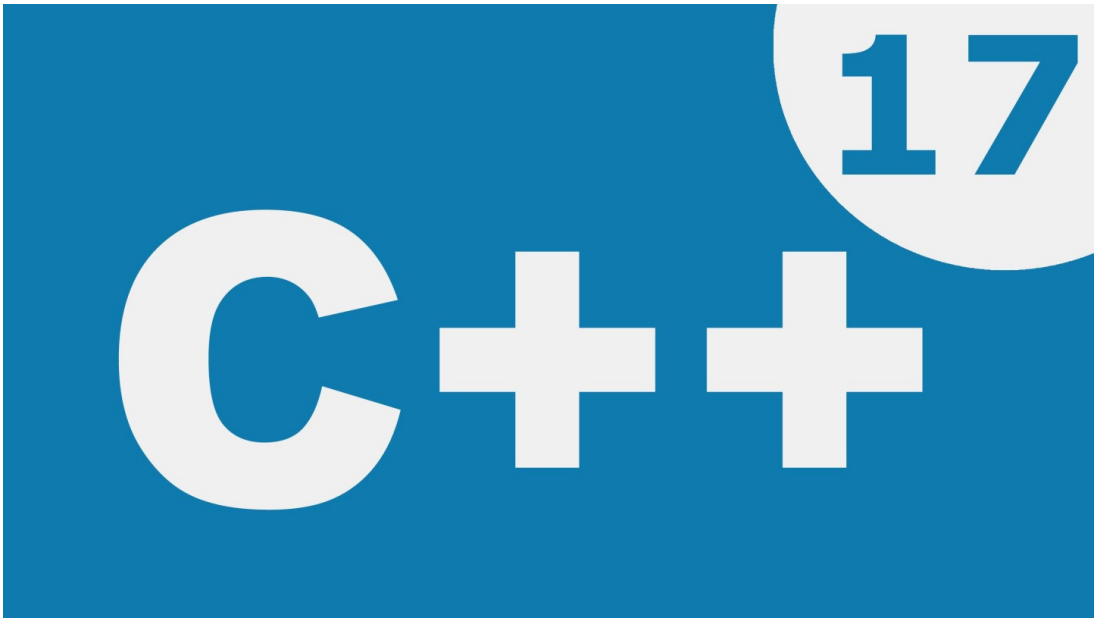

C++17 新特性之 std::optional（上）


茗一
百度 后端研发工程师
已关注

最近在学习 c++ 17 的一些新特性，为了加强记忆和理解，把这些内容作为笔记记录下来，有理解不对的地方请指正，欢迎大家留言交流。

引言

在介绍之前，我们从一个问题出发，C++ 的函数如何返回多个值?

比较有年代感的一种做法是将返回值作为引用参数传入，函数的返回值用来标识运行状态，比如像下面这样

```
#include <iostream>

using namespace std;

int func(const string& in, string& out1, string& out2) {
    if (in.size() == 0)
        return 0;
    out1 = "hello";
    out2 = "world";
    return 1;
}

int main() {
    string out1, out2;
    int status = func("hi", out1, out2);
    if (status) {
        cout << out1 << endl;
        cout << out2 << endl;
    }
    return 0;
```

这种做法性能不错，但可读性会比较差，参数列表里既包含了入参也包含了出参，常见通过变量名前缀来标识，尤其是在出入参比较多的时候，后期维护会非常头疼。

在 C++ 11 中新增了 tuple 这种数据结构的支持，自然也可以使用 tuple 来实现多个返回值

```
#include <iostream>
#include <tuple>

using namespace std;

tuple<bool, string, string> func(const string& in) {
    if (in.size() == 0)
        return make_tuple(false, "", "");
    return make_tuple(true, "hello", "world");
}

int main() {
    if (auto [status, out1, out2] = func("hi"); status) {
        cout << out1 << endl;
        cout << out2 << endl;
    }
    return 0;
}
```

上面这段代码中的 `auto [status, out1, out2] = func("hi");` 是 C++ 17 中叫 Structured Bindings 的新特性，效果就是将多个返回值按照顺序绑定到方括号中的变量名中。

tuple 在这里用起来不是很爽的地方是需要刻意的记忆每个返回值的位置，在返回值数量比较多的时候就会带来比较大的困扰，返回值的语意表达的。

还有一种做法就是将函数返回值定义成一个结构体，同时要返回函数的运行状态，我们可以考虑把这两部分数据定义成一个 pair ，pair 可以理解为一种特殊的 tuple（只有 2 个元素的 tuple）。

```
#include <iostream>

using namespace std;

struct Out {
    string out1 { "" };
    string out2 { "" };
};

pair<bool, Out> func(const string& in) {
    Out o;
    if (in.size() == 0)
        return { false, o };
    o.out1 = "hello";
    o.out2 = "world";
    return { true, o };
}

int main() {
    if (auto [status, o] = func("hi"); status) {
        cout << o.out1 << endl;
        cout << o.out2 << endl;
    }
    return 0;
```

在线玩 -> [Wandbox]三へ(へ՞ਊ ՞)へ ﾊｯﾊｯ

目前这种做法可以做到让返回值更富有语意，并且可以很方便的扩展，如果要增加一个新的返回值，只需要扩展现有的结构体就可以了。正如上文所说，在 CppCoreGuidelines 中对于多返回值更建议使用 tuple 或 struct ，这样做能让返回值的语意更加明确。

最后这种做法中的 pair<bool, Out> 这个数据结构实现的功能就跟本文要介绍 std::optional 很相似了。

std::optional

From cppreference -std::optional

> The class template `std::optional` manages an *optional* contained value, i.e. a value that may or may not be present.
> A common use case for `optional` is the return value of a function that may fail. As opposed to other approaches, such as std::pair<T,bool>, `optional` handles expensive-to-construct objects well and is more readable, as the intent is expressed explicitly.
> 类模板 `std::optional` 管理一个*可选*的容纳值，即可以存在也可以不存在的值。
> 一种常见的 `optional` 使用情况是一个可能失败的函数的返回值。与其他手段，如 std::pair<T,bool> 相比， `optional` 良好地处理构造开销高昂的对象，并更加可读，因为它显式表达意图。

std::optional 是在 C++ 17 中引入到标准库中的，C++ 17 之前的版本可以通过 boost::optional 实现几乎相同的功能。

我们来看一下使用 std::optional 来实现上面那段代码的样子

```
#include <iostream>
#include <optional>

using namespace std;

struct Out {
    string out1 { "" };
    string out2 { "" };
};

optional<Out> func(const string& in) {
    Out o;
    if (in.size() == 0)
        return nullopt;
    o.out1 = "hello";
    o.out2 = "world";
    return { o };
}

int main() {
    if (auto ret = func("hi"); ret.has_value()) {
        cout << ret->out1 << endl;
        cout << ret->out2 << endl;
    }
    return 0;
}
```

这段代码中我们看到了部分 std::optional 的用法，std::nullopt 是 C++ 17 中提供的没有值的 optional 的表达形式，等同于 { } 。

创建一个 optional 的方法：

```
// 空 optiolal
optional<int> oEmpty;
optional<float> oFloat = nullopt;

optional<int> oInt(10);
optional oIntDeduced(10);  // type deduction

// make_optional
auto oDouble = std::make_optional(3.0);
auto oComplex = make_optional<complex<double>>(3.0, 4.0);

// in_place
optional<complex<double>> o7{in_place, 3.0, 4.0};

// initializer list
optional<vector<int>> oVec(in_place, {1, 2, 3});

// 拷贝赋值
auto oIntCopy = oInt;
```

访问 optional 对象中数据的方法：

```
// 跟迭代器的使用类似，访问没有 value 的 optional 的行为是未定义的
cout << (*ret).out1 << endl;
cout << ret->out1 << endl;

// 当没有 value 时调用该方法将 throws std::bad_optional_access 异常
cout << ret.value().out1 << endl;

// 当没有 value 调用该方法时将使用传入的默认值
Out defaultVal;
cout << ret.value_or(defaultVal).out1 << endl;
```

使用 std::optional 带来的好处：

- 省去了运行状态的 bool 值的声明，让代码更简洁，更注重返回值本身的语意
- 不用担心额外的动态内存分配，这一点会在后面的文章里详细展开

总结

通过对多返回值的代码不断的重构，最后通过 std::optional 实现了一个比较满意的版本，不过在这个过程中我们还遗漏了异常处理的部分，目前的实现方式在出异常时我们只知道没有返回值，但为什么出现异常却无从得知，以及 std::optional 在内存和性能上的一些思考，还有 std::optional 其它场景下的应用介绍都放到下一篇文章里啦。

编辑于 2019-05-20

还没有人赞赏，快来当第一个赞赏的人吧！

C++17　C++　Modern C++

文章被以下专栏收录

蓝色的味道　　关注专栏

推荐阅读

C++干货系列——C++17新特性之std::optional

小天狼星不...　发表于C++干货...

C++右值引用（std::move）

niediao

C++17新特性

程序喵之前已经介绍过C++11的新特性和C++14的新特性，链接如下：xxx，今天向亲爱的读者们介绍下C++17的新特性，现在基本上各个编译器对C++17都已经提供完备的支持，建议大家编程中尝试使...

程序喵大人

C++11新特性总结（相比C++98）

jameswhale

36 条评论　　切换为时间排序

写下你的评论...

精选评论（1）

扶余城里小老二　2019-05-08

我刚在简历上面写上精通c++，然后c++20,23就出来了。

13　查看回复

评论（36）

伊斯坦布尔选帝侯　2019-05-08

*ret.out是取不出值的，.优先级比*高

赞

茗一 (作者) 回复 伊斯坦布尔选帝侯　2019-05-08

感谢指正，已修改

赞

伊斯坦布尔选帝侯 回复 茗一 (作者)　2019-05-08

emmmm....不应该是(*ret).out1吗

赞

展开其他 3 条回复

已赞同 164　36 条评论　分享　喜欢　收藏　申请转载　...

我刚在简历上面写上精通c++，然后c++20,23就出来了。

13

茗一 (作者) 回复 扶余城里小老二 2019-05-08

23333

赞

plazum01 回复 扶余城里小老二 2019-11-11

我觉得哪怕只有C++11你也很难说精通吧

2

展开其他 1 条回复

小小的开发崽 2019-05-08

期待 大佬 出Go

赞

茗一 (作者) 回复 小小的开发崽 2019-05-08

[为难] 别闹

赞

pop3imap 2019-05-08

M.....Maybe?

4

KimmyLeo 回复 pop3imap 2019-05-08

是的，Maybe

1

正规子群 回复 pop3imap 2019-05-24

看到中途我就猜有人会在评论区提 Maybe [捂嘴]

2

托老师 2019-05-08

找个和Java8的optional很像，有细节区别。C++也经常有nullptr错误吗?

赞

nnnn123456789 回复 托老师 2019-05-08

C++太经常了吧。。。

1

托老师 回复 nnnn123456789 2019-05-08

对忘了C++有指针这个东西，语言太多容易搞混。[捂脸]

赞

struct 2019-05-08

optional不能放引用

赞

茗一 (作者) 回复 struct 2019-05-09

嗯 是的

赞

放进reference_wrapper应该就行了吧

赞

展开其他 1 条回复

前方 2019-05-09

好像和shared_ptr也没多少差别

赞

王汪汪 2019-05-09

请教下将auto ret=func()放在if()里面和放在if外面有什么区别吗?

赞

zhtz 回复 王汪汪 2019-08-17

作用域不一样

赞

檀十一郎 2019-05-09

c++已经彻底放飞自我了

赞

lhrbu 2019-05-09

不知道用途啊，如果想要知道函数有没有成功，返回错误码或抛异常一九九足够了

赞

plazum01 回复 lhrbu 2019-11-11

显然不如这个优雅

赞

Khellendros 2019-05-09

感觉不算特别有用……大部分时候还是要手动判断是否为空，和直接用智能指针没什么不同呀，而且貌似没有提供map、flatmap这些函数功能太局限了

赞

学不会 回复 Khellendros 2020-01-14

元编程操作类型时比较有用

赞

张一峰 2019-05-17

用variant封装一个result是不是会更好用一点，这样还能携带错误信息回来

赞

钱乎 2019-05-27

期待(下)!

赞

Allan.class 2019-06-05

为什么感觉c++越来越像Python了？编程固然方便了许多，但总觉得失去了许多魅力。。

2

暂时是路人 2019-09-02

这玩意貌似性能不行，浪费内存

赞

请教一下，如果optional使用时发生错误，会抛出bad optional access的异常。有没有办法让此时不抛出异常，而是直接中断？目前项目中遇到不规范的代码，异常现场无法确认，只能通过code review来查找这些潜在的bug，实在是心累

赞

雾漫江南 01-09

下呢？亲

赞