

## 32 | 编程范式游记（3） - 类型系统和泛型的本质

2018-01-18 陈皓

左耳听风

[进入课程 >](#)



前面，我们讨论了从 C 到 C++ 的泛型编程方法，并且初探了更为抽象的函数式编程。正如在上一篇文章中所说的，泛型编程的方式并不只有 C++ 这一种类型，我们只是通过这个过程了解一下，底层静态类型语言的泛型编程原理。这样能够方便我们继续后面的历程。

是的，除了 C++ 那样的泛型，如果你了解其它编程语言一定会发现，在动态类型语言或是某些有语法糖支持的语言中，那个 `swap()` 或 `search()` 函数的泛型其实可以很简单地就实现了。

比如，你甚至可以把 `swap()` 函数简单地写成下面这个样子（包括 Go 语言也有这样的语法）：

复制代码

```
1 b, a = a, b;
```

第一件事是编程语言中的类型问题。

第二件事是对真实世界中业务代码的抽象、重用和拼装。

所以，在这篇文章中，我们还是继续深入地讨论上面这两个问题，着重讨论一下编程语言中的类型系统和泛型编程的本质。

## 类型系统

在计算机科学中，类型系统用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，以及如何操作这些类型，还有这些类型如何互相作用。类型可以确认一个值或者一组值具有特定的意义和目的。

一般来说，编程语言会有两种类型，一种是内建类型，如 `int`、`float` 和 `char` 等，一种是抽象类型，如 `struct`、`class` 和 `function` 等。抽象类型在程序运行中，可能不表示为值。类型系统在各种语言之间有非常大的不同，也许，最主要的差异存在于编译时期的语法，以及运行时期的操作实现方式。

编译器可能使用值的静态类型以最优化所需的存储区，并选取对数值运算时的最佳算法。例如，在许多 C 编译器中，“浮点数”数据类型是以 32 比特表示、与 IEEE 754 规格一致的单精度浮点数。因此，在数值运算上，C 应用了浮点数规范（浮点数加法、乘法等）。

类型的约束程度以及评估方法，影响了语言的类型。更进一步，编程语言可能就类型多态性部分，对每一个类型都对应了一个针对于这个类型的算法运算。类型理论研究类型系统，尽管实际的编程语言类型系统，起源于计算机架构的实际问题、编译器实现，以及语言设计。

程序语言的类型系统主要提供如下的功能。

**程序语言的安全性。** 使用类型可以让编译器侦测一些代码的错误。例如：可以识别出一个错误无效的表达式。如：“Hello, World” + 3 这样的不同数据类型间操作的问题。强类型语言提供更多的安全性，但是并不能保证绝对的安全。

**利于编译器的优化。** 静态类型语言的类型声明，可以让编译器明确地知道程序员的意图。因此，编译器就可以利用这一信息做很多代码优化工作。例如：如果我们指定一个类


**代码的可读性。**有类型的编程语言，可以让代码更易读和更易维护。代码的语义也更清楚，代码模块的接口（如函数）也更丰富和清楚。

**抽象化。**类型允许程序设计者对程序以较高层次的方式思考，而不是烦人的低层次实现。例如，我们使用整型或是浮点型来取代底层的字节实现，我们可以将字符串设计成一个值，而不是底层字节的数组。从高层上来说，类型可以用来定义不同模块间的交互协议，比如函数的入参类型和返回类型，从而可以让接口更有语义，而且不同的模块数据交换更为直观和易懂。

但是，正如前面说的，**类型带来的问题就是我们作用于不同类型的代码，虽然长得非常相似，但是由于类型的问题需要根据不同版本写出不同的算法，如果要做到泛型，就需要涉及比较底层的玩法。**

对此，这个世界出现了两类语言，一类是静态类型语言，如 C、C++、Java，一种是动态类型语言，如 Python、PHP、JavaScript 等。


我们来看一下，一段动态类型语言的代码：

 复制代码

```
1 x = 5;
2 x = "hello";
```

在这个示例中，我们可以看到变量 `x` 一开始好像是整型，然后又成了字符串型。如果在静态类型的语言中写出这样的代码，那么就会在编译期出错。而在动态类型的语言中，会以类型标记维持程序所有数值的“标记”，并在运算任何数值之前检查标记。所以，一个变量的类型是由运行时的解释器来动态标记的，这样就可以动态地和底层的计算机指令或内存布局对应起来。


我们再来看一个示例，对于 JavaScript 这样的动态语言来说可以定义出下面这样的数据结构（一个数组的元素可以是各式各样的类型），这在静态类型的语言中是很难做到的。

 复制代码

```
4 a[2] = {name: "Hao Chen"};
```

注：其实，这并不是一个数组，而是一个 `key:value`。因为动态语言的类型是动态的，所以，`key` 和 `value` 的类型都可以随意。比如，对于 `a` 这个数据结构，还可以写成：`a["key"] = "value"` 这样的方式。

在弱类型或是动态类型的语言中，下面代码的执行会有不确定的结果。

 复制代码

```
1 x = 5;  
2 y = "37";  
3 z = x + y;
```

有的像 Visual Basic 语言给出的结果是 42：系统将字符串 "37" 转换成数字 37，以匹配运算上的直觉。

而有的像 JavaScript 语言给出的结果是 "537"：系统将数字 5 转换成字符串 "5" 并把两者串接起来。

像 Python 这样的语言则会产生一个运行时错误。

但是，**我们需要清楚地知道，无论哪种程序语言，都避免不了一个特定的类型系统。**哪怕是可随意改变变量类型的动态类型的语言，我们在读代码的过程中也需要脑补某个变量在运行时的类型。

所以，每个语言都需要一个类型检查系统。


静态类型检查是在编译器进行语义分析时进行的。如果一个语言强制实行类型规则（即通常只允许以不丢失信息为前提的自动类型转换），那么称此处理为强类型，反之称为弱类型。

动态类型检查系统更多的是在运行时期做动态类型标记和相关检查。所以，动态类型的语言必然要给出一堆诸如：`is_array()`, `is_int()`, `is_string()` 或是 `typeof()` 这

总之，“类型”有时候是一个有用的事，有时候又是一件很讨厌的事情。因为类型是对底层内存布局的一个抽象，会让我们的代码要关注于这些非业务逻辑上的东西。而且，我们的代码需要在不同类型的数据间做处理。但是如果程序语言类型检查得过于严格，那么，我们写出来的代码就不能那么随意。

所以，对于静态类型的语言也开了些“小后门”：比如，类型转换，还有 C++、Java 运行时期的类型测试。

这些小后门也会带来相当讨厌的问题，比如下面这个 C 语言的示例。

 复制代码

```
1 int x = 5;
2 char y[] = "37";
3 char* z = x + y;
```

在上面这个例子中，结果可能和你想的完全不一样。由于 C 语言的底层特性，这个例子中的 `z` 会指向一个超过 `y` 地址 5 个字节的内存地址，相当于指向 `y` 字符串的指针之后的两个空字符处。

静态类型语言的支持者和动态类型自由形式的支持者，经常发生争执。前者主张，在编译的时候就可以较早发现错误，而且还可增进运行时期的性能。


后者主张，使用更加动态的类型系统，分析代码更为简单，减少出错机会，才能更加轻松快速地编写程序。与此相关的是，后者还主张，考虑到在类型推断的编程语言中，通常不需要手动宣告类型，这部分的额外开销也就自动降低了。

在本系列内容的前两篇文章中，我们用 C/C++ 语言来做泛型编程的示例，似乎动态类型语言能够比较好地规避类型导致需要出现多个版本代码的问题。这样可以让我们更好地关注于业务。

但是，我们需要清楚地明白，**任何语言都有类型系统**，只是动态类型语言在运行时做类型检查。动态语言的代码复杂度比较低，并可以更容易地关注业务，在某些场景下是对的，但有



比如：在 JavaScript 中，我们需要做一个变量转型的函数，可能会是下面这个样子：

 复制代码

```
1 function ToNumber(x) {  
2     switch(typeof x) {  
3         case "number": return x;  
4         case "undefined": return NaN;  
5         case "boolean": return x ? 1 : 0;  
6         case "string": return Number(x);  
7         case "object": return NaN;  
8         case "function": return NaN;  
9     }  
10 }
```

我相信，你在动态类型语言的代码中可以看到大量类似 `typeof` 这样的类型检查代码。是的，这是动态类型带来的另一个问题，就是运行时识别（这个是比较耗性能的）。

如果你用过一段时间的动态类型语言，一旦代码量比较大了，我们就会发现，代码中出现“类型问题”而引发整个程序出错的情况实在是太多太多了。而且，这样的出错会让整个程序崩溃掉，太恐怖了。这个时候，我们就很希望提前发现这些类型的问题。

静态语言的支持者会说编译器能帮我们找到这些问题，而动态语言的支持者则认为，静态语言的编译器也无法找到所有的问题，想真正提前找到问题只能通过测试来解决。其实他们都对。

## 泛型的本质

要了解泛型的本质，就需要了解类型的本质。

类型是对内存的一种抽象。不同的类型，会有不同的内存布局和内存分配的策略。

不同的类型，有不同的操作。所以，对于特定的类型，也有特定的一组操作。

所以，要做到泛型，我们需要做下面的事情。

标准化掉类型的内存分配、释放和访问。

标准化掉类型上特有的操作。需要有标准化的接口来回调不同类型的具体操作.....

所以，C++ 动用了非常繁多和复杂的技术来达到泛型编程的目标。

通过类中的构造、析构、拷贝构造，重载赋值操作符，标准化（隐藏）了类型的内存分配、释放和复制的操作。

通过重载操作符，可以标准化类型的比较等操作。

通过 `iostream`，标准化了类型的输入输出控制。

通过模板技术（包括模板的特化），来为不同的类型生成类型专属的代码。

通过迭代器来标准化数据容器的遍历操作。

通过面向对象的接口依赖（虚函数技术），来标准化了特定类型在特定算法上的操作。

通过函数式（函数对象），来标准化对于不同类型的特定操作。

通过学习 C++，我们可以看到一个比较完整的泛型编程里所涉及的编程范式，这些编程范式在其它语言中都会或多或少地体现着。比如，JDK 5 引入的泛型类型，就源自 C++ 的模板。

泛型编程于 1985 年在论文 [Generic Programming](#) 中被这样定义：

Generic programming centers around the idea of abstracting from concrete, efficient algorithms to obtain generic algorithms that can be combined with different data representations to produce a wide variety of useful software.

— Musser, David R.; Stepanov, Alexander A., Generic Programming

我理解其本质就是 —— **屏蔽掉数据和操作数据的细节，让算法更为通用，让编程者更多地关注算法的结构，而不是在算法中处理不同的数据类型。**

## 小结

数上，等等。但是，类型的产生和限制，虽然对底层代码来说是安全的，但是对于更高层次的抽象产生了些负面因素。比如在 C++ 语言里，为了同时满足静态类型和抽象，就导致了模板技术的出现，带来了语言的复杂性。

我们需要清楚地明白，编程语言本质上帮助程序员屏蔽底层机器代码的实现，而让我们可以更为关注于业务逻辑代码。但是因为，编程语言作为机器代码和业务逻辑的粘合层，是在让程序员可以控制更多底层的灵活性，还是屏蔽底层细节，让程序员可以更多地关注于业务逻辑，这是很难两全需要 trade-off 的事。

所以，不同的语言在设计上都会做相应的取舍。比如：C 语言偏向于让程序员可以控制更多的底层细节，而 Java 和 Python 则让程序员更多地关注业务功能的实现。而 C++ 则是两者都想要，导致语言在设计上非常复杂。

以下是《编程范式游记》系列文章的目录，方便你了解这一系列内容的全貌。**这一系列文章中代码量很大，很难用音频体现出来，所以没有录制音频，还望谅解。**

[编程范式游记 \( 1 \) - 起源](#)

[编程范式游记 \( 2 \) - 泛型编程](#)

[编程范式游记 \( 3 \) - 类型系统和泛型的本质](#)

[编程范式游记 \( 4 \) - 函数式编程](#)

[编程范式游记 \( 5 \) - 修饰器模式](#)

[编程范式游记 \( 6 \) - 面向对象编程](#)

[编程范式游记 \( 7 \) - 基于原型的编程范式](#)

[编程范式游记 \( 8 \) - Go 语言的委托模式](#)

[编程范式游记 \( 9 \) - 编程的本质](#)

[编程范式游记 \( 10 \) - 逻辑编程范式](#)

[编程范式游记 \( 11 \) - 程序世界里的编程范式](#)



# 左耳朵耗子

## 全年独家专栏《左耳听风》

20000 名程序员的练级攻略

陈皓

资深技术专家  
骨灰级程序员



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 31 | 编程范式游记 (2) - 泛型编程

下一篇 33 | 编程范式游记 (4) - 函数式编程

### 精选留言 (19)

 写留言



云学

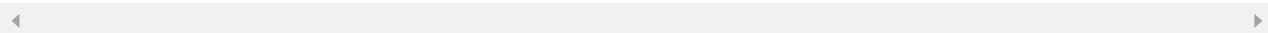
2018-01-12

 14

意犹未尽啊，比追剧还痛苦

展开 

作者回复: 哈哈



shougao

2018-01-19

 5

“类型是对内存的抽象，不同的类型会有不同的内存分布和内配策略”，见识了，用范型

**casey**

2018-01-11

👍 5

本来说就看10分钟，一不小心就从一看到三了，耗哥快更新^\_^。

这个系列除了了解不同的编程范式，最关注的是不同的编程范式如何设计，解决了什么问题。这些皓哥都点到了。我个人喜欢Scala这门语言，觉得它的设计非常优美，函数式编程和类型系统都是它的特点，希望皓哥这个系列或者答疑中能写写您的看法。

对于皓哥掌握那么多不同的编程范式，每种起源发展娓娓道来，我也是钦佩不已，不知...

展开 ▾

**saiyn**

2018-05-31

👍 2

“C 语言偏向于让程序员可以控制更多的底层细节，而 Java 和 Python 则让程序员更多地关注业务功能的实现。而 C++ 则是两者都想要，导致语言在设计上非常复杂”——多么简单而又深刻的诠释，大赞

**不记年**

2018-03-08

👍 1

请教一个问题，为何要引用百度百科的内容，而不用自己的话去讲明白？

**W\_T**

2018-02-03

👍 1

类型是对内存的一种抽象。这是我在这篇文章中最大的收获

**靠人品去赢**

2019-04-10

👍

之前看泛型这一块，什么类型系统，类型检查系统听起来模糊不理解，看了一下JS的相关文章，而且在项目中看到一个声明的变量，在由null变为function的一个操作过程中。

结合动态语言，变量类型可以转换，而静态语言，声明了变量类型少了个类型检查系统，整个语言代码可以简洁不少。

展开 ▾

**靠人品去赢**

2018-12-24



之前对强类型语言和弱类型语言，有点理解不清，现在貌似理解了。

**wessonwang**

2018-12-13



“内存操作”-类型-数据结构（容器）-算法，分层思想。

**章炎**

2018-10-16



“C++两者都想要”. 感觉这就给了Go可乘之机了。泛型做的不如Go彻底，性能和简单的C相比没有绝对优势，到现在C++14不断地扩充和简化语法，感觉到头来两边都做不好。这个和做事情一样，二鸟在林不如一鸟在手。

展开 ∨

**北风一叶**

2018-08-02



泛型编程：将类型泛化的一种编程方式

展开 ∨

**lostsquirr...**

2018-03-27



谢谢指正，是我记错了，是  $x = 5$ ,  $y = \text{"hello"}$ ,  $x + y$  会报错，随便吐槽下这app到处坑，留言不能回复，昨天在小程序听音频，不知道在哪停止。。。

展开 ∨

作者回复:

我帮你转答池建强

作者回复: 当然可以。强类型和动态类型是两码事，前者是判断不同的类型是否可以互转，后者是变量的类型可以动态调整。



**allwmh**

2018-01-17



期待后续

展开 ▾



**茎待佳阴**

2018-01-15



期待函数式编程，希望多讲讲装饰器这种东西的运用场景，以及滥用的一些例子



**JK.Ryan**

2018-01-13



最后的总结高屋建瓴，针对不同的业务场景和需求选择才是更好的方式，每个语言都有其擅长点和解决问题的方式。

展开 ▾



**RZ\_diversi...**

2018-01-13



依照皓哥的讲解，再结合我目前的情况来说，我更侧重业务逻辑，我还是首先选择Python和Java



**蔬菜饼**

2018-01-12



您怎么看待近期热门的区块链，深度学习等技术，程序员应该追热吗？



下载APP

