[BOOST--学了很久的C++](http://blog.csdn.net/chaoyuan899/article/details/8946008)

标签： [boost](http://www.csdn.net/tag/boost)

2013-05-19 09:35 1678人阅读 [评论](http://blog.csdn.net/chaoyuan899/article/details/8946008#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chaoyuan899/article/details/8946008#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

C++（42） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

     学习了很久的C++，不过最近才知道有BOOST这个东西，还是在程序员杂志上看到JJHOU先生写的关于BOOST 使用的系统文章时才决定来关注一下这个神秘的东西。下面是网上找的一篇文章，看完之后作了一些修改，可以对大家有用吧。

       一直流传这么一个说法，想成为高手，一定要多读高手写的源代码。哪些代码是好材料呢？C++标准库的源代码？不，如果您读过，就会发现：要么是各种实现独有的表达方式让人摸不着头脑，要么是恐怖的代码风格（如到处是下划线）憋得人难受。**Boost库的代码则相当清晰，注释合理，命名规范，绝对是适合阅读的典范**。同时，Boost内容广泛，**数值计算、泛型编程、元编程、平台API**……不妨从容选择自己感兴趣的部分，细细品味。   
  
Boost是什么？一套**开放源代码、高度可移植的C++库**。   
  
谁发起的？C++标准委员会库工作组。所以，质量保证，不怕遇到假冒伪劣产品。   
  
  
**其中所包含的内容：**  
  
字符串及文本处理  
**Boost.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。由于C++没有提供正则表达式支持，使得有些用户被迫转向其它支持正则表达式的语言，如Perl, awk, 和 sed。Regex提供了高效和强大的正则表达式支持，基于与STL同样的前提而设计，这使得它很容易使用。Regex已被即将发布的Library Technical Report接受。更多的信息，请见"Library 5: Regex."  
  
  
**Boost.Spirit**        Spirit库是一个多用途的、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[1]。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。分析器非常难写，对于一个特定的问题，它们很快就变得难于维护和看懂。而Spirit解决了这些问题，而且达到了与手工制作的分析器一样或几乎一样的性能。  
  
[1] Wave库使用Spirit实现了一个与C++高度一致的预处理器，就证明了这一点。  
  
  
**Boost.String\_algo**      这是一组与**字符串相关的**[**算法**](http://lib.csdn.net/base/datastructure)。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。  
  
  
**Boost.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。  
  
  
[**数据结构**](http://lib.csdn.net/base/datastructure)**, 容器, 迭代器, 和算法  
Boost.Any  
     Any库支持类型安全地存储和获取任意类型的值**。当你需要一个可变的类型时，有三种可能的解决方案：**无限制的类型**，如 void\*. 这种方法不可能是类型安全的，应该象逃避灾难一样避免它。**可变的类型**，即支持多种类型的存储和获取的类型。**支持转换的类型**，如字符串类型与整数类型之间的转换。**Any实现了第二种方案，一个基于值的可变化的类型，无限可能的类型**。这个库通常用于把不同类型的东西存储到标准库的容器中。更多的说明请见 "Library 6: Any."  
  
  
**Boost.Array**    这个库包装了普通的**C风格数组**，给它们增加了一些来自于标准库容器的函数和typedef 。其结果就是可以把普通的数组视为标准库的容器。这非常有用，因为它增加了类型安全性而没有降低效率，而且它使得标准库容器和普通数组拥有统一的语法。后一点意味着可以把普通数组用于大多数的要求容器类来操作的函数。当要求软件要达到普通数组的性能时，可以用Array来替代std::vector.  
  
  
**Boost.Compressed\_pair**       这个库包括一个参数化的类型, **compressed\_pair, 它非常象标准库中的 std::pair. 与std::pair**不同之处在于， boost::compressed\_pair 对模板参数进行评估，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。**Boost.Compressed\_pair 常用于存放一对对象，其中之一或两个都可能是空的。  
  
Boost.Dynamic\_bitset  
       Dynamic\_bitset库非常象std::bitset**, 除了std::bitset 是用参数来指定位数(即容器的大小), 而**boost::dynamic\_bitset 则支持在运行期指定大小**。dynamic\_bitset 支持与std::bitset一样的接口，还增加了支持运行期特定功能的函数和一些std::bitset中没有的功能。在bitset的大小无法在编译期确定或在程序运行时可能变化的情况下，这个库通常用于替换std::bitset。  
  
  
**Boost.Graph  
      Graph是一个处理图结构的库**，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，拓朴逻辑排序，等等。  
  
  
Boost.Iterator**       这个库**提供一个创建新的迭代器类型的框架**，还提供了许多有用的迭代器适配器，比C++标准中定义的更多。创建遵循标准的新迭代器类型是一件困难且乏味的工作。Iterator通过自动完成大多数细节，如提供所需的 typedef，简化了这件工作。Iterator还可以改编已有的迭代器类型以赋于它新的行为。例如，间接迭代器适配器增加了一个额外的解引用操作，可以把一个包含某种对象的指针(或智能指针)的容器变成象一个包含该对象的容器。  
  
**Boost.MultiArray  
       MultiArray提供了一个多维容器**，它很象标准库的容器，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。  
  
**Boost.Multi-index  
       Multi-index为底层的容器提供多个索引**。这意味着一个底层的容器**可以有不同的排序方法和不同的访问语义**。当std::set 和 std::map不够用时，就可以用Boost.Multi-index，通常是在需要为查找元素而维护多个索引时。  
  
  
**Boost.Range  
       这个库是一组关于范围的概念和工具**。比起在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。  
  
  
**Boost.Tuple**  
       在标准C++中有Pairs(类模板 std::pair), 但它不支持n-tuples。用Tuple.不象用structs 或 classes 来定义n-tuples, 这个类模板支持直接声明和使用，如函数返回类型或参数，并提供一个泛型的方法来访问tuple的元素。关于这个库的详细信息，请见"Library 8: Tuple 8"。Tuple已经被即将发布的Library Technical Report所接受。  
  
  
**Boost.Variant  
       Variant库包含一个不同于union的泛型类，用于在存储和操作来自于不同类型的对象**。这个库的一个特点是支持类型安全的访问，减少了不同数据类型的类型转换代码的共同问题。  
  
  
**函数对象及高级编程  
Boost.Bind  
        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。它还可以通过嵌套绑定器实现函数组合。这个库不要求那些对标准库绑定器的强制约束，最显著的就是不要求你的类提供typedefs result\_type, first\_argument\_type, 和 second\_argument\_type 等。这个库也使得我们不再需要用 ptr\_fun, mem\_fun, 和 mem\_fun\_ref 等适配器。Bind库的说明在"Library 9: Bind 9."。它是对C++标准库的一个重要且很有用的扩充。Bind可以被标准库的算法使用，也经常用于Boost的函数，它提供了一个强大的工具，用于存放后续调用的函数和函数对象。Bind 已被即将发布的Library Technical Report所接受。  
  
  
**Boost.Function  
        Function库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。当然，它与binder库，如Boost.Bind 和 Boost.Lambda一起工作，大大提高了回调(包括带态度的回调函数)的使用机会。这个库的详细介绍请见"Library 11: Function 11."。Function常用于需要把函数指针用于回调的地方。例如：信号/接收者的实现，GUI与业务逻辑的分离，以及在标准库容器中存储不同的类函数类型。Function已被即将发布的Library Technical Report所接受。  
  
  
**Boost.Functional  
         Functional库提供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，这个问题是由对带有一个或多个引用参数的函数使用标准库的绑定器所引起的。Functional同时消除了在标准库算法中使用函数指针时必须用ptr\_fun的问题。  
  
  
**Boost.Lambda  
       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。使用lambdas意味着更少的代码，在哪需要就在哪写，这比分散在代码各处的函数对象更清晰、更好维护。"Library 10: Lambda 10" 详细讨论了这个库。  
  
  
**Boost.Ref**  
       许多函数模板，包括大量标准C++库里的函数模板，它们的参数采用传值的方式传递，有时候会有问题。复制一个对象可能很昂贵或者甚至不可能，或者状态可能取决于特写的实例，因此这时复制是不希望的。在这些情况下，可用的办法是用引用传递取代值传递。 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。Ref 已被即将发布的Library Technical Report所接受。  
  
**Boost.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。很少有大型应用软件不采用这种强大设计模式的某种变形，尽管他们有各自的实现方式。**Signals提供了一个已验证的、高效的手段，将信号(events/subjects)的发生和这些信号要通知的接收者(subscribers/observers)进行了分离。  
  
泛型编程与模板元编程  
Boost.Call\_traits**           这个库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。例如，当传递的是如int 和 double这样的内建类型，最高效的方式是传值。对于用户自定义类型，则传送const引用通常更好。Call\_traits为你自动选择正确的参数类型。这个库还有助于声明参数为引用，而不用冒引用到引用的风险(在C++这是非法的)。Call\_traits常用于要求以最高效方式传递参数而又不知道参数类型的泛型函数，并避免引用到引用的问题。  
  
  
**Boost.Concept\_check**  
      Concept\_check提供一些类模板，用于[**测试**](http://lib.csdn.net/base/softwaretest)特定的概念(需求的集合)。泛型(参数化的)代码要求实例化时的类型必须符合某些抽象概念，如LessThanComparable. 这个库提供了一些方法来明确地声明模板的参数化类型的特定需求。代码的用户可以获益，由于需求的文档化以及编译器可以产生错误信息以明确指出类型不符合这些概念的地方。Boost.Concept\_check提供了超过30个可用于泛型代码的概念，其中一些原型可用于校验包括所有相关概念的组件的实现。它用于在泛型代码中声明和证明概念的需求。  
  
  
**Boost.Enable\_if**  
      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。例如，仅当采用一个整数类型实例化时使能一个函数模板。这个库还为SFINAE(substitution failure is not an error)提供了一个非常有用的研究机会。  
  
**Boost.In\_place\_factory**  
      In\_place\_factory库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。这个库有助于减少传送用于对象初始化的参数所需的工作量。  
  
In\_place\_factory 的作者是 Fernando Cacciola.  
  
Boost.Mpl  
Mpl是一个模板元编程库。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。甚至有编译期的lambda表达式支持！提供编译期的操作，如产生类型或操作类型序列，在现代C++中越来越普遍，而提供这些功能的库是非常重要的工具。就我所知，还没有其它象Mpl这样的库。它填充了C++元编程世界的空白。我可以告诉你在你读本书时有一本关于Boost.Mpl的书正在创作，它就快要面世了，它就是Aleksey Gurtovoy 和 David Abrahams所著的C++ Template Metaprogramming。你应该尽快获得一本。  
  
Mpl 的作者是 Aleksey Gurtovoy, 并有许多其它人的重要贡献。  
  
Boost.Property\_map  
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。这在需要声明必须支持的类型的泛型代码中很有用。C++数组是一个property\_map的例子。这个库包含了Boost.Concept\_check可以测试的概念的定义。  
  
Property\_map 的作者是 Jeremy Siek.  
  
Boost.Static\_assert  
进行编译期编程的一个公共的需求是提供静态断言，即编译期断言。另外，获得一致的错误提示不是必然的，由于静态断言必须会产生失败断言的信号，跨不同的编译器。Static\_assert提供对名字空间、类、函数作用域的静态断言的支持。详细信息见"Library 3: Utility."  
  
Static\_assert 的作者是 Dr. John Maddock.  
  
Boost.Type\_traits  
成功的泛型编程通常需要根据参数化类型进行决策或调整这些类型的属性(如cv-qualification[2])。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。Type\_traits已被加入即将发布的Library Technical Report。  
  
[2] 一个类型可以是cv-unqualified (非 const 或 volatile), const-qualified (const), volatile-qualified (声明为 volatile), or volatile-const-qualified (既 const 并 volatile); 类型的这些版本都是独特的。  
  
Type\_traits 的作者是 Steve Cleary, Beman Dawes, Aleksey Gurtovoy, Howard Hinnant, Jesse Jones, Mat Marcus, John Maddock, 和 Jeremy Siek, 以及其它许多人的贡献。  
  
数学及数字处理  
Boost.Integer  
这个库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[3]，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。还包括从1999年C标准头文件<stdint.h>中的typedef。  
  
[3] std::numeric\_limits 仅能以函数方式提供这些值。  
  
Integer 的作者是 Beman Dawes 和 Daryle Walker.  
  
Boost.Interval  
Interval库帮助你使用数学区间。它提供类模板interval及相关算子。区间的常见用法(除了明显的进行区间计算的情况)是提供模糊结果的计算；区间的使用可以量化舍入误差的传播情况。  
  
Interval 的作者是 Guillaume Melquiond, Sylvain Pion, 和 Hervé Brönniman, 该库从 Jens Maurer的前期工作获得灵感。  
  
Boost.Math  
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。  
  
Math 的作者是 Hubert Holin, Daryle Walker, 和 Eric Ford.  
  
Boost.Minmax  
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。对于n个元素的情况，只要3n/2+1次比较，而使用std::min\_element 和 std::max\_element则需要2n次比较。  
  
Minmax 的作者是 Hervé Brönniman.  
  
Boost.Numeric Conversion  
Numeric Conversion库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。例如，有一个名为numeric\_cast (最早来自于Boost.Conversion)的工具，提供了范围检测的转换以确定数值可被目标类型所表示，否则它会抛出异常。  
  
Numeric Conversion 的作者是 Fernando Cacciola.  
  
Boost.Operators  
Operators库提供了相关操作符及概念(LessThanComparable, Arithmetic,等等)的实现。定义一个类型的操作符时，保证所有操作符都有定义是一件乏味并容易出错的工作。例如，你提供了operator< (LessThanComparable)，通常都要同时提供operator<=, operator>, 和 operator>= 。Operators可以根据给定类型的最小的用户自定义操作符集合，自动声明并定义其它所有的相关操作符。详细讨论见"Library 4: Operators 4."  
  
Operators 的作者是 David Abrahams, Jeremy Siek, Aleksey Gurtovoy, Beman Dawes, 和 Daryle Walker.  
  
Boost.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。Random已被收入即将发布的Library Technical Report.  
  
Random 的作者是 Jens Maurer.  
  
Boost.Rational  
整数类型和浮点数类型都内建成于C++语言，复数类型也是C++标准库的一部分，但有理数类型呢？有理数可以避免浮点数的精度损失问题，因此它们常被用于计算金钱等。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(具有无限精度的类型显然是很有用的).  
  
Rational 的作者是 Paul Moore.  
  
Boost.uBLAS  
uBLAS库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(使用表达式模板)。  
  
uBLAS 的作者是 Joerg Walter 和 Mathias Koch.  
  
输入/输出  
Boost.Assign  
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。除了对原型风格的代码特别有用，这个库的功能在其它时候也很有用，使用这个库有助于提高代码的可读性。使用本库中的list\_of还可以就地生成无名数组。  
  
Assign 的作者是 Thorsten Ottosen.  
  
Boost.Filesystem  
Filesystem库提供对路径、目录和文件操作的可移植性。这种高级抽象使C++程序员可以写出类似于其它编程语言脚本的代码。它提供了便于操作目录和文件的算法。编写要在不同文件系统平台间移植代码的困难工作由于这个库的帮助变得容易了。  
  
Filesystem 的作者是 Beman Dawes.  
  
Boost.Format  
这个library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。相反使用具有相同便利性的printf的最主要问题是参数类型的危险；它不保证格式化串中指定的类型与实际的参数类型是匹配的。除了消除了这种不匹配性的危险以外，Format还可以用于格式化用户自定义的类型。[4]  
  
[4] 格式化函数用省略号表示可变数量的参数是不可以的。  
  
Format 的作者是 Samuel Krempp.  
  
Boost.Io\_state\_savers  
Io\_state\_savers库允许保存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。许多操纵器会永久改变它们操作的流的状态，这可能是你不想要的，而手工重置状态又容易出错。这个状态保存器可以保存控制标志、精度、宽度、异常掩码、流的locale等等。  
  
Io\_state\_savers 的作者是 Daryle Walker.  
  
Boost.Serialization  
这个库允许任意的C++数据结构存进来，再取出去，以及存档。例如，存档可以是文本文件或XML文件。Boost.Serialization是高度可移植的，并提供了非常成熟的特性，如类的版本、C++标准库中的通用类的序列化、共享数据的序列化，等等。  
  
Serialization 的作者是 Robert Ramey.  
  
杂项  
Boost.Conversion  
Conversion库包含有一些函数，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。Conversion为安全的多态转换增加了 polymorphic\_cast 和 polymorphic\_downcast，为安全的数字类型转换增加了 numeric\_cast，为文本转换(如string 和 double间的转换)增加 lexical\_cast。你可为了你自己的类型更好地工作而定制这些类型转换，可能这些类型并不可以使用语言本身所提供的类型转换。这个库的详细讨论在"Library 2: Conversion."  
  
Conversion 的作者是 Dave Abrahams 和 Kevlin Henney.  
  
Boost.Crc  
Crc库提供了循环冗余码(CRC)的计算，常有于校验和类型。CRC被加到一个数据流中(它就是从这些数据中计算得来的)，用来对这些数据进行校验，例如PKZip就使用了CRC32。这个库包含了四个CRC类型：crc\_16\_type, crc\_ccitt\_type, crc\_xmodem\_type, 和 crc\_32\_type5.  
  
Crc 的作者是 Daryle Walker.  
  
Boost.Date\_time  
Date\_time库提供了对日期和时间类型及对它们的操作的广泛支持。如果没有对日期和时间的支持，程序开发任务会变得复杂并容易出错。使用Date\_time，你想要的所有自然概念都被支持：日、周、月、持续时间(及时间间隔)、加、减等等。这个库还提供了其它日期/时间库所忽略的东西，如闰秒处理以及高精度时间源的支持。这个库的设计是可扩展的，允许客户化定制行为或添加功能。  
  
Date\_time 的作者是 Jeff Garland.  
  
Boost.Optional  
要求函数可以指出它的返回值无效是一个很普通的要求，但通常返回类型并不存在某个状态来表示其无效。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。  
  
Optional 的作者是 Fernando Cacciola.  
  
Boost.Pool  
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。当你需要分配和回收许多不的对象或需要更高效的内存控制时，使用内存池是一个好的解决方案。  
  
Pool 的作者是 Steve Cleary.  
  
Boost.Preprocessor  
当你要表示象循环这样的结构时，很难使用预处理器，它没有容器，不提供迭代器，等等。然而预处理器仍是一个强大的可移植的工具。Preprocessor库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。这个库有助于减少重复的代码，减轻你的负担，也使得代码更易读、更清晰、更具可维护性。  
  
Preprocessor 的作者是 Vesa Karvonen 和 Paul Mensonides.  
  
Boost.Program\_options  
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。这个库减轻了程序员手工分析这些数据的负担。  
  
Program\_options 的作者是 Vladimir Prus.  
  
Boost.[**Python**](http://lib.csdn.net/base/python)  
Python库提供了C++与Python[6]的互操作性。它用于将C++类及函数提供给Python，同样把Python对象给C++。它是非插入式的，也就是说已有代码无需修改即可用于Python。  
  
[6] 一种你应该知道的非常流行的编程语言。  
  
Python 的作者是 David Abrahams, 并得到Joel de Guzman 和 Ralf W. Grosse-Kunstleve的重要贡献。  
  
Boost.Smart\_ptr  
智能指针是任何一个程序员工具包中的重要部分。它们用于防止资源泄漏、共享资源、对象生存期管理。有很多好的智能指针库可用，有些是免费的，而有些是商业软件包的组成部分。Smart\_ptr是其中的佼佼者，已被成千上万的用户所证实，并被该领域的专家所推荐。 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。关于智能指针更详细的说明请见"Library 1: Smart\_ptr 1."  
  
Smart\_ptr 的作者是 Greg Colvin, Beman Dawes, Peter Dimov, 和 Darin Adler.  
  
Boost.Test  
Test库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。作为这个库的一个组件，程序执行监视器在某些生产(非测试)环境下也很有用。  
  
Test 的作者是 Gennadiy Rozental (基于Beman Dawes早期的工作).  
  
Boost.Thread  
可移植的线程是很难处理的业务，也无法从C++本身获取帮助，因为语言本身不包括线程支持。当然，我们有POSIX, 它在许多平台上可用，但POSIX使用的是C API。Thread是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。  
  
Thread 的作者是 William Kempf.  
  
Boost.Timer  
Timer库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。虽然每个平台都有特定的 API可以让程序员用于计时，但对于高精度计时还没有可移植的方案。Boost.Timer通过提供最大可能的精度并同时保留可移植性解决了这个问题，从而可以让你自由地确定精度。  
  
Timer 的作者是 Beman Dawes.  
  
Boost.Tribool  
这个库包含一个 tribool 库，它实现了三状态布尔逻辑。三状态布尔类型除了true 和 false以外还有一个额外的状态：indeterminate (这个状态也被称为maybe; 这个名字是可配置的).  
  
Tribool 的作者是 Douglas Gregor.  
  
Boost.Utility  
一些本不应在一个库里出现的有用的东西，只是因为它们每个都不太复杂和广泛，不足够形成一个单独的库。但不是说它们没有什么用外；事实上小的工具通常都有最广泛的用处。在Boost, 这些小工具被集中起来，形成一个称为Utility的库。你可以在这找到checked\_delete, 一个函数，用于确认在删除点的类型是完整的；还有类noncopyable，用于确保类不能被复制；还有enable\_if，用于对函数重载的完全控制。还有其它很多工具，详细请见"Library 3: Utility"。  
  
Utility 的作者是 David Abrahams, Daryle Walker, Douglas Gregor, 和其它人。  
  
Boost.Value\_initialized  
Value\_initialized库帮助你用泛型的方法构造和初始化对象。在C++里，一个新构造的对象可以是零初始化的、缺省构造的，或是不确定的，这依赖于对象的类型。有了Boost.Value\_initialized, 这种不一致的问题就没有了。