day20

今日内容

- 类成员
- 成员修饰符

内容回顾 & 补充

1. 三大特性
    - 封装
        - 函数封装到类
        - 数据封装到对象 *

        ```
        class Foo:
            def __init__(self,name,age):
                self.name = name
                self.age = age

        obj = Foo('alex',19)
        ```

    - 继承
    - 多态
2. 作业

内容详细

1.成员

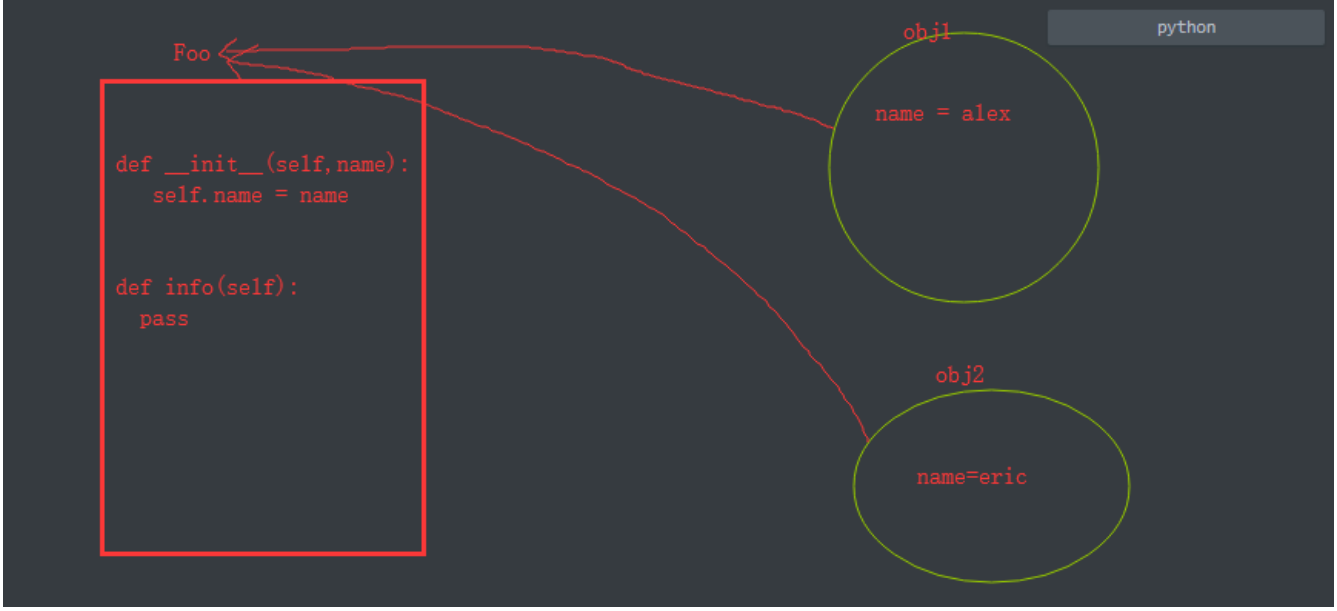
- 类
    - 类变量
    - 绑定方法
    - 类方法
    - 静态方法
    - 属性
- 实例（对象）
    - 实例变量

1.1 实例变量

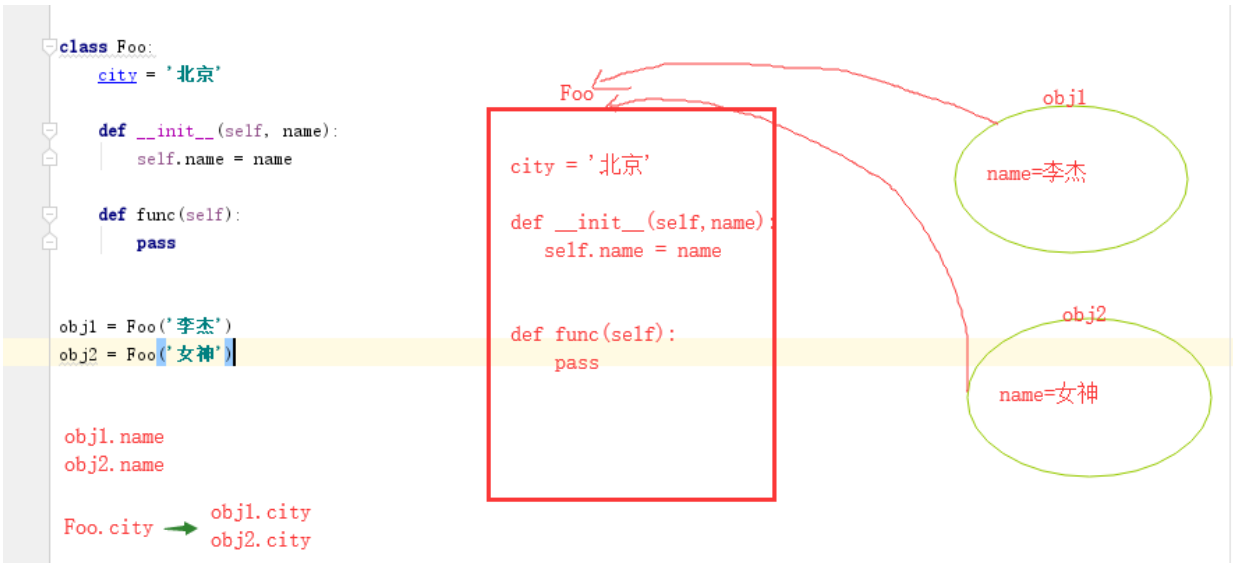
```python
class Foo:
    def __init__(self,name):
        self.name = name

    def info(self):
        pass

obj1 = Foo('alex')
obj2 = Foo('eric')
```



1.2 类变量



- 定义：写在类的下一级和方法同一级。
- 访问：

```
类.类变量名称
对象.类变量名称
```

- 面试题

```
class Base:
    x = 1

obj = Base()


print(obj.x) # 先去对象中找，没有再去类中找。
obj.y = 123  # 在对象中添加了一个y=123的变量。
print(obj.y)
obj.x = 123
print(obj.x)
print(Base.x)
```

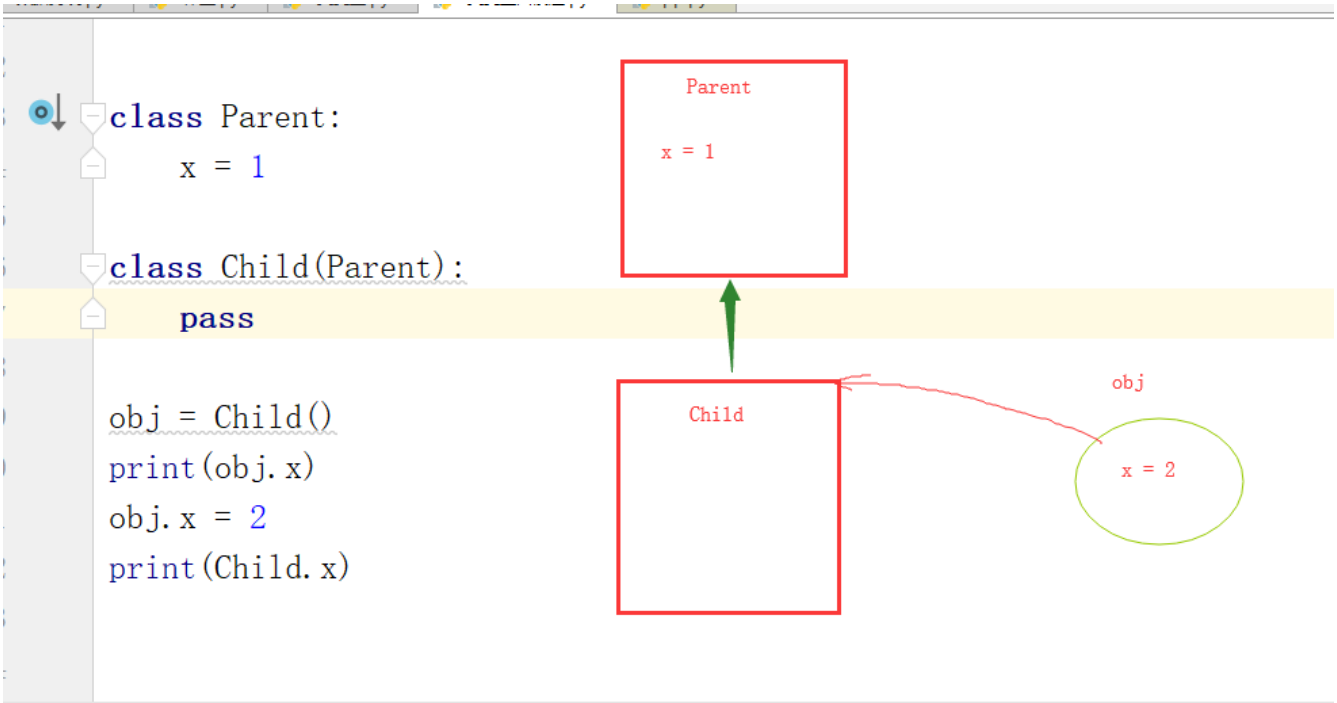
```
class Parent:
    x = 1

class Child1(Parent):
    pass

class Child2(Parent):
    pass

print(Parent.x,Child1.x,Child2.x) # 1 1 1
Child1.x = 2
print(Parent.x,Child1.x,Child2.x) # 1 2 1
Child2.x = 3
print(Parent.x,Child1.x,Child2.x) # 1 2 3
```

总结：找变量优先找自己，自己没有找 类 或 基类；修改或赋值只能在自己的内部设置。



1.3 方法（绑定方法/普通方法）

- 定义：至少有一个self参数
- 执行：先创建对象，由对象.方法()。

```
class Foo:
    def func(self,a,b):
        print(a,b)

obj = Foo()
obj.func(1,2)
# ############################
class Foo:
    def __init__(self):
        self.name = 123

    def func(self, a, b):
        print(self.name, a, b)

obj = Foo()
obj.func(1, 2)
```

1.4 静态方法

- 定义：
  - @staticmethod装饰器
  - 参数无限制
- 执行：
  - 类.静态方法名 ()
  - 对象.静态方法() (不推荐)

```
class Foo:
    def __init__(self):
        self.name = 123

    def func(self, a, b):
        print(self.name, a, b)

    @staticmethod
    def f1():
        print(123)

obj = Foo()
obj.func(1, 2)

Foo.f1()
obj.f1() # 不推荐
```

1.5 类方法

- 定义：
  - @classmethod装饰器

    - 至少有cls参数，当前类。
- 执行：
    - 类.类方法()
    - 对象.类方法() （不推荐）

```
class Foo:
    def __init__(self):
        self.name = 123

    def func(self, a, b):
        print(self.name, a, b)

    @staticmethod
    def f1():
        print(123)

    @classmethod
    def f2(cls,a,b):
        print('cls是当前类',cls)
        print(a,b)

obj = Foo()
obj.func(1, 2)

Foo.f1()
Foo.f2(1,2)
```

面试题：

```
# 问题： @classmethod和@staticmethod的区别?
"""
一个是类方法一个静态方法。
定义：
    类方法：用@classmethod做装饰器且至少有一个cls参数。
    静态方法：用staticmethod做装饰器且参数无限制。
调用：
    类.方法直接调用。
    对象.方法也可以调用。
"""
```

1.6 属性

- 定义：
    - @property装饰器
    - 只有一个self参数
- 执行：
    - 对象.方法 不用加括号。

```python
class Foo:

    @property
    def func(self):
        print(123)
        return 666

obj = Foo()
result = obj.func
print(result)
```

```python
# 属性的应用

class Page:
    def __init__(self, total_count, current_page, per_page_count=10):
        self.total_count = total_count
        self.per_page_count = per_page_count
        self.current_page = current_page
    @property
    def start_index(self):
        return (self.current_page - 1) * self.per_page_count
    @property
    def end_index(self):
        return self.current_page * self.per_page_count


USER_LIST = []
for i in range(321):
    USER_LIST.append('alex-%s' % (i,))

# 请实现分页展示:
current_page = int(input('请输入要查看的页码: '))
p = Page(321, current_page)
data_list = USER_LIST[p.start_index:p.end_index]
for item in data_list:
    print(item)
```

2.成员修饰符

- 公有，所有地方都能访问到。
- 私有，只有自己可以访问到。

```python
class Foo:
    def __init__(self, name):
        self.__name = name

    def func(self):
        print(self.__name)


obj = Foo('alex')
# print(obj.__name)
obj.func()
```

```python
class Foo:
    __x = 1

    @staticmethod
    def func():
        print(Foo.__x)


# print(Foo.__x)
Foo.func()
```

```python
class Foo:

    def __fun(self):
        print('msg')

    def show(self):
        self.__fun()

obj = Foo()
# obj.__fun()
obj.show()
```

3.小补充

```python
class Foo:
    def __init__(self,num):
        self.num = num


cls_list = []
for i in range(10):
    cls_list.append(Foo)

for i in range(len(cls_list)):
    obj = cls_list[i](i)
    print(obj.num)
```

```python
class Foo:
    def __init__(self,num):
        self.num = num

B = Foo
obj = B('alex')
```

```python
class Foo:
    def f1(self):
        print('f1')

    def f2(self):
        print('f2')

obj = Foo()

v = [ obj.f1,obj.f2 ]
for item in v:
    item()
```

```python
class Foo:
    def f1(self):
        print('f1')

    def f2(self):
        print('f2')

    def f3(self):
        v = [self.f1 , self.f2 ]
        for item in v:
            item()

obj = Foo()
obj.f3()
```

```python
class Account:

    def login(self):
        pass

    def register(self):
        pass

    def run(self):
        info = {'1':self.register, '2':self.login }
        choice = input('请选择:')
        method = info.get(choice)
        method()
```

```python
class Foo:
```

```
    pass

class Foo(object):
    pass

# 在python3中这俩的写法是一样，因为所有的类默认都会继承object类，全部都是新式类。


# 如果在python2中这样定义，则称其为：经典类
class Foo:
    pass
# 如果在python2中这样定义，则称其为：新式类
class Foo(object):
    pass

class Base(object):
    pass
class Bar(Base):
    pass
```

赠送

```
# 强制访问私有成员

class Foo:
    def __init__(self,name):
        self.__x = name


obj = Foo('alex')

print(obj._Foo__x) # 强制访问私有实例变量
```

总结

1. 数据封装
2. 继承关系的查找
3. 嵌套

```
class School(object):
    def __init__(self,title,addr):
        self.title = title
        self.address = addr

class ClassRoom(object):
```

```
    def __init__(self,name,school_object):
        self.name = name
        self.school = school_object

s1 = School('北京','沙河')
s2 = School('上海','浦东')
s3 = School('深圳','南山')

c1 = ClassRoom('全栈21期',s1)
c1.name
c1.school.title
c1.school.address
# ##############################################
v = [11,22,33,{'name':'山海','addr':'浦东'}]

v[0]
v[3]['name']
```