1. Qt信号槽机制的实现
2. QT bind()实现机制
3. C++智能指针
4. shared\_ptr是多线程安全的吗 自己如何实现一个shared\_ptr
5. 一个函数 void printNumber(int N)，在这个函数内创建两个线程，一个依次打印0到N之间的偶数0，2，4，、、、、，一个依次打印0到N之间的奇数1，3，5，、、、，现让两个线程交替打印，使输出 0、1、2、3、4、5、、、、

shared\_ptr实现

#ifndef \_\_SHARED\_PTR\_

#define \_\_SHARED\_PTR\_

template <typename T>

class shared\_ptr {

public:

shared\_ptr(T\* p) : count(new int(1)), \_ptr(p) {}

shared\_ptr(shared\_ptr<T>& other) : count(&(++\*other.count)), \_ptr(other.\_ptr) {}

T\* operator->() { return \_ptr; }

T& operator\*() { return \*\_ptr; }

shared\_ptr<T>& operator=(shared\_ptr<T>& other)

{

++\*other.count;

if (this->\_ptr && 0 == --\*this->count)

{

delete count;

delete \_ptr;

}

this->\_ptr = other.\_ptr;

this->count = other.count;

return \*this;

}

~shared\_ptr()

{

if (--\*count == 0)

{

delete count;

delete \_ptr;

}

}

int getRef() { return \*count; }

private:

int\* count;

T\* \_ptr;

};

#endif

一般来说，智能指针的实现需要以下步骤：

1.一个模板指针T\* ptr，指向实际的对象。

2.一个引用次数(必须new出来的，不然会多个shared\_ptr里面会有不同的引用次数而导致多次delete)。

3.重载operator\*和operator->，使得能像指针一样使用shared\_ptr。

4.重载copy constructor，使其引用次数加一。

5.重载operator=，如果原来的shared\_ptr已经有对象，则让其引用次数减一并判断引用是否为零(是否调用delete)。

　然后将新的对象引用次数加一。

6.重载析构函数，使引用次数减一并判断引用是否为零(是否调用delete)。

两个线程交叉打印

#include <thread>

#include <iostream>

#include <mutex>

#include <condition\_variable>

std::mutex data\_mutex;

std::condition\_variable data\_var;

bool label = false;

void printodd()

{

std::unique\_lock<std::mutex> ulock(data\_mutex) ;

for(int odd = 1; odd <= 100; odd += 2 )

{

data\_var.wait(ulock,[]{return label;});

std::cout<< std::this\_thread::get\_id() << ": " << odd <<std::endl;

label = false;

data\_var.notify\_one();

}

}

void printeven()

{

std::unique\_lock<std::mutex> ulock(data\_mutex) ;

for(int even = 0; even < 100; even += 2 )

{

std::cout<< std::this\_thread::get\_id() << ": " << even <<std::endl;

data\_var.notify\_one();

label = true;

data\_var.wait(ulock,[]{return !label;});

}

}

int main()

{

std::thread t1(printeven);

std::thread t2(printodd);

t1.join();

t2.join();

std::cout<<"end!"<<std::endl;

return 0;

}