Objects and Classes - 如何引入类?

属性合并，C语言里使用了结构体。作为基于C的加强语言，有更深入的机制，就是类

存在一个朴素的准则：好的工具应该**实现形式和逻辑的统一。**

封装

程序设计语言中，使用形式上的**语法结构**或者特定**语言机制**来反映逻辑上的强相关

相关属性合并成一个整体：封装【在同一个结构中】

```
struct RECT {
    int width;
    int height;
};

class Rect {
public:
    int width;
    int height;
};
```

数据类型和处理该数据类型的函数，在形式上关联，合并成整体：更多的封装

```
struct RECT {
    int width;
    int height;
};
int Area(struct RECT rc) { //函数处理信息，信息要以参数形式传递
    return rc.width * rc.height; }

class Rect {
public:
    int width;
    int height;
public:
    int Area() { return width * height; }//类中的函数，其数据可以直接访问，不需要额外传
递!
};

int main(){
    struct RECT RC;
    RC.width = 2; RC.height = 1;
    int area1 = Area(RC);

    Rect rc;
    rc.width = 2; rc.height = 1;
    int area2 = rc.Area();

    return 0;
```

```
}
```

进一步抽象成：属性和接口

- **【模块使用】的角度，只关注该模块的功能，并不关注内部结构和实现细节。**
  - 一个模块可以提供多个接口
- 公开接口，并将属性隐藏

单个实体，包含多属性。如，矩形，圆

```
class Rect {
private:
  int width;
  int height;
public:
  void Init(int w, int h)   {
      width = w;
      height = h;
  }

  int Area() {
      return width * height;
  }
};

int main()
{
  Rect rc;
  rc.Init(2, 1);
  int area2 = rc.Area();

  return 0;
}
```

多个实体组合。如，电脑，个人信息等

```
class CPU {};
class RAM {};

class Computer {
private:
    CPU cpu;
    RAM ram;
};
```

容器。存放一组数据，如线性表。

```
class MyList {
private:
    int buffer[1024];
    int size;
public:
    void push_back(int n);
    int pop_back();
};
```

抽象实体，比如一组排序算法【封装的继续抽象】

```
class SortAlgorithm
{
public:
    void Act(int* arr, int n);
};
```

为什么要封装?

- 定义接口，隐藏细节
  - 接口意味着考虑问题从功能逻辑的视角出发，更符合人们解决问题的思维【构建复杂系统，层级思维】
  - 功能提供者和使用者可能是不同的人或者模块，之间的交互方式非常重要（接口用途）
- 实现对接口和实现的分离
  - 保持接口不变，更换内部属性和实现，对于调用者来说完全不受影响。

```
class MyList {
private:
    int *buffer;
    int maxSize;
    int size;
public:
    void push_back(int n);
    int pop_back();
};
```