微软在动态语言支持上超越了Java？ PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/269/2021_2022__E5_BE_AE_E8_BD_AF_E5_9C_A8_E5_c67_269822.htm 当.NET在2000/2001年第一次发布的时候，Java社区认为它仅仅是从语言以及标准库上对Java的一个“ 克隆 ”。我们把二者的简单实例代码进行比较以后就可以很轻易地得出这样一个感受。不过，微软从它多年的Java经验中获益匪浅，并且成功解决了一些Sun现在才后知后觉的问题。Java社区也有人开始认为，。NET和CLR要比Java发展得更加快速。Neil Bartlett称：我认为微软在CLR上的创新速度更快是非常明显的。举例来说：LINQ就是一个极其强大的新特性（补充一下，它基于Haskell语言的monads）；泛型（Generics）在C#中也比在Java中得到更早、更良好的支持（两者的泛型风格都受到Haskell的多态类型类[Polymorphic Type Classes]的启发……嗯！）；CLR提供比JVM更好的多语言支持，而且现在它又有了DLR，而JVM上还需要两年时间才能出现能够相提并论的产品。其它的例子有：模块化以及版本控制，对此。NET采用了Assembly（一些类的集合）离作为基本的部署单元来解决这个问题。Assembly包含了诸如版本信息之类的元数据，与之相反的是Java的Jar文件是缺乏这些版本信息的元数据的。这个缺陷会为那些加载了许多类库，不断增大的大型项目带来许多麻烦。目前，OSGi为这个问题提供了解决方案，而Sun也正在为将类似的解决方案加入Java 7中而忙碌着。通过增加泛型、自动装箱（AutoBoxing）、枚举类型（Enumerated types）和Annotations等特性，Java语言正在不停地追赶。NET，C#现

在提供了对匿名表达式的支持，这个特性是LINQ技术的基础组成之一。LINQ可以被认为是一种针对多种不同数据源的静态类型查询语言，这里说的数据源可以是XML，可以是关系数据库，甚至可以是任意的对象图。与此同时，Java社区还在争论语言的琐碎问题，比如说语言支持的属性（Properties），以及到底四种匿名方法（闭包）的哪一种应该被语言内建支持。随着DLR的发布，微软再次领先了，这一次是在CLR对动态语言或者脚本语言的支持领域再次开始领跑。Java领域目前还没有能够相对应的措施。Mono项目是一个非常纯净的。NET实现，它的发起者Miguel de Icaza对DLR的特点概括如下：一个针对动态语言的共享式类型系统；一个共享的AST，可以被语言开发人员用来创建新的动态语言；针对编译器开发人员的辅助/工具类；一个通用的宿主接口，从而可以将通用脚本语言的接口嵌入你的程序中，并且允许开发人员用一种货多种动态语言扩展系统；控制台支持，DLR甚至提供了一个简单的控制台接口，用于进行交互式编程。共享式类型系统（Shared Type System）是使动态语言之间能够互动及交换对象重要因素。Jim Hugunin揭示了隐藏在这之后的机制，并且演示了Java是如何处理这种情况，以及DLR是如何独辟蹊径的。Jim Hugunin肯定明白之间的区别的，毕竟他开发了Jython项目，一个基于JVM的Python实现。最近他转向微软平台，并且开发了IronPython（基于CLR的Python实现）。Jim Hugunin是这样解释其中一个问题的：使用Wrapper（包装器）的方式也可能会有更深层次的问题，挑战之一就是确定需要传递的对象。举例来说：如果Python有一个PyString对象并且它调用了一个需要Object的C#方法，是应该传递PyString对

象呢，还是应该将它解包成一个String对象呢？这些非常难以捉摸的类型问题永远都没有一个完美的答案。更糟糕的是，想在程序员不知情的情况下对对象进行包装或者解包，而导致对象标识的丢失而引起的一些超级棘手的问题。这些问题毫无疑问也存在于Java领域中，比如说JRuby 1.0在Java和Ruby代码间处理字符串传递的方式：传入Ruby代码的Java字符串将被编码为UTF-8，这暗示了你应该在接收参数的代码中用UTF-8 byte[]来工作。Ruby 字符串传出到Java时也被假定为UTF-8，Java端调用的返回结果应该符合该假定。Java领域并没有实现我们上面提到的那些东西，除了宿主接口（Hosting Interface），它将在Java 6中按照JSR 233的规范实现。（Java中的）宿主接口只是一个框架，该框架提供添加新的语言运行时，并对其进行初始化和访问的标准方式。Jim Hugunin进一步揭示了动态方法分派是如何被处理的，这个过程利用了扩展方法（Extension Methods）以及其它已有的CLR系统。在Java方面，唯一可以相提并论的努力就是JSR 292，其中要做到的一件事就是为了加入一种新的字节码invokedynamic.这个想法来源于Gilad Bracha，他在创建了这个JSR之后很快离开了Sun公司，现在他并不看好这个项目会在短期内有任何解决方案出台：JSR 292是我为了解决这些问题所发起的一项努力。我希望在缺席的情况下它仍能继续下去，但这个项目还需要若干年才能出成果（如果可能的话）。坦白地说，向JVM为这些特性添加支持，然后使Strongtalk变得更稳定是更为困难的一件事情。注：Strongtalk是一个Smalltalk实现，其VM是HotSpot技术的基础，而HotSpot技术已经随着Sun的JVM发布很长一段时间了。JSR 292的规范负

责人Danny Coward则对在性能上带来的改善更有信心：动态语言引擎的创造者们正在忙于将Ruby代码转换成Java的字节码。当JRuby的引擎尝试着将方法调用转化成字节码时，就必须创建一个合成的接口来表现返回类型。这并不是开发人员创建出来的接口，而是由JRuby引擎所创建的，所以引擎可以处理这个方法调用，并且将其转化成字节码。而那就是返回的类型对于方法参数和异常也是同样一个道理。JSR 292消除了对这种合成接口的需要。在今天，动态语言解释器必须输出方法调用的字节码，即使是在解释执行比如说一段Ruby代码的时候。明天，有了JSR 292，解释器将会用到invokedynamic版本。这将使动态语言引擎实现变得更加简单，因为现在的许多引擎对创建新的合成接口以及做许多簿记工作烦恼不已：当一个方法在七八个不同的地方被调用时，引擎就必须在所有（调用的）地方重用那些合成接口。值得关注的是，这些改进都将被写入JVM规范中，这就意味着这些特性都将被内建支持（被硬编码进去）并且在将来就不容易升级了。基于类库的方法好处在于：当处理这些系统更好的方法出现时，这个方法可以很快被采用。基于JVM的方法将在很长一段时间内保持不变，因为JVM常常会有一个很长的使用周期（作为参考：Java 1.3现在还在被许多公司所采用）。JVM真的会采用这种字节码，并且改进动态方法调用的速度吗？这也还有待观察。另一个问题是官方对基于JVM的语言的支持和认可。目前，JRuby有两名开发人员在领着Sun的薪水。其中一位是Charles O. Nutter，他已经加入了Jython和Groovy的社区当中，这些努力是否会开始还有待于观察。考虑到微软有致力于IronPython、IronRuby、JavaScript

以及动态VB支持等各种动态语言的紧密合作的开发团队，微软在这方面具有一定的优势。毕竟，DLR是一个不同团队合作的产品，这些团队在分享他们的经验并将这些经验融入一个通用的类库和知识库中，与之相反的是，基于JVM的开发团队经常不得不重复吸取重要的教训。举例而言，JRuby的特色之一就是它的即时（Just In Time，JIT）编译器，这个编译器将在运行期将Ruby代码转化为Java字节码。问题在于：在当前版本中，这样的代码会使基于set_trace_func的调试器（这些调试器使用回调的方法来实现调试器功能）不能正常工作，因为代码不再调用这个回调。这意味着，JRuby的调试要受到这种方式的影响。而同样的问题肯定要在每种语言中得到处理和解决，因此，共享哪怕是这样的一小部分经验或者代码，都会帮助其他人节省时间和工作。补充观点：微软的DLR还有待证明自己，目前集成了IronPython的DLR已经可以在网上下载到。IronRuby还没有发布，并且它的真实速度以及通用互操作能力还有待检验。比起。NET，Java仍然还有被认为是更加开放，并且能运行在更多平台上的优势。不过，Miguel de Icaza看起来确信，Mono在今年结束之前将能提供对Silverlight的支持。 