第二章 C++封装（上）

一、类和对象

1.基本概念

(1)类的定义

```
class Dog;//关键字class，Dog是类名
{
   char name[20];
   int age;
   int type;//以上数据成员（属性）
   void speak();
   void run();//以上成员函数（方法）
};
```

结论：目的的不同，抽象出的信息也完全不同

封装：选择性暴露，把实现细节封装起来，只暴露需要了解的

```
class TV;
{
   char name[20];
   int type;
   void changeVol();
   void power();//以上希望暴露
  // 电阻调节;
   //像素配色;  （希望隐藏）
};
```

(2)访问限定符

public：公共的

protected：受保护的

private：私有的

```
class TV;
{
public:
   char name[20];
   int type;
   void changeVol();
   void power();//以上希望暴露
private:
  // 电阻调节;
   //像素配色; （希望隐藏）
};
```

2.对象实例化

(1)从栈实例化

```
class TV;
{
public:
   char name[20];
   int type;

   void changeVol();
   void power();
};
int main()
{
  TV tv;
  TV tv[20];
  return 0;
}
```

(2)从堆实例化

```
class TV;
{
public:
   char name[20];
   int type;

   void changeVol();
   void power();
};
int main()
```

```
{
  TV *p=new TV();
  TV *q=new TV[20];
  //todo
  delete p;
  delete []q;//数组释放内存
  return 0;
}
```

3.对象成员的访问

(1)通过栈实例化的对象

```
class TV;
    {
    public:
       char name[20];
       int type;

       void changeVol();
       void power();
    };
    int main()
    {
       TV tv;//用点来访问
       tv.type;
       tv.changeVol();
       return 0;
    }
```

(2)通过堆实例化的对象

```
class TV;
    {
    public:
       char name[20];
       int type;

       void changeVol();
       void power();
    };
    int main()
    {
       TV *p=new TV();
       p->type=0;//调用自己的成员
```

```
        p->changeVol();//调用自己的成员函数
        delete p;
        p=NULL;
        return 0;
    }
```

(3)对象数组的访问

```
int main()
    {
        TV *p=new TV[5];
        for(int i=0;i<5;i++)
        {
            p[i]->type=0;//调用自己的成员
            p[i]->changeVol();//调用自己的成员函数
        }
        delete []p;
        p=NULL;
        return 0;
    }
```

4.编码示例如何进行对象实例化

```
#include<iostream>
#include<stdlib.h>
using namespace std;
//定义一个坐标类
class Coordinate
{
public:
    int x;
    int y;
    void printX()
    {
      cout<<x<<endl;
    }
    void printY()
    {
      cout<<y<<endl;
    }
};
int main()
{
  Coordinate coor;
  coor.x=10;
```

```
  coor.y=20;
  coor.printX();
  coor.printY();//栈的方式
  Coordinate *p=new Coordinate();
  if(NULL==p)
  {
    //failed;//解决内存new失败
    return 0;
  }
  p->x=100;
  p->y=200;
  p->printX();
  p->printY();//堆的方式
  delete p;
  p=NULL;
  return 0;
}
```

二、初始String

1.String类型

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
  string name="ZhangSan";
  string hobby("football");
  cout<<name<<hobby<<endl;
  return 0;
}
```

2.初始化String对象的方式

```
string s1;            //s1为空串
strign s2("ABC");     //用字符串字面值初始化s2
string s3(s2);        //将s3初始化为s2的一个副本
string s4(n,'c');     //将s4初始化为字符'c'的n个副本
```

3.String的常用操作

```
s.empty();          //若s为空串，则返回true，否则返回false
s.size();           //返回s中字符的个数
s[n];               //返回s中位置为n的字符，位置从0开始
s1+s2;              //将两个串连接成新串，返回新生成的串
s1=s2;              //把s1的内容替换为s2的副本
v1==v2;             //判断相等，相等返回true，否则返回false
v1!=v2;             //判断不等，不等返回true，否则返回false
```

程序示例：

```
string s1="hello";
string s2("world");
string s3=s1+s2;
string s4="hello"+s2;
string s5="hello"+s2+"world";
string s6="hello"+"world";      //错误
```

编码实例：

```
#include<iostream>
#include<stdlib.h>
#include<string>
using namespace std;
int main()
{
  string name;
  cout<<"Please input your name:";
  getline(cin,name);
  if(name.empty()){
    cout<<"Input is null..."<<endl;
    return 0;
  }
  if(name=="imooc")
  {
    cout<<"You are a administraor"<<endl;
  }
  cout<<"Hello "+name<<endl;
  cout<<"Your name length :"<<name.size()<<endl;//字符串，但是长度不会是，用+号这样连接有问题
  cout<<"Your name first letter is:"<<name[0]<<endl;
  return 0;
}
```

练习题：定义一个Student类，包含名字和年龄两个数据成员，实例化一个Student对象，并打印出其成两个数据成员

```cpp
#include <iostream>
#include <string>
using namespace std;

/**
  * 定义类：Student
  * 数据成员：名字、年龄
  */
  class Student
{
public:
    // 定义数据成员名字 m_strName 和年龄 m_iAge
    string m_strName;
    int m_iAge;
};

int main()
{
    // 实例化一个Student对象stu
    Student stu;
    // 设置对象的数据成员
    stu.m_strName = "慕课网";
    stu.m_iAge = 2;

    // 通过cout打印stu对象的数据成员
    cout << stu.m_strName<< " " <<stu.m_iAge<< endl;
    return 0;
}
```

或者：

```cpp
int main()
{
    // 实例化一个Student对象stu
    Student *stu=new Student;
    // 设置对象的数据成员
    stu->m_strName = "慕课网";
    stu->m_iAge = 2;

    // 通过cout打印stu对象的数据成员
    cout << stu.m_strName<< " " <<stu.m_iAge<< endl;
    delete stu;
    stu=NULL;
    return 0;
}
```

三、数据的封装

```
class Student
{
   public:
   string name;
   int age;
   ......
};
int main()
{
  Student stu;
  stu.name="Jim";
  stu.age=10;
  return 0;
}
```

有问题：违背了面向对象的指导思想：谁做什么

1.数据封装

```
class Student
{
   public:
     void setAge(int _age){age=_age;}
     int getAge(){return age;}
   private:
     string name;
     int age;
   ......
};
```

2.数据封装的好处

```
class Student
{
   public:
     void setAge(int _age){age=_age;}
     int getAge(){return age;}
   private:
     string name;
     int age;
   ......
};
```

```
  void setAge(int _age)//条件限制
  {
    if(_age>0&&_age<100)
    {
       age=_age;
    }
    else
    {
      ......
    }
   }
  int main()
  {
    Student stu;
    stu.name="Jim";
    stu.age=10;
    return 0;
  }
```

当希望数据只能读不能写时：只读属性

```
  class Car
  {
     public:
       int getWheelCount(){return m_iWheelCount;}
     private:
       int m_iWheelCount;//只读属性
     ......
  };
```

定义一个Student类，包含如下信息：姓名name，性别gender，学分（只读）score，学习study 代码示例：

```
//定义一个Student类，包含如下信息：姓名name，性别gender，学分（只读）score，学习study
#include<iostream>
#include<stdlib.h>
#include<string>
using namespace std;
class Student
{
public:
  void setName(string _name)
  {
    m_strName=_name;
  }
  string getName()
```

```
  {
    return m_strName;
  }
  void setGender(string _gender)
  {
    m_strGender=_gender;
  }
  string getGender()
  {
    return m_strGender;
  }
  int getScore()
  {
    return m_iScore;//只读
  }
  void initScore()//初始化
 {
    m_iScore=0;
 }
 void study(int _score)
 {
   m_iScore+=_score;//m_iScore=m_iScore+_score;
 }
private:
  string m_strName;
  string m_strGender;
  int m_iScore;
};
int main()
{
  Student stu;
  stu.initScore();//必须有初始值
  stu.setName("zhangsan");
  stu.setGender("女");
  stu.study(5);
  stu.study(3);
  cout<<stu.getName()<<" "<<stu.getGender()<<" "<<stu.getScore()<<endl;
  return 0;
}
```

数据封装练习题：定义一个Student类，包含名字一个数据成员，使用get和set函数封装名字这个数据成员。在main函数中通过new实例化对象，并打印其相关函数。

```
#include <iostream>
#include <string>
```

```
using namespace std;

/**
  * 定义类：Student
  * 数据成员：m_strName
  * 数据成员的封装函数：setName()、getName()
  */
class Student
{
public:
    // 定义数据成员封装函数setName()
    void setName(string _name)
    {
        m_strName=_name;
    }
    // 定义数据成员封装函数getName()
    string getName()
    {
        return m_strName;
    }
    //定义Student类私有数据成员m_strName
private:
   string m_strName;
};

int main()
{
    // 使用new关键字，实例化对象
        Student *stu =new Student;
    // 设置对象的数据成员
        stu->setName("慕课网");
    // 使用cout打印对象str的数据成员
    cout<<stu->getName()<<endl;
    // 将对象str的内存释放，并将其置空
    delete stu;
    stu=NULL;
        return 0;
}
```

四、类内定义与内联函数

1.关于内联函数inline

关键字：inline

```
inline void fun()
{
     cout<<"Hello"<<endl;
}
```

2.内联函数与普通函数的区别

普通函数：main()——¿调用fun()——¿结束

内联函数：编译时将函数体代码和实参代替函数调用语句，必须是结构和逻辑都比较简单的代码，复杂的代码即使加上inline，编译器也会拒绝将其按照内联函数的方式进行编译

3.类内定义

```
class Student
{
  public:
       void setAge(int _age){age=_age}
       int getAge(){return age;}
       void study(){//todo}//编译器优先将其编译成内联函数
  private:
       string name;
       int age;
 };
```

4.类外定义

1）同文件类外定义

car.cpp

```
class Car
{
 public:
 void run();
 void stop();
 void changeSpeed();
};
void Car::run(){}//函数属于car汽车的函数，Car::(属于汽车类)
void Car::stop(){}
void Car::changeSpeed(){}
```

2）分文件类外定义

先定义car.h头文件，在有car.cpp,将car.h包含在car.cpp中

5.同类外定义代码演示

```
#include <iostream>
#include <stdlib.h>
using namespace std;
//Teacher类
//数据成员：名字，年龄，性别
//成员函数：数据成员的封装函数，授课teach
class Teacher
{
public:
  void setName(string _name);
  string getName();
  void setGender(string _gender);
  string getGender();
  void setAge(int _age);
  int getAge();
  void teach();
private:
  string m_strName;
  string m_strGender;
  int m_iAge;
};
//类外定义
//名字
void Teacher::setName(string _name)
{
  m_strName=_name;
}
string Teacher::getName()
{
  return m_strName;
}
//年龄
void Teacher::setGender(string _gender)
{
  m_strGender=_gender;
}
string Teacher::getGender()
{
  return m_strGender;
}
//性别
```

```
void Teacher::setAge(int _age)
{
  m_iAge=_age;
}
int Teacher::getAge()
{
  return m_iAge;
}
//授课teach
void Teacher::teach()
{
  cout<<"现在上课"<<endl;
}
int main()
{
  Teacher t;
  t.setName("孔子");
  t.setGender("男");
  t.setAge(30);
  cout<<t.getName()<<" "<<t.getAge()<<" "<<t.getGender();
  t.teach();
  cout<<endl;
  return 0;
}
```

五、构造函数

1.对象结构

内存分区：

1）栈区：int x=0;int *p=NULL;

2）堆区：int *P=new int[20];

3）全局区：存储全局变量及静态变量

4）常量区：string str=“hello”;

5）代码区：存储逻辑代码的二进制

2.对象初始化

```
class Tank
{
 private:
   int m_iPosX;
   int m_iPosY;
 public:
   void init()//初始化函数
   {
     m_iPosX=0;
     m_iPosY=0;
   }
};
int main()
{
  Tank t1;
  t1.init();
  Tank t2;
  t2.init();
  return 0;
}
```

对象初始化：

1）有且仅有一次

思考：初始化函数如何避免误操作，包括忘记调用了初始化函数，重复调用了初始化函数

2）根据条件初始化

3.构造函数

构造函数的规则和特点：

1）构造函数在对象实例化时被自动调用；只需将初始化函数写在构造函数内就可以实现初始化

) 2）构造函数与类同名

3）构造函数没有返回值

4）构造函数可以有多个重载形式

5）实例化对象时仅用到一个构造函数

6）当用户没有定义构造函数时，编译器会自动生成一个构造函数

4.构造函数的定义

1）无参构造函数

```
class Student
{
  public:
     Student()//无参构造函数
     {
       m_strName="Jim";
     }
  private:
     string m_strName;
 };
```

2）有参构造函数

```
class Student
{
  public:
     Student(string name)//有参构造函数
     {
       m_strName=name;
     }
  private:
     string m_strName;
 };
```

3）重载构造函数

```
class Student
{
  public:
     Student(){m_strName="Jim";}   //符合重载规则即可
     Student(string name)//有参构造函数
     {
       m_strName=name;
     }
  private:
     string m_strName;
 };
```

5.构造函数编码示例

构造函数编码示例：

```
#include <iostream>
#include <stdlib.h>
#include <string>
using namespace std;
//Teacher类
//自定义无参构造函数
//自定义有参构造函数
//数据：名字，年龄
//成员函数：数据成员的封装函数
class Teacher
{
public:
  //声明构造函数
  Teacher();
  Teacher(string _name,int _age);
  void setName(string _name);
  string getName();
  void setAge(int _age);
  int getAge();
private:
  string m_strName;
  int m_iAge;
};
//无参构造函数
Teacher::Teacher()
{
  m_strName="Jim";
  m_iAge=5;
  cout<<"Teacher()"<<endl;
}
//有参构造函数
Teacher::Teacher(string _name,int _age=20)
{
  m_strName=_name;
  m_iAge=_age;
  cout<<"Teacher(string _name,int _age)"<<endl;
}
//数据成员的封装函数
void Teacher::setName(string _name)
{
  m_strName=_name;
}
```

```
string Teacher::getName()
{
  return m_strName;
}
//
void Teacher::setAge(int _age)
{
  m_iAge=_age;
}
int Teacher::getAge()
{
  return m_iAge;
}
//
int main()
{
  Teacher t1;
  Teacher t2("Merry",15);
  Teacher t3("James");//默认20

  cout<<t1.getName()<<" "<<t1.getAge()<<endl;//通过初始化实现
  cout<<t2.getName()<<" "<<t2.getAge()<<endl;//通过传参实现
  cout<<t3.getName()<<" "<<t3.getAge()<<endl;//可以使用默认值
  //如果给名字也赋默认值，计算机无法分辨调用哪个构造函数

  return 0;
}
```

6.默认构造函数

```
int main()
{
  Student stu1();//调用的构造函数都不用传参数
  Student *p=NULL;
  p=new Student();//调用的构造函数都不用传参数
  return 0;
}
class Student
{
 public:
  Student(){}
  或者Student(string name="Jim")
 private:
  string m_strName;
};
```

7.构造函数函数初始化列表

```
class Student
{
 public:
  Student():m_strName("Jim"),m_iAge(10){}//初始化，用冒号隔开，赋值用括号，中间用逗号隔开
 private:
  string m_strName;
  int m_iAge;
};
```

构造函数初始化列表的特性：

1）初始化列表先于构造函数执行，意味着编译器先给初始化列表的数据成员赋值，再执行构造函数

2）初始化列表只能用于构造函数

3）初始化列表可以同时初始化多个数据成员

初始化列表存在的必要性：

```
class Circle
{
  public:
    Circle(){m_dPi=3.14}//error,编译器报错
  private:
    const double m_dPi;//const修饰的常量，不能给它再赋值
};
```

修改为用初始化列表来完成：

```
class Circle
{
  public:
    Circle():m_dPi(3.14){}
  private:
    const double m_dPi;
};
```

8.初始化列表编码示例

```
#include<iostream>
#include<stdlib.h>
#include<string>
```

```
using namespace std;
//Teacher类
//自定义有参默认构造函数
//使用初始化列表初始化数据
//数据：名字，年龄
//成员函数：数据成员的封装函数
//拓展：定义可以带最多学生的个数，此为常量
class Teacher
{
public:
  //声明有参默认构造函数
  Teacher(string _name="Jim",int _age=1,int m=100);
  void setName(string _name);
  string getName();
  void setAge(int _age);
  int getAge();
  int getMax();
private:
  string m_strName;
  int m_iAge;
  const int m_iMax;
};
//有参构造函数
/*Teacher::Teacher(string _name,int _age)
{
  m_strName=_name;
  m_iAge=_age;
  cout<<"Teacher(string _name,int _age)"<<endl;
}*/
//使用初始化列表
Teacher::Teacher(string _name,int _age,int m):m_strName(_name),m_iAge(_age),m_iMax(m)
{
  cout<<"Teacher(string _name,int _age)"<<endl;
}
//数据成员的封装函数
void Teacher::setName(string _name)
{
  m_strName=_name;
}
string Teacher::getName()
{
  return m_strName;
}
//
void Teacher::setAge(int _age)
{
```

```
  m_iAge=_age;
}
int Teacher::getAge()
{
  return m_iAge;
}
//
int Teacher::getMax()
{
  return m_iMax;
}
int main()
{
  Teacher t1;
  Teacher t2("Merry",12,150);
  cout<<t1.getName()<<" "<<t1.getAge()<<endl;
  cout<<t2.getName()<<" "<<t2.getAge()<<" "<<t2.getMax()<<endl;
  return 0;
}
```

六、拷贝构造函数

1.定义

定义格式：类名（const 类名 & 变量名）

```
class Student
{
 public:
  Student(){m_strName="Jim";}
  Student(const Student & stu){}//拷贝构造函数
 private:
  string m_strName;
};
```

1）如果没有自定义的拷贝构造函数则系统自动生成一个默认的拷贝构造函数

2）当采用直接初始化或复制初始化实例化对象时系统自动调用拷贝构造函数

2.拷贝构造函数代码示例

```
#include <iostream>
#include <string>
using namespace std;
//Teacher类
//自定义拷贝构造函数
//数据：名字，年龄
//成员函数：数据成员的封装函数
class Teacher
{
public:
  //声明有参默认构造函数
  Teacher(string _name="Jim",int _age=1);
  Teacher(const Teacher & tea);
  void setName(string _name);
  string getName();
  void setAge(int _age);
  int getAge();
private:
  string m_strName;
  int m_iAge;
};
Teacher::Teacher(string _name,int _age):m_strName(_name),m_iAge(_age)
{
  cout<<"Teacher(string _name,int _age)"<<endl;
}
//拷贝构造函数
Teacher::Teacher(const Teacher &tea)
{
  cout<<"Teacher(const Teacher &tea)"<<endl;
}
//数据成员的封装函数
void Teacher::setName(string _name)
{
  m_strName=_name;
}
string Teacher::getName()
{
  return m_strName;
}
//
void Teacher::setAge(int _age)
{
  m_iAge=_age;
```

```
}
int Teacher::getAge()
{
  return m_iAge;
}
void test(Teacher t)
{

}
int main()
{
  Teacher t1;
  test(t1);
  return 0;
}
```

七、析构函数

1.定义

定义格式：～类名（）

```
class Student
{
 public:
  Student(){cout<<"Student"<<endl;}
  ~Student(){cout<<"Student"<<endl;}//析构函数,析构函数不允许加任何参数
 private:
  string m_strName;
};
```

思考：析构函数有存在的必要性吗?

```
class Student
{
 public:
  Student(){m_pName=new char[20];}
  ~Student(){delete[]m_pName;}//释放内存，唯一功能
 private:
  char *m_pName;
};
```

1）如果没有自定义的析构函数则系统自动生成

2）析构函数在对象销毁时被自动调用，与其相对应的构造函数则是在对象实例化时被自动调用

3）析构函数没有返回值，没有参数也不能重载

2.拷贝构造函数代码示例

```
#include<iostream>
#include<stdlib.h>
#include<string>
using namespace std;
//Teacher类
//自定义析构函数
//普通方式实例化的对象，在销毁时是否自动调用析构函数
//通过拷贝构造函数实例化对象，在销毁时是否自动调用析构函数
//数据：名字，年龄
//成员函数：数据成员的封装函数
class Teacher
{
public:
  //声明有参默认构造函数
  Teacher(string _name="Jim",int _age=1);
  Teacher(const Teacher & tea);
  ~Teacher();
  void setName(string _name);
  string getName();
  void setAge(int _age);
  int getAge();
private:
  string m_strName;
  int m_iAge;
};
Teacher::Teacher(string _name,int _age):m_strName(_name),m_iAge(_age)
{
  cout<<"Teacher(string _name,int _age)"<<endl;
}
//拷贝构造函数
Teacher::Teacher(const Teacher &tea)
{
  cout<<"Teacher(const Teacher &tea)"<<endl;
}
//析构函数
Teacher::~Teacher()
{
  cout<<"~Teacher()"<<endl;
```

```
}
//数据成员的封装函数
void Teacher::setName(string _name)
{
  m_strName=_name;
}
string Teacher::getName()
{
  return m_strName;
}
//
void Teacher::setAge(int _age)
{
  m_iAge=_age;
}
int Teacher::getAge()
{
  return m_iAge;
}
int main()
{
  Teacher t1;
  //Teacher *p=new Teacher();
  //delete p;//打印两次
  Teacher t2(t1);//分别执行t1和t2的析构函数
  return 0;
}
```

综合练习：定义一个Student类，包含名字一个数据成员，定义无参构造函数、有参构造函数、拷贝构造函数、析构函数及对于名字的封装函数，在main函数中实例化Student对象，并访问相关函数，观察运行结果。

```
#include <iostream>
#include <string>
using namespace std;
/**
 * 定义类：Student
 * 数据成员：m_strName
 * 无参构造函数：Student()
 * 有参构造函数：Student(string _name)
 * 拷贝构造函数：Student(const Student& stu)
 * 析构函数：~Student()
 * 数据成员函数：setName(string _name)、getName()
 */
class student
{
```

```
public:
    student(){m_strName=" ";}
        student(string _name){m_strName=_name;}
        student(const student& stu){};
        ~student(){};
        void setName(string _name){m_strName=_name;}
        string getName(){return m_strName;}
private:
        string m_strName;
};

int main(void)
{
    // 通过new方式实例化对象*stu
    student *stu = new student();
    // 更改对象的数据成员为\慕课网"
    stu->setName("慕课网");
    // 打印对象的数据成员
        cout<<stu->getName()<<endl;
        delete stu;
        stu=NULL;
        return 0;
}
```