# boost

## 编译

1.打开vs命令提示符（x86本机工具命令提示符）

2.进入boost包目录

3.命令运行bootstrap.bat

4.命令运行b2.exe install

编译后会在c盘根目录下生成boost文件夹

b2.exe stage --toolset=msvc --stagedir="D:\boost\out\bin" link=static threading=multi runtime-link=static --build-type=complete –-build-dir="D:\boost\out\build"

## 时间

<boost\timer.hpp>

timer t;

.........

t.elapsed();//获取从对象创建到当前位置的时间

<boost\progress.hpp>

progress\_timer t;

可以像timer一样使用，还可以在退出作用域时自动输出时间。

<boost\date\_time\gregorian\gregorian.hpp>

gregorian::date d1;

gregorian::date d2(2016, 4, 17);

gregorian::date d3(d2);

if (gregorian::date(boost::date\_time::not\_a\_date\_time) == d1)

{

cout << "not" << endl;

}

if (d2 == d3)

{

cout << "d2 == d3" << endl;

}

获取当前时间

#include <boost/date\_time/posix\_time/posix\_time.hpp>

std::string strTime = boost::posix\_time::to\_iso\_string(boost::posix\_time::second\_clock::local\_time());

// strTime里存放时间的格式是YYYYMMDDTHHMMSS，日期和时间用大写字母T隔开

int pos = strTime.find('T');

strTime.replace(pos, 1, std::string("-"));

strTime.replace(pos + 3, 0, std::string(":"));

strTime.replace(pos + 6, 0, std::string(":"));

## 内存管理

<boost\shared\_ptr.hpp>

可以在多线程中安全使用

shared\_ptr<int> p(new int(5));

cout << \*p << endl;

//使用make\_shared抵消new的使用

shared\_ptr<int> p(make\_shared<int>(5));

## 文件操作

#include<boost/filesystem.hpp>

m\_filePath = "d:\\filename";

boost::filesystem::path path(m\_filePath);

if (boost::filesystem::exists(path))

{

boost::filesystem::remove(path);

}

## 序列化

boost::archive::binary\_oarchive ioa(stream);

//可以是文件流或者字符流

//包括xml、text和binary的io类型

//xml序列化要用oa & BOOST\_SERIALIZATION\_NVP(a)

ioa& Entity;//必须是支持序列化的类型

#include <fstream>

#include<boost\archive\binary\_iarchive.hpp>

#include<boost\archive\binary\_oarchive.hpp>

#include <boost\serialization\vector.hpp>

using namespace std;

class Test

{

protected:

int id;

vector<int> v;

public:

Test() {};

Test(int d) :id(d){}

void Set(int x)

{

v.resize(10, x);

}

private:

friend class boost::serialization::access;

// 如果类Archive 是一个输出存档，则操作符& 被定义为<<. 同样，如果类Archive

// 是一个输入存档，则操作符& 被定义为>>.

template<class Archive>

void serialize(Archive & ar, const unsigned int version)

{

ar & id;

ar & v;

}

};

// 保存数据到存档

{

// 创建类实例

Test g(1);

g.Set(5);

// 创建并打开一个输出用的字符存档

std::ofstream ofs("d:\\filename");

boost::archive::binary\_oarchive oa(ofs);

// 将类实例写出到存档

oa << g;

// 在调用析构函数时将关闭存档和流

}

//将类实例恢复到原来的状态

{

Test newg;

// 创建并打开一个输入用的存档

std::ifstream ifs("d:\\filename");

boost::archive::binary\_iarchive ia(ifs);

// 从存档中读取类的状态

ia >> newg;

// 在调用析构函数时将关闭存档和流

}

## 解析json

#include <boost/property\_tree/ptree.hpp>

#include <boost/property\_tree/json\_parser.hpp>

std::string json = "{\"A\":1,\"B\":{\"C\":2,\"D\":3},\"E\":[{\"F\":4},{\"F\":5}]}";

boost::property\_tree::ptree pt, child1, child2;

std::stringstream ss(json);

boost::property\_tree::read\_json(ss, pt);

child1 = pt.get\_child("B");

for (auto c : child1)

{

cout << c.first << c.second.data() << endl;

}