Module04 – C++ Boost

C++ Boost - 课程目标

Module04 – C++ Boost 是 C++ 标准库的延伸，在这个 Module 中，将介绍以下几个方面的内容：

- 容器相关 (Containers)：
  - boost.any, boost.tuple, boost.array, boost.unordered
- 字符串和文字处理 (String & Text Processing)：
  - boost.lexical_cast, boost.format, boost.string_algo
- 正则表达式 (Regular Expression)
  - boost.regex
- 智能指针 (Smart Pointers)：
  - boost.smart_ptr

C++ Boost - 课程目标（续）

- 函数对象相关 (Function Objects)：
  - boost.bind, boost.mem_fn, boost.function, boost.ref, boost.lambda
- 序列化 (Serialization)：
  - boost.serialization
- 日期与时间 (Date & Time)
  - boost.date_time
- 多线程 (MultiThreads)：
  - boost.thread
- 网络 (Networking)：
  - boost.asio

Module05-01
C++ Boost：容器相关

C++ Boost - 容器相关

- 容器相关
- 字符串和文字处理
- 正则表达式
- 智能指针
- 函数对象相关
- 序列化
- 日期与时间
- 多线程
- 网络

容器相关 - boost.any

- 容器相关：
  - boost.any
  - boost.tuple (In TR1)
  - boost.array (In TR1)
  - boost.unordered (In TR1)

容器相关 - boost.any

- 关于 boost.any
  - 对任意类型进行安全的存储和安全的取回
  - 通过 boost.any 可以将不同类型的对象装进 STL 容器

容器相关 - boost.any

- boost.any 接口

```
// in <boost/any.hpp>

namespace boost {
class bad_any_cast;
class any;
template<typename T>
T any_cast(any&);

template<typename T>
T any_cast(const any&);

template<typename ValueType>
const ValueType* any_cast(const any*);

template<typename ValueType>
ValueType* any_cast(any*);
}
```

容器相关 - boost.any

- boost.any class

```
// in <boost/any.hpp>

class any {
public:
    // construct/copy/destruct
    any();
    any(const any&);
    template<typename ValueType> any(const ValueType&);
    any& operator=(const any&);
    template<typename ValueType>
    any& operator=(const ValueType&);
    ~any();

    // modifiers
    any& swap(any&);
    // queries
    bool empty() const;
    const std::type_info& type() const;
};
```

容器相关 - boost.any

- 示例
  - (DEMO Using Boost Examples)

容器相关 - boost.tuple

- 容器相关：
  - boost.any
  - boost.tuple (In TR1)
  - boost.array (In TR1)
  - boost.unordered (In TR1)

容器相关 - boost.tuple

- 关于 boost.tuple
  - boost.tuple 是 std::pair 的延伸，该组件已包含于 TR1，将会出现在 C++ 新标准中
  - 可以将 std::pair 视作 boost.tuple 的一个特例
  - 与 std::pair 不同的是 boost.tuple 可以容纳更多数量的元素
  - boost::tuple 的典型用途：
    - 作为函数的返回值，可以支持两个以上的返回值
    - 将多个相关的类型组合起来
  - tuple 元素必须有可用的构造函数、复制构造、赋值操作
  - 目前 tuple 元素个数最多为 10 个

容器相关 - boost.tuple

- boost.tuple 相关的头文件
  - 如果仅仅是使用 tuple 类型，只需包含：
    - `#include <boost/tuple/tuple.hpp>`
  - 如需比较 tuple 对象，其比较操作定义于：
    - `#include <boost/tuple/tuple_comparison.hpp>`
  - 如需输出 tuple 对象，则需：
    - `#include <boost/tuple/tuple_io.hpp>`

    (tuple_comparison.hpp 或 tuple_io.hpp 中已经包含 tuple.hpp)

容器相关 - boost.tuple

- 创建 boost.tuple 对象

```
// tuple可以是如下各种类型
tuple<int>
tuple<double&, const double&, const double, double*, const
double*>
tuple<A, int(*)(char, int), B(A::*)(C&), C>
tuple<std::string, std::pair<A, B> >
tuple<A*, tuple<const A*, const B&, C>, bool, void*>
```

```
int b[8] = { };
double d = 3.14;
tuple<int, string, double, char> t2(0, "hello", 3.14, 'o');
tuple<const int(&)[8], void(*)(int&), double&> t1(b, 0, d);
```

```
// 辅助函数make_tuple，作用如同std::make_pair
tuple<int, int, double> add_multiply_divide(int a, int b) {
    return make_tuple(a + b, a * b, double(a) / double(b));
}
```

容器相关 - boost.tuple

- 访问 tuple 元素
  - 可以通过 tuple 成员函数 get<N>() 或自由函数 get<N>(tuple t) 访问 tuple 中的元素

```
double d = 2.7;
A a;
tuple<int, double&, const A&> t(1, d, a);
const tuple<int, double&, const A&> ct = t;
//...
int i = boost::get<0> (t);
i = t.get<0> (); // ok
int j = boost::get<0> (ct); // ok
boost::get<0> (t) = 5; // ok
boost::get<0> (ct) = 5; // error, can't assign to const
//...
double e = boost::get<1> (t); // ok
boost::get<1> (t) = 3.14; // ok
boost::get<2> (t) = A(); // error, can't assign to const
A aa = boost::get<3> (t); // error: index out of bounds
++boost::get<0> (t); // ok, can be used as any variable
```

容器相关 - boost.tuple

- tie
  - tie 创建一个所有元素类型为非 const 引用的 tuple，如：

```
int i; char c; double d;
// 创建 tuple<int&, char&, double&>, 或:
// 调用 make_tuple(ref(i), ref(c), ref(d));
tie(i, c, d);

tie(i, c, d) = tuple<int, char, double> (1, 'A', 0.618);
cout << d << endl; // 0.618
```

  - ignore

```
char c;
tie(tuples::ignore, c) = std::make_pair(1, 'a');
```

容器相关 - boost.tuple

- I/O
  - 可以很方便的将 tuple 对象输出到 ostream 或从 istream 中获取 tuple 对象，注意包含头文件 <boost/tuple/tuple_io.hpp>：

```
tuple<float, int, string> a(1.0f, 2, string("Howdy folks!"));
cout << a; // (1 2 Howdy folks!)
```

  - 使用 3 个操控符： set_open(char), set_close(char), set_delimiter(char)

```
tuple<float, int, string> a(1.0f, 2, string("Howdy folks!"));
cout << set_open('[') << set_close(']')
     << set_delimiter(',') << a;
//output: [1,2,Howdy folks!]
```

```
tuple<int, int, int> i;  cin >> i; // inupt i from cin
tuple<int, int> j;
cin >> set_open('[') >> set_close(']') >> set_delimiter(':');
cin >> j;  // 接受: [12:9] 之类的格式
```

容器相关 - boost.array

- 容器相关：
  - boost.any
  - boost.tuple (In TR1)
  - boost.array (In TR1)
  - boost.unordered (In TR1)

容器相关 - boost.array

- 关于 boost.array
  - 定长数组，接口类似 std::vector，该组件包含于 TR1，将会出现在新的 C++ 标准中
  - 没有自定义的构造函数，没有 private 成员、没有虚函数等

```
// 定义一个容纳4个int型元素的定长数组
boost::array<int, 4> a = { 12, 26, 3 };
for_each(a.begin(), a.end(), cout << _1 << ' ');
```

容器相关 - boost.unordered

- 容器相关：
  - boost.any
  - boost.tuple (In TR1)
  - boost.array (In TR1)
  - boost.unordered (In TR1)

容器相关 - boost.unordered

- 关于 boost.unordered
  - 即基于 hash_table 的 4 种容器：
    - unordered_map (hash_map)
    - unordered_multimap (hash_multimap)
    - unordered_set (hash_set)
    - unordered_multiset (hash_multiset)
  - 该组件已包含于 TR1，将出现在新的 C++ 标准中
  - 接口类似于 std::map、 std::multimap、 std::set、 std::multiset

容器相关 - boost.unordered

- 关于 bucket 、 load_factor 和 rehash
  - 与基于红黑树的容器不同， boost.unordered 容器使用 bucket 容纳元素， bucket 的数量和 load_factor 直接影响容器操作的时间和空间的复杂度：
    - bucket 数量越大，单个 bucket 中元素数量有可能越少，查找速度越快，但占用的空间较大
    - 当 bucket 中的元素数量达到一定程度时，比如超过 load_factor 的比例时，查找的性能就会越来越差，这个时候需要 rehash ，分配更大的空间
    - 一旦 rehash ，将导致原先的迭代器 (Iterator) 失效，如同 vector 的重新分配空间

容器相关 - boost.unordered

- 等于判断式 (==) 和 hash 函数
  - 与基于红黑树的容器不同， boost.unordered 容器使用相等 (==) 比较 key ， 而不是使用 < ，对于自定义类型，必须保证其可进行相等比较
  - 另外，对于自定义类型，还应该提供合适的 hash 函数， hash 函数是基于 hash_table 的容器的操作能否高效的重要保证

容器相关 - boost.unordered

- 等于判断式 (==) 和 hash 函数（续）
  - 示例：

```
struct point {
    int x, y;
};

bool operator==(point const& p1, point const& p2) {
    return p1.x == p2.x && p1.y == p2.y;
}

std::size_t hash_value(point const& p) {
    std::size_t seed = 0;
    boost::hash_combine(seed, p.x);
    boost::hash_combine(seed, p.y);
    return seed;
}

// Now the default function objects work.
boost::unordered_multiset<point> points;
```

容器相关 - boost.unordered

- 操作复杂度保证
  - (See boost.unordered Doc)

容器相关 - More stuffs...

- 除本单元所列的 4 个容器相关的组件外， boost 还提供了其它有用的容器或数据结构组件，如：
  - boost.bimap： 即 map 的 key 和 value 都可以作为 key
  - boost.circular_buffer： 空间循环使用的 buffer
  - boost.multi_array： 多维数组
  - boost.multi_index： 相当于有多个 key 的关联容器
  - boost.pointer_contaner： 指针容器
  - ...