二十三、面向对象基础

目标

- 理解面向对象
- 类和对象
- 添加和获取对象属性
- 魔法方法

23.1 理解面向对象

面向对象是一种抽象化的编程思想，很多编程语言中都有的一种思想。

例如：洗衣服

思考：几种途径可以完成洗衣服？

答：手洗和机洗

手洗：找盆-放水-加洗衣粉-浸泡-搓洗-拧水-倒水-漂洗N次-拧干-晾晒。

机洗：打开洗衣机-放衣服-加洗衣粉-按下开始按钮-晾晒。

思考：对比两种洗衣服途径，发现了什么？

答：机洗更简单

思考：机洗，只需要找到一台洗衣机，加入简单操作就可以完成洗衣服的工作，而不需要关心洗衣机内 部发生了什么事情。

总结：面向对象就是将编程当成是一个事物，对外界来说，事物是直接使用的，不用去管他内部的情况。而编程就是设置事物能够做什么事。

23.2 类和对象

思考：洗衣机洗衣服描述过程中，洗衣机其实就是一个事物，即对象，洗衣机对象哪来的呢？

答：洗衣机是由工厂工人制作出来。

思考：工厂工人怎么制作出的洗衣机？

答：工人根据设计师设计的功能图纸制作洗衣机。

总结：图纸→洗衣机→洗衣服。

在面向对象编程过程中，有两个重要组成部分：类和对象。

类和对象的关系：用类去创建一个对象。

23.3 理解类和对象

23.3.1 类

类是对一系列具有相同特征和行为的事物的统称，是一个抽象的概念，不是真实存在的事物。

- 特征即是属性
- 行为即是方法

类比如是制造洗衣机时要用到的图纸，也就是说类是用来创建对象。

23.3.2 对象

对象是类创建出来的真实存在的事物，例如：洗衣机。

注意：开发中，先有类，再有对象。

23.4 面向对象实现方法

23.4.1 定义类

- 语法

```
class 类名():
    代码
    .....

```

注意：类名要满足标识符命名规则，同时遵循大驼峰命名习惯

- 体验

```
class Washer():
    def wash(self):
        print("我会洗衣服")
```

- 拓展：经典类

```
class 类名:
    代码
    ....
```

23.4.2 创建对象

对象又名实例

- 语法

```
对象名 = 类名()
```

- 体验

```
haier1 = Washer() # 创建对象
print(haier1) # <__main__.Washer object at 0x00000194F1331160>
# 使用wash功能 -- 实例方法/对象方法 -- 对象名.wash()
haier1.wash()
```

23.4.3 self

self指的是调用该函数的对象

```
# 1、定义类
class Washer():
    def wash(self):
        print('我会洗衣服')
        print(self) # <__main__.Washer object at 0x0000021B3A5EE160>

# 2、创建对象
haier1 = Washer()
print(haier1)  # <__main__.Washer object at 0x000002BB8EA31160>

haier1.wash()
# 由于打印对象和打印self得到的内存地址相同，所以self 指的是调用该函数的对象
```

23.4.4 一个类创建多个对象

```
# 1、一个类可以创建多个对象
# 2、多个对象都调用函数的时候，self地址是否相同 ---不相同

# 1、定义类
class Washer():
    def wash(self):
        print('我会洗衣服')
        print(self) # <__main__.Washer object at 0x0000021B3A5EE160>

# 2、创建对象
haier1 = Washer()
haier1.wash()

haier2 = Washer()
haier2.wash() #
```

23.4.5扩展经典类和新式类

- **拓展1**：经典类或旧式类

不由任意内置类型派生出的类，称之为经典类。

```
class 类名:
    ....
```

- **拓展2**：新式类

```
class 类名(object):
    代码
    ....
```

Python面向对象的继承指的是多个类之间的所属关系，即子类默认继承父类的所有属性的方法，具体如

23.5 添加和获取对象属性

属性即是特征，比如：洗衣机的宽度、高度、重量

对象属性既可以在类外面添加和获取，也能在类里面添加和获取。

23.5.1 类外面添加对象属性

- 语法

```
对象名.属性名 = 值
```

- 体验

```
haier1.width =500
haier1.height = 500
```

23.5.2 类外面获取对象属性

- 语法

```
对象名.属性名
```

- 体验

```
print(f'haier1洗衣机的宽度是{haier1.width}')
print(f'haier1洗衣机的高度是{haier1.height}')
```

23.5.3 类里面获取对象属性

- 语法

```
self.属性名
```

- 体验

```

# 1、定义类
class Washer():
    def wash(self):
        print('我会洗衣服')
    # 获取对象属性
    def print_info(self):
        """self.属性名"""
        #print(self.width)
        # 在类里面定义一个实例方法
        print(f'洗衣机的宽度是{self.width}')
        print(f'洗衣机的高度是{self.heigth}')
```

```
13
14 haier1 = Washer()
15
16 # 添加属性
17 haier1.width = 400
18 haier1.heigth = 500
19
20 # 对象调用方法
21 haier1.print_info()
```

23.6 魔法方法

在python中，`__xx__()` 的函数叫做魔法方法，指的是具有特殊功能的函数。

23.6.1 `__init__()`

23.6.1.1 体验 `__init__()`

思考：洗衣机的宽度高度是与生俱来的属性，可不可以在先产过程中就赋予这些属性呢?

答：理应如此。

`__init__()` 方法的作用：初始化对象

```
1  class Washer():
2      # 定义__init__，添加实例属性(通常是创建类时候，与生俱来的属性)
3      def __init__(self):
4          # 添加实例属性
5          self.width = 400
6          self.height = 500
7      def print_info(self):
8          # 类里面调用实例属性
9          print(f'洗衣机的宽度是{self.width},高度是{self.height}')
10
11 haier1 = Washer() # 创建对象
12 haier1.print_info() # 调用实例对象
```

> 注意：
>
> - `__init__()` 方法，在创建一个对象时默认被调用，不需要手动调用
> - `__init__(self)` 中的self参数，不需要开发者传递， python解释器会自动把当前的对象引 用传递过去。

23.6.1.2 带参数的 `__init__()`

思考：一个类可以创建多个对象，如何对不同的对象设置不同的初始化属性那?

答：传参数。

```
1  # 定义类：带参数的__init__：宽度和高度；实例方法：调用实例属性
2  class Washer():
3      def __init__(self,width,height):
```

```
        self.width = width
        self.height = height

    def print_info(self):
        print(f'洗衣机的宽度是{self.width}')
        print(f'洗衣机的高度是{self.height}')

# 创建对象，创建多个对象且属性值不同；调用实例方法
haier1 = Washer(10,20) # 创建对象
haier1.print_info() # 实例化对象

haier2 = Washer(10,30) # 创建对象
haier2.print_info() # 实例化对象

```

23.6.2 `__str__()`

当使用print输出对象的时候，默认打印对象的内存地址。如果类定义了 `__str__()` 方法，那么就会打印从这个方法中return的数据。

```
class Washer():
    def __init__(self,width, height):
        self.width = width
        self.height = height

    def __str__(self):
        return '这是海尔洗衣机说明书'

haier1 = Washer(10,20)
print(haier1) # 这是海尔洗衣机说明书
```

23.6.3 `__del__()`

当删除对象时候，python解释器也会默认调用 `__del__()` 方法。

```
class Washer():
    def __init__(self,width, height):
        self.width = width
        self.height = height

    def __del__(self):
        print(f'{self}对象已经被删除') # <__main__.Washer object at
0x0000020A40431198>对象已经被删除

haier1 = Washer(10,20)
del haier1 #
```

23.7 综合应用

23.7.1烤地瓜

23.7.1.1 需求

需求主线：

1、被烤的时间和对应的地瓜状态

0-3分钟：生的

3-5分钟：半生不熟

5-8分钟：熟的

超过8分钟：烤糊了

2、添加的调料

用户可以按照自己的意愿添加调料

23.7.1.2 步骤分析

需求涉及一个事物：地瓜，故案例涉及一个类：地瓜类

定义类

- 地瓜的属性

  被烤的时间

  地瓜的状态

  添加的调料
- 地瓜的方法

  被烤

  用户根据意愿设定每次烤地瓜的时间

  判断地瓜被烤的总时间是在哪个区间，修改地瓜状态

  添加调料

  用户根据意愿设定添加的调料

  将用户添加的调料存储
- 显示对象信息

23.7.1.3 具体实现

- 地瓜属性

  定义地瓜初始化属性、后期根据程序推荐更新实例属性

```
# 1、定义类：初始化属性、被烤和添加调料的方法、显示对象信息的str
class SweetPotato():
    def __init__(self):
        # 被烤的时间
        self.cook_time = 0
        # 地瓜的状态
        self.cook_static = '生的'
        # 调料列表
        self.condiments = []
# 2、创建对象并调用 对应的实例方法
```

- 定义烤地瓜方法

```
# 1、定义类：初始化属性、被烤和添加调料的方法、显示对象信息的str
class SweetPotato():
    .....

    def cook(self,time):
        """烤地瓜的方法"""
        # 1、先计算地瓜整体烤过的时间
        # 2、用整体烤过的时间再判断地瓜的状态
        self.cook_time += time
        if 0<=self.cook_time<3:
            self.cook_static = '生的'
        elif 3<=self.cook_time <5:
            self.cook_static = '半生不熟'
        elif 5<=self.cook_time<8:
            self.cook_static = '熟了'
        elif self.cook_time>=8:
            self.cook_static = '烤糊了'
# 2、创建对象并调用 对应的实例方法
```

- 书写str魔法方法，用于输出对象状态

```
class SweetPotato():
    ....

    def __str__(self):
        return f'这个地瓜的被烤过时间是{self.cook_time},状态是{self.cook_static}'
```

- 创建对象，测试实例属性和实例方法

```
# 2、创建对象并调用 对应的实例方法
digua1 = SweetPotato() # 创建对象
print(digua1) # 打印对象--打印的是魔法方法str

digua1.cook(2)
print(digua1)
"""
这个地瓜的被烤过时间是0,状态是生的
这个地瓜的被烤过时间是2,状态是生的
这个地瓜的被烤过时间是4,状态是半生不熟

"""
```

- 定义添加调料方法，并调用该实例方法

```
# 1、定义类：初始化属性、被烤和添加调料的方法、显示对象信息的str
class SweetPotato():
    ...
    def add_condiments(self,condiment):
        """用户意愿的调料追加到调料列表中"""
        self.condiments.append(condiment)

    def __str__(self):
```

```
        return f'这个地瓜的被烤过时间是{self.cook_time},状态是{self.cook_static},调料有{self.condiments}'


# 2、创建对象并调用 对应的实例方法

digua1 = SweetPotato() # 创建对象
print(digua1) # 打印对象--打印的是魔法方法str

digua1.cook(2)
digua1.add_condiments('辣椒面')
print(digua1)

digua1.cook(2)
digua1.add_condiments('酱油')
print(digua1)

"""
这个地瓜的被烤过时间是0,状态是生的,调料有[]
这个地瓜的被烤过时间是2,状态是生的,调料有['辣椒面']
这个地瓜的被烤过时间是4,状态是半生不熟,调料有['辣椒面', '酱油']
"""
```

23.7.2 搬家具

将小于房子剩余面积的家具搬放到房子中

23.7.2.1 步骤分析

需求涉及两个事物：房子 和 家具，故被案例涉及两个类：房子类 和 家具类

- 房子类

  实例属性

  房子占地面积

  房子地理位置

  房子剩余面积

  房子内家具列表

  实例方法

  容纳家具

  显示房屋信息

- 家具类

  家具名称

  家具占地面积

23.7.2.2 具体实现

创建对象并调用相关方法

- 家具类

```python
class Furniture():
    def __init__(self,name,area):
        # 家具名字
        self.name = name
        # 家具占地面积
        self.area = area


```

- 房子类

```python
# 房子类
class Home():

    def __init__(self,address,area):
        # 地理位置
        self.address = address
        # 房屋面积
        self.area = area
        # 剩余面积
        self.free_area = area
        # 家具列表
        self.furniture = []

    def __str__(self):
        return f'房子坐落于{self.address},占地面积{self.area},剩余面积{self.free_area},家具有{self.furniture}'
```

- 容纳家具

```python
    def add_furniture(self,item):
        """容纳家具"""
        # 如果家具占地面积<=房子剩余面积：可以搬入（家具列表添加家具名字，并且房子剩余面积更新）
        # 房屋剩余面积-该家具的占地面积
        # 否则：提示用户家具太大，剩余面积不足，无法容纳
        if item.area <= self.free_area:
            self.furniture.append(item.name)
            self.free_area -= item.area
        else:
            print('用户家具太大，剩余面积不足，无法容纳')
```

所以源码：

```python
# 家具类
class Furniture():
    def __init__(self,name,area):
        # 家具名字
        self.name = name
        # 家具占地面积
        self.area = area

```

```python
# 房子类
class Home():

    def __init__(self,address,area):
        # 地理位置
        self.address = address
        # 房屋面积
        self.area = area
        # 剩余面积
        self.free_area = area
        # 家具列表
        self.furniture = []

    def __str__(self):
        return f'房子坐落于{self.address},占地面积{self.area},剩余面积{self.free_area},家具有{self.furniture}'

    def add_furniture(self,item):
        """容纳家具"""
        # 如果家具占地面积<=房子剩余面积：可以搬入（家具列表添加家具名字，并且房子剩余面积更新）
        # 房屋剩余面积-该家具的占地面积
        # 否则：提示用户家具太大，剩余面积不足，无法容纳
        if item.area <= self.free_area:
            self.furniture.append(item.name)
            self.free_area -= item.area
        else:
            print('用户家具太大，剩余面积不足，无法容纳')

# 双人床
bed = Furniture('双人床', 6)
sofa = Furniture('沙发',10)

# 房子1：北京，1000
jia1 = Home('北京',1000)
print(jia1)

jia1.add_furniture(bed)
print(jia1)

ball = Furniture('篮球场', 2000)
jia1.add_furniture(ball)
print(jia1)
```

二十四、继承

目标

- 继承的概念
- 单继承
- 多继承
- 子类重写父类的同名属性和方法
- 子类调用父类的同名属性和方法
- 多层继承

- super()
- 私有属性和私有方法

24.1 体验继承

- **拓展1**：经典类或旧式类

不由任意内置类型派生出的类，称之为经典类。

```
class 类名:
    ....
```

- **拓展2**：新式类

```
class 类名(object):
    代码
    ....
```

Python面向对象的继承指的是多个类之间的所属关系，即子类默认继承父类的所有属性的方法，具体如

```
# 实例属性
# 实例方法
# 继承：子类默认继承父类的所有属性和方法
# 1、定义父类
class A(object):
    def __init__(self):
        self.num = 1

    def info_print(self):
        print(self.num)

# 2、定义子类，继承父类
class B(A):
    pass

# 3、创建对象，验证结论
result = B()
result.info_print() # 1
```

在 Python中，所有类默认继承object类， object类是顶级类或基类；其他子类叫做派生类。

24.2 单继承

> 故事主线：一个煎饼果子老师傅，在煎饼果子界摸爬滚打多年，研发了一套精湛的摊煎饼果子的技术。师父要把这套技术传授给他的唯一的最得意的徒弟。

分析：徒弟是不是要继承师父的所有技术？

```
# 1、师傅类：属性和方法
class Master(object):
    def __init__(self):
        self.kongfu = '古法煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 2、徒弟类：继承师傅类
class Prentice(Master):
    pass

# 3、用徒弟类创建对象，调用实例属性和方法结论验证
daqiu = Prentice()
print(daqiu.kongfu)
daqiu.make_cake()

"""
古法煎饼果子配方
运用古法煎饼果子配方制作煎饼果子
"""
```

24.3 多继承

> 故事推进：daqiu是个爱学习的好孩子想学习更多的煎饼果子技术，于是，在百度搜索到jjk程序员，报班学习煎饼果子技术。

所谓多继承意思就是一个类同时继承了多个父类。

```
# 1、师傅类：属性和方法
class Master(object):
    def __init__(self):
        self.kongfu = '古法煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

#为了验证我们的多继承，添加school父类
class School(object):
    def __init__(self):
        self.kongfu = 'jjk煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 2、徒弟类：继承师傅类 和 学校类
class Prentice(Master,School): # 想要继承谁，就把谁写在第一个位置
    pass

# 3、用徒弟类创建对象，调用实例属性和方法结论验证
daqiu = Prentice()

```

```
print(daqiu.kongfu)

daqiu.make_cake()
#结论：如果一个类继承多个父类，优先继承第一个父类的同名属性和方法

```

注意：当一个类有多个父类的时候，默认使用第一个父类的同名属性和方法。

24.4 子类重写父类同名方法和属性

> 故事：daqiu掌握了师父和培训的技术后，自己潜心钻研出自己的独门配方的一套全新的煎饼果子 技术。

```
# 1、师傅类：属性和方法
class Master(object):
    def __init__(self):
        self.kongfu = '古法煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

#为了验证我们的多继承，添加school父类
class School(object):
    def __init__(self):
        self.kongfu = 'jjk煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 2、徒弟类：继承师傅类 和 学校类， 添加和父类同名的属性和方法
class Prentice(Master,School): # 想要继承谁，就把谁写在第一个位置
    def __init__(self):
        self.kongfu = '独创的煎饼果子技术'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 3、用徒弟类创建对象，调用实例属性和方法结论验证
daqiu = Prentice()

print(daqiu.kongfu)

daqiu.make_cake()

#结论：如果子类和父类拥有同名属性和方法，子类创建对象调用属性和方法的时候，调用到的是子类里面的同名属性和方法。

```

子类和父类具有同名属性和方法，默认使用子类的同名属性和方法。

24.5 拓展 `__mro__` 顺序

`__mro_` ： 查看某个类的继承关系

```
# 1、师傅类：属性和方法
class Master(object):
    def __init__(self):
        self.kongfu = '古法煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

#为了验证我们的多继承，添加school父类
class School(object):
    def __init__(self):
        self.kongfu = 'jjk煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 2、徒弟类：继承师傅类 和 学校类， 添加和父类同名的属性和方法
class Prentice(Master,School): # 想要继承谁，就把谁写在第一个位置
    def __init__(self):
        self.kongfu = '独创的煎饼果子技术'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 3、用徒弟类创建对象，调用实例属性和方法结论验证
daqiu = Prentice()

print(daqiu.kongfu)

daqiu.make_cake()
#结论：如果一个类继承多个父类，优先继承第一个父类的同名属性和方法

# (<class '__main__.Prentice'>, <class '__main__.Master'>, <class
'__main__.School'>, <class 'object'>)
print(Prentice.__mro__)
```

24.6 子类调用父类的同名方法和属性

> 故事：很多顾客都希望也能吃到古法和JK的技术的煎饼果子。

```
# 1、师傅类：属性和方法
class Master(object):
    def __init__(self):
        self.kongfu = '古法煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(object):
    def __init__(self):
        self.kongfu = 'jjk煎饼果子配方'
```

```
    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 2、徒弟类：继承师傅类 和 学校类， 添加和父类同名的属性和方法
class Prentice(Master,School):
    def __init__(self):
        self.kongfu = '独创的煎饼果子技术'

    def make_cake(self):
        # 加自己的初始化原因：如果不加这个自己的初始化，kongfu属性值是上一次调用的init内的kongfu属性值
        # 如果是先调用了父类的属性和方法，父类属性会覆盖子类属性，故在调用属性前，先调用自己子类的初始化
        self.__init__()
        print(f'运用{self.kongfu}制作煎饼果子')

    # 子类调用父类的同名属性和方法：把父类的同名属性和方法再次封装
    # 调用父类方法，但是为了保证调用到的也是父类的属性，必须在调用方法前调用父类的初始化
    def make_master_cake(self):
        # 父类类名.函数()
        # 再次调用初始化的原因：这里想要调用父类的同名方法和属性，属性在init初始化位置，所以需要再次调用init
        Master.__init__(self)
        Master.make_cake(self)

    def make_school_cake(self):
        School.__init__(self)
        School.make_cake(self)


# 3、用徒弟类创建对象，调用实例属性和方法结论验证
daqiu = Prentice()
daqiu.make_cake()

daqiu.make_master_cake()
daqiu.make_school_cake()
daqiu.make_cake()

"""
运用独创的煎饼果子技术制作煎饼果子
运用古法煎饼果子配方制作煎饼果子
运用jjk煎饼果子配方制作煎饼果子
运用独创的煎饼果子技术制作煎饼果子

"""
```

24.7 多层继承

> 故事：N年后， daqiu老了，想要把所有技术传承给自己的徒弟。

```
# 1、师傅类：属性和方法
class Master(object):
```

```
    def __init__(self):
        self.kongfu = '古法煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(object):
    def __init__(self):
        self.kongfu = 'jjk煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')


# 2、徒弟类：继承师傅类 和 学校类， 添加和父类同名的属性和方法
class Prentice(Master,School): # 想要继承谁，就把谁写在第一个位置
    # 加自己的初始化原因：如果不加这个自己的初始化，kongfu属性值是上一次调用的init内的kongfu属性值
    def __init__(self):
        self.kongfu = '独创的煎饼果子技术'

    def make_cake(self):
        self.__init__()
        print(f'运用{self.kongfu}制作煎饼果子')

    # 子类调用父类的同名属性和方法：把父类的同名属性和方法再次f封装
    def make_master_cake(self):
        # 父类类名.函数()
        # 再次调用初始化的原因：这里想要调用父类的同名方法和属性，属性在init初始化位置，所以需要再次调用init
        Master.__init__(self)
        Master.make_cake(self)

    def make_school_cake(self):
        School.__init__(self)
        School.make_cake(self)


# 徒孙类
# 步骤：1、创建类Tusun，用这个类创建对象；2、用这个对象调用父类的属性或方法看能否成功。
class Tusun(Prentice):
    pass

xiaoqiu = Tusun()

xiaoqiu.make_cake()

xiaoqiu.make_master_cake()

xiaoqiu.make_school_cake()
```

24.8 super()调用父类方法

```
# 1、师傅类：属性和方法
```

```python
class Master(object):
    def __init__(self):
        self.kongfu = '古法煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(Master):
    def __init__(self):
        self.kongfu = 'jjk煎饼果子配方'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

        # 2.1 super()带参数写法
        # super(School,self).__init__()
        # super(School,self).make_cake()

        # 2.2 无参数super
        super().__init__()
        super().make_cake() # master类


# 2、徒弟类：继承师傅类 和 学校类， 添加和父类同名的属性和方法
class Prentice(School): # 想要继承谁，就把谁写在第一个位置
    # 加自己的初始化原因：如果不加这个自己的初始化，kongfu属性值是上一次调用的init内的kongfu属性值
    def __init__(self):
        self.kongfu = '独创的煎饼果子技术'

    def make_cake(self):
        self.__init__()
        print(f'运用{self.kongfu}制作煎饼果子')

    # 子类调用父类的同名属性和方法：把父类的同名属性和方法再次f封装
    def make_master_cake(self):
        # 父类类名.函数()
        # 再次调用初始化的原因：这里想要调用父类的同名方法和属性，属性在init初始化位置，所以需要再次调用init
        Master.__init__(self)
        Master.make_cake(self)

    def make_school_cake(self):
        School.__init__(self)
        School.make_cake(self)

    # 需求：一次性调用父类School Master的方法
    def make_old_cake(self):
        # 方法一：如果定义的类名修改，这里也要修改，麻烦；代码量庞大，冗余
        # School.__init__(self)
        # School.make_cake(self)
        # Master.__init__(self)
        # Master.make_cake(self)

        # 方法二：super()
        # 方法2.1 super(当前类名，self).函数()
        # super(Prentice,self).__init__()
        # super(Prentice,self).make_cake() # 调用到了School类的方法
```

```
58
59          # 方法2.2 无参数super
60          super().__init__()
61          super().make_cake() # School类
62
63  # 3、用徒弟类创建对象，调用实例属性和方法结论验证
64  daqiu = Prentice()
65  daqiu.make_old_cake()
66
```

注意：使用 super()可以自动查找父类。调用顺序遵循 `__mro__` 类属性的顺序。比较适合单继承使用。

24.9 私有权限

24.9.1 定义私有属性和方法

在 Python中，可以为实例属性和方法设置私有权限，即设置某个实例属性或实例方法不继承给子类。

> 故事：daqiu把技术传承给徒弟的同时，不想把自己的钱（200000个亿）继承给徒弟，这个时候就要为钱这个实例属性设置私有权限。

设置私有权限的方法：在属性名和方法名前面加上两个下划线__。

```
1
```