# 1 C++库基本认识

## 标准的C++库和STL库

C++标准库是类库和函数的集合，其使用核心语言写成，由C++标准委员会制定，并不断的维护和更新的库，C++的标准库文件总共在50个标准头文件中定义，STL是C++标准库的一个子集。在C++的开发中尽量的使用标准库的好处：成本低、质量高、效率高、良好的编程风格。标准库分为10类：

1、标准库中与语言支持功能相关的头文件

（1）<cstddef>：定义宏NULL和Offsetof，以及其他标准类型的size\_t和ptrdiff

（2）<limits>：提供与基本数据类型相关的定义。例如，对于每个数值数据类型，它定义了可以表示出来的最大值和最小值以及二进制数字的位数。

（3）<climits>：提供了与基本整数数据类型相关的C样式定义。这些信息的c++样式定义在<limits>中

（4）<cfloat>：提供与基本浮点型数据相关的C样式定义。这些c++的样式定义在<limits>中

（5）<cstdlib>：提供支持程序启动和终止的宏和函数。这个头文件还声明了许多其他杂项函数，例如搜索和排序函数，从字符串转换成数值的函数， 他与对应的标准c头文件stdlib.H不同，定义了abort(VOID).abort函数还有额外的功能，它不为静态或者自动对象调用析构函数，也不调用 atexit()函数，它还定义了exit()函数的额外功能，可以释放静态对象，以注册的逆序调用用atexit注册的函数。清楚并关闭所打开的c流，把 控制权返回给主机环境

（6）<new>：支持动态内存分配

（7）<typeinfo>：支持变量在运行期间的类型标识

（8）<exception>：支持异常处理，这是处理程序中可能发生的错误的一种方式

（9）<cstdarg>：支持接收数量可变的参数的函数，记载调用函数时，可以给函数传送数量不等的数据项，他定义了宏va\_arg,va\_end,va\_start,以及va\_list类型

（10）<csetjmp>：为C样式的非本地跳跃提供函数。这些函数在c++中不常用

（11）<csignal>：为中断处理体从c样式支持

2、支持流输入输出的头文件

（1）<iostream>：支持标准流cin、cout、cerr和clog的输入和输出，并且支持多字节的输入输出

（2）<iomanip>：提供操纵程序，允许改变流的状态，从而改变输出的格式

（3）<ios>：定义iostream的基类

（4）<istream>：管理输出流缓存区的输出定义模板类

（5）<ostream>：为管理输出流缓存区的输出定义模板类

（6）<sstream>：支持字符串的流输入输出

（7）<fstream>：支持文件流输入输出

（8）<iosfwd>：为输入输出对象提供 向前的声明

（9）<streambuf>：支持流输入和输出的缓存

（10）<cstdio>：为标准流提供C样式的输入输出

（11）<cwchar>：支持多字节字符的输入输出

3、与诊断功能相关的头文件

（1）<stdexcept>：定义标准异常。异常时处理错误的方式

（2）<cassert>：定义断言宏，用于检查运行期间的情形

（3）<cerrno>：支持c样式的错误信息

4、定义工具函数的头文件

（1）<utility>：定义重载的关系运算符，简化关系运算符的写入，它还定义了pair类型

（2）<functional>：定义了许多函数对象类型和支持函数对象的功能，函数对象是支持operator（）函数调用运算符的任意对象

（3）<memory>：给容器、管理内存的函数和auto\_ptr模板类定义标准内存分配器

（4）<ctime>：支持系统时钟函数

5、支持字符串处理的头文件

（1）<string>：为字符串类型提供支持和定义

（2）<cctype>：单字节字符类别

（3）<cwctype>：多字节字符类别

（4）<cstring>：为处理非空字节序列和内存块提供函数，这不同于对用的标准的C库头文件， 几个C样式字符串的一般c库函数被返回值为const和非const的函数对替代了

（5）<cwchar>：为处理执行I/O和转换多字节序列提供函数，这不同于对应的标准c库头文件

（6）<cstdlib>：为把单字节字符串转换为数值、在多字节字符和多字节字符串之间提供函数

6、定义容器类的模板的头文件

（1）<vertor>：定义vector序列模板，这是一个大小可以重新设置的数组类型，比普通的数组更安全，更灵活

（2）<list>：定义list序列模板，这是一个序列的链表，常常在任意位置插入和删除元素

（3）<deque>：定义deque序列模板，支持在开始和结尾的高效插入和删除操作

（4）<queue>：为队列（先进先出）数据结构定义序列适配器queue和priority\_queue

（5）<stack>：为堆栈（后进先出）数据结构定义序列适配器stack

（6）<map>： map是一个关联容器类型，允许根据键值是唯一的，且按照升序存储。

（7）<set>：set是一个关联容器类型，用于以升序方式存储唯一值。

（8）<bitset>:为固定长度的位序列定义bitset模板，它可以看做固定长度紧凑型bool数组

7、支持迭代器的头文件

（1）<iterator>：给迭代器提供定义和支持

8、有关算法的头文件

（1）<algorithm>：提供一组给予算法的函数，包括置换、排序、合并和搜索

（2）<cstdlib>：声明C标准库函数bsearch（）和qsort（），进行搜索和排序

（3）<ciso646>：允许在代码中使用and代替&&

9、有关数值操作的头文件

（1）<complex>：支持复杂数值的定义和操作

（2）<valarray>：支持数值矢量的操作

（3）<numeric>：在数值序列上定义一组一般数学操作

（4）<cmath>：这是c数学库，其中还附加了重载函数，以支持c++约定

（5）<cstdlib>：提供的函数可以提取整数的绝对值，对整数进行取余数操作

10、有关本地化的头文件

（1）<locale>：提供的本地化包含字符类别，排序序列以及货币和日期表示

（2）<clocale>：对本地化提供c样式支持

## BOOST

BOOST是一个准标准库，相当于STL的延续和扩充，它的设计理念和STL比较接近，都是利用泛型让复用达到最大化。不过对比STL，BOOST更加实用。STL集中在算法部分，而BOOST包含了不少工具类，可以完成比较具体的工作。

BOOST主要包含一下几个大类：字符串及文本处理、容器、迭代子(Iterator)、算法、函数对象和高阶编程、泛型编程、模板元编程、预处理元编程、并发编程、数学相关、纠错和测试、数据结构、输入/输出、跨语言支持、内存相关、语法分析、杂项。有一些库是跨类别包含的，就是既属于这个类别又属于那个类别。下面对一些常用的类进行简单介绍：

1. 字符串及文本处理

（1）Regex：正则表达式是解决大量模式匹配问题的基础。它们常用于处理大的字符串，子串模糊查找，按某种格式tokenize字符串，或者是基于某种规则修改字符串。

（2）Spirit：是一个多用途的、递归的语法分析器生成框架。有了它，你可以创建命令行分析器，甚至是语言预处理器。它允许程序员直接在C++代码里使用(近似于)EBNF的语法来指定语法规则。

（3）String\_algo：这是一组与字符串相关的算法。包括很多有用的算法，用于大小写转换，空格清除，字符串分割，查找及替换，等等。这组算法是目前C++标准库里已有功能的扩展。

（4）Tokenizer：这个库提供了把字符序列分割成记号(token)的方法。通用的语法分析任务包括了在已分割的文本流里查找数据。如果可以把字符序列视为多个元素的容器将很有帮助，容器中的元素被执照用户定义的规则所分割。语法分析就成为了在这些元素上进行操作的单个任务，Tokenizer正好提供了这种功能。用户可以决定字符序列如何被分割，在用户请求新的元素时，库将找出相应的记号。

1. 数据结构, 容器, 迭代器, 和算法

（1）Any:支持类型安全地存储和获取任意类型的值。通常用于把不同类型的东西存储到标准库的容器中。

（2）Array：包装了普通的C风格数组，给他们增加了一些来自标准库容器的函数和typedef。其结果是可以把普通的数组视为标准库的容器。

（3）Pair：这个库包含一个参数化得类型，compressed\_pair，它比较像标准库中的std：：pair。与std：：pair的不同之处是boost：：compressed\_pair对参数模板进行评估，看其中有没有空的参数，如果有使用空类优化技术来压缩pair的大小。

（4）Dynamic\_bitset：与std：：bitset比较像，除了std：：bitset是用参数来指定位数（即容器的大小），而boost：：dynamic\_bitset则支持运行期指定大小。dynamic\_bitset支持与std：：bitset一样的接口，还增加了支持运行期特定的函数和一些std：：bitset中没有的功能。

（5）Graph：处理图结构的库，它的设计受到STL的重要影响。它是泛型的，高度可配置并且包括多个不同的数据结构：邻接链表、邻接矩阵和边列表。Graph还提供了大量的图算法，如Dijsktra最短路径算法，Kruskal最小生成树算法，拓扑逻辑排序等。

（6）Iterator：提供一个创建新的迭代器类型的框架，还提供许多有用的迭代器适配器，比C++标准中定义的更多。

（7）MultiArry：提供了一个多维容器，它很想标准库的容器，但比向量的向量更有效、更高效、更直接。容器的维数在声明时指定，但它支持限制（slicing）和映身（projecting）不同的视图（view），也可以在运行期改变维数。

（8）Multi-index：为底层的容器提供多个索引。这以为着一个底层的容器可以有不同的排序方法和不同的访问语义。

（9）Rang：这个库时一组关于范围的概念和工具。比起算法中使用一对迭代器来指定范围，使用ranges更简单，并提升了用户代码的抽象水平。

（10）Tuple：在标准C++中有Pair（类模板std：：pair），但它不支持n-tuples。用TU盘了不像structs或classes来定义n-tuples，这个类模板支持直接声明和使用，如函数返回参数类型或参数，并提供一个泛型的方法来访问tuple的元素。

（11）Variant：包含一个不同于union的泛型类，用于在存储和操作来自不同类型的对象。这个库的一个特点是支持类型安全地访问，减少了不同数据类型的类型转换代码的共同问题。

1. 函数对象及高级编程

（1）Bind：是对标准库的绑定器bind1st 和 bind2nd的泛化。这个库支持使用统一的语法将参数绑定到任何类似于函数行为的东西，如函数指针、函数对象，以及成员函数指针。

（2）Function：实现了一个泛型的回调机制。它提供了函数指针、函数对象和成员函数指针的存储和后续的调用。

（3）Functional：提供C++标准库的适配器的加强版。主要的优势是它有助于解决引用到引用(这是非法的)的问题。

（4）Lambda：为C++提供lambda表达式及无名函数。在使用标准库算法时特别好用，Lambda允许函数在呼叫点创建，避免了创建多个小的函数对象。

（5）Ref：包装了一个对象的引用，并把它放入一个对象以便被复制。这就允许了通过引用去调用那些采用传值参数的函数。

（6）Signals：信号和接收系统，基于称为publisher-subscriber 和 observer的模式，它是在一个最小相关性系统中管理事件的重要工具。

1. 泛型编程与模板元编程

（1）Call\_traits：提供了传递参数给函数的最好方法的自动演绎，依据参数的类型。

（2）Concept\_check：提供一些类模板，用于测试特定的概念(需求的集合)。

（3）Enable\_if：允许函数模板或类模板的特化体包括/排除在一组匹配的函数或特化体之中/之外。主要的用例是包括/排除基于某些特性的特化体。

（4）In\_place\_factory：是一个直接构造所含对象的框架，包括用于初始化的可变参数列表。它可以消除对所含类型必须是CopyConstructible的要求，并减少了创建不必要的临时对象的需要，该临时对象仅用于提供复制所需的源对象。

（5）Mpl：模板元编程库。它包含了与C++标准库十分相象的数据结构和算法，但它们是在编译期使用的。

（6）Property\_map：是一个概念库而不是一个真正的实现。它引入了 property\_map 概念以及property\_map类型的一组要求，从而给出了对一个key和一个value的映射的语法和语义要求。

（7）Type\_traits：提供关于类型的编译期信息，如某个类型是否指针或引用，以及增加或去除类型基本属性。

（8）Static\_assert：提供对名字空间、类、函数作用域的静态断言的支持。

1. 数学及数字处理

（1）Integer：提供了对整数类型的有用功能，如编译期的最小、最大值常数，基于给定位长的合适大小的类型，静态二进制对数计算等等。

（2）Interval库帮助你使用数学区间。它提供类模板interval及相关算子。

（3）Math是一组数学模板：quaternions 和 octonions (复数的特化)；数学函数如acosh, asinh, 和 sinhc；计算最大公约数(GCD)和最小公倍数(LCM)的函数等等。

（4）Minmax：可以同时计算最小和最大值，而使用std::min 和 std::max则要两次比较。

（5）Numeric Conversion：是一组用于在不同数字类型的值之间进行安全及可预言的转换的工具。

（6）Operators库提供了相关操作符及概念(LessThanComparable, Arithmetic,等等)的实现。

（7）Random：这是一个对随机数的专业使用的库，包括大量的生成器和分配器，可适用于多个不同的领域，如仿真和加密。

（8）Rational：提供的有理数类型可以基于任意整数类型，包括用户自定义的整数类型。

（9）uBLAS：使用数学符号提供对向量和矩阵的基本线性代数操作，采用操作符重载，它可以生成紧凑的代码

1. 输入/输出

（1）Assign帮助你把一系列的值赋给容器。它通过对operator, (逗号操作符) and operator()() (函数调用操作符)的重载，带给用户一种数据赋值的很容易的方法。

（2）Filesystem库提供对路径、目录和文件操作的可移植性。

（3）Format：加入了按格式化串进行格式化的功能，类似于printf, 但增加了类型安全性。

（4）Io\_state\_savers：允许保存IOStream对象的状态，用于以后的恢复，以取消可能发生的任何状态的变化。

1. Serialization：允许任意的C++数据结构存进来，再取出去，以及存档。
2. 其他Boost库

（1）Conversion：包含有一些函数，它们是现有的强制类型转换操作符(static\_cast, const\_cast, 和 dynamic\_cast)的增强。

（2）Crc：提供了循环冗余码(CRC)的计算，常有于校验和类型。

（3）Date\_time：提供了对日期和时间类型及对它们的操作的广泛支持。

（4）Optional：提供了类模板optional, 它是一个在语义上有额外状态的类型，它可以有效地表明optional的实例是否包含被封装对象实例。

（5）Pool库提供了一个内存池分配器，它是一个工具，用于管理在一个独立的、大的分配空间里的动态内存。

（6）Preprocessor：提供了在预处理器之上的抽象。它包括lists, tuples, 和 arrays, 还有操作这些类型的algorithms。

（7）Program\_options库提供了程序选项配置(名字/值对), 程序选项通常是通过命令行参数或配置文件提供。

（8）Python：提供了C++与Python的互操作性。

（9）Smart\_ptr：包括了非插入的智能指针用于限制范围(scoped\_ptr 和 scoped\_array)，用于共享资源(shared\_ptr 和 shared\_array), 一个配合shared\_ptr使用的智能指针(weak\_ptr), 还有一个插入式的智能指针类(intrusive\_ptr). Smart\_ptr的shared\_ptr (包括它的助手enable\_shared\_from\_this) 以及 weak\_ptr 已被收入即将发布的Library Technical Report。

（10）Test：提供了一整组用于编写测试程序的组件，可以把测试组织成简单的测试用例及测试套装，并控制它们的执行。

（11）Thread：一个提供可移植线程的库，它包含大量线程的原始概念和高度抽象。

（12）Timer：包含计时所需的特性，它的目标是尽可能做到跨平台的一致性。

（13）Tribool：实现了三状态布尔逻辑，即true 和 false以外一个额外的状态：indeterminate (这个状态也被称为maybe; 这个名字是可配置的)。

（14）Value\_initialized库帮助你用泛型的方法构造和初始化对象。

## QT

Qt是一个全面的C++应用程序开发框架，它提供给应用程序开发者建立艺术级的图形用户界面所需的所用功能。Qt是完全面向对象的很容易扩展，并且允许真正地组件编程。它包含一个类库和用于跨平台开发及国际化工具。

Qt类库拥有超过400 C++类，同时不断在不断扩展扩展的库，它封装了用于端到端应用程序开发所需要的所有基础结构。拥有优秀的Qt应用程序接口，包括城市的对象模型，内容丰富的集合类，图形用户界面编程与布局设计功能等，数据库编程，网络，XML，国际化等。Qt较常用到的类有：

（1）QApplication 应用程序类；

（2）QLabel 标签类；

（3）QPushButton 按钮类；

（4）QButtonGroup 按钮组合类；

（5）QGroupBox 群组类；

（6）QDateTimeEdit 日期时间编辑框类

（7）QLineEdit 行编辑框类

（8）QTextEdit 文本编辑框类

（9）QComboBox 组合框类

（10）QProgressBar 进度条类

（11）QLCDNumber 数字显示框类

（12）QScrollBar 滚动条类

（13）QSpinBox 微调框类

（14）QSlider 滑动条类

（15）QIconView 图标视图类

（16）QListView 列表视图类

（17）QListBox 列表框类

（18）QTable 表格类

（19）QValidator 有效性检查类

（20）QImage 图像类

（21）QMainWindow 主窗口类

（22）QPopupMenu 弹出性菜单类

（23）QMenuBar 菜单栏类

（24）QToolButton 工具按钮类

（25）QToolTip 提示类

（26）QWhatsThis 这是什么类

（27）QAction 动作类

（28）QHBoxLayout 水平布局类

（29）QVBoxLayout 垂直布局类

（30）QGridLayout 表格布局类

（31）QT对话框类

（32）QMessageBox 消息对话框类

（33）QProgressDialog 进度条对话框类

（34）QWizard 向导对话框类

（35）QFileDialog 文件对话框类

（36）QColorDialog 颜色对话框类

（37）QFontDialog 字体对话框类

（38）QPrintDialog 打印对话框类

Qt是应用于本地化跨平台应用开发的领先性框架。Qt应用程序接口与工具兼容所有支持平台，让开发员们掌握一个应用程序接口便可执行与平台非相关的应用开发与配置。通过使用Qt开发小组们从主要平台可为主要操作系统创建本地化的应用程序。

MFC同样也可用来进行C++图形用户界面应用程序开发，在这里刚好可以拿来与Qt进行对比：

（1）MFC效率较高，但大量的Windows API和消息机制使得其较难理解，不易用；QT封装较好，易用且跨平台，但效率较低；

（2）MFC（微软基础类库）是专门为windows设计的一个用于开发图形用户界面的类库。MFC或多或少使用了面向对象的方法包装了Win32的API，正因如此，这些API有时是C++，有时是C，甚至是C和C＋＋的混合体。Qt这个C++的图形库由Trolltech在1994年左右开发。它可以运行在Windows,Mac OS X, Unix，还有像Sharp Zaurus这类嵌入式系统中。Qt是完全面向对象的。

（3）MFC编程需要使用Document/View模式以及模板(template),如果不使用的话，编程将变得异常困难。而且，模板(template) 设定了固定的结构，若所需结构乃模板未定义之结构，则编程难已。例如，划分一区域使显示两个视图(view)于两个文档(document)。还有一个经常的问题是：模板(template)创建了视图(view)却无法访问(access)它，文档（document）要做完所有事情，但是这经常会出现问题。Qt不强制使用任何设计模式。如果你认为恰当，使用Document/view没有任何问题。不使用也没有任何问题。

（4）MFC的根本目的是访问包装起来的用C语言写的windows的API。 这绝非好的面向对象的设计模式，在很多地方，你必须提供一个包含15个成员的C语言的struct，但是其中只有一个与你所期望的相关，或者必须用旧式的参数来调用你的函数。Qt恰恰相反，它的架构明显是经过精心设计的面向对象的。Qt因此在命名，继承，类的组织等方面保持了优秀的一致性。你只需要提供唯一一个方法的参数，仅此一个。在不同的类中调用方式也是有很强的连贯性。返回值也很有逻辑性。所有一切达到了简单和强大的和谐统一。一旦你使用了其中一个类，其他的类也就触类旁通，因为他们是一致的。

（5）MFC是事件驱动的架构。要执行任何操作，都必须是对特定的消息作出响应。Windows对应用程序发送的信息数以千计，遗憾的是，要分清楚这些分繁芜杂的消息是很困难的。Qt的消息机制是建立在SIGNAL()发送和SLOT()接受的基础上的。这个机制是对象间建立联系的核心机制。利用SIGNAL()可以传递任何的参数。他的功能非常的强大。可以直接大传递信号给SLOT()，因此可以清楚的理解要发生的事情。

（6）MFC无法创建大小动态可变的子窗口，必须重新手动修改代码来改变窗口的位置（这恰好解释了为什么windows里的dialog是不可以改变的）这个问题在软件进行国际化翻译的时候更加严重，因为许多国家表达相同意思需要更长的词汇和句子，必须要对每个语言的版本重新修改自己的软件。在Qt中，任何东西都可以手动的敲出来。Qt拥有非常简单而又不失强大的layout机制，以至于不使用它就是在浪费时间了。Qt还提供了一个图形用户工具，Qt Designer，可以用来帮助建立用户界面。

（7）使用MFC是可以国际化的，但是需要将每一个字符串放在一个字符串表中，在代码中到处使用LoadString(IDENTIFIET)。然后转化这些资源到DLL中，翻译字符串到所需要的语言，改变图形界面，然后调用程序使用这个DLL。使用Qt的时候，只需要将字符串置于函数tr()中，在程序开发中这算是举手之劳。可以直接在代码中改变字符串的参考。Qt Linguist,Qt的一个工具，能够提取所有待翻译的string并按照友好的界面显示出来。

（8）在发布基于MFC的软件时，必须依靠存在于客户电脑上的MFC。但是这是不安全的，同样是MFC42.dll，可以基于相同的库得到3个不同的版本。Qt则没有这个风险，因为Qt压根就没有“升级整个系统”这个概念。