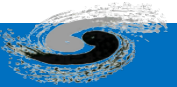
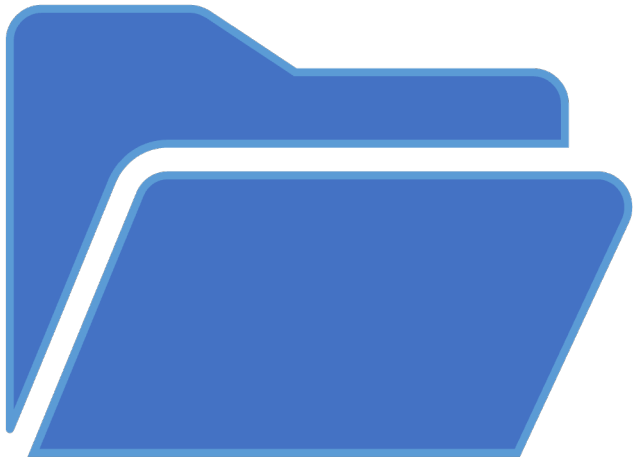
软件框架编程实践
Python/C++为例

田浩来

IHEP School of Computing 2021



目录



- 从结构化编程到面向对象编程
- 从面向对象编程到面向接口编程
- Python/C++跨语言调用

Data, Code, Loop and Branching

- 基本数值类型

name	Example
int	3
float	2.0e-9
complex	1+2j
bool	True/False

- 集合类型

name	Example
str	’hello world’
list	[1, 2, 3]
tuple	(‘a’, ‘b’, ‘c’)
dict	{‘a’:1, ‘b’:2, ‘c’:3}
set	{‘a’, ‘b’, ’c’}

- 语句，运算符

```
In [1]: c1 = 1+2j

In [2]: c2 = 2+3j

In [3]: c1+c2
Out[3]: (3+5j)

In [4]: c1*c2
Out[4]: (-4+7j)
```

- 列表操作

```
In [1]: l = [i for i in range(10)]

In [2]: print(l)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [3]: [i for i in l if i >= 5]
Out[3]: [5, 6, 7, 8, 9]
```

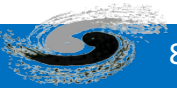
Function

- 函数定义
  - 关键字: def
  - 函数名: displacement
  - 输入参数: t, a (默认输入值)
  - 返回值: return s

```
In [1]: def displacement(t, a = 9.81):
   ...:     s = 0.5*a*t**2
   ...:     return s
   ...:

In [2]: displacement(2)
Out[2]: 19.62
```

- 匿名函数 lambda

```
In [4]: l
Out[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [5]: [displacement(i) for i in l]
Out[5]: [0.0, 4.905, 19.62, 44.145, 78.48, 122.625, 176.58, 240.345, 313.92, 397.305]

In [6]: list(map(lambda t: 0.5*9.81*t**2, l))
Out[6]: [0.0, 4.905, 19.62, 44.145, 78.48, 122.625, 176.58, 240.345, 313.92, 397.305]
```



Class

- 面向对象编程(Object-oriented Programming, OOP)
  - 类(对象的类型)
  - 对象(实例化的类)
  - 属性(对象的数据)
  - 方法(对象的函数)
- 派生和继承
  - 基类/派生类
  - 方法的多态

```python
class School:
    def __init__(self, school_name):
        self.name = school_name
        self.students = {}

    def getName(self):
        return self.name

    def getStudents(self):
        return self.students

    def getStudent(self, s_id):
        return self.students[s_id]

    def setStudent(self, s_id, s_name):
        self.students[s_id] = s_name
```

```python
class SummerSchool(School):
    def __init__(self, school_name, year):
        super().__init__(school_name=school_name)
        self.year = year

    def getYear(self):
        return self.year

    def getName(self):
        ret = self.name + '@' + str(self.year)
        return ret
```

Module

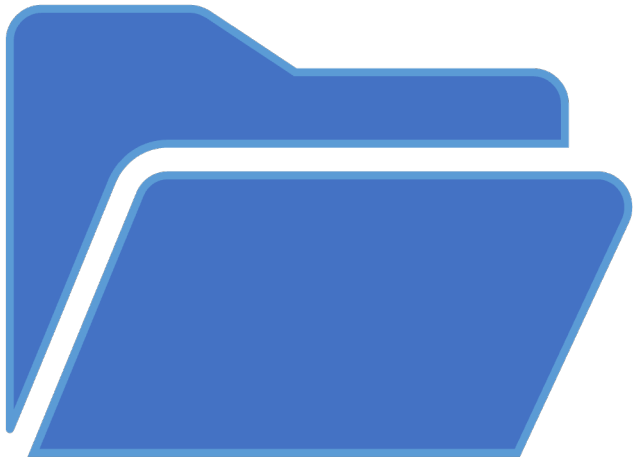
- 模块的导入

```
In [1]: from SCHOOL import School
SCHOOL MODULE

In [2]: sc1 = School.SingleSchool()

In [3]: sc2 = School.SingleSchool()

In [4]: id(sc1) == id(sc2)
Out[4]: True
```

- 模块的定义

```
(base) haolaitian@haolaitiandeMacBook-Air SCHOOL % ls
School.py       __init__.py     __pycache__
(base) haolaitian@haolaitiandeMacBook-Air SCHOOL % cat __init__.py
print("SCHOOL MODULE")
```

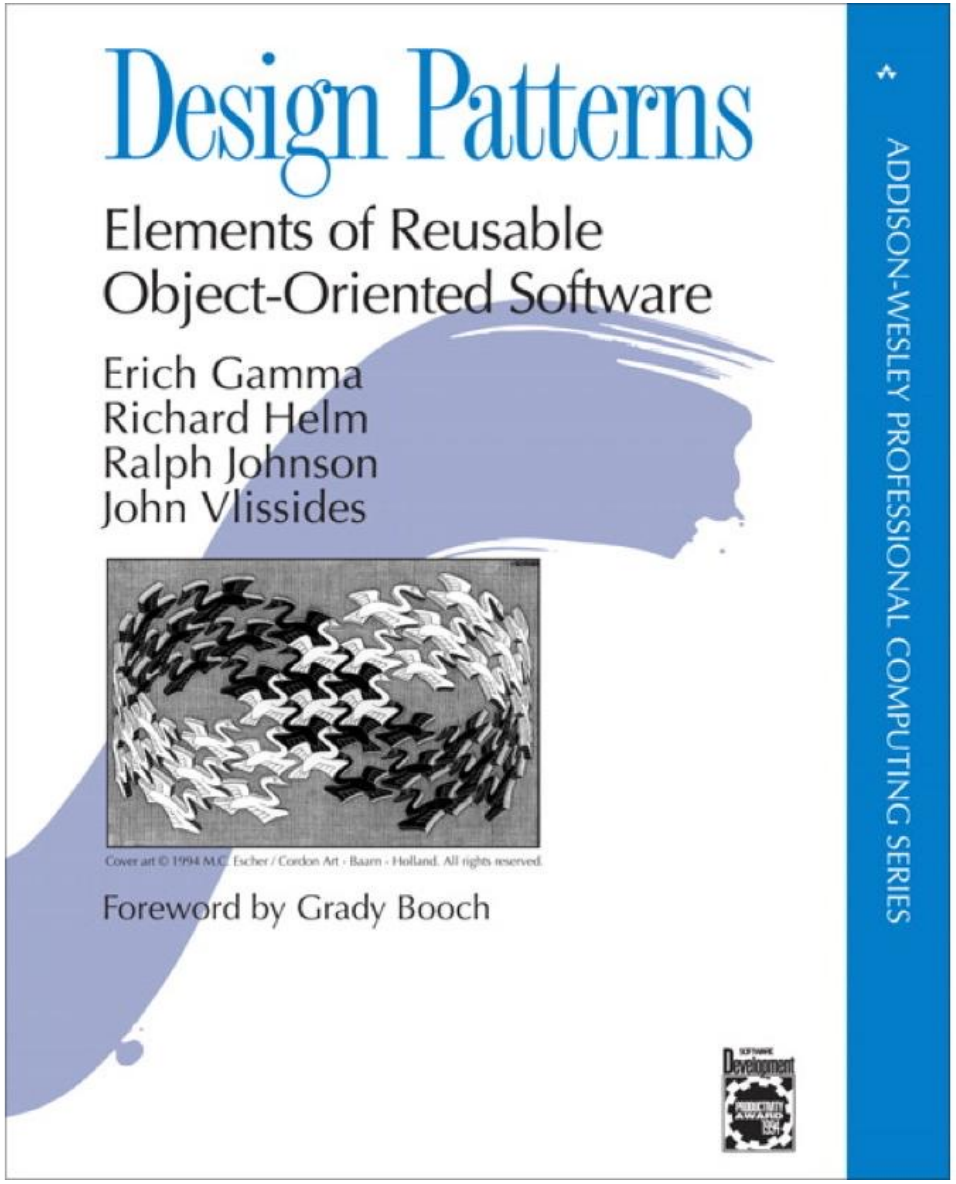
目录



- 从结构化编程到面向对象编程
- 从面向对象编程到面向接口编程
- Python/C++跨语言调用

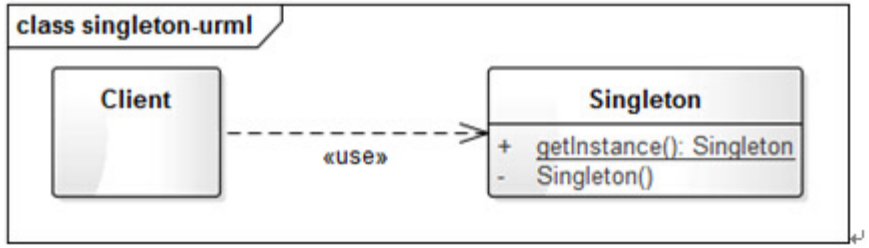
设计模式(GoF23)

- 面向接口编程6个原则
  - 开放封闭(接口隔离原则)
  - 高内聚低耦合(单一职责原则)
  - 依赖倒置(子类替换原则)
- 创建型模式(5)
  - 单实例模式(Singleton)
- 结构型模式(7)
  - 装饰器(Decorator)
- 行为型模式(11)
  - 观察者模式(Observer)



单实例模式(Singleton)

- 一个类只有一个实例
  - 该单例对象必须由单例类自行创建
  - 单例类对外提供一个访问该单例的全局访问点



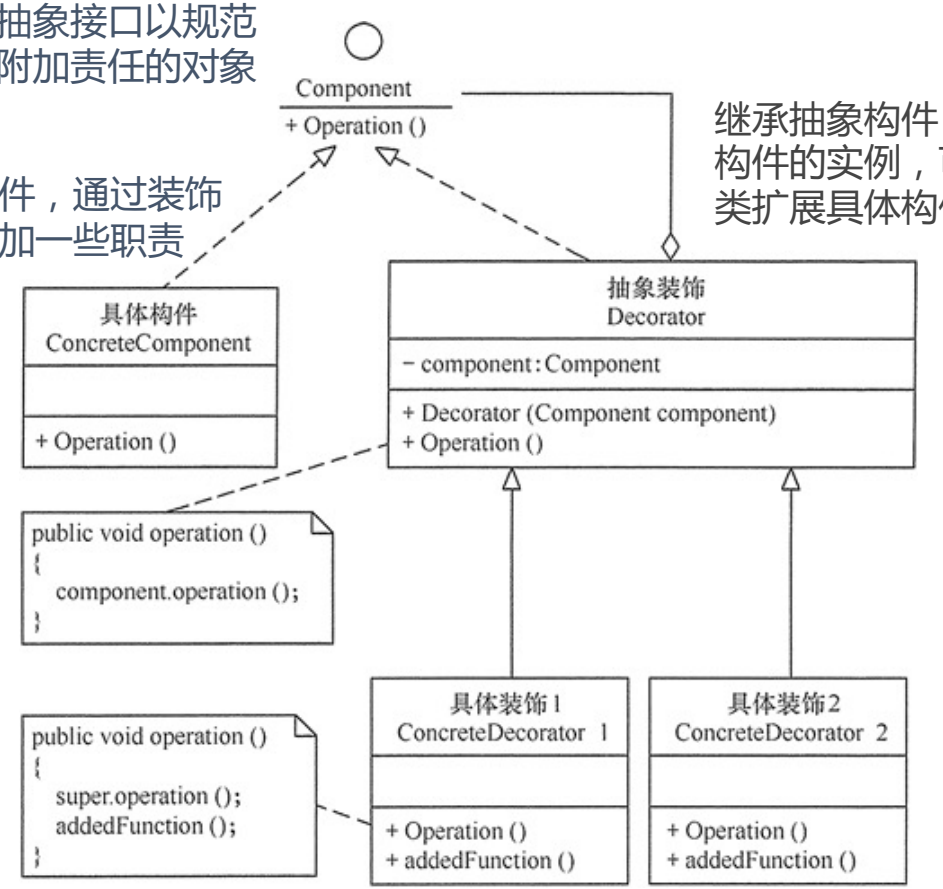
- 实现方法
  - 私有化构造函数，不让外部访问，提供一个对外方法该单例实例对象方法

```
class SingleSchool(object):
    _instance = None

    def __new__(cls, *args, **kw):
        if cls._instance is None:
            cls._instance = super().__new__(cls, *args, **kw)
        return cls._instance
```

装饰器(Decorator)

- 实现抽象装饰的相关方法，并给具体构件对象添加附加的责任。

定义一个抽象接口以规范
准备接收附加责任的对象



实现抽象构件，通过装饰
角色为其添加一些职责

继承抽象构件，并包含具体
构件的实例，可以通过其子
类扩展具体构件的功能

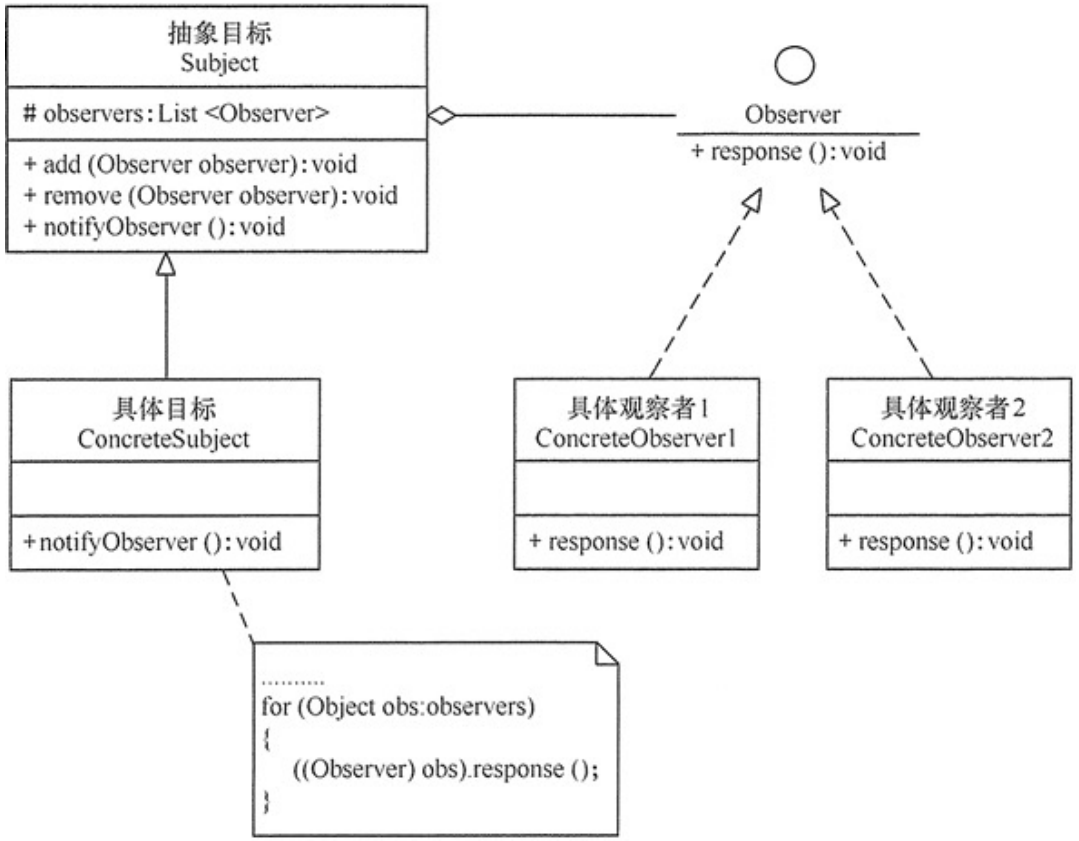
实现抽象装饰的相关方法，并给具体构件对象添加附加的责任

Python中采用闭包(函数中嵌套函数)来实现装饰器
在语言层次实现了装饰器@singleton

```
def singleton(cls):
    _instance = {}

    def inner(name, year):
        if cls not in _instance:
            _instance[cls] = cls(name, year)
        return _instance[cls]
    return inner

@singleton
class SummerSchool(School):
    def __init__(self, school_name, year):
        super().__init__(school_name=school_name)
        self.year = year
```

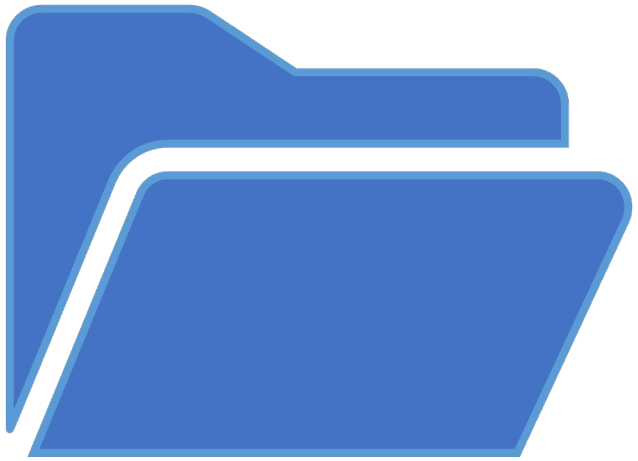
观察者模式(Observer)

- 发布-订阅：当一个对象的状态发生改变时，所有依赖于它的多个对象都得到通知并被自动更新




```python
class University:
    actions = []

    @classmethod
    def attach(cls, action):
        cls.actions.append(action)

    @classmethod
    def notify(cls):
        print("上级通知")
        for action in cls.actions:
            print(action())


sc1 = School('Math')
sc2 = School('Phys')
sc3 = School('EE')

u = University()
u.attach(sc1.getName)
u.attach(sc2.getName)
u.attach(sc3.getName)
u.notify()
```

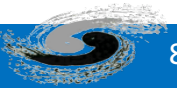
目录



- 从结构化编程到面向对象编程
- 从面向对象编程到面向接口编程
- Python/C++跨语言调用

Boost.Python简介

- 安装开发环境
  - Boost头文件和动态链接库
  - Python头文件和动态链接库
- C++代码开发
  - 普通C++头文件(*.h)
  - 普通C++源代码(*.cpp)
  - Boost.Python封装类源代码(*_wrapper.cpp)
  - 在编译过程中指定编译链接的对象(Makefile)



Boost.Python简介

```cpp
#include "SniperKernel/DLElement.h"
#include "SniperKernel/SniperPtr.h"
#include "SniperKernel/SniperDataPtr.h"
#include "SniperKernel/SniperLog.h"

class ToolBase;

class AlgBase : public DLElement
{
public:
    AlgBase(const std::string &name);

    virtual ~AlgBase();

    virtual bool execute() = 0;

    //for handling tools if needed
    ToolBase *createTool(const std::string &toolName);
    ToolBase *findTool(const std::string &toolName);

    //template version of retrieving a Tool instance
    template <typename Type>
    Type *tool(const std::string &toolName);

    //Declared in base class DLElement
    //virtual bool initialize() = 0;
    //virtual bool finalize() = 0;

    //the json value of this object
    virtual SniperJSON json();
    // eval this Algorithm from json
    virtual void eval(const SniperJSON &json) override;

protected:
    std::map<std::string, ToolBase *> m_tools;
};

template <typename Type>
Type *AlgBase::tool(const std::string &toolName)
{
    return dynamic_cast<Type *>(this->findTool(toolName));
}
```

```cpp
#include "SniperKernel/AlgBase.h"
#include "SniperKernel/ToolBase.h"
#include <boost/python.hpp>

namespace bp = boost::python;

struct AlgBaseWarp : AlgBase, bp::wrapper<AlgBase>
{
    AlgBaseWarp(const std::string& name)
        : AlgBase(name)
    {
    }

    bool initialize() {
        return this->get_override("initialize")();
    }

    bool execute() {
        return this->get_override("execute")();
    }

    bool finalize() {
        return this->get_override("finalize")();
    }
};

void export_Sniper_AlgBase()
{
    using namespace boost::python;

    class_<AlgBaseWarp, bases<DLElement>, boost::noncopyable>
        ("AlgBase", init<const std::string&>())
        .def("initialize", pure_virtual(&AlgBase::initialize))
        .def("execute",    pure_virtual(&AlgBase::execute))
        .def("finalize",   pure_virtual(&AlgBase::finalize))
        .def("createTool", &AlgBase::createTool,
                return_value_policy<reference_existing_object>())
        .def("findTool",   &AlgBase::findTool,
                return_value_policy<reference_existing_object>())
        ;
}
```


enjoy it!