# 社区精选 | 手写编程语言-实现运算符重载

crossoverJie SegmentFault思否 2022-10-05 12:00 发表于山东

今天小编为大家带来的是社区作者 **crossoverJie** 的文章，让我们一起来学习手写编程语言-实现运算符重载。

## 前言

先带来日常的 GScript 更新：新增了可变参数的特性，语法如下：

```
int add(string s, int ...num){
    println(s);
    int sum = 0;
    for(int i=0;i<len(num);i++){
        int v = num[i];
        sum = sum+v;
    }
    return sum;
}
int x = add("abc", 1,2,3,4);
println(x);
assertEqual(x, 10);
```

得益于可变参数，所以新增了格式化字符串的内置函数：

```
//formats according to a format specifier and writes to standard output.
printf(string format, any ...a){}

//formats according to a format specifier and returns the resulting string.
string sprintf(string format, any ...a){}
```

下面重点看看 GScript 所支持的运算符重载是如何实现的。

## 使用

运算符重载其实也是多态的一种表现形式，我们可以重写运算符的重载函数，从而改变他们的计算规则。

```
println(100+2*2);
```

以这段代码的运算符为例，输出的结果自然是：104.

但如果我们是对两个对象进行计算呢，举个例子:

```
class Person{
    int age;
    Person(int a){
        age = a;
    }
}
Person p1 = Person(10);
Person p2 = Person(20);
Person p3 = p1+p2;
```

这样的写法在 Java/Go 中都会报编译错误，这是因为他们两者都不支持运算符重载；

但 Python/C# 是支持的，相比之下我觉得 C# 的实现方式更符合 GScript 语法，所以参考 C# 实现了以下的语法规则。

```
Person operator + (Person p1, Person p2){
    Person pp = Person(p1.age+p2.age);
    return pp;
}
Person p3 = p1+p2;
println("p3.age="+p3.age);
assertEqual(p3.age, 30);
```

有几个硬性条件：

- 函数名必须是 operator

- 名称后跟上运算符即可。

目前支持的运算符有：`+-*/ == != < <= > >=`

## 实现

以前在使用 Python 运算符重载时就有想过它是如何实现的？但没有深究，这次借着自己实现相关功能从而需要深入理解。

其中重点就为两步：

1. 编译期间：记录所有的重载函数和运算符的关系。
2. 运行期：根据当前的运算找到声明的函数，直接运行即可。

第一步的重点是扫描所有的重载函数，将重载函数与运算符存放起来，需要关注的是函数的返回值与运算符类型。

```
// OpOverload 重载符
type OpOverload struct {
    function  *Func
    tokenType int
}

// 运算符重载自定义函数
opOverloads []*symbol.OpOverload
```

在编译器中使用一个切片存放。
而在运行期中当两个入参类型相同时，则需要查找重载函数。

```
// GetOpFunction 获取运算符重载函数
// 通过返回值以及运算符号(+-*/) 匹配重载函数
func (a *AnnotatedTree) GetOpFunction(returnType symbol.Type, tokenType int) *symbol.Func {
    for _, overload := range a.opOverloads {
```

```
        isType := overload.GetFunc().GetReturnType().IsType(returnType)
        if isType && overload.GetTokenType() == tokenType {
            return overload.GetFunc()
        }
    }
    return nil
}
```

查找方式就是通过编译期存放的数据进行匹配，拿到重载函数后自动调用便实现了重载。

感兴趣的朋友可以查看相关代码：

- 编译期：
- https://github.com/crossoverJie/gscript/blob/ae729ce7d4cf39fe115121993fcd2222716755e5/resolver/type_scope_resolver.go#L127
- 运行期：
- https://github.com/crossoverJie/gscript/blob/499236af549be47ff827c6d55de1fc8e5600b9b3/visitor.go#L387

## 总结

运算符重载其实并不是一个常用的功能；因为会改变运算符的语义，比如明明是加法却在重载函数中写为减法。

这会使得代码阅读起来困难，但在某些情况下我们又非常希望语言本身能支持运算符重载。

比如在 Go 中常用的一个第三方精度库decimal.Decimal，进行运算时只能使用 d1.Add(d2) 这样的函数，当运算复杂时：

```
a5 = (a1.Add(a2).Add(a3)).Mul(a4);
```

```
a5 = (a1+a2+a3)*a4;
```

就不如下面这种直观，所以有利有弊吧，多一个选项总不是坏事。

GScript 源码：https://github.com/crossoverJie/gscript

---

点击左下角阅读原文，到 SegmentFault 思否社区 和文章作者展开更多互动和交流，“**公众号后台**“回复“ **入群** ”即可加入我们的**技术交流群**，收获更多的技术文章~

**- END -**

阅读原文

“越来越像新语言的 C++，我与它结缘、痴迷、深耕的 14 年！”

CSDN