



C++ Programming

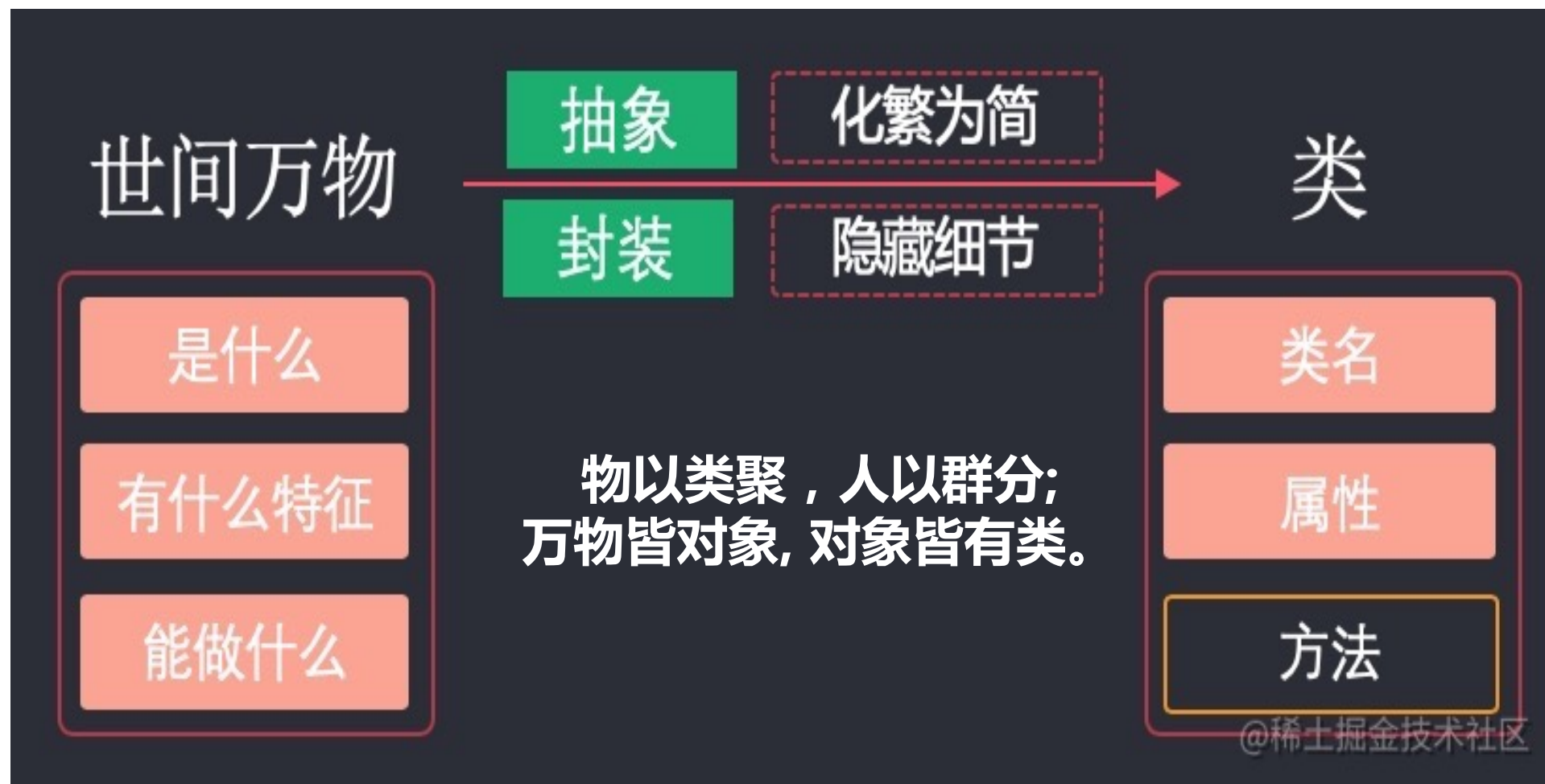
多态性与虚函数

Polymorphism & Virtual Functions

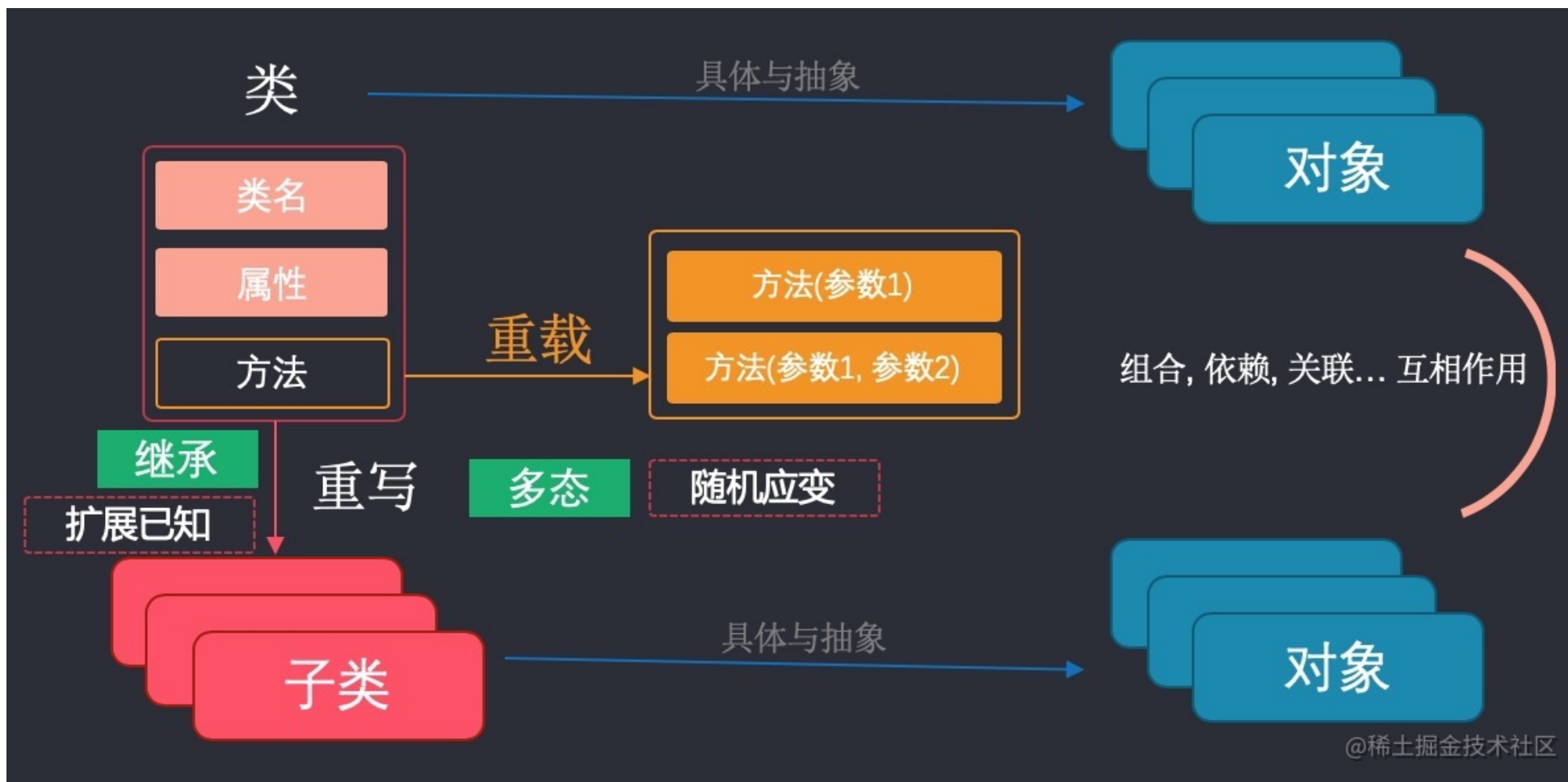
2023年5月10日

学而不厌 诲人不倦

面向对象编程：抽象、封装、信息隐藏



面向对象编程：模板、继承、多态



@稀土掘金技术社区

- ➡ **6.1 多态性的概念**
- ➡ **6.2 多态的典型实例**
- ➡ **6.3 虚函数**
- ➡ **6.4 纯虚函数与抽象类**



6.1 多态性的概念

➤ 多态的概念

多态 (polymorphism) 指的是同一名字的事物可以完成不同的功能。

多态可以分为编译时的多态和运行时的多态。

1. 编译时的多态 (**静态多态**) 主要是指**函数的重载** (包括运算符的重载)、对重载函数的调用，在编译时就能根据实参确定应该调用哪个函数；
2. 运行时的多态 (**动态多态**) 则和**继承、虚函数**等概念有关，是本章内容。



6.1 多态性的概念

➤ Demo01 :

```
#include <iostream>
using namespace std;
//基类People
class People{
public:
    People(string name, int age);
    void display();
protected:
    string m_name;
    int m_age;
};
People::People(string name, int age): m_name(name), m_age(age){}
void People::display(){
    cout<<m_name<<"今年"<<m_age<<"岁了，是个无业游民。"<<endl;
}
```



6.1 多态性的概念

➤ Demo01 :

```
//派生类Teacher
class Teacher: public People{
public:
    Teacher(string name, int age, int salary);
    void display();
private:
    int m_salary;
};
Teacher::Teacher(string name, int age, int salary): People(name, age),
m_salary(salary){}
void Teacher::display(){
    cout<<m_name<<"今年"<<m_age<<"岁了，是一名教师，每月有"<<m_salary<<"元
的收入。"<<endl;
}
```

6.1 多态性的概念

➤ Demo01 :

```
int main(){  
    People *p = new People("王志刚", 23);  
    p -> display();  
  
    p = new Teacher("赵宏佳", 45, 8200);  
    p -> display();  
    return 0;  
}
```

运行结果：

王志刚今年23岁了，是个无业游民。

赵宏佳今年45岁了，是个无业游民。

6.1 多态性的概念

➤ Demo02 :

```
#include <iostream>
using namespace std;
//基类People
class People{
public:
    People(string name, int age);
    virtual void display();
protected:
    string m_name;
    int m_age;
};
People::People(string name, int age): m_name(name), m_age(age){}
void People::display(){
    cout<<m_name<<"今年"<<m_age<<"岁了, 是个无业游民。"<<endl;
}
```



6.1 多态性的概念

➤ Demo01 :

```
//派生类Teacher
class Teacher: public People{
public:
    Teacher(string name, int age, int salary);
    void display();
private:
    int m_salary;
};
Teacher::Teacher(string name, int age, int salary): People(name, age),
m_salary(salary){}
void Teacher::display(){
    cout<<m_name<<"今年"<<m_age<<"岁了，是一名教师，每月有"<<m_salary<<"元
的收入。"<<endl;
}
```

6.1 多态性的概念

➤ Demo02 :

```
int main(){  
    People *p = new People("王志刚", 23);  
    p -> display();  
  
    p = new Teacher("赵宏佳", 45, 8200);  
    p -> display();  
    return 0;  
}
```

运行结果：

王志刚今年23岁了，是个无业游民。

赵宏佳今年45岁了，是一名教师，每月有8200元的收入。



6.1 多态性的概念

➤ 多态的用途

有了虚函数，基类指针指向基类对象时就使用基类的成员（包括成员函数和成员变量），指向派生类对象时就使用派生类的成员。

换句话说，基类指针可以按照基类的方式来做事情，也可以按照派生类的方式来做事情，它有多种形态，或者说有多种表现方式，我们将这种现象称为多态（Polymorphism）。

6.1 多态性的概念

➤ 多态的用途：Demo03

```
#include <iostream>
using namespace std;
//军队
class Troops{
public:
    virtual void fight(){ cout<<"Strike back!"<<endl; }
};
//陆军
class Army: public Troops{
public:
    void fight(){ cout<<"--Army is fighting!"<<endl; }
};
```



6.1 多态性的概念

➤ 多态的用途：Demo03

```
//99A主战坦克
class _99A: public Army{
public:
    void fight() { cout<<"----99A(Tank) is fighting!"<<endl; }
};

//武直10武装直升机
class WZ_10: public Army{
public:
    void fight() { cout<<"----WZ-10(Helicopter) is fighting!"<<endl; }
};

//长剑10巡航导弹
class CJ_10: public Army{
public:
    void fight() { cout<<"----CJ-10(Missile) is fighting!"<<endl; }
};
```



6.1 多态性的概念

➤ 多态的用途：Demo03

```
//空军
class AirForce: public Troops{
public:
    void fight(){ cout<<"--AirForce is fighting!"<<endl; }
};
//J-20隐形歼击机
class J_20: public AirForce{
public:
    void fight(){ cout<<"----J-20(Fighter Plane) is fighting!"<<endl; }
};
//CH5无人机
class CH_5: public AirForce{
public:
    void fight(){ cout<<"----CH-5(UAV) is fighting!"<<endl; }
};
//轰6K轰炸机
class H_6K: public AirForce{
public:
    void fight(){ cout<<"----H-6K(Bomber) is fighting!"<<endl; }
};
```

6.1 多态性的概念

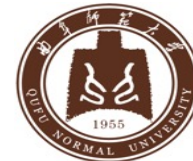
➤ 多态的用途：Demo03

```
int main() {  
    Troops *p = new Troops;  
    p ->fight();  
    //陆军  
    p = new Army;  
    p ->fight();  
    p = new _99A;  
    p -> fight();  
    p = new WZ_10;  
    p -> fight();  
    p = new CJ_10;  
    p -> fight();  
}
```

```
    //空军  
    p = new AirForce;  
    p -> fight();  
    p = new J_20;  
    p -> fight();  
    p = new CH_5;  
    p -> fight();  
    p = new H_6K;  
    p -> fight();  
    return 0;  
}
```


- ➡ 6.1 多态性的概念
- ➡ 6.2 多态的典型实例
- ➡ 6.3 虚函数
- ➡ 6.4 纯虚函数与抽象类

6.2 多态的典型实例



➤ Demo04 Point类

```
#include <iostream>
using namespace std;
class Point
{
protected:
    float x, y;

public:
    Point(float = 0, float = 0);
    void setPoint(float, float);
    float getX() const { return x; }
    float getY() const { return y; }
    friend ostream
    &operator<<(ostream &, const
    Point &);
};
```

```
// Point的构造函数
Point::Point(float a, float b)
{
    x = a;
    y = b;
}
// 设置x和y的坐标值
void Point::setPoint(float a, float b)
{
    x = a;
    y = b;
}
// 输出点的坐标
ostream &operator<<(ostream &output, const
Point &p)
{
    output << "[" << p.x << "," << p.y <<
    "]" << endl;
    return output;
}
```



6.2 多态的典型实例

➤ Demo04 Point类

```
int main()
{
    Point p(3.5, 6.4);
    cout << "x=" << p.getX() << ",y=" << p.getY() << endl;
    p.setPoint(8.5, 6.8);
    cout << "p(new):" << p << endl;
    return 0;
}
```



6.2 多态的典型实例

➤ Demo05 Circle类

```
class Circle : public Point
{
protected:
    float radius;

public:
    Circle(float x = 0, float y = 0, float r = 0);
    void setRadius(float);
    float getRadius() const;
    float area() const;
    friend ostream &operator<<(ostream &, const Circle &);
};
```



6.2 多态的典型实例

➤ Demo05 Circle类

```
Circle::Circle(float a, float b, float r) : Point(a, b), radius(r) {}

void Circle::setRadius(float r) { radius = r; }
float Circle::getRadius() const { return radius; }
float Circle::area() const
{
    return 3.14159 * radius * radius;
}
ostream &operator<<(ostream &output, const Circle &c)
{
    output << "Center=[" << c.x << "," << c.y << "], Radius=" << c.radius <<
    ", area=" << c.area() << endl;
    return output;
}
```



6.2 多态的典型实例

➤ Demo05 Circle类

```
int main()
{
    Circle c(3.5, 6.4, 5.2);
    cout << "original circle:\n x=" << c.getX() << ", y=" << c.getY()
    << ", r = " << c.getRadius() << ", area = " << c.area() << endl;
    c.setRadius(7.5);
    c.setPoint(5, 5);
    cout << "new circle:\n" << c;
    Point &pRef = c;
    cout << "pRef:" << pRef;
    return 0;
}
```



6.2 多态的典型实例

➤ Demo06 Cylinder类

```
class Cylinder : public Circle
{
public:
    Cylinder(float x = 0, float y = 0, float r = 0, float h = 0);
    void setHeight(float);
    float getHeight() const;
    float area() const;
    float volume() const;
    friend ostream &operator<<(ostream &, const Cylinder &);

protected:
    float height;
};
```



6.2 多态的典型实例

➤ Demo06 Cylinder类

```
Cylinder::Cylinder(float a, float b, float r, float h)
: Circle(a, b, r), height(h) {}

void Cylinder::setHeight(float h) { height = h; }
float Cylinder::getHeight() const { return height; }
float Cylinder::area() const
{
    return 2 * Circle::area() + 2 * 3.14159 * radius * height;
}
float Cylinder::volume() const {return Circle::area() * height;}
ostream &operator<<(ostream &output, const Cylinder &cy)
{
    output << "Center=[" << cy.x << ", " << cy.y << "], r=" << cy.radius << ",
h=" << cy.height << " \narea=" << cy.area() << ", volume=" << cy.volume()
<< endl;
    return output;
}
```




6.2 多态的典型实例

➤ Demo06 Cylinder类

```
int main()
{
    Cylinder cy1(3.5, 6.4, 5.2, 10);
    cout << "\n original cylinder:\n x=" << cy1.getX() << ", y="
    << cy1.getY() << ", r=" << cy1.getRadius() << ", h=" <<
    cy1.getHeight() << "\narea=" << cy1.area()<< ", volume=" <<
    cy1.volume() << endl;
    cy1.setHeight(15);
    cy1.setRadius(7.5);
    cy1.setPoint(5, 5);
    cout << "\nnew cylinder:\n" << cy1;
    Point &pRef = cy1;
    cout << "\npRef as a point:" << pRef;
    Circle &cRef = cy1;
    cout << "\ncRef as a Circle:" << cRef;
    return 0;
}
```

Chapter 6 多态性与虚函数

- ➡ 6.1 多态性的概念
- ➡ 6.2 多态的典型实例
- ➡ 6.3 虚函数
- ➡ 6.4 纯虚函数与抽象类

多态的典型实例

1. 设计基类Point
2. 设计Point基类的派生类Circle类
3. 从Circle类派生Cylinder类

实验作业四 C++ 类的设计与实现 (题目三选一)

- 1. 题目1：泛化的链表类设计与实现**：设计泛化的链表类LinkedList，每个节点包含data和next两个域，data用来存取各种不同类型数据，next为指向节点的指针，实现链表元素的增加、删除、查找、修改等功能。
- 2. 题目2：图书馆系统读者类设计与实现**：图书馆系统有两类读者：学生读者、教师读者。每位读者信息包括卡号，姓名、单位、已借阅数量、已借阅记录。
- 3. 围绕C++类的继承和多态等技术点设计自选题目。**
4. 认真按格式撰写实验作业报告，补充目的、原理等各部分内容；
5. 准备演讲PPT；

2023年5月8日-2023年5月23日



Thank You !

Q & A