设计C++函数传参时如何决定使用指针还是引用？ - 知乎

https://www.zhihu.com/question/31276547/answer/233683553

神奇先生 Mon Nov, 23 17:50

这个问题问得好。该用引用还是指针，要分情况来看。

1. 函数不是构造函数，且参数是只读：用const引用。

如果函数参数是只读的，且不是构造函数，const引用最好。原因有二：

1. 如果用指针，用户使用时可能会传nullptr作为参数。用引用的话，用户基本不可能犯这种错误。
2. const reference可以接受匿名变量作为参数。比如，

```
int say_hello(const std::string& name) {
    std::cout << 'hello ' << name << std::endl;
}

// 然后你可以这样调用
say_hello('world');
```

这里'world'被隐式构造为一个匿名的std::string变量。而const reference可以接受匿名变量。如果name被定义为指针或者普通引用都不行。

2. 函数不是构造函数，且参数是作为输出(out)参数：用指针。

这种情况用指针最好。因为用户在传指针的时候需要用到取地址操作符(&)，这样代码看起来更易懂。比如：

```
void copy1(const std::string &a, std::string *b);
void copy2(const std::string &a, std::string &b);

copy1(foo, &bar);
copy2(foo, bar);
```

这两个copy函数，很明显第一种一眼就能看出来是把foo复制到bar。如果是第二种就必须得看函数定义才能知道。

3. 函数不是构造函数，且不是只读，但也不是输出参数：用引用。

这种情况用引用比用指针好是引用可以防止不小心传nullptr这种情况。这种参数多半是类似stream这种，虽然只是“读”，但还是会产生副作用，改变变量的状态。我个人很讨厌用起来必须改变状态的类型，因为对于函数式编程和并行程序非常不友好。所以我都尽量不定义这样的类型（除了负责IO的类）。

4. 函数是构造器，但参数并不参与组成构造类的一部分，那就参考以上几点。

如果参数不参与构造的话其实跟不是构造函数的情况一样。

以上只是比较简单的情况，下面才是比较有趣的部分。也就是如果函数是构造器的话，该传什么类型的参数。

5. 函数是构造器，且参数参与构造，且参数是可移动的类型：传值

这种情况直接传值是最好的。比如：

```
class person {
  public:
    person(std::string name) : name_{std::move(name)} {}
  private:
    std::string name_;
};
```

传值比传const引用好处在于，取决于用户是否需要保留原参，可以省略复制。比如，

```
Person p{'Jack'}; // 创建一个临时变量，如果一个move；多半情况下编译器会直接创建在Person里
std::string name{'John'};
Person p1{name}; // 这种情况跟用const引用一样，都是一次copy
Person p2{std::move(name)}; // 因为被转化成了右值，name会被直接移动到p2里，省掉一次copy
```

当然，如果string是像这个例子中这么小的话，move和copy开销是一样的，因为small string optimization。但如果参数是长的string，或者类似于vector或者map这种heap allocated变量的话，move就比copy开销小非常多。

5. 函数是构造器，且参数参与构造，但构造的类只保留参数的引用：传shared_ptr或者自定义指针

首先我非常不提倡这种类型，因为如果这样定义，用户就必须保证name的生命周期比person长。但只看构造器的签名的话又没办法看出来这点。用起来非常麻烦，每次都需要查文档或者读代码才能知道用name构建了person之后能不能销毁name。比如如果你这样定义这个类：

```
class person {
public:
  person(const family_tree &family) : family_{&name} {
      family_->add(this);
  }
private:
  const family_tree *family_;
};

family_tree family{...};
person p{family}; //只看这两行代码你能知道可不可以return p吗？
```

我一般只有当类是作为函数类型用的时候才会定义这种类型。其他情况，我一般会用shared_ptr作为参数，然后类里也保存shared_ptr作为成员。比如像这样，

```
class person {
public:
 person(std::shared_ptr<family_tree> family) : family_{std::move(family)} {}
private:
 std::shared_ptr<family_tree> family_;
};
```

但这样如果family是stack上的变量且我可以保证family的生命周期比person长怎么办？其实很容易，这样就行了：

```
int main() {
    family_tree family{...};
    person p{shared_ptr<family_tree>(&family, [](family_tree *) {})};
    ...
}
```