Effective C++

第一章 让自己习惯C++

条款1：视C++为一个语言联邦

基础 C Object-Oriented C++ Template C++ STL

条款2：尽量以const，enum，inline替换#define

#define A 2在预处理器处理，也许从未被编译器看见

名称未进记号表

const int A = 2;

条款3：尽可能使用const

1.允许指定一个语义约束，即指定一个“不被改动的对象”

2.令函数返回一个常量值，往往可以降低因客户错误而造成的以外，而又不至于放弃安全性和高效性

3.const施用于成员函数，是为了确认该成员函数可作用于const对象身上

4.两个成员函数常量性不同（一个返回值有const，一个没有）可以被重载

5.const成员函数不可以更改任何non-static成员变量

6.在const和non-const成员函数中避免重复

代码重复时，使用const\_cast移除const

条款4：确定对象被使用前已被初始化

1.确保每一个构造函数都将对象的每一个成员初始化，C++规定，对象成员变量的初始化动作发生在进入构造函数本体之前

2.C++成员初始化次序 base classes早于derived classes

3.而class成员变量的初始化总是以其声明的次序被初始化，所以初值列出的成员变量，其次序应该和在class中的声明次序相同

成员函数初值列

Class A

{

int a;

A(int b):a(b) //初值列

{

本体

}

}

4.static对象： 函数内的static对象称为local static对象，其他static对象称为non-local static对象，程序结束时static对象会被自动销毁

C++对于”定义于不同编译单元内的non-local static对象”的初始化次序并无明确定义

原因：多个编译单元内的non-local static对象经由“模板隐式具现化”形成（后者自己也可能经由“模板隐式具现化”形成）

解决方法：将每个non-local static对象搬到自己的专属函数内（该对象在此函数内被声明为static） 这些函数返回一个reference指向它所含的对象。然后用户调用这些函数，而不直接指涉这些对象，换句话说，non-local static对象被local static对象替换了-------单例模式

第二章 构造 析构 赋值运算

条款5：了解C++默默编写并调用哪些函数

1.一个空类

Class A{};

编译器内部默认存在 public型的default构造函数，copy构造函数，析构函数，copy=操作符

class A

{

public:

A(){...}

A(const A&a){...}

~A(){...}

A& operator=(const A&a){...}

};

如果你不声明，却调用的话，编译器会自动声明

2.如果你声明了一个构造函数，编译器就不再为他创建默认的构造函数‘

3.C++不允许让reference改指向不同的对象

4.当类中含有reference成员 const成员和base classes 将copy assignment声明为private时，编译器不生成默认的copy assignment操作符

条款6：若不想使用编译器自动生成的函数，就该明确拒绝

1.只要将这些函数声明为private并且不予实现，即可驳回编译器自动提供的功能，，客户企图使用该函数时，编译器会阻挠他

条款7：为多态基类声明virtual析构函数

1.一个base类的指针指向一个derived的对象，调用delete销毁base的指针，肯可能derived的成分并没被销毁

解决办法： 给base class一个virtual析构函数

2.当一个类中不含virtual函数，表示它不意图被用作一个base class 所以不要把析构函数声明为virtual，会增加对象的体积

3.含有纯虚函数pure virtual的基类称为抽象类，不可以被实例化

如果你希望拥有抽象类，但是又没有纯虚函数，可以把析构函数声明为纯虚函数

条款8：别让异常逃离析构函数

1.c++并不禁止析构函数吐出异常，但不鼓励这样做，因为可能一个vector里面有10个类的对象，当vector销毁时，10个元素在析构第一个时，发出异常，其他就还没有销毁，还会再调用，当第二个再抛出异常，就有两个异常同时存在，程序若不是结束执行就是导致不明确行为

2.析构函数绝对不要吐出异常，如果一个被析构函数调用的函数可能抛出异常，析构函数应该捕捉任何异常，然后吞下他们(不传播)或结束程序

3.如果客户需要对某个操作函数运行期间抛出的异常做出反应，那么class应该提供一个普通函数（而非析构函数）执行该操作

条款9：绝不在构造和析构过程中调用virtual函数

1.在base class构造，析构期间，virtual函数就不是virtual函数，绝不会下降到derived class阶层，如果base的构造函数调用的是纯虚函数，编译器会报错，但是如果是虚函数，并且该虚函数在基类中有实现，则程序会顺利执行，只是在构造derived class时会先构造基类，调用基类的虚函数实现，为了使调用的函数实现就是derived的实现，

解决方法： 将base构造函数中调用的函数设为non-virtual，然后在derived class构造函数中，向base的构造函数传递正确的参数即可

条款10：令operator=返回一个reference to \*this

1.为了实现连锁赋值，赋值操作符必须返回一个reference指向操作符的左侧

int x,y,z

x=y=z=15;

条款11：在operator=中处理“自我赋值”

“自我赋值”发生在对象被赋值给自己时

class Wight{...};

Wight w;

w =w;

先验证thie和参数是否为同一对象

if(this == &rhs) return \*this

否则再赋值

条款12：复制对象时勿忘每一个成分

1.任何时候为derived Class 撰写copying函数时，必须很小心地也赋

复制base class成员变量，那些成员往往是private，所以无法直接访问他们，应该让derived Class的copying函数调用相应的base Class 函数

2.当编写一个copying函数时，确保

(1).复制所有local成员变量

(2).调用base class内适当的copying函数

3.不要尝试以某个copying函数实现另一个copying函数，不要用copy assignment操作符调用copy构造函数，因为这就像试图构造一个已经存在的对象，不要让copy构造函数调用copy assignment操作符，就像在一个尚未初始化的对象身上做“只对已初始化对象才有意义”的事一样，构造函数是用来初始化新对象，而assignment操作符只施行于已初始化的对象身上。

如果copy构造函数和copy assignment操作符有相近的代码，消除的做法：共同的放入第三个函数，然后让他们调用

条款13：以对象管理资源

1.在一个函数里面定义的对象，如果不能执行delete，就算函数退出，该对象的内存也不会被释放

2.使用auto\_ptr“类指针对象”，就是所谓的智能指针，其析构函数自动对其所指对象调用delete

void f()

{

std::auto\_ptr<A> a(createA()); //a指向createA的返回

}

注意 别让多个auto\_ptr同时指向同一个对象

1).受auto\_ptr管理的资源必须绝对没有一个以上的auto\_ptr同时指向它。

std::auto\_ptr<A> b(a); //则b指向对象，a被设为null

复制动作会使被复制物指向null

2).替代方案：引用计数型智慧指针RCSP（shared\_ptr）

持续追踪共有多少对象指向某笔资源，并在无人指向它时自动删除该资源

但是无法打破环状引用（两个其实已经没被使用的对象彼此互指，因而好像还处在“被使用”状态）

std::share\_ptr<A> a(createA());

std::share\_ptr<A> b(a); //a,b指向同一对象

3).注意：auto\_ptr和share\_ptr两者都在其析构函数内做delete而不是delete[]，所以不要使用在array身上 boost::scoped\_array和boost::shared\_array classes可以

3.已对象管理资源的关键

1).获得资源后立刻放进管理对象内

2).管理对象运用析构函数确保资源被释放

条款14：在资源管理类中小心coping行为

1.复制底部资源：针对一份资源拥有其任意数量的复本，而“资源管理类”就是当不再需要某个复件时确保他释放

2.转移底部资源的拥有权：auto\_ptr的复制

条款15：在资源管理类中提供对原始资源的访问

1.当一个函数的参数是一个对象指针时，而传入的参数是一个对象，则会编译报错，此时要进行转换

1).share\_ptr和auto\_ptr都提供一个get成员函数，用来执行显示转换，即返回智能指针内部的原始指针（的复件）

void fun(A \*a);

std::share\_ptr<A> a(createA());

fun(a.get());

2).share\_ptr和auto\_ptr也重载了指针取值操作符(operator->和operator\*),它们允许隐式转换至底部原始指针

class A{

public:

bool isa() const;

}

可以使用 a->isa()和(\*a).isa()来访问资源

2.对原始资源的访问可能经由显式转化或隐式转换，一般而言显示转换比较安全，但是，隐式转换对客户比较方便

条款16：成对使用new和delete时要采用相同形式

如果在new表达式中使用[],则必须在相应的delete表达式中也使用[]

如果在new表达式中不使用[],则必须不要在相应的delete表达式中也使用[]

delete 删除一个对象

delete[] 删除一个有对象组成的数组

条款17：以独立语句将newed对象置入智能指针

以独立语句将newed对象存储于智能指针内，即一个函数参数是智能指针时不要在传参时直接写入share\_ptr<A> a (new A),而是先 shart\_ptr<A> a (new A),再把a传入，因为，如果不这样，一旦异常抛出，有可能导致new的对象没被智能指针接收，资源泄漏

第四章

条款18：让接口容易被正确使用，不易被误会

条款19：设计class犹如设计type

设计规范：

1).新type的对象应该如何创建和销毁？ 构造函数 析构函数

2).对象的初始化和对象的赋值该有怎样的差别？ 构造函数和赋值

3).以值传递 copy构造函数

4).type的“合法值”

5).继承

6).新的type需要怎样的转换？类型转换

7).操作符、函数

8).成员public private

9).未声明接口

10).type一般化

11).是否需要一个新的type

条款20：宁以pass-by-reference-to-const替换pass-by-value

1.缺省情况下C++以值传递方式传递对象给函数，会调用拷贝构造函数 是一个副本耗时，而且值传递，参数是什么类型，不管实参是什么类型，都是形参确定的类型，会出现切割问题

2.内置类型以及STL的迭代器和函数对象，往往pass-by-value比较适当

条款21： 必须返回对象时，别妄想返回其reference

绝不要返回pointer或reference指向一个local stack对象，因为该对象在调用函数返回时，就已经被释放了 ，不要返回reference指向一个heap-allocated对象，或返回pointer或reference指向一个local static对象而有可能同时需要多个这样的对象

条款22:将成员变量声明为private

1.切记将成员变量声明为private，这可赋予客户访问数据的一致性、可细微划分访问控制、允诺约束条件获得保证，并提供class作者以充分的事项弹性

2.protected并不比public更具有封装性

条款23：宁以non-member non-friend替换member函数

宁可拿non-member non-friend函数替换member函数，这样做可以增加封装性、包裹弹性和机能扩充性

条款24：若所有参数皆需类型转换，请为此采用non-member函数

只有当参数被列为参数数列内，这个参数才是隐式类型转换的合格参与者。调用成员函数的对象不可以隐式转换

条款25：考虑写出一个不抛出异常的swap函数

1.当std::swap对你的类型效率不高时，提供一个swap成员函数，并确定这个函数不抛出异常

2.如果你提供一个member swap 也该提供一个non-member swap用来调用前者，对于classes(而非template),也请特化std::swap

3.为“用户定义类型”进行std template 全特化是好的，但是千万不要尝试在std内加入某些对std而言的新东西。

第五章 实现

条款26：尽可能延后变量定义式的出现时间

尽可能延后变量定义式的出现，这样做可增加程序的清晰度并改善程序效率

条款27：尽量少做转型动作

1. C++的四种新式转型

const\_cast: 通常被用来将对象的常量性转除，也是唯一有此能力的C++style转型操作符

dynamic\_cast: 主要用来执行“安全向下转型”，也就是用来决定某对象是否归属继承体系中的某个类型，他是唯一无法由旧式语法执行的动作，也是唯一可能耗费重大运行成本的转型动作

reinterpret\_cast:意图执行低级转型，实际动作(及结果)可能取决于编译器，用于不同类型的转换，如将int指针转换为int

static\_cast:强迫隐式转换，同类之间如将non-const对象转化为const对象 int到double

2.如果可以，尽量避免转型，特别是在注重效率的代码中避免dynamic\_casts。如果有个设计需要转型动作，试着发展无需转型的替代设计

dynamic\_casts用于继承，当base中的函数不是virtual时，而用base的指针指向，derived需要调用自己的，就，而不是base的 就可以用转型，

3.如果转型是必要的，试着将他隐藏于某个函数背后，客户随后可以调用该函数，而不需要将转型放进他们自己的代码中

4.宁可使用C++的新式转型，不要使用旧式转型，前者很容易辨识出来，而且分门别类

条款28：避免返回handles指向对象内部成分

避免返回handle（包括reference、指针、迭代器）指向对象内部，遵守这个条款可以增加封装性，帮助const成员函数的行为像个const

条款29：为“异常安全”而努力是值得的

1.异常安全函数即使发生异常也不会泄漏资源或允许任何数据结构败坏。这样的函数区分为三种可能的保证：基本型、强烈型、不抛异常型

2.“强烈保证”往往能够以copy-and-swap实现出来，但“强烈保证”并非对所有函数都可实现或具备现实意义

1. 函数提供的“异常保证”通常最高只等于其调用之各个函数的“异常安全保证”中的最弱者

条款30：透彻了解inlining的里里外外

1. inline函数对编译器而言必须是可见的，以便它能够在调用点内展开该函数。

2.绝大多数inlinging限制在小型、被频繁调用的函数身上，这可使日后的调试过程和二进制升级更容易，也可使潜在的代码膨胀问题最小化，使程序的速度提升机会最大化

3.不要只因为function templates出现在头文件，就将他们声明为inline

4.定义在类声明之中的成员函数将自动地成为内联函数

5.内联能提高函数的执行效率，为什么不把所有的函数都定义成内联函数？

如果所有的函数都是内联函数，还用得着“内联”这个关键字吗？

内联是以代码膨胀（复制）为代价，仅仅省去了函数调用的开销，从而提高函数的执行效率。如果执行函数体内代码的时间，相比于函数调用的开销较大，那么效率的收获会很少。另一方面，每一处内联函数的调用都要复制代码，将使程序的总代码量增大，消耗更多的内存空间。以下情况不宜使用内联：

（1）如果函数体内的代码比较长，使用内联将导致内存消耗代价较高。

（2）如果函数体内出现循环，那么执行函数体内代码的时间要比函数调用的开销大。

类的构造函数和析构函数容易让人误解成使用内联更有效。要当心构造函数和析构

函数可能会隐藏一些行为，如“偷偷地”执行了基类或成员对象的构造函数和析构函数。

条款31：将文件间的编译依存关系降至最低

1. 支持“编译依存性最小化”的一般构想是：相依于声明式，不要相依于定义式
2. 程序库头文件应该以“完全且仅有声明式”的形式存在，这种做法不论是否涉及template都适用

第六章

条款32：确定你的public继承塑模出is-a关系

“public继承”意味着is-a 适用于base classes身上的每一件事情一定也适用于derived classes身上，因为每一个derived class对象也都是一个base class对象，反之，则不成立

条款33：避免遮掩继承而来的名称

1.内层作用域的名称会遮掩外围作用域的名称

2.名字掩盖规则：即使base classes和derived classes内的函数有不同的参数类型也适用，而且不论函数是virtual或non-virtual一体适用

3.为了让被掩盖的名称再见天日，可使用using声明式或转交函数（在derived函数中加base作用域调用base函数）

条款34：区分接口继承和实现继承

1.接口继承是只继承成员函数的接口（也就是声明），实现继承，就是继承成员函数的定义，实现，在public继承之下，derived classes总是继承base class的接口

2.声明一个pure virtual函数的目的是为了让derived classes只继承函数接口

声明简朴的（非纯）virtual函数的目的是为了让derived classes继承该函数的接口和缺省实现

声明non-virtual函数的目的是为了令derived classes继承函数的接口及一份强制性实现

条款35：考虑virtual函数以外的其他选择

1.使用non-virtual interface手法，那是template method设计模式的一种特殊形式，它以public non-virtua;成员函数包裹较低访问性（private或protected）的virtual函数，这样derived调用base的函数，函数中调用的virtual函数会使derived的

2.将virtual函数替换为“函数指针成员变量”，这是strategy设计模式的某种形式

3.以trl::function成员变量替换virtual函数，因而允许使用任何可调用物搭配一个兼容于需求的签名式，这也是strategy设计模式的某种形式

4.将继承体系内的virtual函数替换为另一个继承体系内的virtual函数，这是strategy设计模式的传统实现手法

条款36：绝不重新定义继承而来的non-virtual函数

1.non-virtual函数时静态绑定的意思是 一个声明为base的指针，调用的函数永远是base的函数

2.virtual函数是动态绑定的函数调用真正的对象的函数

条款37：绝不重新定义继承而来的缺省参数值

1. virtual函数是动态绑定的，而缺省的参数值却是静态绑定的，意思是：可能会调用一个定义于derived class内的virtual函数的同时使用base class为它指定的缺省参数值

条款38：通过复合塑模出has-a或“根据某物实现出”

1.复合：在某种类型的对象内含它种类型的对象

2.在应用域，复合意味着has-a（有一个），在实现域，复合意味着is-implemented-in-terms-of(根据某物实现出) 为人工制品

条款39：明智而审慎地使用private继承

1.private继承，编译器不会自动的将一个derived class对象转换为一个base class对象，由private base class继承而来的所有成员，在derived Class中都会变成private属性，纵使它们在base class中原本是protected或public属性

2.private继承意味着is-implemented-in-terms-of(根据某物实现出) ，他通常比复合的级别低，但是当derived Class需要访问protected base class的成员，或需要重新定义继承而来的virtual函数时，这么设计是合理的

3.和复合不同，private继承可以造成empty base最优化，这对致力于“对象尺寸最小化”的程序库开发者而言，可能很重要

条款40：明智而审慎地使用多重继承

1.多重继承要避免歧义

2.virtual继承会增加大小，速度，初始化（及赋值）复杂度等等成本，乳沟virtual base classes不带任何数据，僵尸最具有实用价值的情况

3多重继承的确有正当用途，其中一个情节涉及“public继承某个interface class”和“private继承某个协助实现的Class”的两种组合

第七章

条款41：了解隐式接口和编译期多态

1.class和templates都支持接口和多态

2.对classes而言接口是显示的，以函数签名（size normalize swap）为中心，多态则是通过vittual函数发生于运行期

3.对template参数而言，接口是隐式地，奠基于有效表达式，多态则是通过template具现代化和函数重载解析

4.重载和模板是编译器多态

条款42：了解typename的双重意义

1.声明template参数时，前缀关键字class和typename可以互换

2.请使用关键字typename标识嵌套从属类型名称，但不得在base class lists（基类列）或member initialization list（成员初值列）内以他作为base class 修饰符

条款43：学习处理模板化基类内的名称

可在derived class template内通过“this->”指涉base Class templates内的成员名称，或藉由一个明白写出的“base class资格修饰符”

条款44：将与参数无关的代码抽离templates

1.templates生成多个classes和多个函数，所以任何template代码都不该与某个造成膨胀的template参数产生依赖关系

2.因非类型模板参数而造成的代码膨胀，往往可以消除，做法使用以函数参数或class成员变量替换template参数

3.因类型参数而造成的代码膨胀，往往可降低，做法是让带有完全相同二进制表述的具体类型共享实现码

条款45：运用成员函数模板接受所有兼容类型

1.请使用member function template（成员函数模板）生成“可接受所有兼容类型”的函数

2.如果你声明member template用于“泛化assignment操作”，你还是需要正常的copy函数和copy assignment操作符

条款46：需要类型转换时请为模板定义非成员函数

条款47：请使用traits classes表现类型信息

条款48：认识template元编程

模板元编程是编写template-based C++程序执行于编译期的过程

第八章

条款49：了解new-handler行为

1.当operator new无法满足某一内存分配需求时，他会抛出异常。它会先调用一个客户指定的错误处理函数，一个所谓的new-handler为了指定这个“用以处理内存不足”的函数，客户必须调用set\_new\_handler，那是声明于<new>的标准程序库函数

2.设计一个new-handler函数要做的是

1).让更多内存可被使用

2).安排另一个new-handler，这个无法取得更多内存时使用

3).卸除new-handler 一旦没有安装new-handler ，operator new会在内存分配不成功，抛出异常

4).抛出bad\_alloc异常，这样的异常不会被operator new捕获，因此会被传播到内存索求处

5).不返回，通常调用abort或exit

条款50： 了解new和delete的合理替换时机

1.替换operator new和operator delete的理由

1).用来检测运行上的错误：内存泄漏问题

2).为了强化效能

3).为了收集使用上的统计数据

条款51：编写new和delete时需固守常规

1.operator new应该内含一个无穷循环，并在其中尝试分配内存，如果它无法满足内存需求，就应该调用new-handler。它也应该有能力申请0bytes申请。class专属版本则还应该处理“比正确大小更大的（错误）申请”

2.operator delete 应该在收到null指针时不做任何事，calss专属版本则还应该处理“比正确大小更大的（错误）申请”

条款52：写了placement new也要写placement delete

1.当你写一个placement operator new ，请确定也写出了对应的placement operator delete 如果没有这样做，你的程序可能会发生隐微而时断时续的内存泄漏

2.当你声明placement new 和placement delete 请确定不要无意识地 遮掩了他们的正常版本

第九章

条款53：不要轻忽编译器的警告

条款54：让自己熟悉包括TR1在内的标准程序库

1.tr1::shared\_ptr和tr1::weak\_ptr

tr1::shared\_ptr引用计数 智能指针，在非环形数据结构中防止资源泄漏，但是如果两个或多个对象内含tr1::shared\_ptrs并形成环状，这个环状会造成每个引用对象的引用次数超过0---即使指向这个环形的所有指针都已被销毁

tr1::weak\_ptrs的设计“非环形tr1::shared\_ptr-based数据结构”中的环形感生指针，不参与引用计数的计算，当最后一个指向某对象的tr1::shared\_ptr被销毁，纵使还有个tr1::weak\_ptrs继续指向同一对象，该对象依旧会被删除，这种情况下的tr1::weak\_ptrs会被自动视为无效

2.tr1::function 表示任何函数或函数对象，只要其签名符合目标

void registerCallback(std::tr1::function<std::string<int> func); 参数func接受任何可调用物，只要该“可调用物”的签名与“std::string(int)一致”

3.tr1::bind 能够做STL绑定器

4.其他TR1组件

1).Hash tables 实现sets multisets maps multi-maps

2).正则表达式

3).Tuples（变量组），可持有任意个数的对象

4).tr1::array STL化的数组，只是大小固定，并不使用动态内存

5).tr1::mem\_fn 语句构造上和成员函数指针一致

6).tr1::reference\_wrapper 可以造成容器“犹如持有references”

7).随机数生成工具

8).数字特殊工具

9).C99兼容扩充

10).Type traits 提供类型的编译期信息，是否是内置类型，是否提供virtual构造函数，是否是个empty class 可隐式转换为其他类型吗

11).tr1::result\_of是个template，用来推导函数调用的返回类型

条款55：熟悉boost

1.字符串与文本处理:具备类型安全的printf-like格式化动作，正则表达式，以及语汇单元切割和解析

2.容器：接口与STL相似且大小固定的数组，大小可变的bitsets以及多维数组

3.函数对象和高级编程

4.泛型编程

5.模板元编程

6.数字和数值

7.正确性与测试

8.数据结构

9.语言件的支持：包括允许C++和python之间的无缝互操作性

10.内存：智能指针

11.杂项: CRC校验，日期和时间的处理，在文件系统上来回移动