型校正牺牲了速度，而且过多地做了整型操作，产生像Integer.MAX_VALUE 1的整型，和Integer.MIN_VALUE等价，可是Ruby类型校正整型只是在需要的时候。 不论Ruby有什么优点，Java的静态类型可以让它在大规模的项目里面作为首选：Java工具能够在开发时候明白代码意思。IDE能够在类之间依赖跟踪，找到方法和类的用处，自动检标识符而且帮助你检测代码。同样的虽然Ruby工具在这些功能上存在限制，它缺乏类型信息所以不能够完成上边这些工作。 解释性语言 Ruby是在解释器里面执行的，所以你不需要等待编译可以直接测试你的代码；你能够很平坦地利用交互方式执行Ruby，执行你键入的每一行。除开这些反馈，解释性语言在处理程序Bug地时候很少有明显的优点。使用编译性语言，能够分析代码造成的程序Bug，这些领域工程师必须决定发布过的应用程序的确切版本而且需要在源代码配置管理系统里面查找这些代码。在真实生活中，这些通常很简单。使用解释性语言，另一方面，分析的时候源代码是可以马上利用的

而且在需要的时候如果突然异常是可以修复的。 可是解释性语言因为执行慢有一个不幸的名声。对比Java的历史，一种“ semi-compiled ”语言：在早些年，JVM经常在运行时解释字节码，Java因为速度慢捐献了它的名声。然而很多应用程序花费了开发者很多时间等待用户或者是网络输入输出而且这些瓶颈一只保持着，Java开发者快速学习高效算法来优化它强于使用低级计算机语言进行开发。最近几年里，Java速度有了一定的提升；例如，基于动态分析的最优化及时编译器（just-in-time）有时候让Java胜过C ，这些在基于静态编译中最优化是受到限制的。 在很多应用程序中，Ruby不比其他语言速度慢。在不久的将来，Ruby将得到更大的进步就像Ruby本地解释器转移到基于字节码的系统，而且JVM JRuby解释器获得了将Ruby编译成Java字节码的能力。最终的，任何平台对于很多功能来说都逐渐被忽略，由此获得了平台独立性。功能性编程 Ruby支持熟练的多样化设计程序范例。另外的对于它的纯面向对象的风格，它同样支持一次性脚本的程序化风格：你可以在类和函数外任何地方写代码，就像函数在任何类之外一样。 更有趣的是，对于Java程序员寻找的新的和有用的观点，Ruby支持功能范例。你可以在Java里面使用更多的功能程序结构，使用很多类库支持就像Jakarta Commons Collections系列的集合包一样，但是这些语法是笨拙的。Ruby，虽然不是一种纯功能性语言，但是对待功能和匿名块如同完整的语言成员一样，这些都是能够像其他简单对象一样被传递到周围和利用的。 在Java里面，你经常反复使用集合的Iterator（迭代器），从逻辑上去遍历集合里面的每一个元素。但是这种遍历仅仅是这样一个细节实现：一般来说，你

仅仅尝试去接受对集合里面的每一个元素采取同样的逻辑。在Ruby里，你可以将一段代码块传给一个执行它的方法，遍历是在执行后台运行的。例如，[1,2,3,4,5].each{|n|print n*n, “ “ }将打印这样的字符串：1 4 9 16 25；迭代器将会遍历这个列表里面的每一个元素将他作为一个变量n传到代码块里面。利用在代码块之前和之后执行的指令将代码块封装起来的功能程序是很有用的。比如，在Java里面，你可以使用Command模式保证一个文件的打开和关闭、数据库的事务处理、或者其他的资源调用。这些匿名内部类和回调函数式的无用框架将会使代码变得冗余沉重；除此之外，这里还有一个变迁规则就是变量在匿名Command类里面传递的时候必须定义成为final。而且为了确保最后的代码块在逻辑上能够执行，整个工作就会使用try{…}finally{…}封装起来。 在Ruby里面，你可以封装任何函数或者代码块，不需要任何方法定义或者匿名内部类。在Listing 2里面，这个文件打开，然后在write()方法执行过后关闭。没有任何需要使用transaction()方法的代码和代码块。