清華大學
Tsinghua University

第8章 类与对象

清华大学计算机系 韩文弢

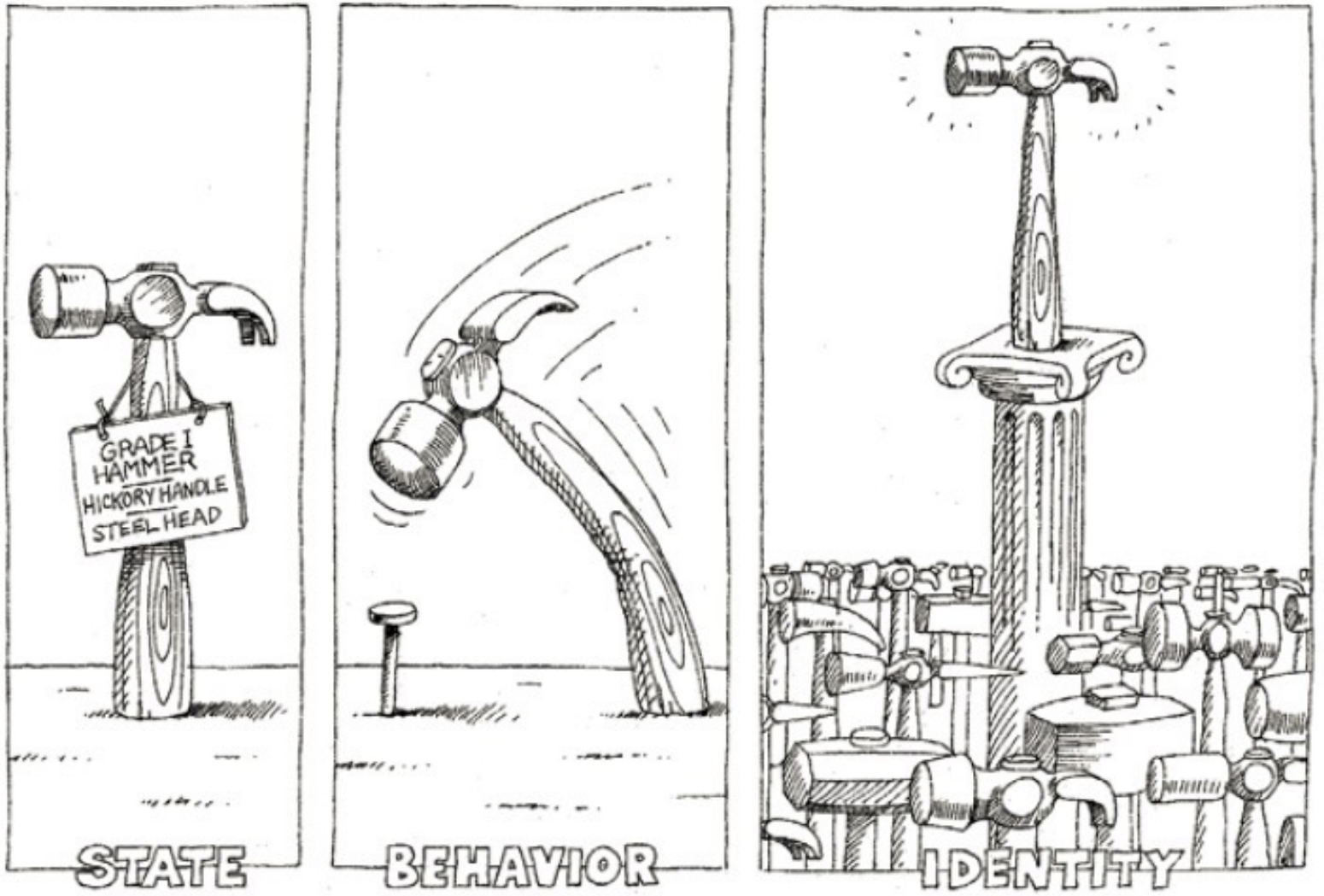
教学内容

1. 类和对象
2. 类的定义与使用
3. 进一步的内容
4. 常用的C++类

1、对象（Object）

- 什么是对象
  - ☺ 在认知心理学中，对象是指可触摸、可见或思维可理解的东西
  - ☺ 软件工程中的定义：对象是一个具有状态、行为和标识的实体。相似对象的结构和行为在它们的共有的类中定义。

状态、行为和标识



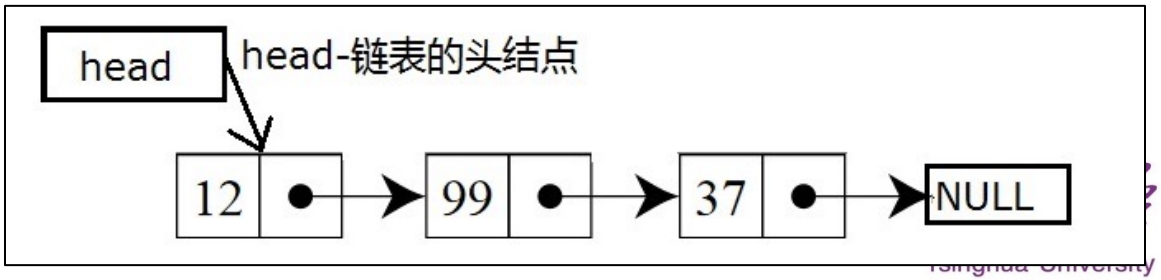
对象

物理实体：

概念实体：

dream

软件实体：



对象的属性

- 对象的属性：对象所具有的一些特征称为属性，以一个人为例，其姓名、性别、年龄、肤色、居住地等可以作为他的属性。这些属性会有其对应的值
- 对象的状态：一个对象的状态包括该对象的所有属性及每个属性的值

属性	值
姓名	喜羊羊
性别	男
年龄	7
肤色	白色
居住地	青青草原-羊村
食物	青草

对象的行为与操作

- 对象的行为：对象不是孤立存在的，一个对象可以作用于其他对象，也可被其他对象所作用，从而导致状态的变化。
- 对象的操作：一个对象（类）对外提供的服务。

喜羊羊
+ Study() + Run() + Play_tricks_on()

对象太多了怎么办？

物以类聚，人以群分

清華大學 Tsinghua University

2、类（class）

- 什么是类？

☺ 一组具有类似属性和行为的对象。

羊：



狼：


类与对象

- 类与对象
  - ☺ 类：抽象地定义了该类对象的本质特征（属性和操作），类型定义、模版
  - ☺ 对象：类的实例，具有各自的属性值，占用存储空间

教学内容

1. 类和对象
2. 类的定义与使用
3. 进一步的内容
4. 常用的C++类

1、类的定义

如何描述现实世界中的

Baby

宝宝胖乎乎的
宝宝眼睛很大
宝宝长了2颗牙齿

类的描述

Class

Baby

Properties

- Name
- Gender
- Age
- Weight
- # poops so far

Behavior

- Eat
- Poop
- Smile
- SayHi

Baby类的定义

```
class Baby {
```

成员变量

成员函数

```
};
```


清華大學 Tsinghua University

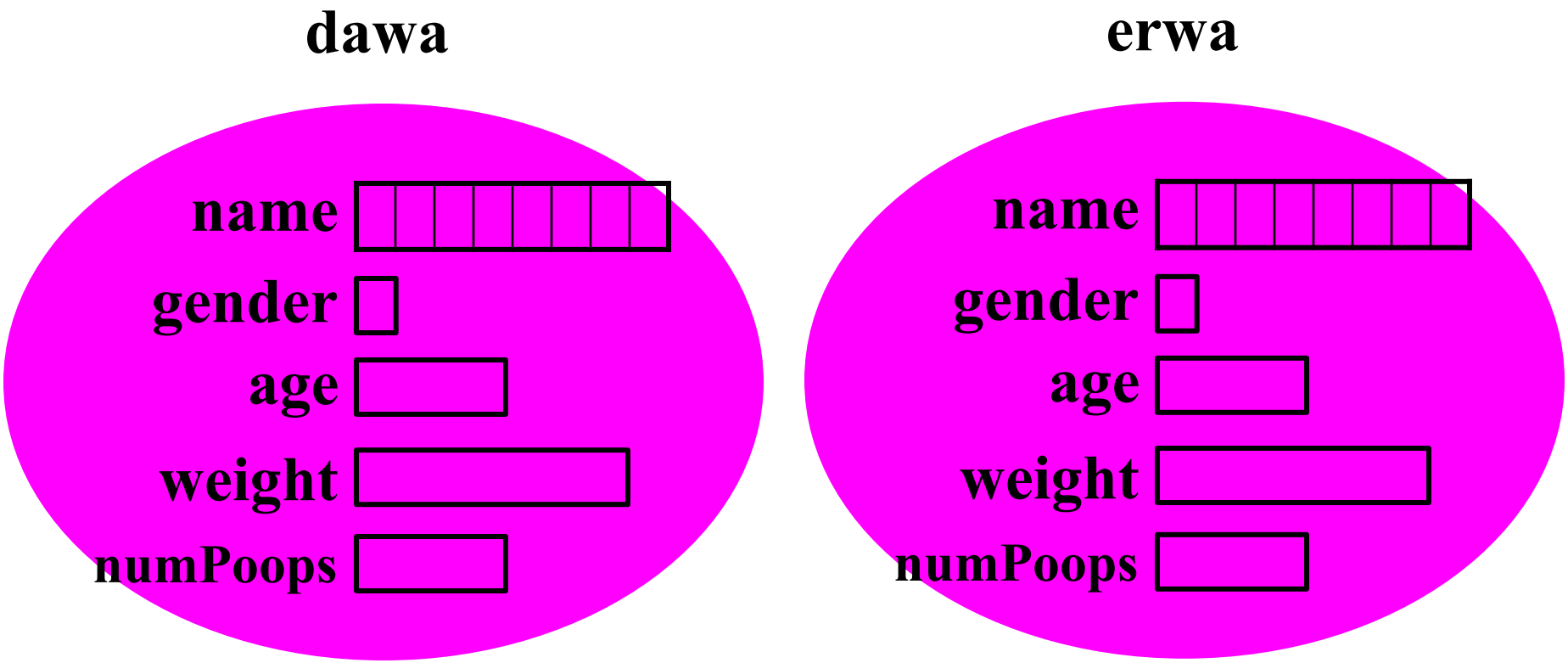
成员变量

```
class Baby {
public:
    char name[20];
    char gender;
    int age;
    double weight;
    int numPoops;
};
```


```
// 类定义
class Baby {...};

// 类实例
Baby dawa;
```

实例（对象）

```
Baby dawa, erwa;
```



成员变量的初始化

- 成员变量的初始化
  - ☺ 在定义类时即可确定，用构造函数，且无需参数
  - ☺ 在创建对象时即可确定，用带参数的构造函数
  - ☺ 在创建对象后才能确定，用成员函数

构造函数（Constructors）

```
class CLASSNAME {
    CLASSNAME() {
    }
    CLASSNAME([ARGUMENTS]) {
    }
}
```

构造函数

- 构造函数名字＝类的名字
- 没有返回值类型：不返回任何数据
- 创建对象时自动调用，初始化成员变量
- 每个类需要至少一个构造函数，若不写，则默认为：

```
CLASSNAME() {

    }
```

构造函数的执行过程

- 类初始化时构造函数的调用顺序
  1. 初始化对象的存储空间为零或null
  2. 按顺序分别调用类成员变量和对象成员的初始化
  3. 调用构造函数

构造函数的执行过程

```
#include <iostream>
using namespace std;
class Dollar {
public:
    int n;
    Dollar() {
        n = 100;
        cout << n << " dollars come to my home" << endl;
    }
};
class Money {
public:
    Dollar d;
    Money() {
         cout << "All money come to my home" << endl;
    }
};
```

构造函数的执行过程

```
int main()
{
    Money m;

    return 0;
}
```

```
100 dollars come to my home
All money come to my home
```

默认构造函数

- 调用时可以不需要实参的构造函数
  - ✓ 参数列表为空
  - ✓ 全部参数都有默认值（略）
- 每个类只能有一个默认的构造函数，否则将编译错误

Baby类的默认构造函数

```
class Baby {
public:
    char name[20];
    char gender;
    int age;
    double weight;
    int numPoops;
    Baby() {
        strcpy(name, "");  // 创建对象时确定
        gender = 'F';   // 创建对象时确定
        weight = 6.0;   // 创建对象时确定
        age = 0;    numPoops = 0;   //定义类时确定
    }
};
```

Baby类的构造函数

```
class Baby {
public:
    char name[20];
    char gender;
    int age;
    double weight;
    int numPoops;
    Baby(char myname[], char g, double w)
                                : age(0), numPoops(0) {
        strcpy(name, myname);
        gender = g;
        weight = w;
    }
};
```

初始化成员列表

调用带参数的构造函数

```
// 创建对象
Baby dawa = Baby("大力士", 'M', 20);
// 或者
Baby dawa("大力士", 'M', 20);
```

成员函数

```
class Baby {
public:
   ...
   void poop() {
      numPoops ++;
      cout << "Dear mother," <<
      "I have pooped." << endl;
   }
```

成员方法(2)

```
    void sayHi(){
       cout << "Hi, my name is " <<
            name << endl;
    }
    void eat(double foodWeight) {
       weight += foodWeight;
    }
}
```

完整的Baby类

```
class Baby {
public:
   char name[20];
   char gender;
   int age;
   double weight;
   int numPoops;
   Baby(...){...}
   void poop() {...}
   void sayHi() {...}
   void eat(double foodweight){...}
};
```

2、函数声明与实现分离

- 对于普通函数，函数的原型与函数的实现可以分离，即把函数原型放在文件开头，而把函数的实现放在后面
- 类似的，类的成员函数也可以这样做，函数的实现需要用::来表示它是哪一个类的成员函数

函数的声明

```
// baby.h 头文件
class Baby {
public:
    char name[20];
    char gender;
    int age;
    double weight;
    int numPoops;
    Baby(char *, char, double); //函数原型
    void poop();  //函数原型
    void sayHi(); //函数原型
    void eat(double foodweight); //函数原型
};
```

函数的实现

```
// baby.cpp 源文件
#include "baby.h"
Baby::Baby(char myname[], char g, double w) {
      ……
}
void Baby::poop( ) {
    ……
}
 ……
void Baby::eat(double foodweight) {
       weight += foodWeight;
}
```

::表示当前实现的是哪一个类的成员函数

3、类的使用

```
// 类定义
class Baby {...}

// 类实例
Baby dawa = Baby("大力士", 'M', 20);
Baby erwa = Baby("千里眼", 'M', 16);
Baby sanwa = Baby("钢筋铁骨", 'M', 18);
Baby siwa = Baby("火神", 'M', 16);
```

访问成员变量

```
Object.FIELD_NAME
```

```
cout << dawa.name << endl;
cout << erwa.weight << endl;
cout << sanwa.numPoops << endl;
```

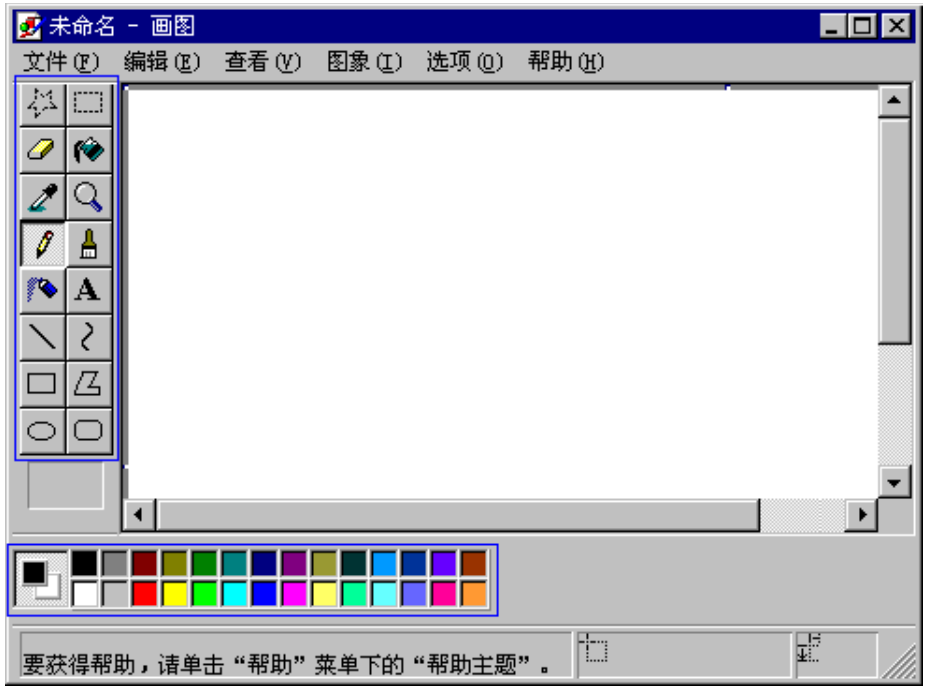
调用成员函数

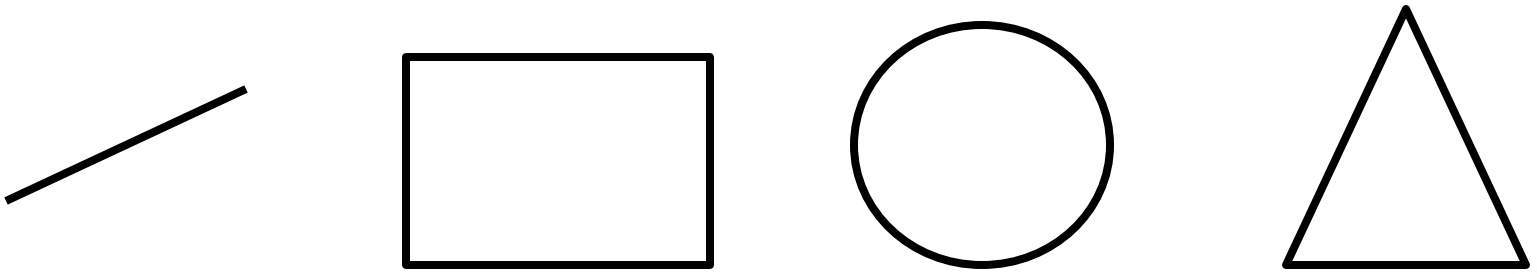
```
Object.METHOD_NAME([参数])

dawa.sayHi();  //把sayHi作用于dawa
erwa.eat(1);   //把eat(1)作用于erwa
sanwa.poop();  //把poop作用于sanwa
```

程序 = 代码+数据，参考**递归函数**

成员函数能访问哪些变量？

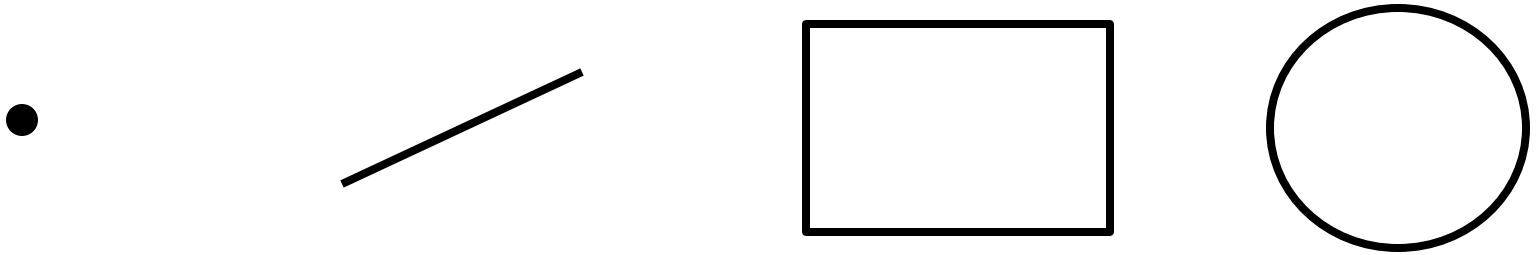
4、类的设计





清華大學 Tsinghua University

类的定义



Point
int x, y;
Point(); SetPosition(); GetPosition(); SetX(); ……

Line
Point start; Point end;
Line(); DrawLine(); GetLength(); ……

Rectangle
Point p; int width; int height;
Rectangle(); DrawRectangle(); Area(); ……

Circle
Point center; int radius;
Circle(); DrawCircle(); Area(); ……

教学内容

1. 类和对象
2. 类的定义与使用
3. 进一步的内容
4. 常用的C++类

1、对象的赋值

```
class Point {
public:
    int x, y;
    Point(){ x = 0; y = 0;}
};
int main()
{
    Point start, end;
    start.x = 3;
    start.y = 4;
    end = start;   //对象的赋值
    cout << end.x << ' ' << end.y << endl;
    return 0;
}
```

内存块拷贝

```
int main()
{
    ......
    int i, n, size;
    char *p, *q;
    p = (char *) &start;
    q = (char *) &end;
    size = sizeof(start);
    n = 0;
    for(i = 0; i < size; i++) {
        if(p[i] == q[i]) n++;
    }
    if(n == size)
      cout<< size << "个字节全部相同！"<<endl;
    return 0;
}
```

对象作为函数参数

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

class DaGongRen {   // 打工人
public:
    int salary, nCars, nHouses;
};
void GetRich(DaGongRen p, int s, int c,int h)
{
    p.salary += s;
    p.nCars += c;
    p.nHouses += h;
}
```

对象作为函数的参数(2)

```
int main()
{
    DaGongRen poor;
    poor.salary = 6000;
    poor.nCars = 0;
    poor.nHouses = 0;
    GetRich(poor, 12000, 2, 1);
    cout << poor.salary << ' ' << poor.nCars
         << ' ' << poor.nHouses << endl;
    return 0;
}
```

传值导致美梦一场，如何成真？引用或指针！

用另一对象来初始化

卓别林成名后，一次参加一个模仿秀比赛，模仿自己，然后获得了第二名。

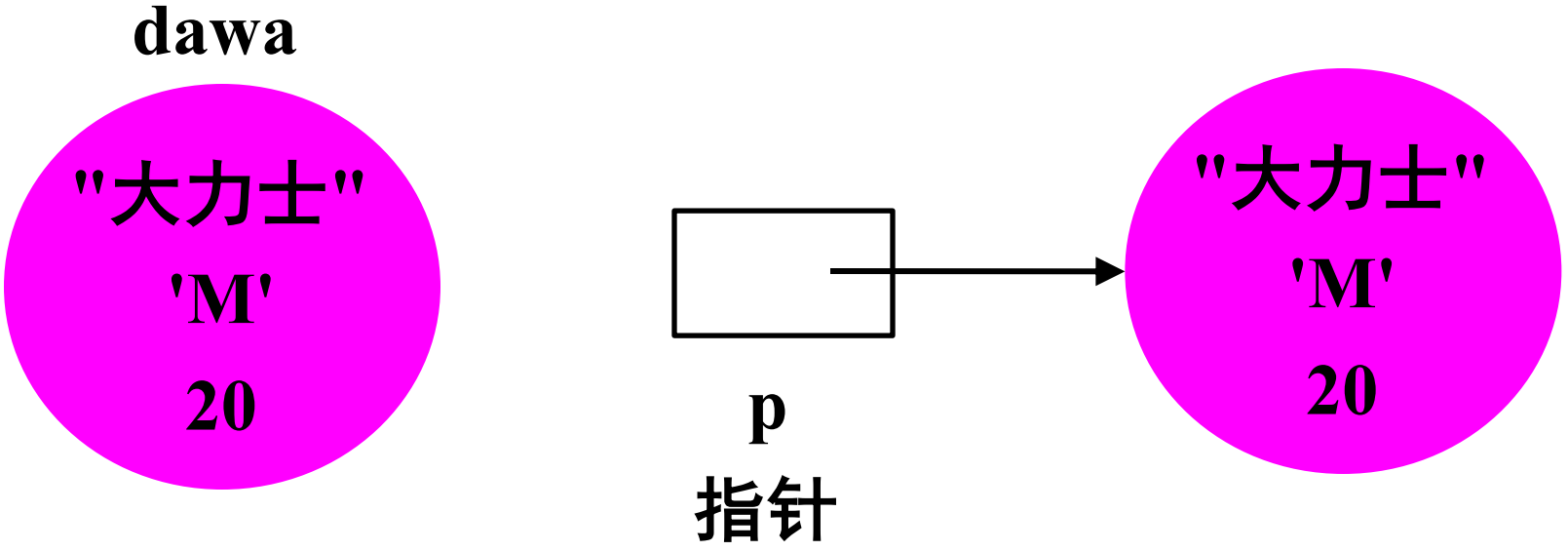
王杰克逊，河南小伙，因模仿MJ被好莱坞签约，在全球范围内巡演。

用另一对象来初始化(2)

```
class Singer {
public:
    string name;
    int birth;
    Singer() {}   //默认构造函数
    Singer(Singer &o){ //复制构造函数，形参为引用
        name = o.name; birth = o.birth;
    }
};
int main() {
{
    Singer MJ; //使用默认构造函数
    MJ.name = "Michael Jackson";
    MJ.birth = 1958;
    Singer wang(MJ); //使用复制构造函数
}
```

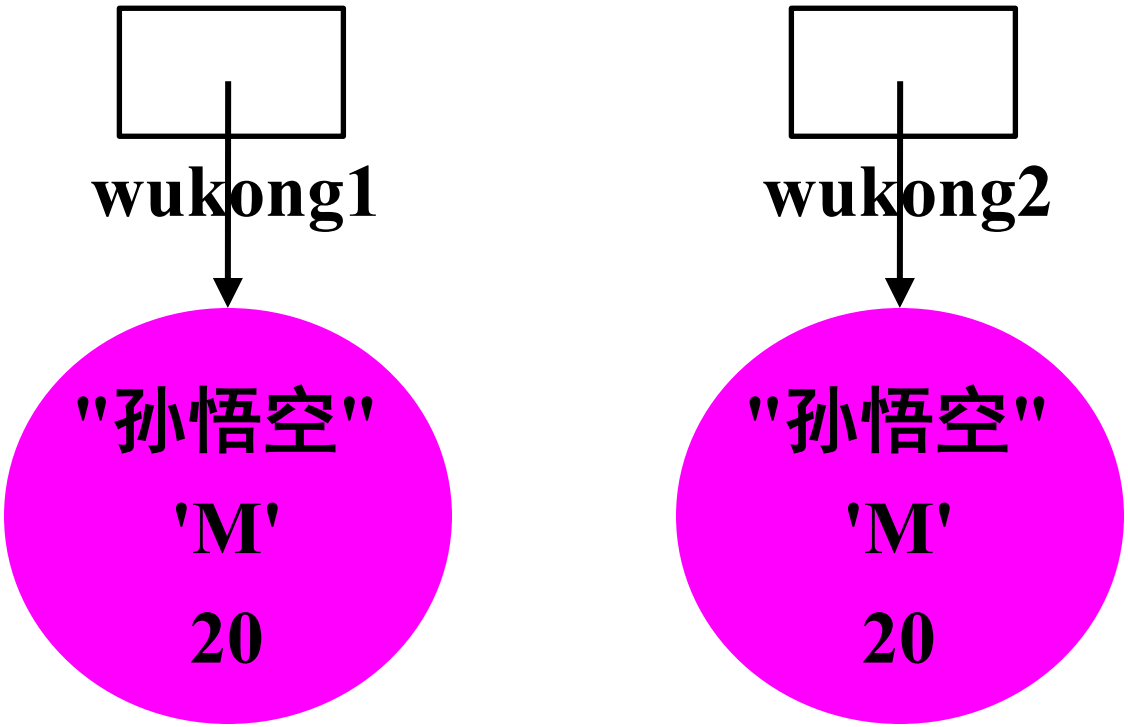
2、指向对象的指针

```
Baby dawa("大力士", 'M', 20);
Baby *p = new Baby("大力士", 'M', 20);
cout << p->name << ' ' << p->gender
```



比较运算

```
Baby *wukong1 = new Baby("孙悟空", 'M', 20);
Baby *wukong2 = new Baby("孙悟空", 'M', 20);
cout << (wukong1 == wukong2) << endl;
```



分析结果

```
class Reference {
public:
    void plus(int i){
        i++;
    }
    void plus(Counter c){
        c.i++;
    }
    void create(Counter *c){
        c = new Counter();
        (c->i)++;
    }
};
void main( ) {
    int i = 0;
    Reference r = Reference();
    Counter c1 = Counter();
    Counter *c2 = new Counter();
    r.plus(i);
    cout << i << endl;
    r.plus(c1);
    cout << c1.i << endl;
    r.create(c2);
    cout << c2->i << endl;
}
```

```
class Counter{
public:
    int i;
    Counter(){
        i = 0;
    }
};
```

```
0
0
0
```

3、静态类型（static）

- static类型表示“静态”或“全局”的意思，适用于成员变量和方法
- 对于静态成员变量
  - ☺ 是类一级的定义，独立于该类的任何对象（实例），为所有对象所共享。
  - ☺ 必须在类外初始化，用::指明类
  - ☺ 可以在任何类对象创建之前访问（静态成员函数也是如此）

静态成员变量

```
class Baby {
public:
    static int numBabiesMade;
};
int Baby::numBabiesMade = 0;
void main() {
    Baby::numBabiesMade = 100;
    Baby b1, b2;
    b1.numBabiesMade = 1;
    b2.numBabiesMade = 2;
    cout << Baby::numBabiesMade<<' '
       << b1.numBabiesMade << ' '
       << b2.numBabiesMade << endl;
}
```

static示例

- 如何记录被创建的宝宝个数

```
class Baby{
public:
     int numBabiesMade = 0;
     Baby() {
          numBabiesMade ++;
     }
}
```

编译错误！

请问：上述方法是否可行？

static示例(2)

- 如何记录被创建的宝宝个数

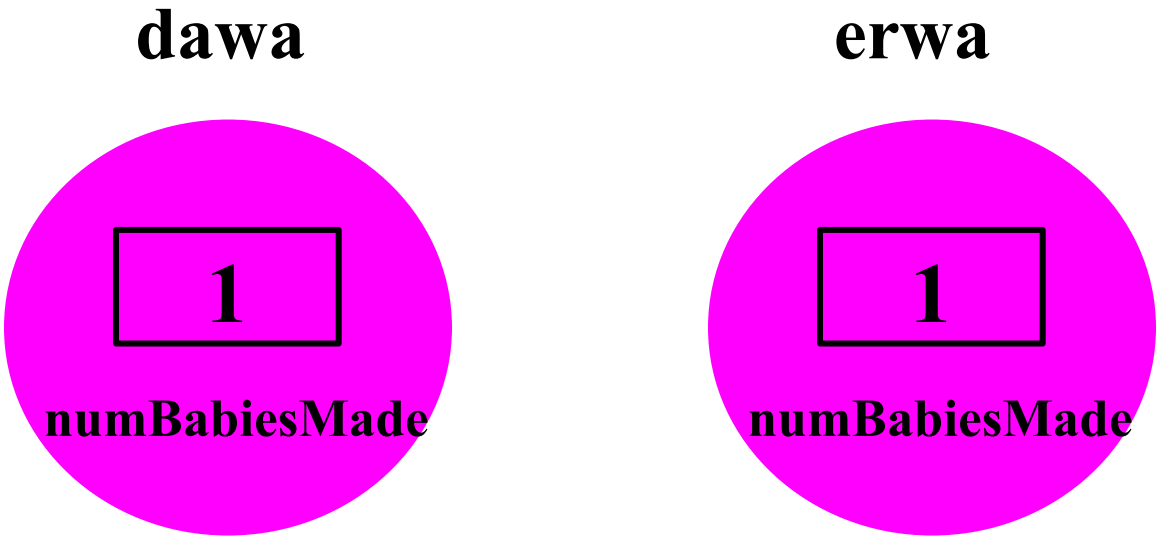
```
class Baby{
public:
      int numBabiesMade;
      Baby():numBabiesMade(0){
            numBabiesMade ++;
      }
}
```

NO!

请问：上述方法是否可行？

static示例(3)

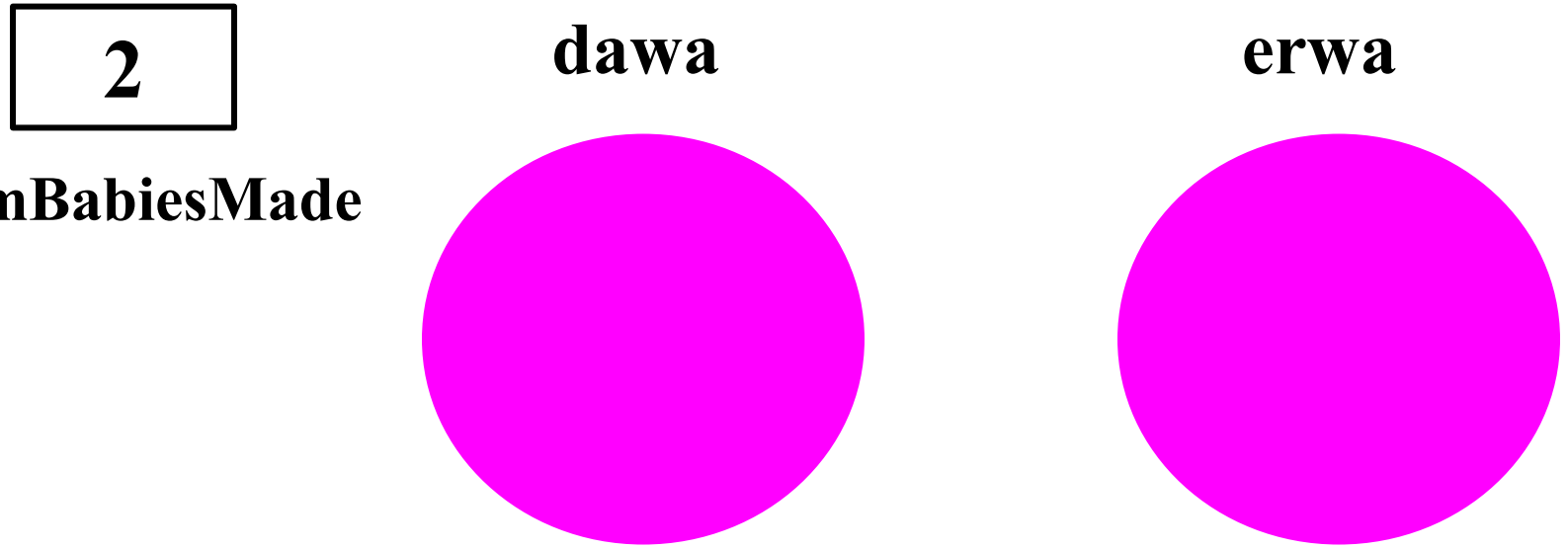
```
Baby dawa, erwa;
cout << dawa.numBabiesMade << ' '
     << erwa.numBabiesMade << endl;
```



修改为static类型后

```
class Baby {
public:
     static int numBabiesMade;
     Baby() {
          numBabiesMade ++;
     }
};
int Baby::numBabiesMade = 0;
void main() {
     Baby dawa, erwa;
     cout << Baby::numBabiesMade <<' '
        << dawa.numBabiesMade << ' '
        << erwa.numBabiesMade << endl;
}
```

修改为static类型后(2)



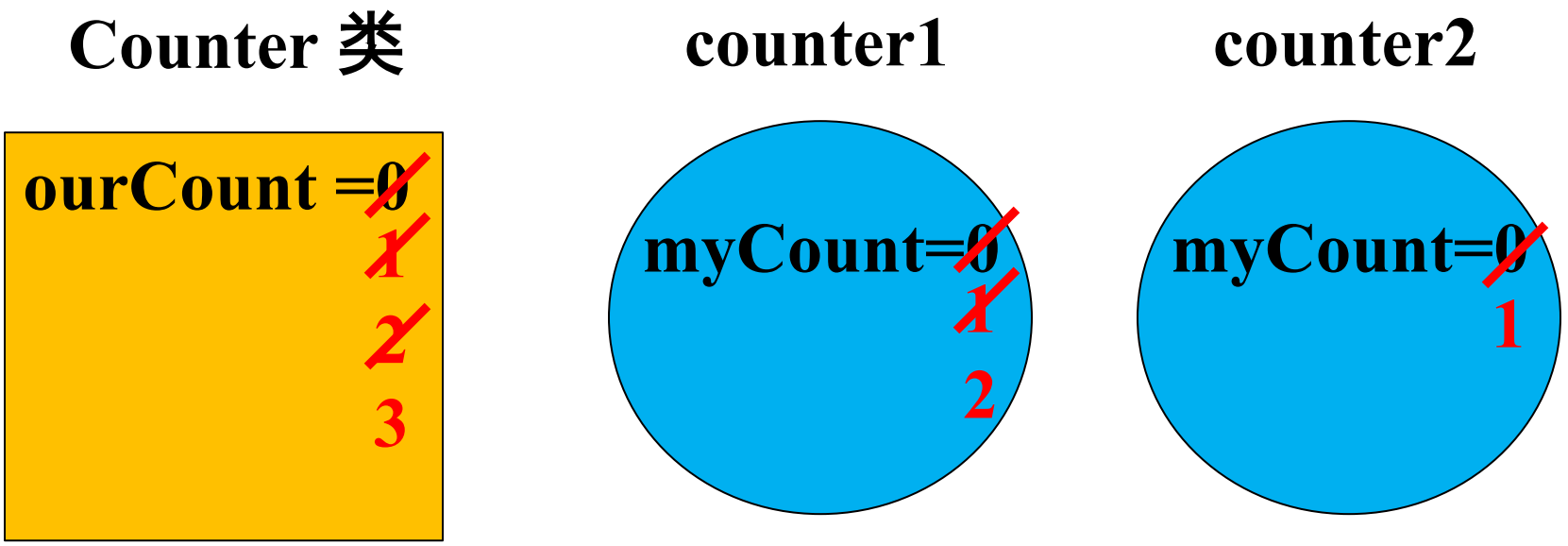
另一个例子

```
class Counter {
public:
    int myCount;
    static int ourCount;
    Counter() : myCount(0) {}
    void increment(){
        myCount ++;
        ourCount ++;
    }
};
int Counter::ourCount = 0;
```

另一个例子

```
int main( )
{
    Counter counter1;
    Counter counter2;
    counter1.increment();
    counter1.increment();
    counter2.increment();
    cout << Counter::ourCount << ' '
         << counter1.myCount << ' '
         << counter2.myCount << endl;
}
```

另一个例子(2)




```
Counter counter1;
Counter counter2;
counter1.increment();
counter1.increment();
counter2.increment();
```

静态成员函数

- 非静态函数可以访问静态函数和变量，而反过来则不行，why？

```
class Baby{
public:
    int age;
    static void how_old() {
        cout << age;
    }
};
```

4、常类型（const）

- 对于既需要共享、又需要防止改变的数据应声明为常类型（用**const**来修饰）
- **const**可以适用于
  - ☺ 普通变量
  - ☺ 指针
  - ☺ 对象
  - ☺ 成员变量、成员函数
  - ☺ 引用

常变量

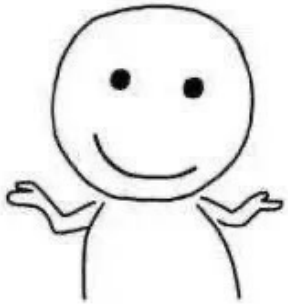
```
//必须初始化，随后其值不能被修改
const int a = 6;

cout << a << endl;  // ok
x = a + 2; // ok
a = a + 1; // ???
```

清華大學
Tsinghua University

常指针

```
void  test(char  *p)
{
    char   s[ ] = "hello";
    const  char  *pc  =  s;  // 指针, 指向常量字符
    pc[3]  =  'g';           // 错误, 常量不能被修改
    pc  =  p;                // 正确, 指针本身可以变
    char  *const  cp  =  s;  //指针本身是常量
    cp[3] = 'a';             //正确, 指针指向的是变量
    cp  =  p;                //错误, 指针本身是常量
    const char  *const  cpc  =  s;  // 都是常量
    cpc[3] = 'a';                   // 错误
    cpc  =  p;                      // 错误
}
```

常指针(2)



这突如其来的心塞
是怎么回事

倒过来念：

```
const  char  *pc;              // pointer to const char
char   *const   cp;            // const pointer to char
const  char *const  cpc;       // const pointer to const
                               //  char
```

说明

const只是用来修饰数据的类型，即限制了一个数据的使用方式，而不是指明了它的存放方式。

```
void test(const int *p)
{
    (*p)++;    //错误,这里不能修改*p的内容
}
void main( )
{
    int x;     //x的值在这里是可以修改的
    test(&x);
}
```

常对象

```
//对象被创建后其值不能被修改
const Baby dawa("大力士",'M',8);

cout << dawa.name << endl; //ok
Baby erwa = dawa; //ok
dawa.weight += 10; // ???
```

清華大學
Tsinghua University

常成员函数

- 对于一个常对象，其成员变量不能被修改，那么哪些成员函数能调用呢？
- 可在一个成员函数的末尾加上**const**
  - ☺ 常成员函数保证不会去修改对象的成员变量，普通成员函数没有这个保证
  - ☺ 对于一个常对象，只能调用它的常成员函数

教学内容

1. 类和对象
2. 类的定义与使用
3. 进一步的内容
4. 常用的C++类

1、string类

- 字符串
  - 用双引号括起来的若干个字符
  - 如"你好"、"no zuo no die"
- C语言：存储在字符数组，以0结尾
- C++：string类
  - 表示字符串
  - 实际上是对字符数组操作的封装

string类的构造函数

- string(); //默认构造函数，建立长度为0的串

  string s1;

- string(const char *s); //用s字符串来初始化

  string s2 = "abc"; //或 string s2("abc");

- string(const string& s) //复制构造函数

  string s3 = s2; //或 string s3(s2);

字符串存储

- 字符串中的字符下标从0开始：

```
String name = "Ultimate";
```

下标	0	1	2	3	4	5	6	7	8
字符	U	l	t	i	m	a	t	e	0

- ✓ 首字符下标为0，尾字符下标为N-1
- ✓ 每个元素的类型为char

string类的一些成员函数

函数名称	功能描述
+、=	字符串的拼接和赋值
==、!=、<、<=、>、>=	各种关系运算
int length()	返回字符串的长度
s[i]	访问字符串s中下标为i的字符
string substr (int pos, int n)	返回pos开始的n个字符组成的子串
int find(const char *s, int pos) int find(char c, int pos)	从pos开始查找字符串s或字符c在当前字符串中的位置
string&insert(int p0, char*s)	在当前字符串的p0位置插入字符串s
string&erase(int pos, int n)	删除pos开始的n个字符，返回新串

string类举例

```
string s1 = "too ";
string s2 = s1 + "how";
cout << s2 << endl;  // "too how"

// index      012345678901
string s3 = "Stuart Reges";

cout << s3.length() << endl;          // 12
cout << s3.find("e", 0) << endl;      // 8
cout << s3.substr(7, 3) << endl;      // "Reg"
```

本讲小结

- 类和对象
- 构造函数
- 静态成员
- 常类型与常成员
- std::string类