跨越边界:Java模型以外的类型策略 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022__E8_B7_A8_E8_B6_8A_E8_BE_B9_E7_c104_144745.htm 当谈到 Java 语言的类型方法时，Java 社区分为两大阵营。一些人喜欢编译时错误检查，更好的安全性，以及改善的工具 这些都是静态类型所能提供的特性。而另一些人则偏爱更动态的类型体验。这一次在 跨越边界 中，您将看到两种高生产力的非 Java 语言所使用的一些截然不同的类型策略，并发现在 Java 编程中提高类型灵活性的一些方法。 在对任何编程语言的讨论中，争议较大的一个问题就是类型模型。类型决定可以使用哪些种类的工具，并影响到应用程序的设计。很多开发人员将类型与生产率或可维护性联系起来（我就是其中的一个）。典型的 Java 开发人员通常都特别乐于维护 Java 语言的类型模型的地位，强调 Java 语言可采用更好的开发工具，在编译时捕捉某些种类的 bug（例如类型不兼容和拼写错误），以及性能等方面的优势。 如果您想理解一种新的编程语言，甚至一系列语言，那么通常应该从类型策略着手。在本文中，您将看到 Java 之外的一些语言中的类型模型。我首先简要介绍任何语言设计者在类型模型中必须考虑的一些决策，着重介绍静态类型和动态类型的一些不同的决策。我将展示一些不同极端的例子 Objective Caml 中的静态类型和 Ruby 中的动态类型。我还将谈到 Java 语言的类型限制，以及如何突破 Java 类型的限制快速编程。来源：www.examda.com 类型策略 至少可以从三个角度来看待类型： 静态类型还是动态类型，这取决于何时 实施类型模型。静态类型语言在编译时实施类型。而动态

类型语言通常基于一个对象的特征在运行时实施类型。 强类型还是弱类型，这取决于如何 实施类型模型。强类型严格地实施类型，如果发现有违反类型规则的情况，则会抛出运行时或编译时错误。而弱类型则留有更多的余地。极端情况下，弱类型语言（例如 Assembler）允许将任意数据类型赋给另一种类型（不管这种赋值是否有意义）。静态类型的语言既可以有强类型，也可以有弱类型；而动态类型系统通常是强类型的，但也不完全是。 显式类型还是隐式类型，这取决于语言如何确定一个给定对象的类型。显式类型语言要求声明每个变量和每个函数参数。而隐式类型语言则根据语言中的语法和结构线索来确定对象的类型。静态类型语言通常是显式类型的，但也不完全是；而动态类型语言几乎都是隐式类型的。 