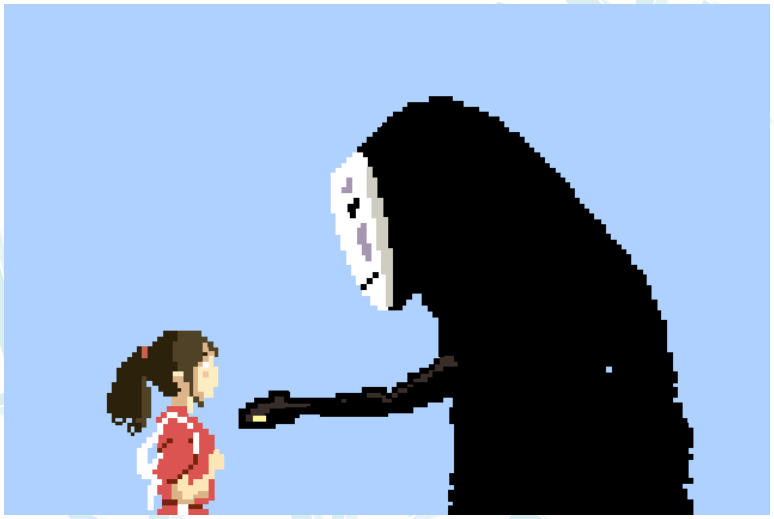
C/C++程序设计案例实战

——浅拷贝与深拷贝

华中农业大学信息学院　李小霞

像素画



案例：像素点

```
#include <iostream>
#include<cstring>
using namespace std;
class Pixel{
```

定义像素点类

案例：像素点

```
#include <iostream>
#include<cstring>
using namespace std;
class Pixel{
private:
    char *name;//像素点名字指针
    int x,y;//像素点坐标
```

定义像素点类

案例：像素点

```
#include <iostream>
#include<cstring>
using namespace std;
class Pixel{
private:
    char *name;//像素点名字指针
    int x,y;//像素点坐标
public:
    Pixel(char *nm="noname",int xx=0,int yy=0){
        name=nm; x=xx; y=yy;}
    void show(){cout<<x<<" "<<y<<" "<<name<<endl;}
};
```

定义像素点类

问题引入：

```
int main()
{
    char str[10]="Tom";
    Pixel px1(str);
    px1.show();
    strcpy(str,"Winnie");
    px1.show();
    return 0;
}
```

名字被修改

问题引入：

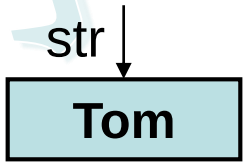
```
int main()
{
    char str[10]="Tom";
    Pixel px1(str);
    px1.show();
    strcpy(str,"Winnie");
    px1.show();
    return 0;
}
```

对象浑然不知

```
0 0 Tom
0 0 Winnie
```

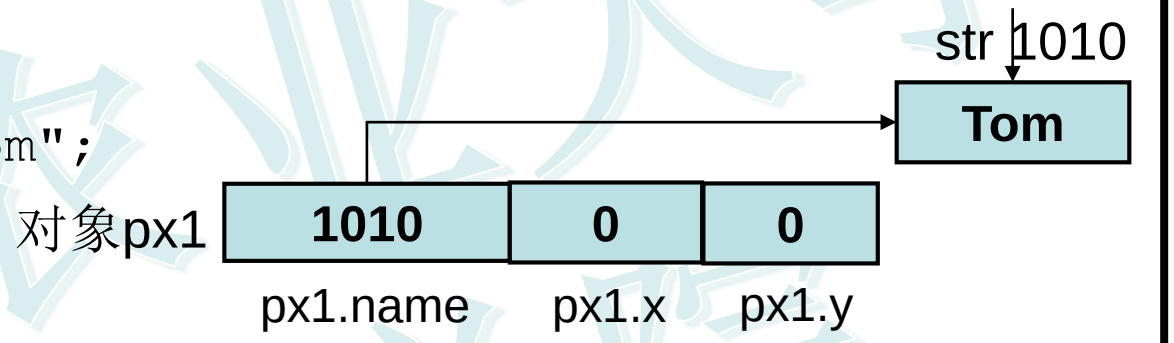
案例分析：

```
int main()
{
    char str[10]="Tom";
```



案例分析：

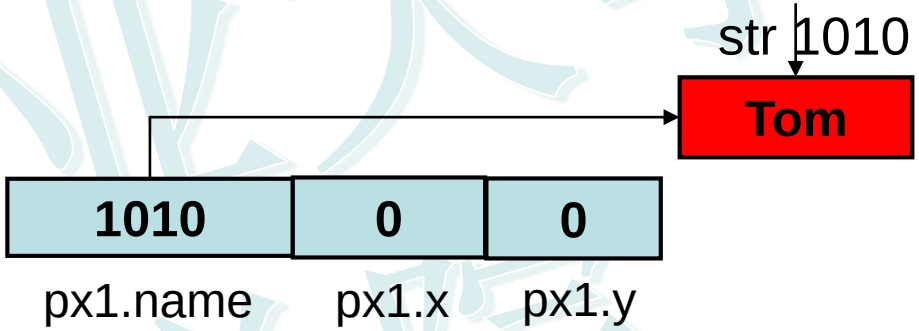
```
int main()
{
    char str[10]="Tom";
    Pixel px1(str);
```




```
Pixel(char *nm="noname",int xx=0,int yy=0){
    name=nm; x=xx; y=yy;}
```

案例分析：

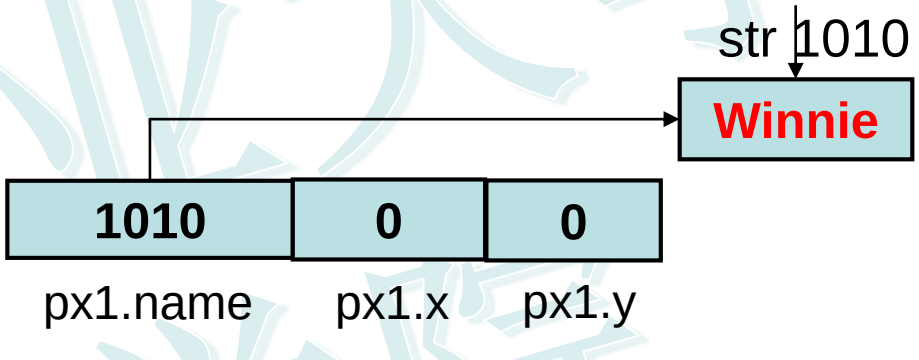
```
int main()
{
    char str[10]="Tom";
    Pixel px1(str);
    px1.show();
}
```




```
0 0 Tom
```

案例分析：

```
int main()
{
    char str[10]="Tom";
    Pixel px1(str);
    px1.show();
    strcpy(str,"Winnie");
```

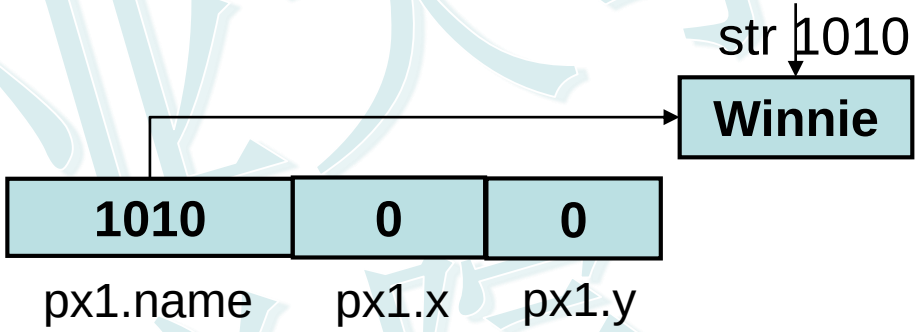



```
0 0 Tom
```

案例分析：

```
int main()
{
    char str[10]="Tom";
    Pixel px1(str);
    px1.show();
    strcpy(str,"Winnie");
    px1.show();
    return 0;
}
```

对象px1

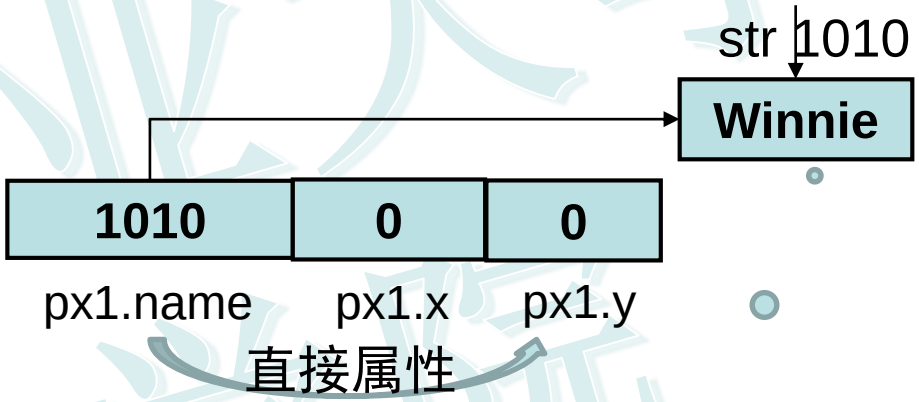



```
0 0 Tom
0 0 Winnie
```

案例分析：

```
int main()
{
    char str[10]="Tom";
    Pixel px1(str);
    px1.show();
    strcpy(str,"Winnie");
    px1.show();
    return 0;
}
```

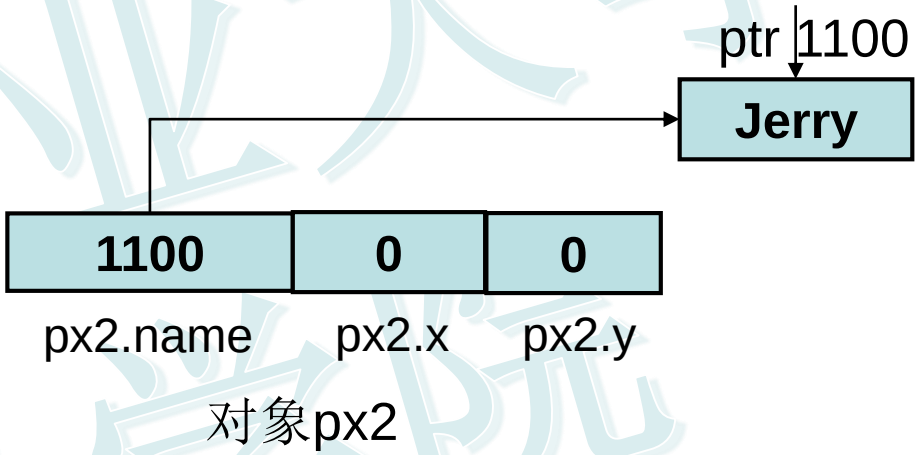
对象px1



对象的间接属性

案例进阶

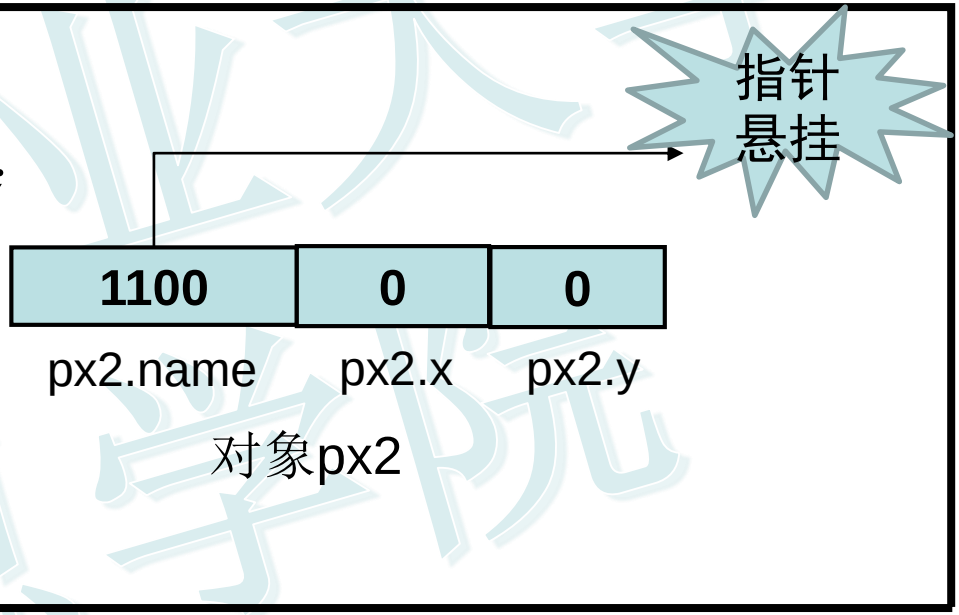
```
int main()
{
    char *ptr=new char[10];
    strcpy(ptr,"Jerry");
    Pixel px2(ptr);
    px2.show();

}
```




```
0 0 Jerry
```

案例分析：

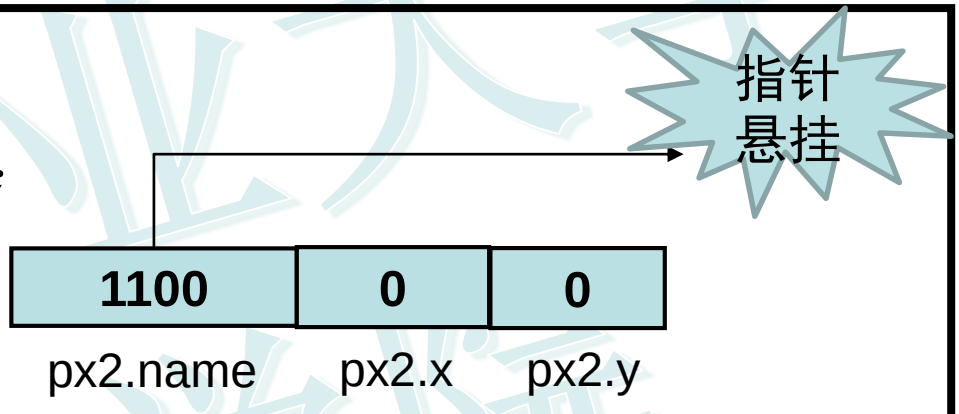
```
int main()
{
    char *ptr=new char[10];
    strcpy(ptr,"Jerry");
    Pixel px2(ptr);
    px2.show();
    delete [] ptr;

}
```




```
0 0 Jerry
```

案例分析：

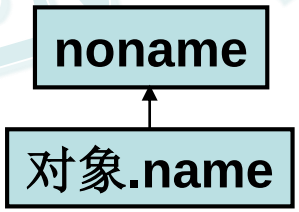
```
int main()
{
    char *ptr=new char[10];
    strcpy(ptr,"Jerry");
    Pixel px2(ptr);
    px2.show();
    delete [] ptr;
    px2.show();
    return 0;
}
```




```
0 0 Jerry
0 0 `&
```

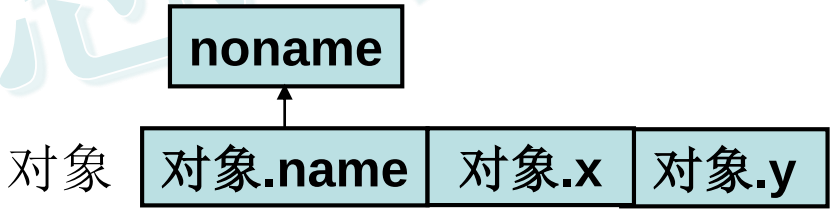
```
#include <iostream>
#include<cstring>
using namespace std;
class Pixel{
private:
    char *name;//像素点名字指针
    int x,y;//像素点坐标
public:
    Pixel(char *nm="noname",int xx=0,int yy=0){
        name=new char[strlen(nm)+1]; strcpy(name,nm);
```

修改构造函数



```
#include <iostream>
#include<cstring>
using namespace std;
class Pixel{
private:
    char *name;//像素点名字指针
    int x,y;//像素点坐标
public:
    Pixel(char *nm="noname",int xx=0,int yy=0){
        name=new char[strlen(nm)+1]; strcpy(name,nm);
        x=xx; y=yy;}
```

修改构造函数

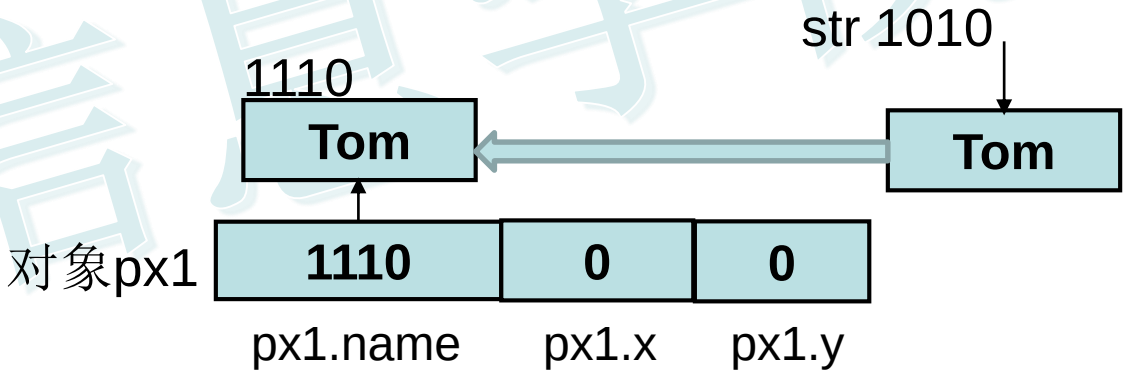


```
#include <iostream>
#include<cstring>
using namespace std;
class Pixel{
private:
    char *name;//像素点名字指针
    int x,y;//像素点坐标
public:
    Pixel(char *nm="noname",int xx=0,int yy=0){
        name=new char[strlen(nm)+1]; strcpy(name,nm);
        x=xx; y=yy;}
    ~Pixel(){ if(name!=NULL) delete [] name;}
    void show(){cout<<x<<" "<<y<<" "<<name<<endl;}
};
```

定义析构函数

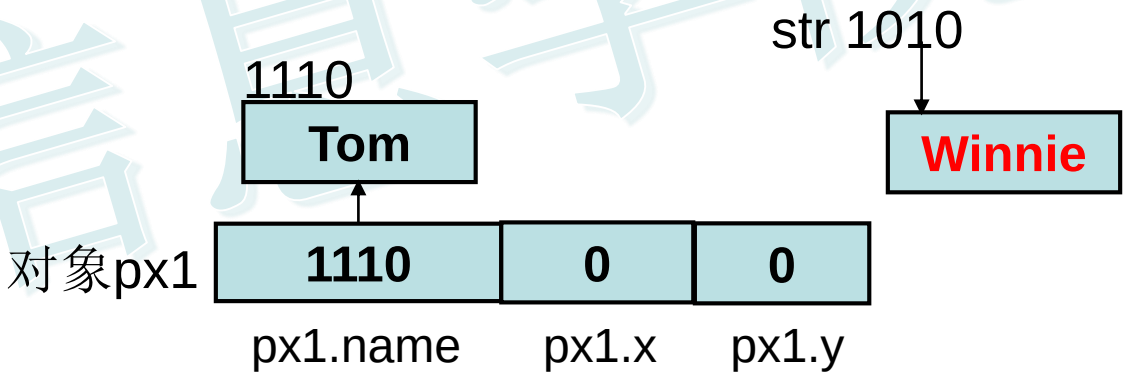
```
int main()
{
    char str[10]="Tom";
    Pixel px1(str);
    px1.show();
    strcpy(str,"Winnie");
    px1.show();
    return 0;
}
```

```
Pixel(char *nm="noname",int
xx=0,int yy=0){
   name=new char[strlen(nm)+1];
   strcpy(name,nm);
   x=xx; y=yy;}
```



```
int main()
{
    char str[10]="Tom";
    Pixel px1(str);
    px1.show();
    strcpy(str,"Winnie");
    px1.show();
    return 0;
}
```

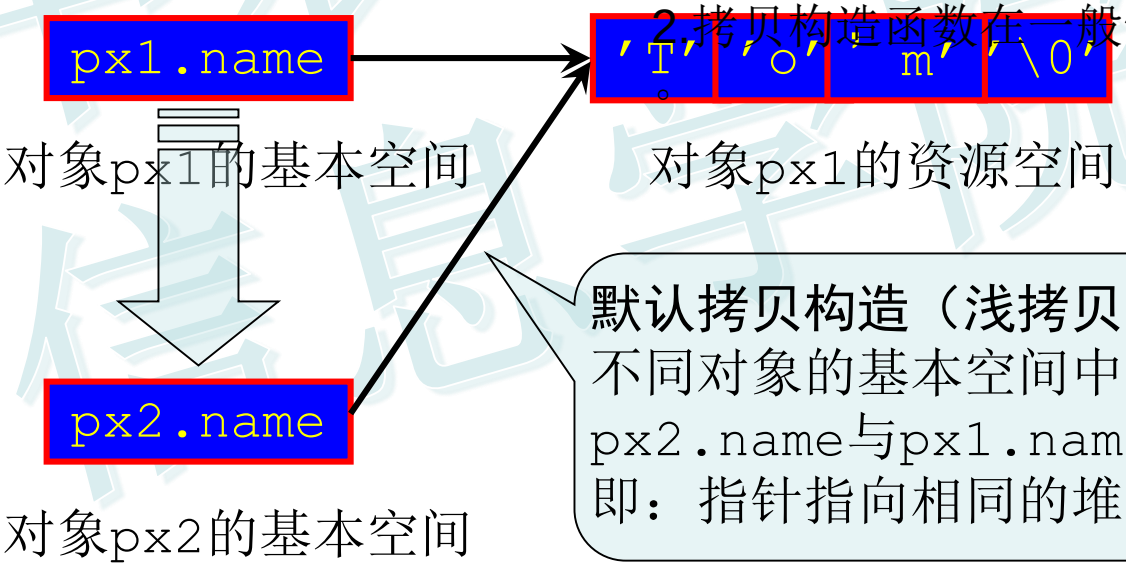
```
Pixel(char *nm="noname",int
xx=0,int yy=0){
   name=new char[strlen(nm)+1];
   strcpy(name,nm);
   x=xx; y=yy;}
```



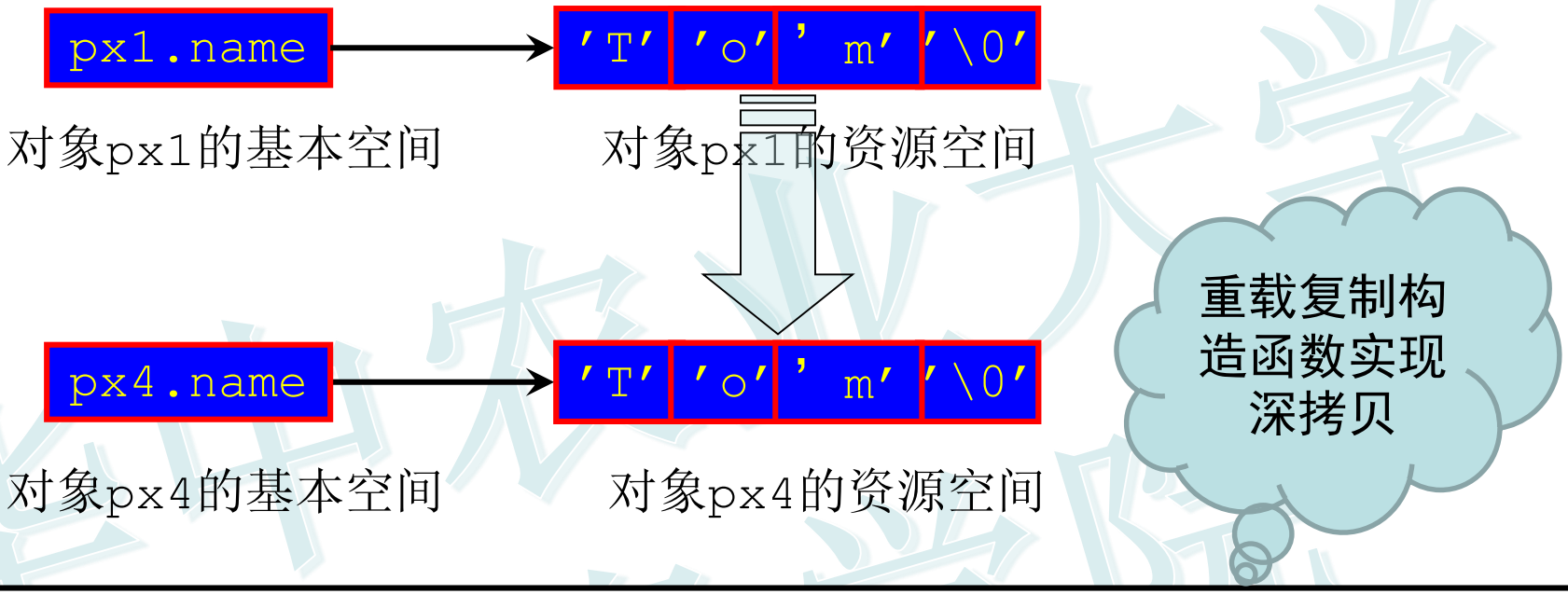
案例进阶

```
int main(){
    char str[10]="Tom";
    Pixel px1(str);
    Pixel px2=px1;
    return 0;}
```

浅拷贝问题，要么变成引用，要麽重载拷
构造函数或者复制运算符，重载成深拷贝
1.在对象复制时，会调用默认的拷贝构造
数。
其作用是将对象的各个数据成员值一一赋
给新的对象中对应的数据成员；
2.拷贝构造函数在一般情况下都能正常工



默认拷贝构造（浅拷贝）的结果：
不同对象的基本空间中的数据成员px2.name与px1.name的值相等。
即：指针指向相同的堆地址。




```
Pixel(const Pixel &px)
{              // 拷贝构造函数
    name = new char[strlen(px.name)+1];
               // 构造（申请）属于对象自己的资源空间
    strcpy(name, px.name);
          // 拷贝（复制）另一对象的资源内容
}
```

小结

浅拷贝

深拷贝

小结

浅拷贝

深拷贝

延申

默认的对象赋值运算也是浅拷贝的过程，需重载赋值运算为深拷贝。重载赋值运算,解决像素点类例中默认赋值运算存在的浅拷贝问题，请编写代码实现。