编程语言 编程 Rust（编程语言） Swift 语言

关注者 72 被浏览 14,711

强类型语言的优缺点是什么?

平时自己写东西的时候没觉得那么难受, 但是拿强类型语言刷算法题实在是有点惨不忍睹, 比如: if ((x as f32 - y).sqrt().floo...显示全部

关注问题 写回答 邀请回答 12 条评论 分享 举报 ···

查看全部 18 个回答

HOOCCOOH
Rustacean，C++黑，信息就是生命

14 人赞同了该回答

题主在写算法题，看起来想问为什么 Rust 在这里这么“啰嗦”而非像 C++ （猜测）那样自动隐式解决。

。

（先反对下某些其他答案， Haskell 字面量多态确实是个好东西，Rust 也有整数字面量多态了。但题主问的是“已经给定不同类型需要参与运算”的问题，这点是强类型设计上就不能解决，或者说是**刻意避免**的。）

Rust 自己的设计目标包含高效（运行时）、安全。作为一个能写操作系统的语言，需要有**底层控制能力**，数字分这么多类型是因为机器本来就有这么多类型，其计算/存储/效率都有区别。它希望你在写的时候**能够搞清楚你到底想干嘛**，需要何种精度，以何种效率执行何事。

C++ 的弱类型很容易写出 sqrt(1 * 2U + 3LL) 这种代码，这里包含机器操作：三次运算、一次整形扩展、一次有符号转无符号、一次整形转浮点。虽然看起来只有运算，但转无符号和转浮点都不是完美转换，你考虑边界情况及其可能导致的问题了吗？比如这里的乘法负数会爆炸，还可能溢出 u32，因为是先乘再转i64，你看出来了吗？（写算法题用 v.size() 就踩过这个坑）

相对的， Rust 要求你显式写出 as，让你**知道确实有这么个操作以及在哪一步执行**，这样你就自然得评估这个操作的效果、性能与风险。“安全”的一个方面是**让“跑起来和看起来不太一样”的代码过不了编译。**

另一个例子是下标必须 usize ，就要求你考虑“下标不能为负”和“内存可能超过 2G”两个问题，你考虑过了，那就写上“as usize”告诉编译器，就像安转软件时必须点上“我已知这个操作可能的风险”，不是让你无脑点，是真考虑过再点的。

因为 **Rust 觉得你容易忘事所以默认你忘考虑边界情况，而 C++ 觉得你永远无限聪明所以默认你什么都考虑了。**你写 C++ 是爽了，但你猜结果代码的安全/可靠性对比如何？（笑，单说算法题 RE 比例就挺高的

总之，这个显式转换的问题是 Rust 设计理念相信“人可能忘事，编译器应该提醒”导致的，命和所有权也是基于这个目标；**当然它确实带来了一定程序编写效率的损失，这是个鱼和熊掌选择问题。**

对于你的“麻烦”问题，我觉得你应该：整数数据永远 i32，浮点永远 f64，表示数据长度或位置永远 usize。那么你可以把 as 减到最少，**剩下的，就真的都是你必须好好考虑一下正确性的地方。**

编辑于 2019-04-18

被子飞

已赞同 14 4 条评论 分享 收藏 感谢 ··· 收起

关于作者

HOOCCOOH
Rustacean，C++黑，信息就

「已注销」、金少海、SuperSodaSea 也关注了他

回答 303 文章 7 关注者 668

关注 发私信

被收藏 1 次

待 0 人关注
Yang 创建

相关问题

强类型语言的优缺点是什么? 18 个回答

js柯里化的代码实现，js柯里化概念提出背景或说js柯里化适用场景，优缺点分别是什么，还有个反柯里化? 9 个回答

TypeScript 与 ReasonML 各有何优缺点? 7 个回答

JavaScript 的原型链继承方式与 C++, C# 等编译型语言的继承方式相比有何优缺点? 12 个回答

scala相对于python有何优缺点? 18 个回答

相关推荐


杨澜 & 彭凯平：幸福力必修
共 11 节课 ▶ 试听

怎样用 Python 进行数据分
★★★★★ 1212 人参与


深宫谍影-青春
颜灼灼
43 人读过 阅读

12 人赞同了该回答

很明显题主也是Rustacean哈哈，我和题主遇到的是一模一样的问题，在cw上做题时被整型报错搞得很烦……

Rust搞这一套很明显是对C++隐式类型转换的PTSD。C艹里比较常见的问题就是如果你写

```
auto len = sth.size();
len = calc(sth_else);
```

而恰好sth.size()返回的是usize（划掉）size_t，clac()返回的是int而且还是负数，那么len就会变成一个很大的数，可以理解为下溢。

https://blog.csdn.net/songbai_pu/article/details/9172689
blog.csdn.net
展开阅读全文

于是Rust里，各种整数的转换虽然是safe的，但必须使用as来进行，还对其行为进行了详细的规

▲ 赞同 12 ▼ 8 条评论 分享 收藏 感谢 ···

MaxwellGeng
九流游戏程序

25 人赞同了该回答

首先，对于计算机来说，只有而且只能有强类型。所有的类型都会在真正的执行开始之前被决定。如果你不决定那么就由编译器自行决定，此为前提。至于喜欢自己决定还是计算机决定全看个人爱好，没有客观对错。

我个人是反对弱类型语言的，首先有可能造成问题隐藏。举个例子，C++的Template和C#的Generic在一定程度上有相似之处，但是C++的Template更加注重灵活而非稳定，它就是弱类型的。譬如template <typename T>加在一段函数之前，使该函数变成模板函数，那么这个T可以是任何东西，底下无论怎么写都不会有文本编写期的报错，这可以理解成弱类型，也确实达到了灵活的目的，而这种灵活性会导致一些程序duck programming，在有些时候是很要命的，因为有可能写错了但是当时看不出来，然后编译的时候才出问题，而这时候发现问题可能已经晚了。但是C#的Generic则是有强类型的限制的，比如必须要靠where指定类型存在的方法，所以Generic基本不存在合法的文本与非法的编译。

第二就是可读性，可读性对于大型工程是很重要的，在看到一个函数，如果不确定它的类型，就有可能造成误解，也有可能给后续的开发留下隐患。所谓千里之堤毁于蚁穴大概如此。

从我个人的角度来讲，牺牲一定的编写快感获得后续维护的方便是非常赚的一件事，因为我永远忘不了维护自己独立开发数月甚至数年的怪物要付出多少努力。哪怕是自己独立开发，需求也可能经常要变的，可能实现一个效果要尝试多个方案，或是为了压榨一点点的性能用尽各种骚操作，在这种情况下我别无所求但求一稳，在使用Template这种灵活强大的功能时更是如此，任何可能引起后续维护的不便的用法，我都会慎之又慎。

以上。

发布于 2019-04-12

▲ 赞同 25 ▼ 27 条评论 分享 收藏 感谢 ···

查看全部 18 个回答