

Branch: master hello-world / C++远征之继承篇 / 第二章-为什么继承 / 为什么继承.md

Find file Copy path

YHim 3 3511d06 4 minutes ago

1 contributor

29 lines (16 sloc) 922 Bytes

为什么继承


从一个例子讲起：

```
class Person
{
public:
    void eat();
    string m_strName;
    int m_iAge;
};
```

```
class Worker
{
public:
    void eat();
    void work();
    string m_strName;
    int m_iAge;
    int m_iSalary;
};
```

慕课网

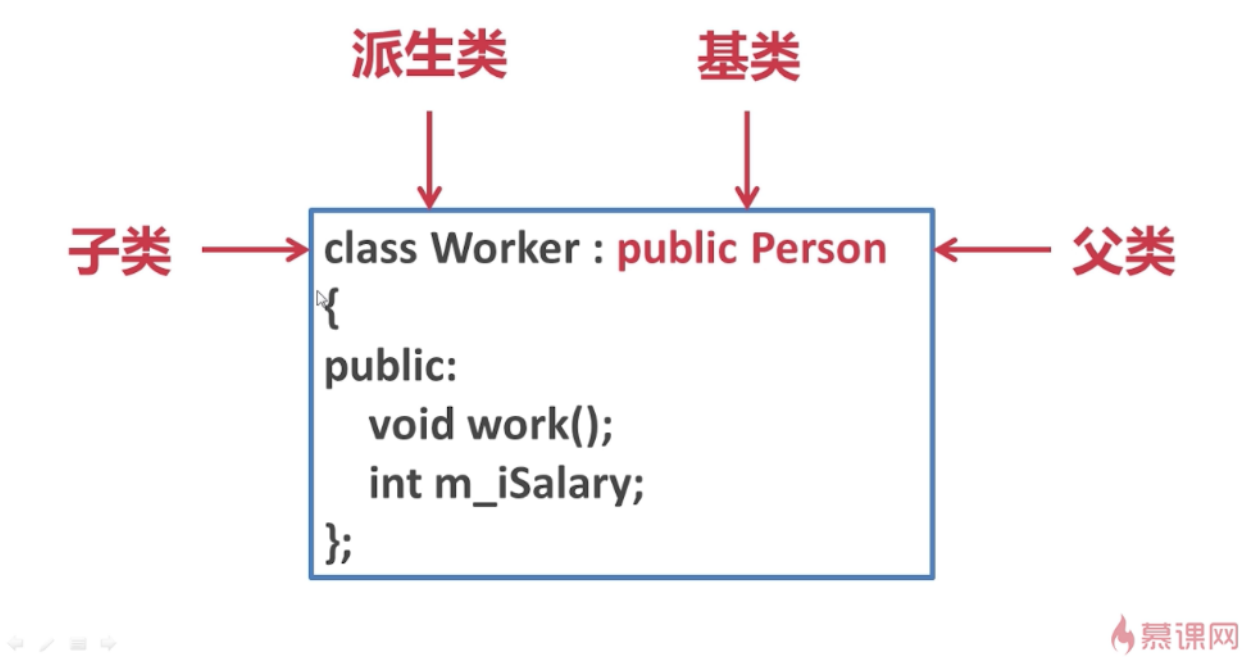
定义了人这个类和工人这个类，工人是人的一种。所以肯定也有名字、年龄、吃。



慕课网

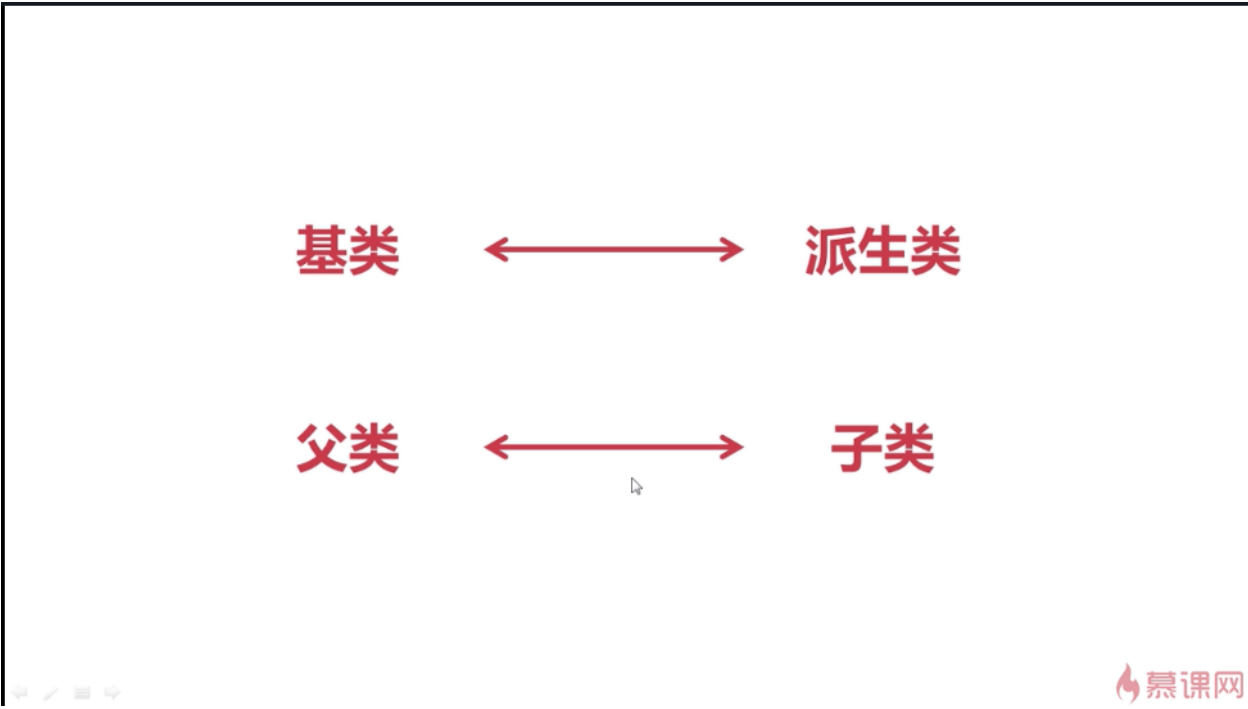
人类是工人类的超集，工人类是人类的子集。

有了这样的概念关系，就可以将程序优化成下面这样：



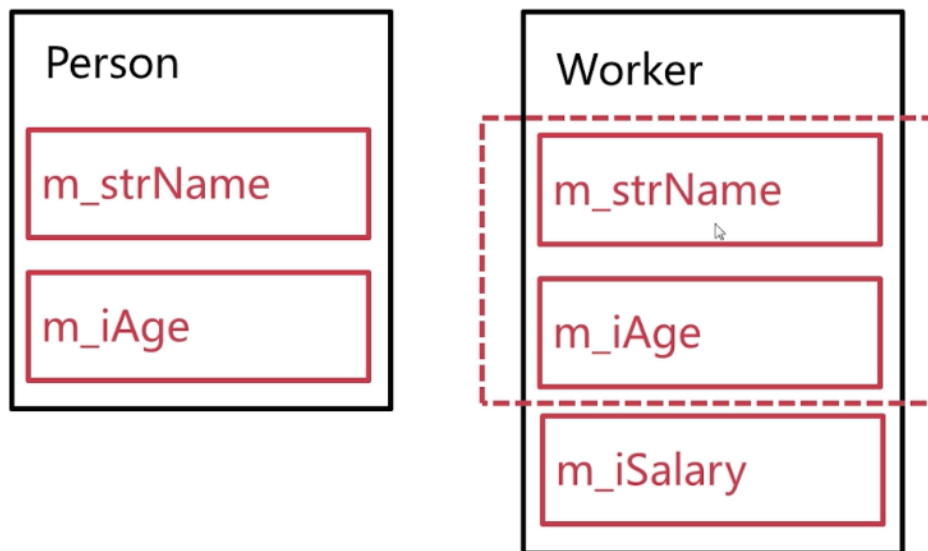
public是一种继承方式，这样写了以后，工人类的定义里面就不需要写人类中的内容了，只需要写出自己特有的内容就行了。

写成这样的继承关系后，可以说工人类是人类的派生类或子类，人类是工人类的基类或父类。



配套使用，不要混搭。

在内存中的关系：



实例化后，Person类有两个数据成员，Worker类虽然在定义中没写Person类中的相关定义，但是还是有3个数据成员。

Branch: master ▾

hello-world / C++远征之继承篇 / 第二章-为什么继承 / 为什么继承-例子.md

Find file

Copy path

YHim 3

3511d06 4 minutes ago

1 contributor

133 lines (107 sloc) 1.79 KB

为什么继承-例子

要求：

```
/*
继承
要求：
1. 定义Person类，要求含有m_strName和m_iAge两个数据成员及构造函数和析构函数、eat函数
2. 定义Worker类，要求共有继承Person类，含有数据成员m_iSalary、构造函数、析构函数、work函数
*/
```

Person.h

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

class Person
{
public:
    Person();
    ~Person();
    void eat();
    string m_strName;
    int m_iAge;
};
```

Person.cpp

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

Person::Person()
{
    cout << "Person()" << endl;
}

Person::~Person()
{
    cout << "~Person()" << endl;
}

void Person::eat()
{
    cout << "eat()" << endl;
}
```

Worker.h

```
#include "Person.h" //因为要继承Person类，所以包含。

class Worker : public Person
```

```
{
public:
    Worker();
    ~Worker();
    void work();
    int m_iSalary;
};
```

Worker.cpp

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

Worker::Worker()
{
    cout << "Worker()" << endl;
}

Worker::~Worker()
{
    cout << "~Worker()" << endl;
}

void Worker::work()
{
    cout << "work()" << endl;
}
```

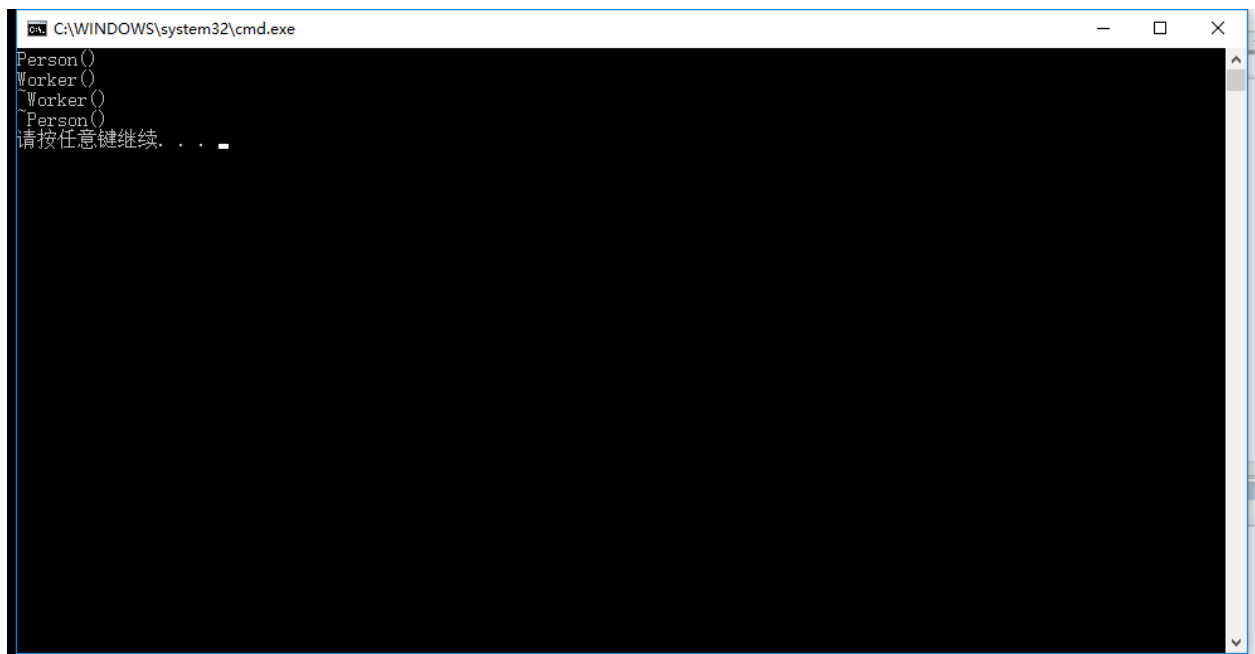
demo.cpp

```
#include <iostream>
#include <stdlib.h>
#include "Worker.h"
using namespace std;

int main()
{
    Worker *p = new Worker();
    delete p;
    p = NULL;

    system("pause");
    return 0;
}
```

运行结果：



```
C:\WINDOWS\system32\cmd.exe
Person()
Worker()
~Worker()
~Worker()
Person()
~Person()
请按任意键继续. . .
```

可见，要想实例化一个子类，必然要先实例化基类（这种实例化是隐性的）。销毁的时候，先执行Worker的析构函数，再执行父类的析构函数。

将demo.cpp代码更改成：

```
#include <iostream>
#include <stdlib.h>
#include "Worker.h"
using namespace std;

int main()
{
    Worker *p = new Worker();
    p->m_strName = "Jim";
    p->m_iAge = 10;
    p->eat();
    p->m_iSalary = 1200;
    p->work();

    delete p;
    p = NULL;

    system("pause");
    return 0;
}
```

运行结果：

可见，子类可以访问父类的成员函数和数据成员，它自己的成员函数和数据成员也可以正确的访问。