作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师
https://github.com/drink-cat/Book_Program-Principles

面向对象的含义

面向对象是一种软件开发方式，在建模、提高扩展性、提高可读性等方面有重要作用。许多高级语言支持面向对象。C 语言是结构化语言，也可以实现面向对象，主要借助 struct 和指针的功能。

面向对象的特性：继承，封装，多态。
继承：类和接口的继承关系。继承可以有很多层次。
封装：类包含属性、方法。
多态：一个方法有多种实现。

使用 struct 和指针实现面向对象

编写代码：　object.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 动物。
struct Animal
{
    // 名称
    char name[30];
};

// 可以飞行
struct FlyAnimal
{
    // 继承父类
    struct Animal animal;
    // 飞行高度
    double flyHeight;
    // 函数，飞行
    void (*flyFunc)(struct FlyAnimal *flyAnimal);
};

// 鸟。
struct Bird
{
    // 继承父类
    struct FlyAnimal flyAnimal;
```

```
    // 唱歌时长
    int singMinute;
    // 鸟，唱歌。
    void (*birdSingFunc)(struct Bird *bird);
};

// 蜜蜂
struct Bee
{
    // 继承父类
    struct FlyAnimal flyAnimal;
    // 鸟，采花蜜。
    void (*beeFlowerFunc)(struct Bee *bee);
};

void func_fly(struct FlyAnimal *flyAnimal)
{
    printf("%s 飞行，高度 %.1f 米 \n", flyAnimal->animal.name, flyAnimal->flyHeight);
}

void func_bird_sing_less(struct Bird *bird)
{
    printf("鸟 %s 唱歌 %d 分钟 \n", bird->flyAnimal.animal.name, bird->singMinute);
}

void func_bird_sing_more(struct Bird *bird)
{
    printf("鸟 %s 在白杨树的枝头，唱歌 %d 分钟，美丽动听 \n",
           bird->flyAnimal.animal.name, bird->singMinute);
}

void func_bee_flower_less(struct Bee *bee)
{
    printf("蜜蜂 %s 采花蜜，只采了 1 分钟 \n", bee->flyAnimal.animal.name);
}

void func_bee_flower_more(struct Bee *bee)
{
    printf("蜜蜂 %s 慢悠悠地采花蜜 5 小时 \n", bee->flyAnimal.animal.name);
}

int main()
{
    printf("\n Bird : \n");
    struct Bird bird_tom;

    // 名称，写到父类的属性。
    sprintf((bird_tom.flyAnimal.animal.name), "bird_tom");
```

```
    // 设置子类的属性
    bird_tom.singMinute = 3;

    // 设置父类的属性和方法
    bird_tom.flyAnimal.flyHeight = 200;
    bird_tom.flyAnimal.flyFunc = func_fly;

    // 调用父类的方法
    bird_tom.flyAnimal.flyFunc((struct FlyAnimal *)&bird_tom);

    // 调用子类的方法
    bird_tom.birdSingFunc = func_bird_sing_less;
    bird_tom.birdSingFunc(&bird_tom);

    // 更换方法，调用方法
    bird_tom.birdSingFunc = func_bird_sing_more;
    bird_tom.birdSingFunc(&bird_tom);

    // --------------------------
    printf("\n Bee : \n");
    struct Bee bee_jerry;
    sprintf((bee_jerry.flyAnimal.animal.name), "bee_jerry");

    // 设置父类的属性和方法
    bee_jerry.flyAnimal.flyHeight = 3;
    bee_jerry.flyAnimal.flyFunc = func_fly;

    // 调用父类的方法
    bee_jerry.flyAnimal.flyFunc((struct FlyAnimal *)&bee_jerry);

    // 调用子类的方法
    bee_jerry.beeFlowerFunc = func_bee_flower_less;
    bee_jerry.beeFlowerFunc(&bee_jerry);

    // 更换方法，调用方法
    bee_jerry.beeFlowerFunc = func_bee_flower_more;
    bee_jerry.beeFlowerFunc(&bee_jerry);

    return 0;
}
```

编译代码：

```
gcc object.c -o object
```

运行代码：

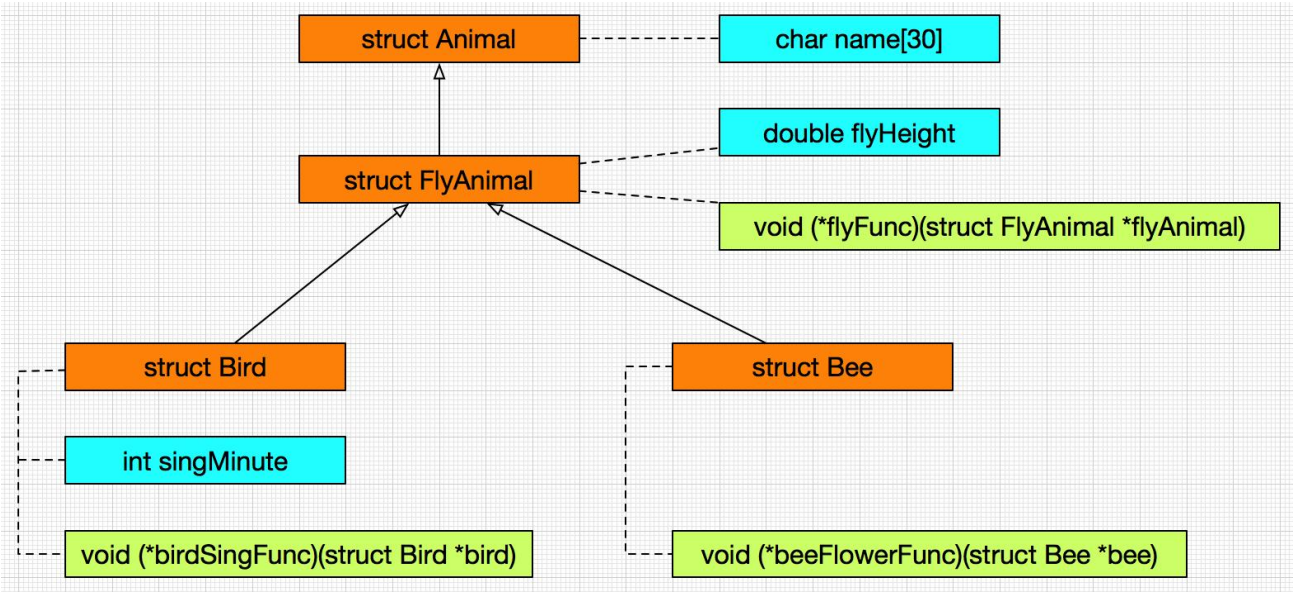
```
[root@local object]# ./object

 Bird :
bird_tom 飞行，高度 200.0 米
```

```
鸟 bird_tom 唱歌 3 分钟
鸟 bird_tom 在白杨树的枝头，唱歌 3 分钟，美丽动听

 Bee :
bee_jerry 飞行，高度 3.0 米
蜜蜂 bee_jerry 采花蜜，只采了 1 分钟
蜜蜂 bee_jerry 慢悠悠地采花蜜 5 小时
```

继承的关系图：
struct FlyAnimal 继承 struct Animal。
struct Bird 继承 struct FlyAnimal。
struct Bee 继承 struct FlyAnimal。



方法的多态：
struct Bee 包含函数属性 `void (*beeFlowerFunc)(struct Bee *bee)` 。beeFlowerFunc 可以动态指定具体的函数。
指定函数 `beeFlowerFunc = func_bee_flower_less` ，打印结果 `蜜蜂 bee_jerry 采花蜜，只采了 1 分钟` 。
指定函数 `beeFlowerFunc = func_bee_flower_more` ，打印结果 `蜜蜂 bee_jerry 慢悠悠地采花蜜 5 小时` 。

用 C 分析函数集合

1 个函数占用 1 个指针。对象可以包含很多函数，把这些函数打包为函数集合。

编写代码： func_pack.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Bird;

// struct 只包含方法。一类方法打包在一起。
```

```
struct BirdFunc
{
    void (*fly)(struct Bird *bird);  // 飞行
    void (*sing)(struct Bird *bird); // 唱歌
};

// struct 实体
struct Bird
{
    char name[30];   // 属性
    double flySpeed; // 属性

    struct BirdFunc *birdFunc; // 方法集合
};

void func_fly_fast(struct Bird *bird)
{
    printf(" %s 飞行很快，速度达到了 %.1f \n", bird->name, bird->flySpeed);
}
void func_fly_slow(struct Bird *bird)
{
    printf(" %s 飞行很慢，速度只有 %.1f \n", bird->name, bird->flySpeed);
}
void func_sing_good(struct Bird *bird)
{
    printf(" %s 唱歌好听 good \n", bird->name);
}
void func_sing_bad(struct Bird *bird)
{
    printf(" %s 唱歌一般 bad \n", bird->name);
}

// 麻雀的方法
struct BirdFunc sparrowFunc = {
    .fly = func_fly_fast,
    .sing = func_sing_good};

// 大雁的方法
struct BirdFunc gooseFunc = {
    .fly = func_fly_slow,
    .sing = func_sing_bad};

int main()
{
    // 一个对象
    struct Bird blue_bird;
    sprintf(blue_bird.name, "蓝色的小鸟");
    blue_bird.flySpeed = 100;
```

```
    // 小鸟，如果是麻雀，使用麻雀的方法集合
    blue_bird.birdFunc = &sparrowFunc;
    blue_bird.birdFunc->fly(&blue_bird);
    blue_bird.birdFunc->sing(&blue_bird);

    printf("\n ====  切换函数集合前后  ====  \n\n");

    // 小鸟，如果是大雁，使用大雁的方法集合
    blue_bird.birdFunc = &gooseFunc;
    blue_bird.birdFunc->fly(&blue_bird);
    blue_bird.birdFunc->sing(&blue_bird);

    return 0;
}
```

编译代码：

```
gcc func_pack.c -o func_pack
```

运行代码：

```
[root@local object]# ./func_pack
 蓝色的小鸟 飞行很快，速度达到了 100.0
 蓝色的小鸟 唱歌好听 good

 ====  切换函数集合前后  ====

 蓝色的小鸟 飞行很慢，速度只有 100.0
 蓝色的小鸟 唱歌一般 bad
```

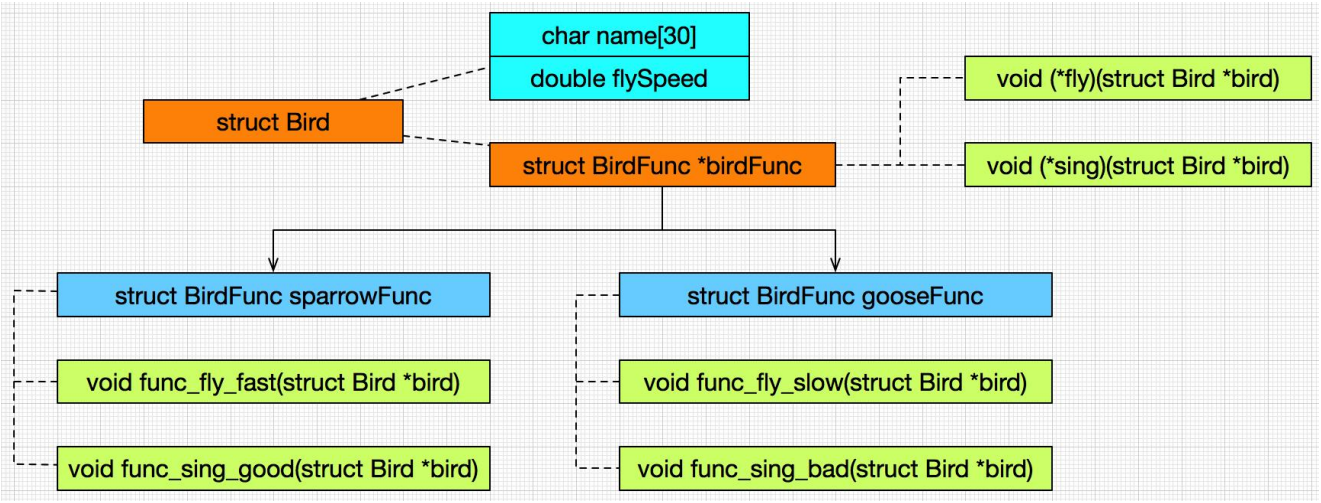
函数集合的逻辑图：

struct Bird 包含 `struct BirdFunc *birdFunc` 。

struct BirdFunc 包含函数 fly、sing。

struct BirdFunc sparrowFunc 包含函数 func_fly_fast、func_sing_good。

struct BirdFunc gooseFunc 包含函数 func_fly_slow、func_sing_bad。



把 blue_bird.birdFunc 指向 sparrowFunc，调用函数 fly、sing，实际执行函数 func_fly_fast、func_sing_good。

把 blue_bird.birdFunc 指向 gooseFunc，调用函数 fly、sing，实际执行函数 func_fly_slow、func_sing_bad。