Kotlin 和 Checked Exception

最近 JetBrains 的 Kotlin 语言忽然成了热门话题。国内小编们传言说，Kotlin 取代了 Java，成为了 Android 的“钦定语言”，很多人听了之后热血沸腾。初学者们也开始注意到 Kotlin，问出各种“傻问题”，很“功利”的问题，比如“现在学 Kotlin 是不是太早了一点？” 结果引起一些 Kotlin 老鸟们的鄙视。当然也有人来信，请求我评价 Kotlin。

对于这种评价语言的请求，我一般都不予理睬的。作为一个专业的语言研究者，我的职责不应该是去评价别人设计的语言。然而浏览了 Kotlin 的文档之后，我发现 Kotlin 的设计者误解了一个重要的问题——关于是否需要 checked exception。对于这个话题我已经思考了很久，觉得有必要分享一下我对此的看法，避免误解的传播，所以我还是决定写一篇文章。

可以说我这篇文章针对的是 checked exception，而不是 Kotlin，因为同样的问题也存在于 C# 和其它一些语言。

冷静一下

在进入主题之前，我想先纠正一些人的误解，让他们冷静下来。我们首先应该搞清楚的是，Kotlin 并不是像有些国内媒体传言的那样，要“取代 Java 成为 Android 的官方语言”。准确的说，Kotlin 只是得到了 Android 的“官方支持”，所以你可以用 Kotlin 开发 Android 程序，而不需要绕过很多限制。可以说 Kotlin 跟 Java 一样，都是 Android 的官方语言，但 Kotlin 不会取代 Java，它们是一种并存关系。

这里我不得不批评一下有些国内技术媒体，他们似乎很喜欢片面报道和歪曲夸大事实，把一个平常的事情吹得天翻地覆。如果你看看国外媒体对 Kotlin 的报道，就会发现他们用词的迥然不同：

> Google’s Java-centric Android mobile development platform is adding the Kotlin language as an officially supported development language, and will include it in the Android Studio 3.0 IDE.
>
> 译文：Google 的以 Java 为核心的 Android 移动开发平台，加入了 Kotlin 作为官方支持的开发语言。它会被包含到 Android Studio 3.0 IDE 里面。

看明白了吗？不是“取代了 Java”，而只是给了大家另一个“选择”。我发现国内的技术小编们似乎很喜欢把“选择”歪曲成“取代”。前段时间这些小编们也有类似的谣传，说斯坦福大学把入门编程课的语言“换成了 JavaScript”，而其实别人只是另外“增加”了一门课，使用 JavaScript 作为主要编程语言，原来以 Java 为主的入门课并没有被去掉。我希望大家在看到此类报道的时候多长个心眼，要分清楚“选择”和“取代”，不要盲目的相信一个事物会立即取代另一个。

Android 显然不可能抛弃 Java 而拥抱 Kotlin。毕竟现有的 Android 代码绝大部分都是 Java 写的，绝大部分程序员都在用 Java。很多人都知道 Java 的好处，所以他们不会愿意换用一个新的，未经时间考验的语言。所以虽然 Kotlin 在 Android 上得到了和 Java 平起平坐的地位，想要程序员们从 Java 转到 Kotlin，却不是一件容易的事情。

我不明白为什么每当出现一个 JVM 的语言，就有人欢呼雀跃的，希望它会取代 Java，似乎这些人跟 Java 有什么深仇大恨。他们已经为很多新语言热血沸腾过了，不是吗？Scala，Clojure…… 一个个都像中国古代的农民起义一样，煽动一批人起来造反，而其实自己都不知道自己在干什么。Kotlin 的主页也把“drastically reduce the amount of boilerplate code”作为了自己的一大特色，仿佛是在暗示大家 Java 有很多“boilerplate code”。

如果你经过理性的分析，就会发现 Java 并不是那么的讨厌。正好相反，Java 的有些设计看起来“繁复多余”，实际上却是经过深思熟虑的决定。Java 的设计者知道有些地方可以省略，却故意把它做成多余的。不理解语言“可用性”的人，往往盲目地以为简短就是好，多写几个字就是丑陋不优雅，其实不是那样的。关于 Java 的良好设计，你可以参考我之前的文章《为 Java 说句公道话》。另外在《对 Rust 语言的分析》里面，我也提到一些容易被误解的语言可用性问题。我希望这些文章对人们有所帮助，避免他们因为偏执而扔掉好的东西。

实际上我很早以前就发现了 Kotlin，看过它的文档，当时并没有引起我很大的兴趣。现在它忽然火了起来，我再次浏览它的新版文档，却发现自己还是会继续使用 Java 或者 C++。虽然我觉得 Kotlin 比起 Java 在某些小地方设计相对优雅，一致性稍好一些，然而我并没有发现它可以让我兴奋到愿意丢掉 Java 的地步。实际上 Kotlin 的好些小改进，我在设计自己语言的时候都已经想到了，然而我并不觉得它们可以成为人们换用一个新语言的理由。

Checked Exception（CE）的重要性

有几个我觉得很重要的，具有突破性的语言特性，Kotlin 并没有实现。另外我还发现一个很重要的 Java 特性，被 Kotlin 的设计者给盲目抛弃了。这就是我今天要讲的主题：checked exception。我不知道这个术语有什么标准的中文翻译，为了避免引起定义混乱，下文我就把它简称为“CE”好了。

先来科普一下 CE 到底是什么吧。Java 要求你必须在函数的类型里面声明它可能抛出的异常。比如，你的函数如果是这样：

```
void foo(string filename) throws FileNotFoundException
{
  if (...)
  {
    throw new FileNotFoundException();
  }
  ...
}
```

Java 要求你必须在函数头部写上“throws FileNotFoundException”，否则它就不能编译。这个声明表示函数在某些情况下，会抛出 FileNotFoundException 这个异常。由于编译器看到了这个声明，它会严格检查你对 foo 函数的用法。在调用 foo 的时候，你必须使用 try-catch 处理这个异常，或者在调用的函数头部也声明 “throws FileNotFoundException”，把这个异常传递给上一层调用者。

```
try
{
    foo("blah");
}
catch (FileNotFoundException e)
{
    ...
}
```

这种对异常的声明和检查，叫做“checked exception”。很多语言（包括 C++，C#，JavaScript，Python……）都有异常机制，但它们不要求你在函数的类型里面声明可能出现的异常类型，也不使用静态类型系统对异常的处理进行检查和验证。我们说这些语言里面有“exception”，却没有“checked exception”。

理解了 CE 这个概念，下面我们来谈正事：Kotlin 和 C# 对 CE 的误解。

Kotlin 的文档明确的说明，它不支持类似 Java 的 checked exception（CE），指出 CE 的缺点是“繁琐”，并且列举了几个普通程序员心目中“大牛”的文章，想以此来证明为什么 Java 的 CE 是一个错误，为什么它不解决问题，却带来了麻烦。这些人包括了 Bruce Eckel 和 C# 的设计者 Anders Hejlsberg。

很早的时候我就看过 Hejlsberg 的这些言论。他的话看似有道理，然而通过自己编程和设计语言的实际经验，我发现他并没有抓住问题的关键。他的论述里有好几处逻辑错误，一些自相矛盾，还有一些盲目的臆断，所以这些言论并没能说服我。正好相反，实在的项目经验告诉我，CE 是 C# 缺少的一项重要特性，没有了 CE 会带来相当麻烦的后果。在微软写 C# 的时候，我已经深刻体会到了缺少 CE 所带来的困扰。现在我就来讲一下，CE 为什么是很重要的语言特性，然后讲一下为什么 Hejlsberg 对它的批评是站不住脚的。

首先，写 C# 代码时最让我头痛的事情之一，就是 C# 没有 CE。每调用一个函数（不管是标准库函数，第三方库函数，还是队友写的函数，甚至我自己写的函数），我都会疑惑这个函数是否会抛出异常。由于 C# 的函数类型上不需要标记它可能抛出的异常，为了确保一个函数不会抛出异常，你就需要检查这个函数的源代码，以及它调用的那些函数的源代码……

也就是说，你必须检查这个函数的整个“调用树”的代码，才能确信这个函数不会抛出异常。这样的调用树可以是非常大的。说白了，这就是在用人工对代码进行“全局静态分析”，遍历整个调用树。这不但费时费力，看得你眼花缭乱，还容易漏掉出错。显然让人做这种事情是不现实的，所以绝大部分时候，程序员都不能确信这个函数调用不会出现异常。

在这种疑虑的情况下，你就不得不做最坏的打算，你就得把代码写成：

```
try
{
    foo();
}
catch (Exception)
{
    ...
}
```

注意到了吗，这也就是你写 Java 代码时，能写出的最糟糕的异常处理代码！因为不知道 foo 函数里面会有什么异常出现，所以你的 catch 语句里面也不知道该做什么。大部分人只能在里面放一条 log，记录异常的发生。这是一种非常糟糕的写法，不但繁复，而且可能掩盖运行时错误。有时候你发现有些语句莫名其妙没有执行，折腾好久才发现是因为某个地方抛出了异常，所以跳到了这种 catch 的地方，然后被忽略了。如果你忘了写 catch (Exception)，那么你的代码可能运行了一段时间之后当掉，因为忽然出现一个测试时没出现过的异常……

所以对于 C# 这样没有 CE 的语言，很多时候你必须莫名其妙这样写，这种做法也就是我在微软的 C# 代码里经常看到的。问原作者为什么那里要包一层 try-catch，答曰：“因为之前这地方出现了某种异常，所以加了个 try-catch，然后就忘了当时出现的是什么异常，具体是哪一条语句会出现异常，总之那一块代码会出现异常……” 如此写代码，自己心虚，看的人也糊涂，软件质量又如何保证？

那么 Java 呢？因为 Java 有 CE，所以当你看到一个函数没有声明异常，就可以放心的省掉 try-catch。所以这个 C# 的问题，自然而然就被避免了，你不需要在很多地方疑惑是否需要写 try-catch。Java 编译器的静态类型检查会告诉你，在什么地方必须写 try-catch，或者加上 throws 声明。如果你用 IntelliJ，把光标放到 catch 语句上面，可能抛出那种异常的语句就会被加亮。C# 代码就不可能得到这样的帮助。

```
OutputStream symOut = null, refOut = null, docOut = null;
try {
    docOut = new BufferedOutputStream(new FileOutputStream(docFilename));
    symOut = new BufferedOutputStream(new FileOutputStream(symFilename));
    refOut = new BufferedOutputStream(new FileOutputStream(refFilename));
    $.msg( m: "graphing: " + srcpath);
    graph(srcpath, inclpaths, symOut, refOut, docOut);
    docOut.flush();
    symOut.flush();
    refOut.flush();
} catch (FileNotFoundException e) {
    System.err.println("Could not find file: " + e);
    return;
} finally {
    if (docOut != null) {
        docOut.close();
```

CE 看起来有点费事，似乎只是为了“让编译器开心”，然而这其实是每个程序员必须理解的事情。出错处理并不是 Java 所特有的东西，就算你用 C 语言，也会遇到本质一样的问题。使用任何语言都无法逃脱这个问题，所以必须把它想清楚。在《编程的智慧》一文中，我已经讲述了如何正确的进行出错处理。如果你滥用 CE，当然会有不好的后果，然而如果你使用得当，就会起到事半功倍，提高代码可靠性的效果。

Java 的 CE 其实对应着一种强大的逻辑概念，一种根本性的语言特性，它叫做“union type”。这个特性只存在于 Typed Racket 等一两个不怎么流行的语言里。Union type 也存在于 PySonar 类型推导和 Yin 语言里面。你可以把 Java 的 CE 看成是对 union type 的一种不完美的，丑陋的实现。虽然实现丑陋，写法麻烦，CE 却仍然有着 union type 的基本功能。如果使用得当，union type 不但会让代码的出错处理无懈可击，还可以完美的解决 null 指针等头痛的问题。通过实际使用 Java 的 CE 和 Typed Racket 的 union type 来构建复杂项目，我很确信 CE 的可行性和它带来的好处。

现在我来讲一下为什么 Hejlsberg 对于 CE 的批评是站不住脚的。他的第一个错误，俗话说就是“人笨怪刀钝”。他把程序员对于出错处理的无知，不谨慎和误用，怪罪在 CE 这个无辜的语言特性身上。他的话翻译过来就是：“因为大部分程序员都很傻，没有经过严格的训练，不小心又懒惰，所以没法正确使用 CE。所以这个特性不好，是没用的！”

他的论据里面充满了这样的语言：

- “大部分程序员不会处理这些 throws 声明的异常，所以他们就给自己的每个函数都加上 throws Exception。这使得 Java 的 CE 完全失效。”
- “大部分程序员根本不在乎这异常是什么，所以他们在程序的最上层加上 catch (Exception)，捕获所有的异常。”
- “有些人的函数最后抛出 80 多种不同的异常，以至于使用者不知道该怎么办。”……

注意到了吗，这种给每个函数加上 `throws Exception` 或者 `catch (Exception)` 的做法，也就是我在《编程的智慧》里面指出的经典错误做法。要让 CE 可以起到良好的作用，你必须避免这样的用法，你必须知道自己在干什么，必须知道被调用的函数抛出的 exception 是什么含义，必须思考如何正确的处理它们。

另外 CE 就像 union type 一样，如果你不小心分析，不假思索就抛出异常，就会遇到他提到的“抛出 80 多种异常”的情况。出现这种情况往往是因为程序员没有仔细思考，没有处理本来该自己处理的异常，而只是简单的把下层的异常加到自己函数类型里面。在多层调用之后，你就会发现最上面的函数累积起很多种异常，让调用者不知所措，只好传递这些异常，造成恶性循环。终于有人烦得不行，把它改成了“throws Exception”。

我在使用 Typed Racket 的 union type 时也遇到了类似的问题，但只要你严格检查被调用函数的异常，尽量不让它们传播，严格限制自己抛出的异常数目，缩小可能出现的异常范围，这种情况是可以避免的。CE 和 union type 强迫你仔细的思考，理顺这些东西之后，你就会发现代码变得非常缜密而优雅。其实就算你写 C 代码或者 JavaScript，这些问题是同样存在的，只不过这些语言没有强迫你去思考，所以很多时候问题被稀里糊涂掩盖了起来，直到很长时间之后才暴露出来，不可救药。

所以可以说，这些问题来自于程序员自己，而不是 CE 本身。CE 只提供了一种机制，至于程序员怎么使用它，是他们自己的职责。再好的特性被滥用，也会产生糟糕的结果。Hejlsberg 对这些问题使用了站不住脚的理论。如果你假设程序员都是糊里糊涂写代码，那么你可以得出无比惊人的结论：所有用于防止错误的语言特性都是没用的！因为总有人可以懒到不理解这些特性的用法，所以他总是可以滥用它们，绕过它们，写出错误百出的代码，所以静态类型没用，CE 没用，…… 有这些特性的语言都是垃圾，大家都写 PHP 就行了 ;)

Hejlsberg 把这些不理解 CE 用法，懒惰，滥用它的人作为依据，以至于得出 CE 是没用的特性，以至于不把它放到 C# 里面。由于某些人会误用 CE，结果就让真正理解它的人也不能用它。最后所有人都退化到最笨的情况，大家都只好写 `catch (Exception)`。在 Java 里，至少有少数人知道应该怎么做，在 C# 里，所有人都被迫退化成最差的 Java 程序员 ;)

另外，Hejlsberg 还指出 C# 代码里没有被 catch 的异常，应该可以用“静态分析”检查出来。可以看出来，他并不理解这种静态检查是什么规模的问题。要能用静态分析发现 C# 代码里被忽略的异常，你必须进行“全局分析”，也就是说为了知道一个函数是否会抛出异常，你不能只看这个函数。你必须分析这个函数的代码，它调用的代码，它调用

的代码调用的代码…… 所以你需要分析超乎想象的代码量，而且很多时候你没有源代码。所以对于大型的项目，这显然是不现实的。

相比之下，Java 要求你对异常进行 throws 显式声明，实质上把这个全局分析问题分解成了一个个模块化（modular）的小问题。每个函数作者完成其中的一部分，调用它的人完成另外一部分。大家合力帮助编译器，高效的完成静态检查，防止漏掉异常处理，避免不必要的 try-catch。实际上，像 Exceptional 一类的 C# 静态检查工具，会要求你在注释里写出可能抛出的异常，这样它才能发现被忽略的异常。所以 Exceptional 其实重新发明了 Java 的 CE，只不过 throws 声明被写成了一个注释而已。

说到 C#，其实它还有另外一个特别讨厌的设计错误，引起了很多不必要的麻烦。感兴趣的人可以看看我这篇文章：《可恶的 C# IDisposable 接口》。这个问题浪费了整个团队两个月之久的时间。所以我觉得作为 C# 的设计者，Hejlsberg 的思维局限性相当大。我们应该小心的分析和论证这些人的言论，不应该把他们作为权威而盲目接受，以至于让一个优秀的语言特性被误解，不能进入到新的语言里。

结论？

所以我对 Kotlin 是什么“结论”呢？我没有结论，这篇文章就像我所有的看法一样，仅供参考。显然 Kotlin 有的地方做得比 Java 好，所以它不会因为没有 CE 而完全失去意义。我不想打击人们对新事物的兴趣，我甚至鼓励有时间的人去试试看。

我知道很多人希望我给他们一个结论，到底是用一个语言，还是不用它，这样他们就不用纠结了，然而我并不想给出一个结论。一来是因为我不想让人感觉我在“控制”他们，如何看待一个东西是他们的自由，是否采用一个东西是他们自己的决定。二来是因为我还没有时间和机会，去用 Kotlin 来做实际的项目。另外，我早就厌倦了试用新的语言，如果一个大众化的语言没有特别讨厌，不可原谅的设计失误，我是不会轻易换用新语言的。我宁愿让其他人做我的小白鼠，去试用这些新语言。到后来我有空了，再去看看他们的成功或者失败经历 :P

所以对我个人而言，我至少现在不会去用 Kotlin，但我并不想让其他人也跟我一样。因为 Java，C++ 和 C 已经能满足我的需求，它们相当稳定，而且我对它们已经很熟悉，所以我为什么要花精力去学一个新的语言，去折腾不成熟的工具，放下我真正感兴趣的算法和数据结构等问题呢？实际上不管我用什么语言写代码，我的头脑里都在用同一个语言构造程序。我写代码的过程，只不过是在为我脑子里的“万能语言”找到对应的表达方式而已。

（本文建议零售价 ￥15）