

第三讲 类和对象（三）

一、类和对象的使用

二、const关键字

三、this指针

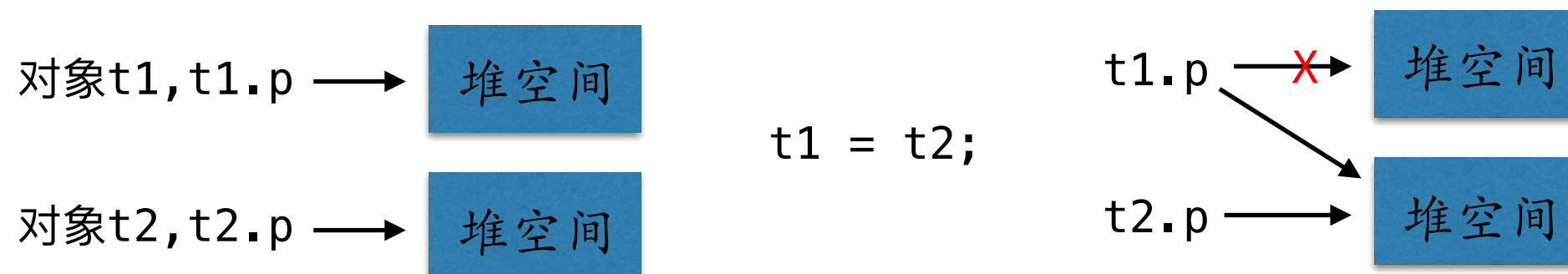
一、类和对象的使用

和结构体一样，一个类的对象也可以以值、指针或者引用的形式作为函数的参数或返回值。一般情况下，为了避免创建临时对象通常传对象的引用，并且为了避免在函数体内修改对象的值，通常使用常引用。

```
const Point middle(const Point& p1, const Point& p2);
```

如果函数以传值形式返回一个类的对象，编译器会创建一个临时对象来保存这个值。但有些编译器会做优化。

浅拷贝：可以把一个对象赋值给另外一个对象，对象的每个成员的值，将一对一的拷贝到新的对象。这种拷贝叫逻辑拷贝，或浅拷贝。但是，如果对象含有指针成员变量，而指针变量又指向堆上空间，将只拷贝指针成员变量本身的价值，造成两个对象的指针指向同一块堆上的内存空间，删除对象时将造成二次删除。



【例1-1】 类和对象的使用

```
/** Point.h */  
#ifndef Point_h  
#define Point_h  
  
#include <iostream>  
using namespace std;  
  
class Point  
{  
public:  
    Point(float a = 0, float b = 0);  
    float get_x( ) const ;  
    float get_y( ) const;  
    void move(float a, float b); //偏移量  
    void print ( ) const;  
  
private:  
    float x;  
    float y; //一个点有一个x坐标和y坐标  
};  
#endif /* Point_h */
```

```
/** Point.cpp */
#include "Point.h"

Point::Point(float a, float b):x(a),y(b) {
}

float Point::get_x( ) const {
    return x;
}

float Point::get_y( ) const {
    return y;
}

//偏移量
void Point::move(float a, float b) {
    x += a;
    y += b;
}

void Point::print ( ) const {
    cout << "("<< x << ", " << y << ")"<<endl;
}
```

```
/** main.cpp Part-1 */  
#include <math.h>  
#include "Point.h"  
  
float distance(const Point& p1, const Point& p2) {  
    float dx = p1.get_x() - p2.get_x();  
    float dy = p2.get_y() - p2.get_y();  
    return sqrt(dx*dx+dy*dy);  
}  
  
const Point middle(const Point& p1, const Point& p2) {  
    return Point((p1.get_x()+p2.get_x())/2, (p1.get_y()+p2.get_y())/2);  
}  
  
int main(int argc, const char * argv[]) {  
    Point p;  
    Point q(1.0, 2.0);  
  
    float dx, dy;  
    cout<<"第一个点的坐标为:";  
    p.print();  
  
    cout<<"第二个点的坐标为:";  
    q.print();  
    return 0;  
}
```

```
/** main.cpp Part-2 */  
int main(int argc, const char * argv[]) {  
    q = p; //经过赋值，两个点都在原点位置 浅拷贝  
    cout<<"赋值之后，第二个点的坐标为:";  
    q.print();  
    cout<<endl;  
    //测试move函数  
    cout<<"从键盘输入两个值，分别表示第一个点x、y方向的偏移量:";  
    cin>>dx>>dy; //从键盘输入数据到dx和dy中，相当于C语言中的scanf函数  
    p.move(dx, dy); //通过对象p调用成员函数move，对p进行移动  
  
    cout<<"移动之后第一个点的坐标为:";  
    p.print();  
    cout<<endl;  
  
    cout<<"从键盘输入两个值，分别表示第二个点x、y方向的偏移量:";  
    cin>>dx>>dy;  
  
    q.move(dx, dy);  
    cout<<"移动之后第二个点的坐标为:";  
    q.print();  
    cout<<endl;  
    return 0;  
}
```

```
/** main.cpp Part-3 */  
int main(int argc, const char * argv[]) {  
    //测试distance函数  
    cout<<"两个点之间的距离为:"<<distance(p, q)<<endl;  
  
    //测试middle函数  
    cout<<"两个点的中间点坐标为:";  
    middle(p, q).print();  
    cout<<endl;  
    return 0;  
}
```

示例分析:

- (1) 访问函数get_x, get_y以及print函数均为const成员函数;
- (2) distance和middle函数为了避免传参时的拷贝和函数体内对实参的修改, 指定形参为常引用类型;

(3) middle函数返回了一个const Point对象。middle函数也可以使用如下方式进行实现：

```
const Point middle(const Point& p1,const Point& p2) {  
    float mid_x = (p1.get_x()+p2.get_x())/2;  
    float mid_y = (p1.get_y()+p2.get_y())/2;  
    Point local_p(mid_x,mid_y);  
    return local_p;  
}
```

但是需要注意middle函数的返回值不能是传引用，const Point& middle(...), 因为不能返回指向局部变量的引用。

(4) 利用middle函数的返回值（一个点对象）再调用print成员函数。这被称作“函数链”；

(5) 在main函数中，声明了两个对象p和q，用p给q赋值，实现的是“逻辑拷贝”，又称“浅拷贝”；

二、const 关键字

1.1 什么是const?

const是C/C++语言中保留的一个关键字，它用来限定一个变量是只读的，即不可变的。程序中使用const可以在一定程度上提高程序的健壮性。

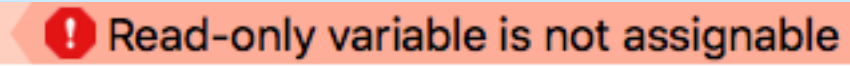
1.2 const的使用

(1) 定义const常量：常量意味着初始化完成后，其值不能再被修改。

```
#include <stdio.h>

int main(int argc, const char * argv[]) {

    const float pi = 3.14;
    pi = 3.14159; //错误
    return 0;
}
```



(2) const和指针

- 常量指针：不能通过指针修改指针所指向的变量的值。但是指针可以指向别的变量。

```
int a = 5;  
const int *p = &a;  
*p = 20;    ✗    //不能通过指针修改指向的变量的值  
  
int b = 10;  
p = &b;      ✓    //指针可以指向别的变量
```

- 指向常量的指针（指针常量）：指针常量的值不能被修改，即不能存一个新的地址，不能指向别的变量。但是可以通过指针修改它所指向的变量的值。

```
int a = 5;
int *const p = &a;
*p = 20;    ✓    //可以通过指针修改指向的变量的值

int b = 10;
p = &b;    ✗    //指针不能指向别的变量
```

总结：

- (1) const修饰谁，谁不可变；如 `int *const p`；修饰指针p则p不可变；
- (2) 从前向后读（翻译），`const int *p`；常量指针；
- (3) 谁在前谁不可变，如指针常量则指针不可变，常量指针则指针指向的值不可变；

(3) const和函数: const在函数中根据修饰位置分三种,

```
const int func(const int a) const;
```

- 修饰返回值 `const int func();` 不能修改返回值。
- 修饰函数形参 `int func(const int a);` 函数体内不能修改形参a的值。

```
int func(const int a){  
    a += 3; //在函数体内不能修改形参的值  
    return a;  
}
```

! Read-only variable is not assignable

- 修饰类的成员函数 `int func() const;` 函数体内不能修改成员变量的值, 增加程序的健壮性或鲁棒性。

【例2-1】const成员函数

```
/** Point.h */  
#ifndef Point_h  
#define Point_h  
  
#include <iostream>  
using namespace std;  
  
class Point  
{  
public:  
    Point(float a = 0, float b = 0);  
    float get_x( ) const ;  
    float get_y( ) const;  
    void move(float a, float b); //偏移量  
    void print ( ) const; //const成员函数  
  
private:  
    float x;  
    float y; //一个点有一个x坐标和y坐标  
};  
#endif /* Point_h */
```

```
/** Point.cpp */
#include "Point.h"

Point::Point(float a, float b):x(a),y(b) {
}

float Point::get_x( ) const {
    return x;
}

float Point::get_y( ) const {
    return y;
}

//偏移量
void Point::move(float a, float b) {
    x += a;
    y += b;
}

void Point::print ( ) const {
    cout << "(" << x << ", " << y << ")";
}
```

(4) const对象: `const Point p;`

在示例2-1中加入下面的主函数代码:

```
#include <iostream>
#include <math.h>
#include "Point.h"
using namespace std;

int main(int argc, const char * argv[]){
    const Point p;
    p.print(); //正确

    p.move(1,1); //错误

    return 0;
}
```

总结: const对象只能调用const成员函数不能调用普通成员函数;
而普通对象既可以调用const成员函数也可以调用普通成员函数。

三、this指针

每一个对象都“包含”一个指针指向对象自身，称之为this指针。

【例3-1】 this指针使用示例

```
/** Circle.h **/  
#ifndef Circle_h  
#define Circle_h  
  
#include <iostream>  
using namespace std;  
class Circle  
{  
public:  
    Circle();  
    void printRadiusOnly();  
  
private:  
    float radius;  
    float x_centre;  
    float y_centre;  
};  
#endif /* Circle_h */
```



```
/** Circle.cpp */
#include "Circle.h"

Circle::Circle() {
    radius = 10.0;
    x_centre = 0.0;
    y_centre = 0.0;
}

void Circle::printRadiusOnly() {
    //隐式使用this指针
    cout<<"radius "<<radius<<endl;

    cout<<"this->radius "<<this->radius<<endl;

    cout<<"(*this).radius "<<(*this).radius<<endl;

    //测试this指针的值
    cout<<"this指针的值: "<<this<<endl;
}
```

```
/** main.cpp */  
  
#include <iostream>  
#include "Circle.h"  
  
int main(int argc, const char * argv[]) {  
    Circle c;  
    c.printRadiusOnly();  
  
    cout<<"对象c的地址: "<<&c<<endl;  
    return 0;  
}
```

程序运行结果如下:

```
radius 10  
this->radius 10  
(*this).radius 10  
this指针的值: 0x7fff5fbff790  
对象c的地址: 0x7fff5fbff790
```

示例解析:

(1) 通过输出结果可知this指针的值就是对象c的地址。

(2) 当对象调用成员函数时, 会默认将对象自身传递给该函数, 而在函数体中不直接使用对象名, 而是使用this指针, 即this指针指向该对象, 指向调用者。哪个对象调用该函数, 那么this指针就指向谁。

有两种情况需要显式使用this指针：

- 1、在类的成员函数体中返回对象本身时，使用return *this;
- 2、当形参与成员变量同名时，需显示使用this指针。如：this->radius=radius;

【例3-2】 显式使用this指针

```
/** Circle.h */
#ifndef __test__Circle__
#define __test__Circle__
#include <iostream>
using namespace std;
class Circle
{
public:
    Circle(float radius=0.0, float x=0.0, float y=0.0);
    Circle& setRadius(float r);
    Circle& setX(float x);
    Circle& setY(float y);
    void print()const;

private:
    float radius;
    float x;
    float y;
};
#endif
```

```
/** Circle.cpp */
#include "Circle.h"
Circle::Circle(float radius, float x, float y) {
    this->radius = radius; //形参与实例变量同名, 需显示使用this指针
    this->x = x;
    (*this).y = y;
}

Circle& Circle::setRadius(float r) {
    radius = r;
    return *this;
}

Circle& Circle::setX(float x) {
    this->x = x;
    return *this;
}

Circle& Circle::setY(float y) {
    this->y = y;
    return *this;
}

void Circle::print() const {
    cout<<x<<" "<<y<<" "<<radius<<endl;
}
```

```
/** main.cpp */  
#include <iostream>  
using namespace std;  
#include "Circle.h"  
  
int main(int argc, const char * argv[]) {  
    Circle c;  
    c.print();  
  
    c.setRadius(10.0).setX(7.0).setY(8.0);  
    c.print();  
    return 0;  
}
```

程序运行结果如下：

0.0 0.0 0.0

7.0 8.0 10.0

示例分析：

(1) `c.setRadius(10.0).setX(7.0).setY(8.0);` 为函数链；

(2) 函数 `setRadius`、`setX`、`setY` 的返回值以及类型均为 `*this` 与引用类型，则修改的都是同一个对象 `c` 的信息；

(3) 如果上述三个函数传值返回，则只能修改对象 `c` 的半径，圆心坐标无法被修改；