**字符串和文本处理**

* boost.regex 为 c++ 提供了正则表达式；
* boost.spirit 是 functional recursive-decent parser framework，支持使用 C++ 描述类似 EBNF 语法，可用于创建命令行 parser、预处理器。boost.wave 就是利用这个 framework 实现的 C++ 预处理器；
* boost.string\_algo 提供了 std::string 以外的字符串处理程序，包括 trimming、case conversion、splitting、finding、replacing 等功能；
* boost.tokenizer 用于将字符串类型的容器切分成为一个一个 token；
* （1.44）boost.iostreams 用于自定义 stream；
* （1.44）boost.wave 是利用 spirit 写的 C++ 的预处理程序，可以利用它实现很多 c++ 0x 的特性；
* （1.44）boost.xpressive 也是一个 regex 的库，但是使用 context free 语法；

**数据结构、容器、迭代器（iterator）和算法**

* boost.any 的目标是存储异构数据。一般来说异构数据需要使用不同的指针，而一种解决方案是使用 void\* 指针（但这种方式并不是 type safe 的）；另外一种是使用串行化，将所有异构的数据转换成为某种格式（如 string），要求所有的数据提供 serialization 和 deserialization 的方法。而 any 实现的类型安全的（value-based variant type with an unbounded set of possible types）；当我们在容器里面需要存储异构数据的时候，any 是一个非常好的选择；
* boost.array 是对 C++ 的数组进行的封装，由于 std::vector 等容器是动态的，性能上相对静态的数组是有一定的差距的，如果性能上需要使用数组但希望使用封装的更好的对象的时候，可以使用 array；
* boost.compressed\_pair 是与 std::pair 相近的功能；在 std::map 这类关联容器中我们常常使用 std::pair 创建需要的一对 key/value；boost 的版本允许在 pair 中存在空对象时利用 empty base optimization 压缩这个 pair 的内存占有率；
* boost.dynamic\_bitset 是对 std::bitset 的扩展，后者是一个静态的数据结构，而 dynamic\_bitset 允许在运行时确定 bitset 的大小；
* boost.graph 是对图结构、相关算法的封装，提供了 adjacency matrix、adjacency list、edge list 这些方式表示图，同时提供了最短路径、minimum spanning tree、topological sort 等算法的实现；
* boost.iterator 用于创建符合 stl 标准的 iterator，提供自动的 typedef，并且可以将已有的 iterator 的行为改变，创建 indirect iterator adaptor 可以将放在容器里面的智能指针 deference，这样可以像一般的容器（里面盛放的是正常的对象）一样使用；
* boost.multiarray 是对 C++ 中多维数组的封装；
* boost.multi-index 允许一个容器存在多个 index，这意味着可以让同一个容器使用不同的序，这样抽取其中元素的时候能够更加便捷；特别在 std::set、std::map 不能够提供需要的多重序关系访问时，multi-index 就会产生作用了；
* boost.range 提供的主要是一种概念性的东西：一般的 algorithm 输入都是两个 iterator 表示操作的范围；ranges 提供了更好的表达方式，这样使用 algorithm 会更加方便，另外也使得用户代码更加抽象；
* boost.tuple 是对 std::pair 的另一种扩展，也就是提供了所谓的 n-tuple（n 元组）结构；与直接使用 struct、class 创建 n-tuple 不同之处在于可以 direct declaration、作为函数返回值或者参数以及一致的方法访问 tuple 的任意成员；
* boost.variant 是对 union 结构的封装，主要为了解决一般 union 结构为了判断其类型而写的类型转换代码；variant 提供的是 type safe visitation；
* （1.44）boost.bimap 提供了双向关联容器；
* （1.44）boost.circular\_buffer 提供了循环的 buffer；
* （1.44）boost.gil 提供的是图片处理的封装；
* （1.44）boost.intrusive 提供了所谓的 intrusive container；intrusive container 是相对于 non-intrusive container 来说的，STL 的容器都是 non-intrusive，被存储的对象不需要有特别的代码上的修改，任何对象都可以存下来，代价是只能复制一份存放在容器里面；intrusive 容器需要对象本身提供互相连接的机制，因此需要修改原来的对象，好处是进入容器的是原来的对象，而不是其复制体；
* （1.44）boost.pointer\_container 用于在堆上的对象需要的指针；
* （1.44）boost.property\_tree 用于存放配置；
* （1.44）boost.unordered 是用于存放无序的数据；
* （1.44）boost.foreach 是一个 macro，可以方便的写遍历代码，使用 foreach 类型的语法；

**functor 和 higher-order programming**

* boost.bind 是非常有用的工具，是对 std::bind1st 和 std::bind2nd 的扩展；bind 允许用统一的接口对函数、函数指针、函数对象、成员函数指针进行封装，产生 functor 以供后面的 algorithm 使用；定义的 functor 提供了 stl 中需要的 typedef；这使得我们没有必要使用 std::ptr\_fun、std::mem\_fun 这类产生 functor 的 helper function；
* boost.function 提供的是一种包装函数指针、函数对象以及成员函数的机制，由此能够产生 stateful callback function（估计就是将产生函数对象时的相关信息存放在函数对象中）等用例；
* boost.functional 主要是对 STL 中存在的一个问题进行了处理：当函数存在引用类型的参数的时候会产生“引用的引用”这种非法的结果导致的错误；使用 functional 将使得使用 std::ptr\_fun 失去使用的意义；
* boost.lambda 是 Boost 的 lambda 算子，用于方便的创建 functor，并且与 STL 中的 algorithm 一起使用将非常方便；避免写非常多的 functors；
* boost.ref 是用来创建引用，这个引用和 C++ 自身的引用不同的是它可以被复制；与指针的用例不同之处在于，引用允许你一直用 . 而不是 -> 来访问成员、方法；
* boost.signals(2) 用于创建 signal/slot 的通讯机制，这在 design pattern 中对应着 observer 或者 publisher/subscriber 这种，GUI 开发中非常常见；
* （1.44）boost.mem\_fun 产生成员函数的 functor；
* （1.44）boost.functional/factory 用于动态或者静态的生成对象；
* （1.44）boost.functional/forward 允许使用任意个参数的 generic programming；
* （1.44）boost.functional/hash 可扩展到用户定义的数据上的 hash；
* （1.44）result\_of 用于确定函数的返回类型；

**generic programming 与 template metaprogramming**

关于这些诡异的概念见上一篇 blog。

* boost.call\_traits 提供了自动推理函数传递参数时最合适的方法，如果函数不修改参数本身，可以传值也可以传引用，在写 generic function 的时候需要这种机制选择合适的类型，一个例子比如我希望写一个串行化的东西，要求传入对象都支持 operator<< 插入到 std::ostream，可是 int 和 std::string 很显然前者希望传值，后者希望传引用，call\_traits 的作用就在于可以让编译器自动选择最合适的方式生成对应的 template；
* boost.concept\_check 用于检查 concept，我们知道 generic programming 里面最重要的就是搞清楚什么样的东西能够被传递给 template 而不会出错（满足某种 concept），那么这个 concept\_check 的用途就是在编译时能够检测出来；另外也提供了 30 多个 concept 用于检查；
* boost.enable\_if 用于根据某些条件特例化或者排除某种特例化；
* boost.in\_place\_factory 实现的是 in-place 创建对象，这对于将一些不能进行 copy 的对象放在容器里面是非常有用的；
* boost.mpl 本身就可以写一本书了，这是用来做 template metaprogramming 的非常有用的库（而在此之前几乎没有）；它实现了很多 C++ STL 的 algorithm，但是都是在编译时执行的；
* boost.property\_map 提供了用于创建 mapping 机制的 concept，与 concept\_check 一起可以用于有效的检查 generic programming 的正确性；
* boost.static\_assert 嗯这个就是进行编译时检查所用的技术，和 assert 宏类似，后者是运行时检查；
* boost.type\_traits 提供编译时判断 template 的类型，如是否指针/引用等等；
* （1.44）boost.function\_type 可调用（callable）对象的 traits；
* （1.44）boost.typed\_in\_place\_factory？
* （1.44）boost.fusion 用于与 tuple 一起使用的一些工具；

**数学与数值计算**

* boost.integer 提供了编译时关于整数的信息，对应于 C99 的 stdint.h；
* boost.interval 提供了对区间的封装；
* boost.math 提供了对复数的推广 quaternion 和 octonion 的封装以及 LCD 和 GCM 的计算；
* boost.minmax 是对 std::min 和 std::max 的补充；
* boost.numeric 给出了 numeric\_cast 保证类型转换的时候不会出现溢出；
* boost.operator 提供了常用的比较 operator；
* boost.random 用于产生随机数；
* boost.rational 封装了有理数；
* boost.ublas 是对 BLAS 的 C++ 实现，不同之处是 BLAS 原来是 fortran 写的 columnwise 的，而 uBLAS 是 rowwise 的；
* （1.44）boost.accumulator 用于处理增量式问题，比如求和，求最大值等；
* （1.44）boost.math/special\_function 包括了许多特殊函数；

**输入输出**

* boost.assign 重载了逗号和括号 operator，可以更方便的给容器赋值；
* boost.filesystem 提供了对基本文件操作的封装；
* boost.format 提供了与 printf 类似的功能，但是是 type safe 的；
* boost.io\_state\_savers 允许对 std::istream 和 std::ostream 进行保存，并在后面 undo ；
* boost.serialization 用于对数据进行串行化，并给出了对 STL 某些容器串行化的封装；

**并行处理**

* boost.thread 是一种 portable 的线程实现；
* （1.44）boost.asio 用于低级网络通讯；
* （1.44）boost.interprocess 用于进程之间的通讯；
* （1.44）boost.MPI 是一个 MPI 实现；

**其他**

* boost.conversion 提供了另外几种 cast 如 polymorphic\_cast、polymorphic\_downcast 等；
* boost.crc 用于计算 CRC；
* boost.date\_time 用于时间方面的操作；
* boost.optional 用于返回值可能存在需要表达“错误”或者“空”但是原来的类型可能不存在这种值的情况；
* boost.pool 提供了内存池；
* boost.preprocessor 提供了预处理程序，支持 recursion 和容器等；
* boost.program\_option 用于 parse 输入的命令行参数；
* boost.python 和 SWIG 做的类似，提供 C++ 的 python binding，当然 boost 这个仅仅面向 python 一个语言；
* boost.smart\_ptr 提供了几种智能指针；
* boost.test 提供了 unit test 的 framework；
* boost.timer 用于计算时间，尽量提供最高精度的时间；
* boost.tribool 提供三值（bool 是两值）；
* boost.utility 包括很多小工具，如 check\_deleted、noncopyable 等；
* boost.value\_initialized 用于处理各种不同方式的初始化，避免之间可能存在的不一致；
* （1.44）boost.uuid 用于计算 UUID；
* （1.44）boost.base\_from\_member 提供了从成员初始化基类的方法；
* （1.44）boost.exception 允许在 exception 中携带别的对象；
* （1.44）boost.flyweight 用于处理含有大量冗余的大规模数据的 design pattern；
* （1.44）boost.scope\_exit 提供在退出环境时执行的函数；
* （1.44）boost.statechart 用于实现有限状态机；
* （1.44）boost.swap 交换对象；
* （1.44）boost.system 提供对操作系统方面的支持；
* （1.44）boost.TR1 是 C++ 标准的 technical report；
* （1.44）boost.typeof 是 typeof 的一种模拟；
* （1.44）boost.units 是用来做 dimensional analysis 和 unit/quantity manipulation 的；

许多 boost 的库已经存在 C++ technical report，这意味着它们已经进入 C++ 标准委员会审查进入 STL 的阶段，有望在 C++0x 版本中成为新的标准。