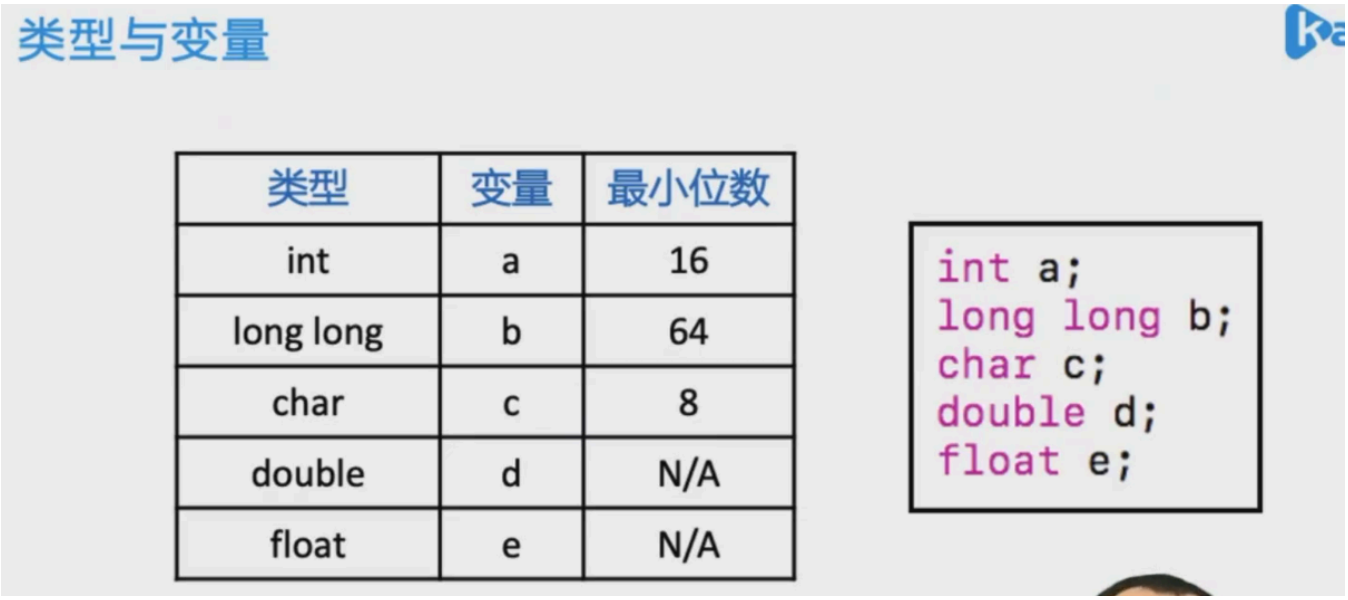
struct结构体和class类的区分

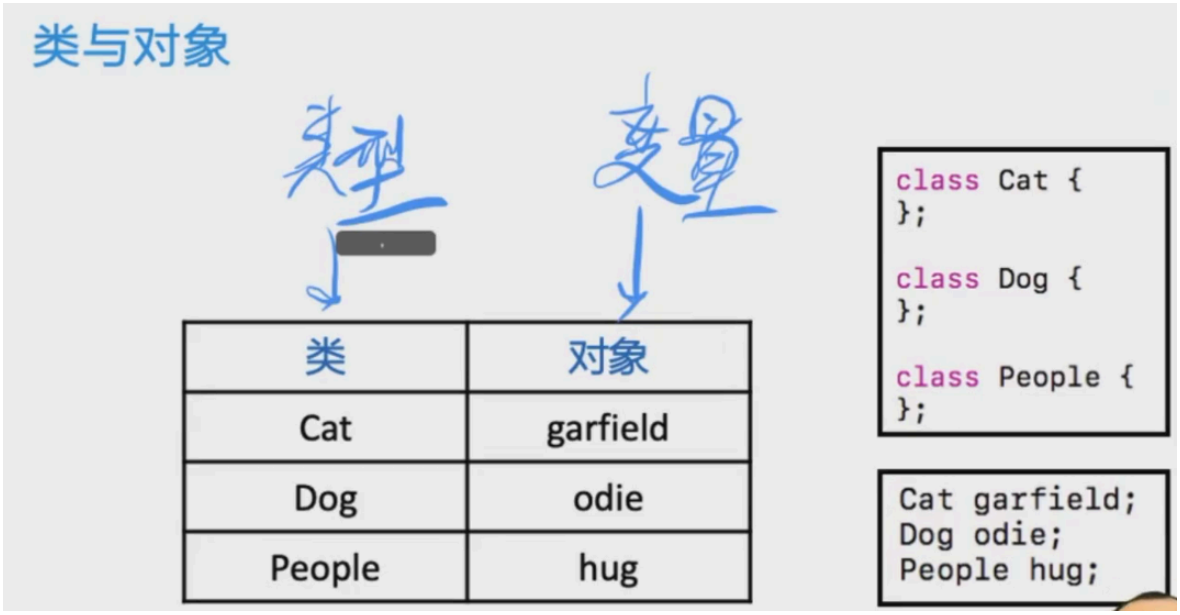
struct和class 本质上是一样的。只是为了兼容C语言代码，保留了struct关键字。在C++中，struct已经不再代表结构体了，更准确地说，在C++中，它的本质也是类。

- struct 默认访问权限为public
- class 默认访问权限是private

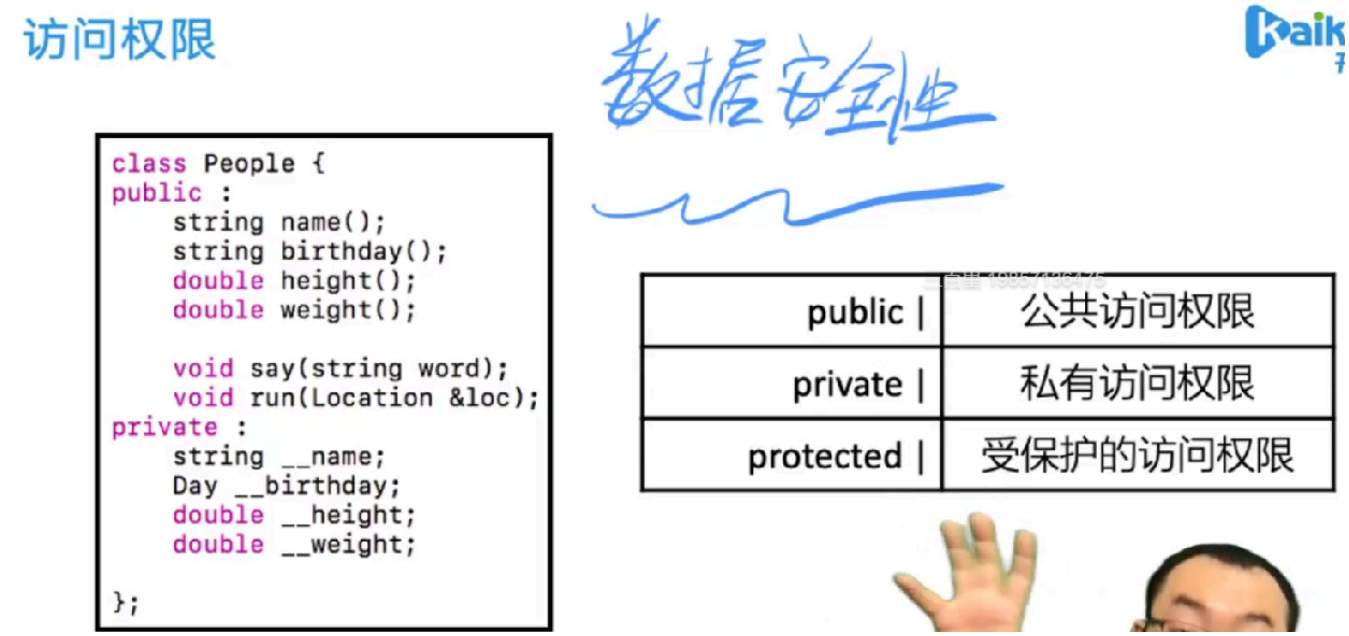
类与对象



- 变量存储的是某个具体类型下的值。
- **类型=类型数据+类型操作**，本质上就是数据结构。
- 研究的是数据的表示方法，**把数据和行为封装到了一起**
-



访问权限



现实世界的功能映射到程序世界。程序往往是为了解决现实世界的问题。

protected：自己和子类能访问，其他不能访问。

friendly：隔壁老王（笑话）。

分清楚内部和外部：类中和类外

命名空间:1.namespace.cpp

```cpp
#include<iostream>

namespace haizei {
    int a, b;
}
namespace haizei {
    int c, d;
}

using namespace haizei;

int main(){
    haizei::a = haizei::b = haizei::c = haizei::d = 1;
    std::cout << haizei::a << std::endl;
    a = b = c = d = 2;
    std::cout << a << std::endl;

    return 0;
}
```

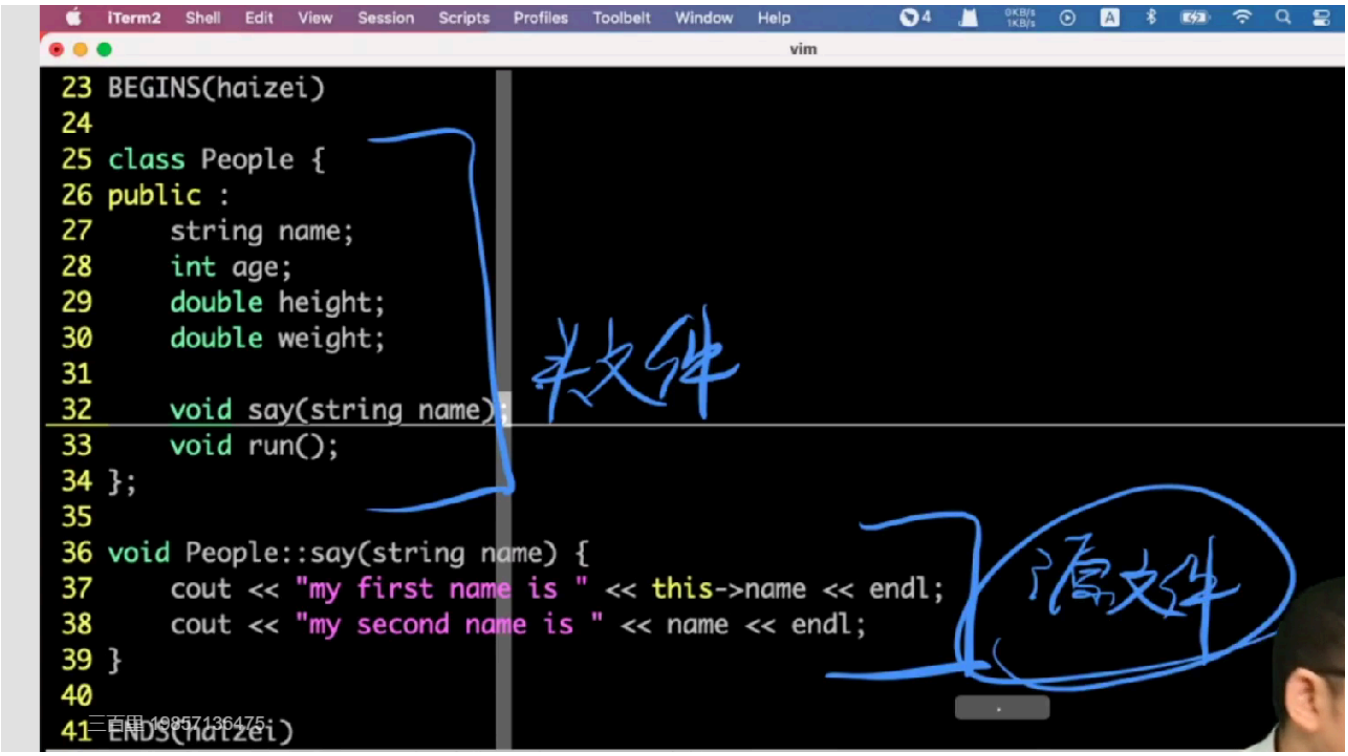
对匿名空间的理解：要把相应的变量和内容打包进我的命名空间中。

```
using namespace haizei
这段代码是把所有haizei::都可以去掉。
但是一般不建议这样用。
```

声明和定义

写在头文件里的是声明，写在源文件里的是定义。

工程实现中一般都像下面这样。**声明和定义分开**


iTerm2 Shell Edit View Session Scripts Profiles Toolbelt Window Help
vim
23 BEGINS(haizei)
24
25 class People {
26 public :
27 string name;
28 int age;
29 double height;
30 double weight;
31
32 void say(string name);
33 void run();
34 };
35
36 void People::say(string name) {
37 cout << "my first name is " << this->name << endl;
38 cout << "my second name is " << name << endl;
39 }
40
41 ENDS(haizei)
头文件
源文件
三百里 19857136475

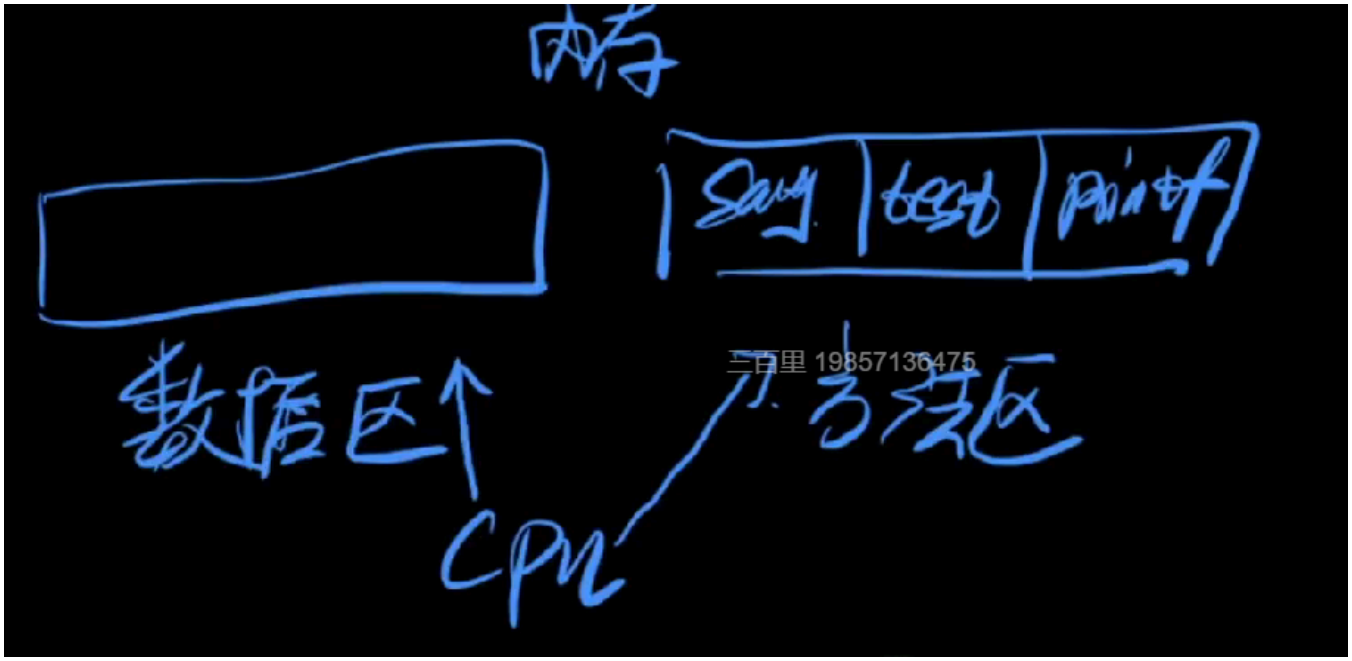
代码演示：2.class.cpp

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

#define BEGINS(x) namespace x {
#define ENDS(x) } // end of namespace x

BEGINS(haizei)

class A_with_int {
    int a;
    double b;
    void say();
};
void A_with_int::say() {
    cout << "hello world" << endl;
}

class People {
public:
    //成员属性
    string name;
    int age;
    double height;
    double weight;

    //成员方法（和对象强相关）
    void say(string name);
    void run();//声明但是没定义，如果不使用就不会报错
};

ENDS(haizei)

void haizei::People::say(string name) {
    cout << "my first name is " << this->name << endl;
    cout << "my second name is " << name << endl;
}
//this是个关键字，又是一个指针变量，指向的是当前对象。

int main() {
    cout << sizeof(haizei::A_with_int) <<endl;
    haizei::People hug;
    haizei::People zhangpei;
    haizei::People hanmingtao;
    haizei::People zhangxueqi;
    hug.name = "Captain Hu";
    zhangpei.name = "Doctor Zhang";
```

```
    hanmingtao.name = "Boss Han";
    zhangxueqi.name = "Leader Zhang";
    hug.say("Xiao Hu");
    zhangpei.say("Xiao Zhang");
    hanmingtao.say("Xiao Han");
    zhangxueqi.say("Xiao Zhang");

    return 0;
}
```

对象占用的存储空间大小是怎么算的？

- 一个int：4个字节
- 再加一个double：16个字节。
- 结构体的对齐方式沿用到C++中的类。
- 如果放一个成员方法：存储空间不变。
- 函数实现的功能也对应了一段二进制代码（存放在**方法区**）。计算机会把相关功能的二进制代码加载到内存里面。只有数据和方法都存在内存里了，CPU才知道怎么做，做什么。
- **sizeof()计算的是相关类型在数据区**中的存储空间。
- 为什么？——方法区的大小不会随着对象的增加而增加。



```
ls -hl a.out
# 显示有57K，方法区的大小在编译的时候就已经确定了
```

这个cout到底是个啥？:3.cout.cpp

1.函数。2.对象。

- << 是一个运算符，运算符的两边放两个值，变量用来存值，而cout本质上是一个对象。

```cpp
#include<iostream>
#define BEGINS(x) namespace x {
#define ENDS(x) }

BEGINS(haizei)
class istream {
public:
    istream &operator>>(int &x);
};
class ostream {
public:
    ostream &operator<<(int x);
    ostream &operator<<(char ch);
};
istream &istream::operator>>(int &x) {
    scanf("%d", &x);
    return *this;
}
ostream &ostream::operator<<(int x) {
    printf("%d", x);
    return *this;
}
ostream &ostream::operator<<(char x) {
    printf("%c", x);
    return *this;
}
ostream cout;
istream cin;
const char endl = '\n';
ENDS(haizei)

int main() {
    int n, m;
    std::cin >> n >> m;
    std::cout << n << m << std::endl;
    haizei::cin >> n >> m;
    haizei::cout << n << m << haizei::endl;
    return 0;
}
```