java认证:C 和Java传统中积极的一面Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/571/2021_2022_java_E8_AE_A4_E8_AF_81_c104_571330.htm 摘要： 在最近的讨论中，有些人断定C 并不是一个设计完美的语言。在我在C 标准委员那8年里，我目睹所有关于C 的决议的诞生。我希望本文有助于帮读者理解C 和JAVA的设计选择，从而可以让大家更全面的来看待他们。 有人说，我很少再使用C .当我使用C 时，我只是为了测试一下陈旧的代码，或者写一个和性能密切相关的程序，通常这个程序非常小，并且通过其他的语言来调用。（我喜欢的做法是，用Python快速开发一个程序，用 profile辅助程序对其进行性能优化，如果需要的话，通过Python的ctypes调用C 写的程序来改善性能）。 因为我曾经是C 标准委员会的一员，我目睹了这些决议的产生。这些C 决议都是在经过超级深思熟虑的考虑之后在做出，他们远比大多数Java的决议更为谨慎小心。 然而，就像有些人准确地指出那样，C 是复杂而难于使用的，并且充满了各种个样容易让人忘记的古怪的规则。当我在写书的时候，我只能从规范中找到这些规则的说明，而不是自己能记住这些规则。 为了让人们理解C 这门语言如何即难用、复杂，同时还要有良好的设计，你必须记住一条C 中最主要的设计原则兼容C语言。这是Stroupstru最正确的决定，这样做将会出现一条让大量的C程序员通向C 程序的捷径：这条捷径允许C程序员不需要做任何修改就可以在C 下编译程序。然而，这也成为了C 语言巨大的约束，它给C 带来了强大的力量，同时也给C 带来了无尽的痛楚。正是因为这个约束导致了C 如此的成功，并且也如

此的复杂。 这些C 古怪的条约使那些没有完全了解C 的Java的设计者们犯了傻。例如，他们认为程序员能用好操作符重载将会是非常困难的一件事。但是操作符重载在C 中却是必须的，因为在C 中有栈分配，同时又有堆上的分配，你只有通过重载好操作符来处理好不同类型的内存分配，并保证不会产生内存泄漏，的确是难！但对Java来说，因为Java只有单一的一种内存分配机制（译者注：Java基本上是采用堆分配）和垃圾回收机制，这样操作符重载在Java中就变得多余（正如C#的操作符重载，和更早之前的Python操作重载，但是Python出现的要比Java早）。但是多年以来，来自Java的团队就一致认为“ 操作符重载太过复杂 ”。这一决议或其他的一些Java决议，明显说明了很多Java的设计者在做出决议的时候没有做足自己的工作，这也是为什么我有了一个藐视由Gosling和他的Java团队所做决议的名声。 同样还有太多太多的例子，基本类型“ 因为性能原因被引入 ”。真正的原因是为了坚持“ 所有都是对象 ”，并且同时为底层具有效率要求的程序提供一个后门（同时这也使得一些热点技术执行起来更有效率）。噢，但是事实是，你没有办法直接使用浮点处理器来进行超越函数的计算（译者注：Transcendental Functions，一种微积分的函数），而只能使用软件来计算，但原本这类函数就可以使用浮点计算处理器来计算的。我尽我所能将类似这样的问题罗列出来，但是我听到的结果却总是那些无用的回答“ 这就是Java的方式 ”。 当我写下泛型是个如何糟糕的设计时，我得到了同样的回应，“ 我们必须兼容之前的（糟糕的）Java的决议 ”。最后越来越多的人们获得了足够关于泛型是多难用的经验的确，C 的泛型更强大，一

致性更好（尤其现在当编译器的错误信息越来越清晰后，泛型也比以前更好使用），因为Java泛型设计很差，很难，所以人们又开始回到认真对待具现化而不是泛型，当然，这对语言是有帮助的，因为具现化这个东西并不会消弱太多的语言设计，也不会因为这些自我限制而导致语言缺陷。那个Java的问题列表在这些沉闷的回应面前只能显得单调乏味。那么，是不是这样就意味着Java是失败的语言设计呢？绝对不是，Java将主流程序员带入到了一个垃圾收集器、虚拟机、一致的错误处理模型的世界（如果你不使用异常处理，这类异常可能是非常有用的异常，正如我在《Think in Java》4ed中演示的那样）。伴随着它设计上种种缺陷，Java把我们带领到了一个更高的层次，在这个层次上我们正在准备着迎接更为高级别的语言。（译者注：作者在这里大意是说Java干了很多事虽然很不成熟，可能还有点失败，但他的成功之外是能让我们找到一种通往更为高级别的语言的铺路石。作者在这里有讥讽的意思。） 另一个观点，人们一直认为C 是语言中的先驱，许多人也认为Java是语言的先驱。但是因为虚拟机，Java使得自己更容易被别的语言替代。现在任何人都有可能快速创建一门新的语言，并且和Java具有一样的效率；而以前，要得到一个正确的，有效率的编译器花去了开发一门新语言的大部分时间。（译者注：作者在这里大意是说C 是先驱，而Java因为虚拟机让其性能比较差，有时还不如别的语言。作者在这里再次讥讽了Java的高不成低不就。） 现在，我们正在见证这一切的发生不管是更高级的静态语言，例如Scala，或者说是动态语言（译者注：Dynamic Language，如Python或Ruby），不管是新的还是移植的，例如Groovy ，JRuby

和Jython.这就是未来的趋势，并且其过度将会非常的平滑，因为你可以在已有的Java代码中使用这些新语言，如果有需要，你甚至可以重写Java中产生有性能瓶颈的地方。 正如C 会消亡一样，Java自生有可能消亡，或着被用于特殊环境之下（或仅仅是为了支持以前遗留的代码，因为Java并不像C 那样会被用于硬件编程）。但是Java 真正的亮点，也是意料之外的收获，就是如果当Java已经到了自身没法在进化的地步时，Java已经为其替代者创建一条平滑之路。所有未来的语言都将从这里学到：要么为自己创建一种可以不断重构（进化）（正如Python和Ruby做的那样）的文化，要么就让其竞争者发展壮大。 译者注：作者本文的大意是在从侧面鄙视Java标准委员会的很多决议，作者认为Java这种静态语言性能上强不过C，然后还不停地加入太多的动态语言的东西，搞不好还不如那些动态语言（如Python或Ruby），整得自己高不成低不就，其现在还不如其过去，Java再这样搞下去，未来的Java必然被别的语言所取代。 