1. 让自己习惯C++

\*Item 1: 视C++为一个语言联邦

+子语言：C,object-oriented, template , stl

\*Item 2: 尽量以const, enum, inline 替换 #define

+

\*Item 3: 尽可能使用const

+

class Rational { ... };

const Rational operator\*(const Rational& lhs, const Rational& rhs);

避免如下错误

Rational a, b, c;

...

(a \* b) = c;

\*Item 4: 确定对象被使用前已先被初始化

+class的成员变量总是以其声明次序被初始化。 和构造函数初始化列表顺序无关。

+定义于不同变异单元内的non-local static对象 的初始化相对次序并无明确定义。为了解决这个问题使用Singleton模式。

1. 构造/析构/赋值运算

\*Item 5: 了解C++默默编写并调用哪些函数

\*Item 6: 若不想使用编译器自动生成的函数，就该明确拒绝

+Uncopyable来拒绝拷贝

\*Item 7: 为多态基类声明virtual析构函数

+任何class只要带有virtual函数都几乎确定应该也有一个virtual析构函数。

+希望成为抽象的那个class声明一个pure virtual析构函数。

class Base

{

public:

virtual ~Base() = 0;

};

\*Item 8: 别让异常逃离析构函数

+析构函数不要调用可能出现异常的函数

~DBConn()

{

db.close();

}

应该在析构函数前手动调用 db.close() 避免在析构函数里调用而引起异常。

\*Item 9: 绝不在构造和析构过程中调用virtual函数

+

\*Item 10: 令operator=返回一个\*this

+

\*Item 11 : 在operator=中处理“自我赋值”

+两种实现方式，第一种要注意new的时候引起异常导致的问题。

Widget& Widget::operator=(const Widget& rhs)

{

Bitmap\* pOrig = m\_pb;

m\_pb = new Bitmap(\*rhs.pb);

delete pOrig;

return \*this;

}

Widget& Widget::operator=(Widget rhs)

{

swap(rhs);

return \*this;

}

\*Item 12: 复制对象时勿忘其每一个成分

+复制所有local成员变量；

+调用所有 base class内的适当的copying函数。

+不要尝试以某个copying函数实现另一个copying函数。应该定义一个private的init()函数来安全消除copy构造函数和copy assignment操作符之间的代码重复。

1. 资源管理

\*Item 13: 以对象管理资源

+为防止资源泄露，请使用RAII对象，在构造函数获得资源并在析构函数中释放资源。

\*Item 14: 在资源管理类中小心copying行为

+

\*Item 15: 在资源管理类中提供对原始资源的访问

+

\*Item 16: 成对使用new 和 delete时要采取相同形式

+

\*Item 17: 以独立语句将newed对象置入智能指针

+

1. 设计与声明

\*Item 18: 让接口容易被正确使用，不易被误用

+

\*Item 19:设计class犹如设计type

+

\*Item 20: 宁以pass-by-reference-to-const 替换 pass-by-value

+bool validateStudent(const Person& s);

+也可以避免 slicing

+内置类型和STL的迭代器和函数对象可以用 pass-by-value

\*Item 21: 必须返回对象时，别妄想返回其reference

+

\*Item 22:将成员变量声明为private

+

\*Item 23: 宁以non-member, non-friend 替换 member函数

+namespace WebBrowserStuff

{

class WebBrowser { ... };

void clearBrowser(WebBrowser& wb);

}

\*Item 24: 若所有参数皆需类型转换，请为此采用non-member函数

+

\*Item 25: 考虑写出一个不抛异常的swap函数

+

1. 实现

\*Item 26: 尽可能延后变量定义式的出现时间

+

\*Item 27: 尽量少做转型

+注重效率的代码中避免 dynamic\_cast 转换。

\*Item 28: 避免返回handles 指向对象内部部分

+

\*Item 29: 为“异常安全”而努力是值得的

+copy-and-swap 策略

+修改对象数据的副本，然后在一个不抛异常的函数中将修改后的数据和原件置换。

\*Item 30: 透彻理解inlining的里里外外

+将大多数inlining限制在小型，被频繁调用的函数身上。

+不要只因为function templates出现在头文件，就将它们声明为inline。

\*Item 31: 将文件间的编译依存关系降至最低

+通过pImpl方法来降低依赖 Person类包含 PersonImpl数据类来访问数据

+abstract base class 来实现 Person基类是纯虚函数

1. 继承与面向对象设计

\*Item 32: 确定你的public继承塑造出is-a 关系

+

\*Item 33: 避免遮掩继承而来的名称

+已在C++11通过override关键字解决此问题

\*Item 34: 区分接口继承和实现继承

+纯虚函数，虚函数，普通成员函数的解释

\*Item 35: 考虑virtual函数以外的其他选择

+NVI: non-virtual-interface; 它是所谓 Template Method 设计模式。

+通过strategy模式抽象出某个函数的继承关系。

\*Item 36: 绝不重新定义继承而来的non-virtual函数

+

\*Item 37: 绝不重新定义继承而来的缺省参数值

+

\*Item 38: 通过复合塑模出has-a 或“根据某物实现出”

+

\*Item 39: 明智而审慎地使用private继承

+

\*Item 40: 明智而审慎地使用多重继承

+virtual 继承比较慢

+如果必须使用virtual base class，尽可能避免在其中放置数据

1. 模板与泛型编程（模板不熟悉，需要重新复习模板）

\*Item 41: 了解隐形接口和编译器多态

+某个模板类调用很多T类型的函数， 则T类型必须实现那些函数。

\*Item 42: 了解typename的双重意义

+

\*Item 43: 学习处理模板化基类内的名称

+

\*Item 44: 将与参数无关的代码抽离

+

\*Item 45: 运用成员函数模板接受所有兼容类型

+

\*Item 46: 需要类型转换时请为模板定义非成员函数

+

\*Item 47: 请使用traits classes 表现类型信息

+

\*Item 48: 认识template 元编程

+对程序库开发人员比较有用。

1. 定制new和delete

\*Item 49: 了解new-handler的行为

+std::set\_new\_handler(outOfMem);

\*Item 50: 了解new和delete的合理替换时机

+

\*Item 51: 编写new和delete时需固守常规

+

\*Item 52: 写了placement new 也要写 placement delete

+

1. 杂项讨论

\*Item 53: 不要轻忽编译器的警告

+

\*Item 54: 让自己熟悉包括TR1在内的标准程序库

+