知识点1【new delete】

1、new 给基本类型申请空间

2、new 申请 基本类型数组空间

3、new delete 给类对象申请空间

知识点2【对象数组】

本质是数组 只是数组的每个元素是类的对象

2、如果想让对象数组中的元素调用有参构造 必须人为使用 有参构造初始化。

3、用new delete申请 对象数组

4、尽量不要用delete释放void *

注意：没有析构？为啥?

5、malloc、free和new、delete 不可以混搭使用

知识点3【静态成员】static修饰的成员

1、static 修饰成员变量

静态变量，是在编译阶段就分配空间，对象还没有创建时，就已经分配空间。

静态成员变量必须在类中声明，在类外定义。

静态数据成员不属于某个对象，在为对象分配空间中不包括静态成员所占空间。

2、static修饰静态成员函数

静态成员函数：

注意：

1、静态成员函数的目的 操作静态成员数据。

2、静态成员函数 不能访问 非静态成员数据。（静态成员函数内部没有this指针）

3、普通成员函数 可以操作 静态成员数据 非静态成员数据。

4、静态成员变量 和 静态成员函数 都有权限之分。

3、const 修饰静态成员

如果一个类的成员，既要实现共享，又要实现不可改变，那就用 static const 修饰

知识点4【静态成员案例】

案例1：静态成员 统计类 实例化对象的 个数

案例2：单例模式设计--打印机（重要）

步骤1：在单例类内部定义了一个Singleton类型的静态对象，作为外部共享的唯一实例

步骤2：提供一个公共静态的方法，让客户可以访问它的唯一实例。

步骤3：为了防止在外部对实例化其他对象，将其默认构造函数和拷贝构造函数设计为私有

知识点5【this指针】（重要）

1、this指针的引入

1、this指针是隐含在对象成员函数内的一种指针

2、成员函数通过this指针即可知道操作的是那个对象的数据

3、静态成员函数内部没有this指针，静态成员函数不能操作非静态成员变量

知识点6【this指针的使用】

1、当形参和成员变量同名时，可用this指针来区分

2、在类的普通成员函数中返回对象本身（*this）（重要）

知识点1【new delete】

1、new 给基本类型申请空间

```
1  void test01()
2  {
3   //基本类型
4   int *p = NULL;
5   //p = (int *)calloc(1,sizeof(int));
6   //p = new int(100);// *p = 100
```

```
7   p = new int;
8   *p = 100;
9   cout<<"*p = "<<*p<<endl;//100
10   //释放 free(p)
11   delete p;
12  }
```

2、new 申请 基本类型数组空间

```
15  void test02()
16  {
17      //申请 int数组
18      int *arr = NULL;
19      //arr = (int *)calloc(5,sizeof(int));
20      arr = new int[5];//申请空间的时候 内容没有初始化 值随机
21
22      int i=0;
23      for(i=0;i<5; i++)
24      {
25          cout<<arr[i]<<" ";
26      }
27      cout<<endl;
28
29      //释放 new时加了[]  delete必须加[]
30      delete [] arr;
31  }
```

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
7696744 7667904 0 0 1644167266
```

```
32  void test03()
33  {
34      //申请 int数组
35      int *arr = NULL;
36      //arr = (int *)calloc(5,sizeof(int));
37      arr = new int[5]{1,2,3,4,5};//申请空间的时候 内容没有初始化 值随机
38
39      int i=0;
40      for(i=0;i<5; i++)
41      {
42          cout<<arr[i]<<" ";
43      }
44      cout<<endl;
45
46      delete [] arr;
47  }
```

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
1 2 3 4 5
```

```
#include<string.h>
void test04()
{
    //char *arr = new char[32]{"hehe"};//错误
    char *arr = new char[32];
    strcpy(arr,"hehe");
    cout<<arr<<endl;
    delete [] arr;
}
in
{

}
```

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
hehe
```

注意：

new 没有加[]　delete释放的时候 就不加[]

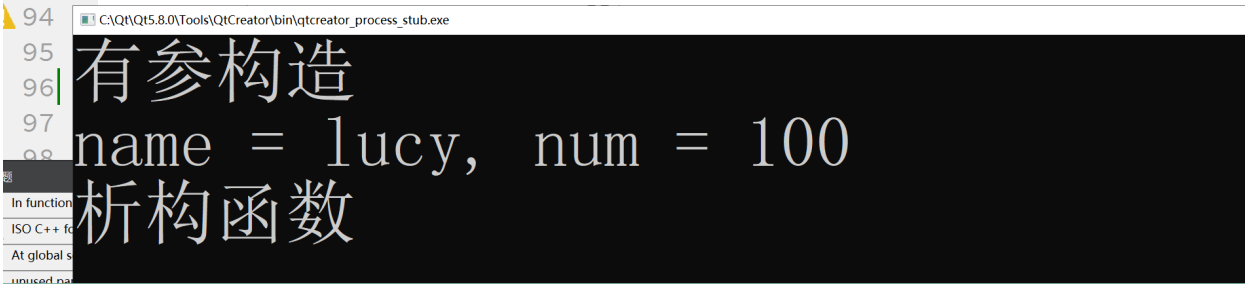
new 加[]　delete释放的时候 就加[]

3、new delete 给类对象申请空间

```
void test05()
{
    //new 按照Person申请空间　如果申请成功 就会 自动调用Person类的构造函数
    Person *p = new Person;

    //delete 先调用析构函数　再释放堆区空间
    delete p;
}
in
{

}
```

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
无参构造
析构函数
```

```
83  void test05()
84  {
85      //new 按照Person申请空间  如果申请成功 就会 自动调用Person类的构造函数
86      Person *p = new Person("lucy",100);
87
88      //由于p是指针 所以使用->  如果p是普通对象 使用.
89      p->showPerson();
90
91      //delete 先调用析构函数  再释放堆区空间
92      delete p;
93  }
```

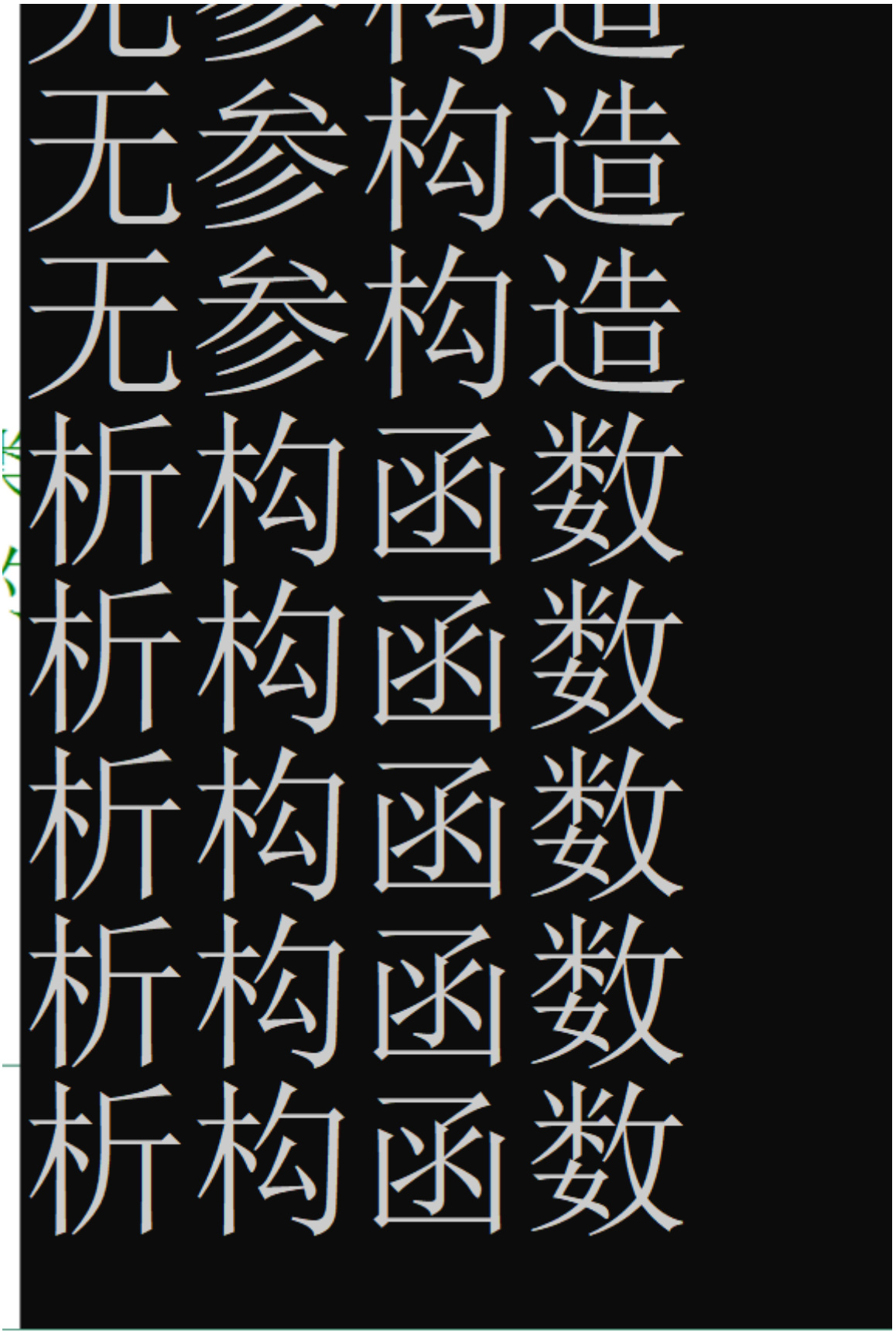


知识点2【对象数组】

本质是数组 只是数组的每个元素是类的对象

```
83  void test06()
84  {
85      //对象数组 arr1是数组 每个元素是Person类型的对象
86      //定义对象数组的时候 系统会自动给 数组中的每个元素 调用构造函数
87      //自动调用无参构造
88      Person arr1[5];
89
90  }
```

运行结果;

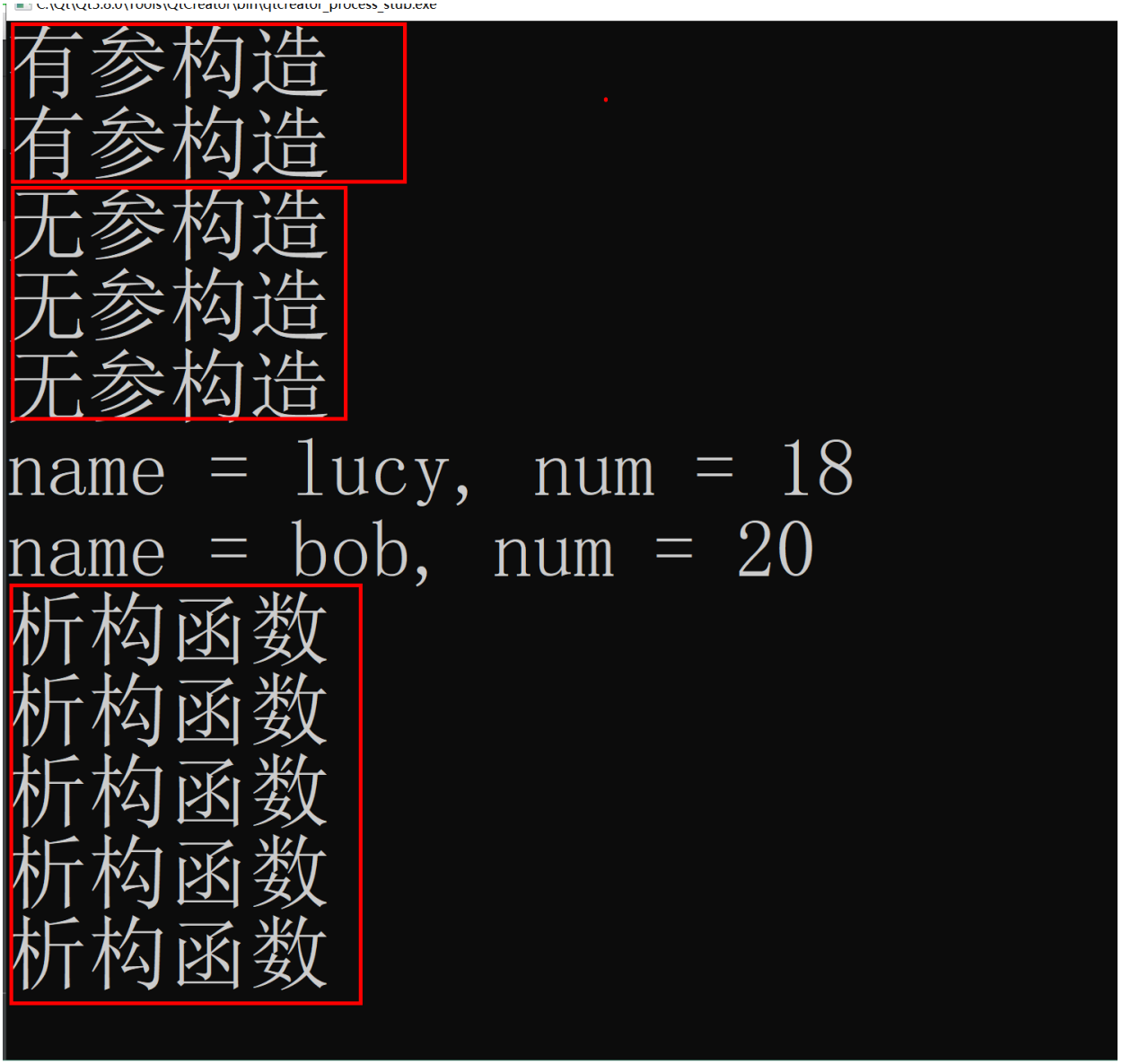


2、如果想让对象数组中的元素调用有参构造 必须人为使用 有参构造初始化。

```
1  void test07()
```

```
{
 //对象数组 arr1是数组 每个元素是Person类型的对象
 //定义对象数组的时候 系统会自动给 数组中的每个元素 调用构造函数
 //人为 为元素 调用有参构造
 //初始化部分 调用有参构造 为初始化部分自动调用默认构造（无参构造）
 Person arr1[5]={ Person("lucy",18), Person("bob",20)};
 //arr1[0] 第0个元素 就是Person的对象
 arr1[0].showPerson();
  arr1[1].showPerson();
}
```

运行结果：



3、用new delete申请 对象数组

```
void test08()
{
 //第一种方式
```

```
    Person *arr = NULL;
    arr = new Person[5];//调用无参构造

    delete [] arr;

    //第二种方式:
     //初始化的元素 调用有参构造 没有初始化 的调用无参构造
     Person *arr2 = new Person[5]{Person("lucy",18), Person("bob",20)};
     (*(arr2+0)).showPerson();
     arr2[0].showPerson();
     (arr2+1)->showPerson();
     delete [] arr2;
}
```

4、尽量不要用delete释放void *

```
void test09()
{
    Person *p = new Person("lucy",18);
    p->showPerson();

    void *p1 = p;

    delete p1;

}
int main(int argc, ch
```

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
有参构造
name = lucy, num = 18
```

注意：没有析构？为啥?

delete发现p1指向的类型 为void 无法从void中寻找响应析构函数

```
void test09()
{
    Person *p = new Person("lucy",18);
    p->showPerson();

    void *p1 = p;

    delete p;

}
int main(int argc, cha
{
    test09();
```

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
有参构造
name = lucy, num = 18
析构函数
```

5、malloc、free和new、delete 不可以混搭使用

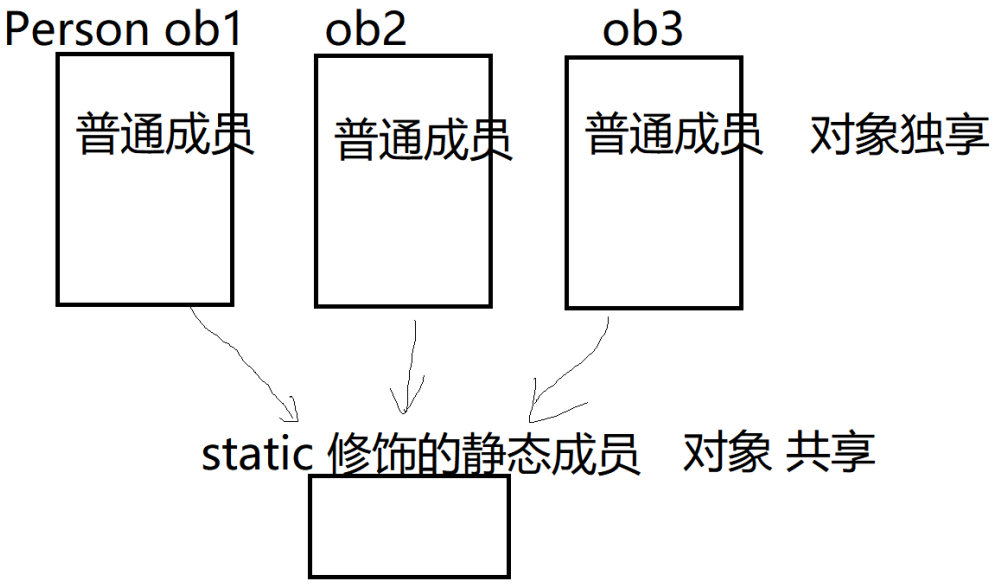
知识点3【静态成员】static修饰的成员

成员:成员变量 成员函数

static 修饰成员变量 修饰成员函数

static声明为静态的，称为静态成员。 不管这个类创建了多少个对象，静态成员只有一个拷贝，这个拷贝被所有属于这个类的对象共享。

静态成员 属于类 而不是对象。



1、static 修饰成员变量

静态变量，是在编译阶段就分配空间，对象还没有创建时，就已经分配空间。

静态成员变量必须在类中声明，在类外定义。

静态数据成员不属于某个对象，在为对象分配空间中不包括静态成员所占空间。

```
class Data
{
public:
 int num;//普通成员变量
 static int data;//静态成员变量(类内声明)
};
//定义的时候 不需要加static
int Data::data=100;//类外定义+初始化

 void test01()
 {
  //data是静态成员变量 是属于类 可以通过类名称::直接访问
  cout<<Data::data<<endl;//100
  //赋值
  Data::data = 200;
```

```
 cout<<Data::data<<endl;//200

 //data静态变量 是所有对象 共享的 可以通过对象名访问
 Data ob1;
 ob1.data = 300;
 cout<<Data::data<<endl;//300

 Data ob2;
 cout<<ob2.data<<endl;//300

 //普通成员变量 属于对象的 只能通过对象名访问
 ob1.num = 100;
 cout<<"ob2.num = "<<ob2.num<<endl;//随机值
 //cout<<Data::num<<endl;//普通成员变量不能通过类名称访问
}
```

2、static修饰静态成员函数

引出：

```
class Data
{
private:
 int num;//普通成员变量
 static int data;//静态成员变量(类内声明)
public:
 //普通成员函数 依赖于 对象的 必须对象调用
 int getData(void)
 {
  return data;
  }
};

//定义的时候 不需要加static
int Data::data=100;//类外定义+初始化

void test01()
{
 //cout<<Data::data<<endl;//err 静态data是私有的 类外不能直接访问
 //cout<< Data::getData()<<endl;//err getData() 必须对象调用

 Data ob;
 cout<<ob.getData()<<endl;
```

```
 //存在问题：data静态的 在创建对象之前 就已经存在
 //如果类没有实例化对象 难道 就不能使用data了吗？
 //解决上述问题 就要用到静态成员函数
}
```

静态成员函数：

```
class Data
{
private:
 int num;//普通成员变量
 static int data;//静态成员变量(类内声明)
public:
 //普通成员函数 依赖于 对象的 必须对象调用
 int getData(void)
 {
  return data;
  }

 //静态成员函数 属于类 而不属于对象
 static int getDataStatic(void)
 {
 return data;
 }
};

//定义的时候 不需要加static
int Data::data=100;//类外定义+初始化

void test01()
{
 //cout<<Data::data<<endl;//err 静态data是私有的 类外不能直接访问
 //cout<< Data::getData()<<endl;//err getData() 必须对象调用

 Data ob;
 cout<<ob.getData()<<endl;
 //存在问题：data静态的 在创建对象之前 就已经存在
 //如果类没有实例化对象 难道 就不能使用data了吗？
 //解决上述问题 就要用到静态成员函数

 //1、静态成员函数 属于类 就可以通过类名称直接访问
 cout<<Data::getDataStatic()<<endl;
```

```
36
37   //2、也可以通过对象名访问（对象共享静态成员函数）
38   cout<<ob.getDataStatic()<<endl;
39  }
```

注意：

1、静态成员函数的目的 操作静态成员数据。

2、静态成员函数 不能访问 非静态成员数据。（静态成员函数内部没有this指针）

```
//静态成员函数   属于类 而不属于对象
static int getDataStatic(void)
{
    //num = 200;//err 静态成员函数 不能访问普通成员变量
    return data;
}
```

3、普通成员函数 可以操作 静态成员数据 非静态成员数据。

```
private:
    int num;//普通成员变量
    static int data;//静态成员变量(类内声明)
public:
    //普通成员函数 依赖于 对象的 必须对象调用
    int getData(void)
    {
        num = 200;//non-static
        return data;//static
    }
```

4、静态成员变量 和 静态成员函数 都有权限之分。

3、const 修饰静态成员

如果一个类的成员，既要实现共享，又要实现不可改变，那就用 **static const** 修饰

```
1  class Data
2  {
3  public:
4   const static int data;//静态成员变量(类内声明)
5  public:
6   //静态成员函数 属于类 而不属于对象
7   static int getDataStatic(void)
8   {
```

```
 //num = 200;//err 静态成员函数 不能访问普通成员变量
  return data;
  }
};

//定义的时候 不需要加static
const int Data::data=100;//类外定义+初始化

void test02()
{
 //访问
 cout<<Data::data<<endl;
 //赋值
 //Data::data = 200;//err data静态成员只读
 cout<<Data::data<<endl;
}
```

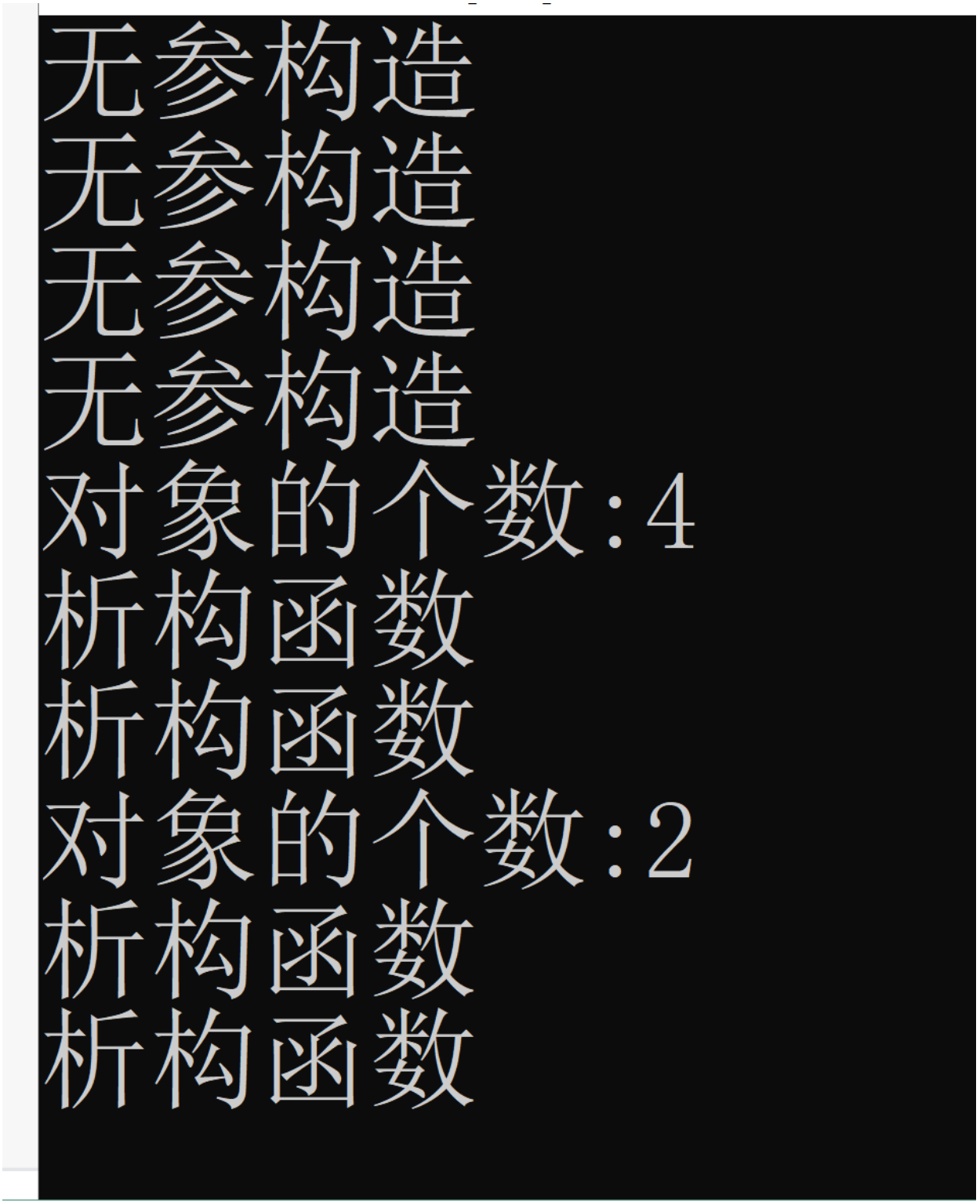
知识点4【静态成员案例】

案例1：静态成员 统计类 实例化对象的 个数

```
#include <iostream>

using namespace std;
class Data
{
public:
 Data()
 {
 cout<<"无参构造"<<endl;
  count++;
  }
  Data(const Data &ob)
  {
  cout<<"拷贝构造函数"<<endl;
  count++;
  }

  ~Data()
  {
  count--;
  cout<<"析构函数"<<endl;
```

```
 }

 static int count;
};

int Data::count = 0;

int main(int argc, char *argv[])
{
 Data ob1;
 Data ob2;
 {
 Data ob3;
 Data ob4;
 cout<<"对象的个数:"<<Data::count<<endl;
 }
 cout<<"对象的个数:"<<Data::count<<endl;
 return 0;
}

```

运行结果：



案例2：单例模式设计--打印机（重要）

步骤1：在单例类内部定义了一个**Singleton**类型的静态对象，作为外部共享的唯一实例

步骤2：提供一个公共静态的方法，让客户可以访问它的唯一实例。

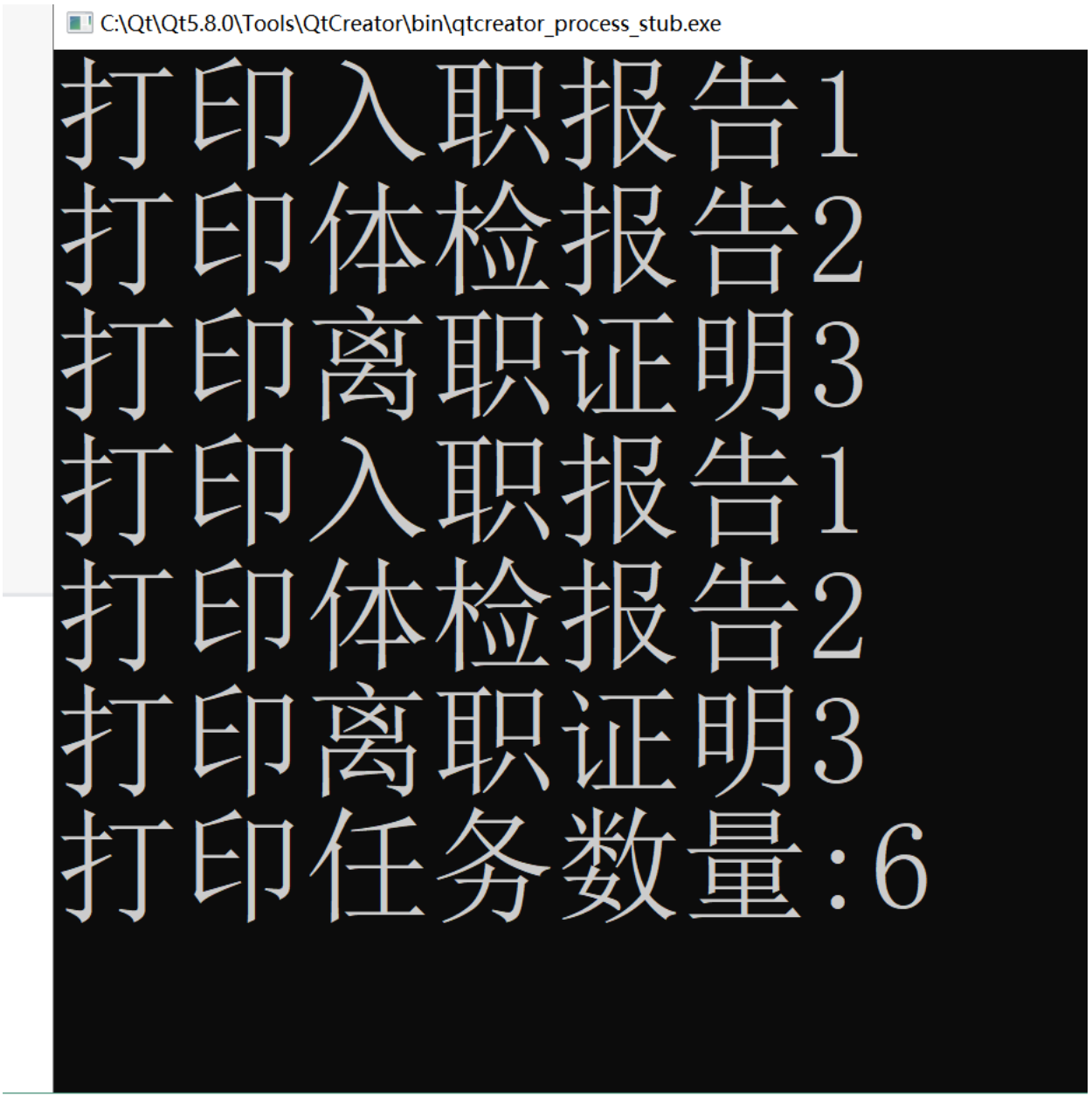
步骤3：为了防止在外部对实例化其他对象，将其默认构造函数和拷贝构造函数设计为私有

```
#include <iostream>

using namespace std;
class Printer
```

```
{
public:
 //2、提供一个方法 获得单例指针
 static Printer* getSignlePrint(void)
 {
  return signlePrint;
  }

  //4、设置功能函数(自定义)
  void printText(char *str)
  {
  cout<<"打印"<<str<<endl;
  count++;

  }
  int count;
private:
 //1、定义一个静态的 对象指针变量 保存唯一实例地址
 static Printer *signlePrint;
private:
 //3、防止 该类实例化其他对象 将构造函数全部 私有
 Printer(){count=0;}
 Printer(const Printer &ob){}

};
Printer *Printer::signlePrint = new Printer;

int main(int argc, char *argv[])
{
 //打印任务1
 Printer *p1 = Printer::getSignlePrint();
 p1->printText("入职报告1");
 p1->printText("体检报告2");
 p1->printText("离职证明3");

 //打印任务2
 Printer *p2 = Printer::getSignlePrint();
 p2->printText("入职报告1");
 p2->printText("体检报告2");
 p2->printText("离职证明3");
```

```
45
46      cout<<"打印任务数量:"<<p2->count<<endl;
47      return 0;
48  }
49
```

运行结果：



知识点5【this指针】（重要）

c++的封装性：将数据 和 方法 封装在一起

数据 和 方法 是分开存储。

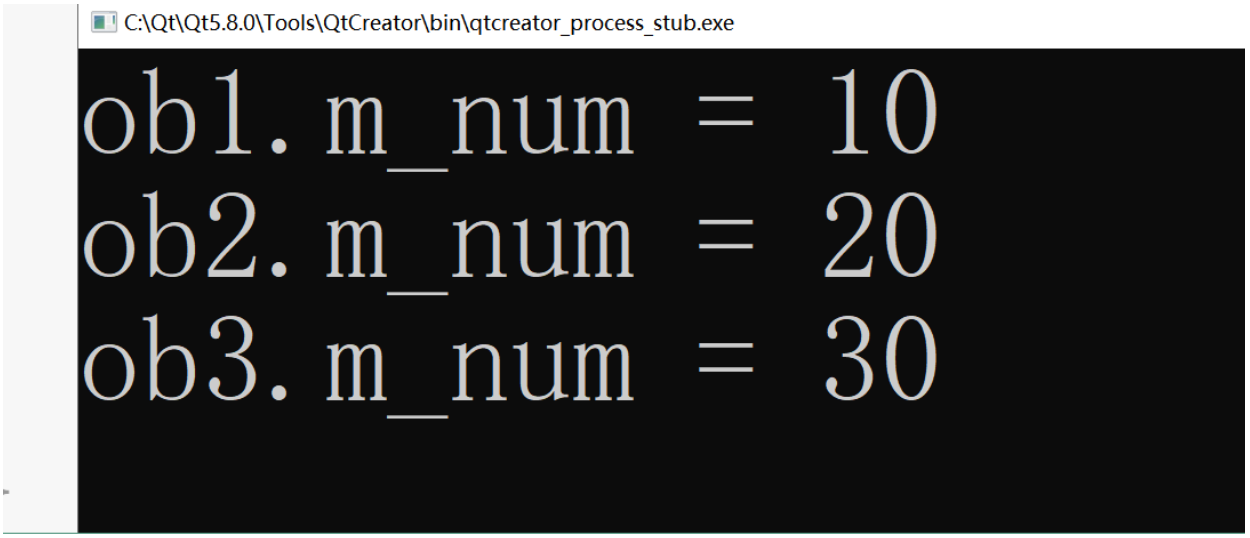
每个对象 拥有独立的数据.

每个对象 共享同一个方法。

1、this指针的引入

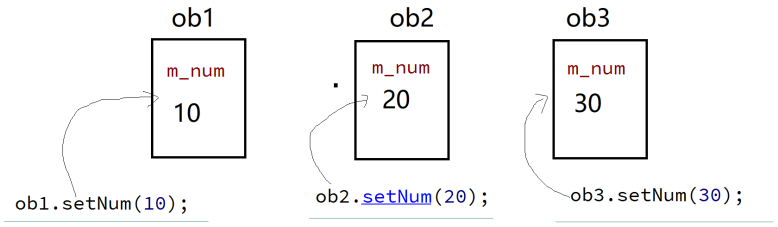
```
class Data
{
public:
 int m_num;

 void setNum(int num)
 {
 m_num = num;
 }

};

void test01()
{
 Data ob1;
 ob1.setNum(10);
 cout<<"ob1.m_num = "<<ob1.m_num<<endl;

 Data ob2;
 ob2.setNum(20);
 cout<<"ob2.m_num = "<<ob2.m_num<<endl;

 Data ob3;
 ob3.setNum(30);
 cout<<"ob3.m_num = "<<ob3.m_num<<endl;
}
```

运行结果：



值得思考：

```
class Data
{
public:
    int m_num;

    void setNum(int num)
    {
        m_num = num;
    }
};
```



数据 是对象独有

方法 是对象共享

this = &ob1

代码区

```
void setNum(int num)
{
    this->m_num = num;
    m_num = num;
}
```

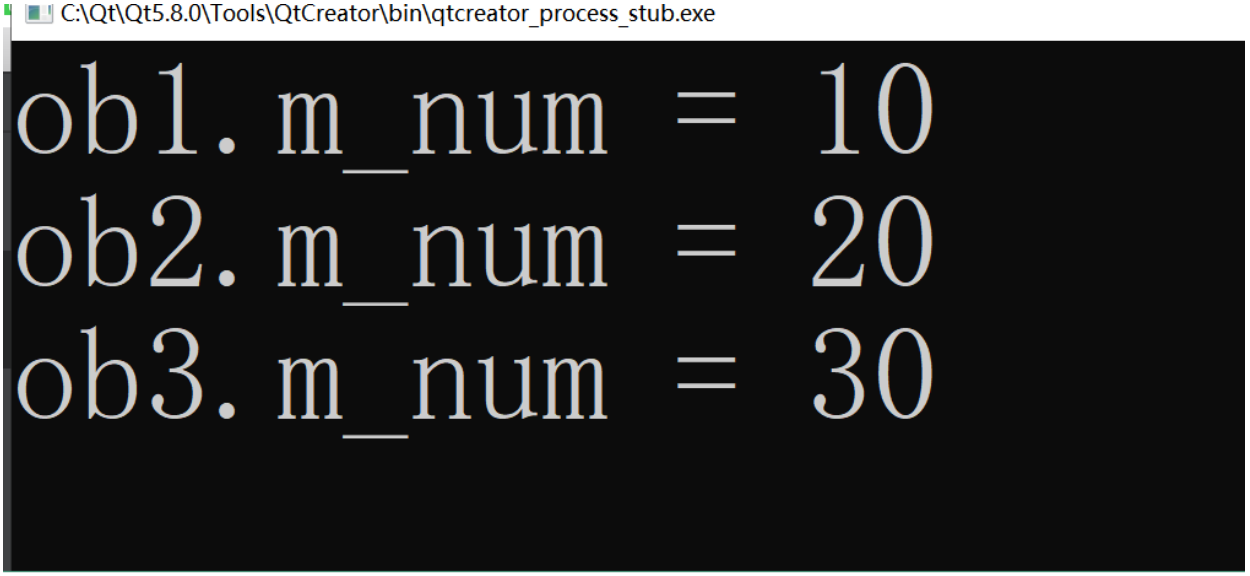
当一个对象 调用setNum方法时 会在setNum方法中产生一个this指针，this指向所调用方法的对象

哪个对象调用方法，那么方法中的this 就指向哪个对象

```
class Data
{
public:
 int m_num;

 void setNum(int num)
 {
 //m_num = num;
 this->m_num = num;
  }
};

void test01()
{
 Data ob1;
```

```
16  ob1.setNum(10);
17  cout<<"ob1.m_num = "<<ob1.m_num<<endl;
18
19  Data ob2;
20  ob2.setNum(20);
21  cout<<"ob2.m_num = "<<ob2.m_num<<endl;
22
23  Data ob3;
24  ob3.setNum(30);
25  cout<<"ob3.m_num = "<<ob3.m_num<<endl;
26 }
```

运行结果：



注意：

1、this指针是隐含在对象成员函数内的一种指针

2、成员函数通过**this**指针即可知道操作的是那个对象的数据

3、静态成员函数内部没有this指针，静态成员函数不能操作非静态成员变量

(静态成员函数 是属于类 函数内部 没有this指针)

知识点6【this指针的使用】

1、当形参和成员变量同名时，可用this指针来区分

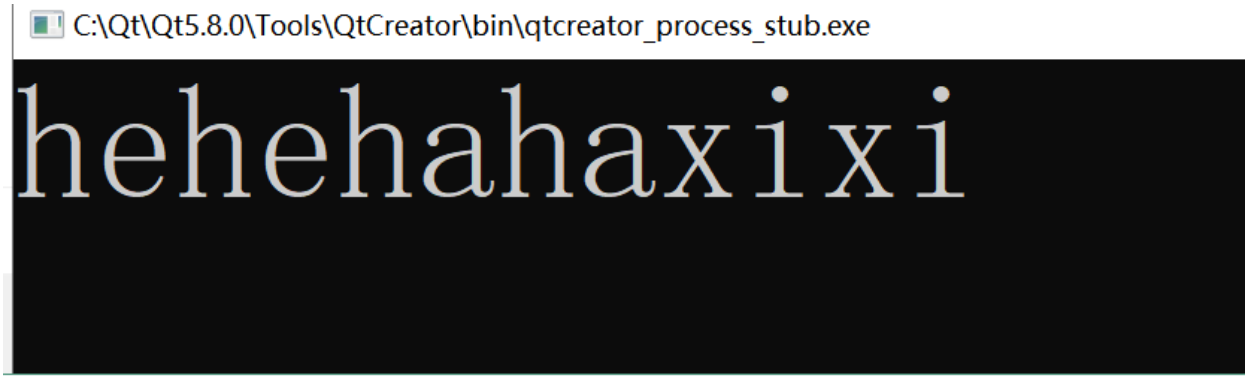
```
1  class Data
2  {
3  public:
4   int num;
5
6   //形参 和成员名相同
```

```
 void setNum(int num)
 {
 //形参num
  //成员num this->num
  //num= num;//就近原则 形参num 赋值给 形参num
  this->num = num;
  //将形参num 赋值给 对象中成员num
  }
};
```

2、在类的普通成员函数中返回对象本身（*this）（重要）

```
class MyCout
{
public:
 MyCout& myCout(char *str)
 {
 cout<<str;
 return *this;//*this 代表就是 当前调用该函数的对象
 }
};
int main(int argc, char *argv[])
{
 MyCout ob;
 /*
 ob.myCout("hehe");
 ob.myCout("haha");
 ob.myCout("xixi");*/

 ob.myCout("hehe").myCout("haha").myCout("xixi");//ob.myCout("hehe") == ob

 return 0;
}
```

运行结果：


C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
hehehahaxixi