Python面向对象语法精讲

嵩 天

Python类的继承

嵩 天

Python类的继承

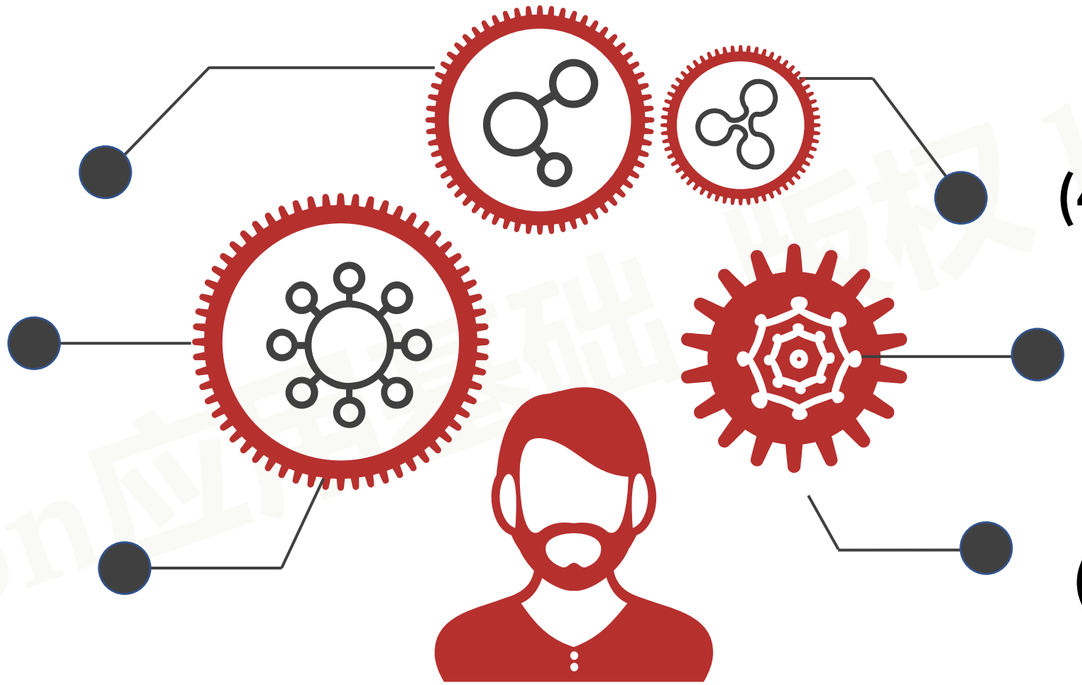
单元开篇

单元开篇

(3) Python最基础类

(2) 类继承的构建

(1) 继承的理解



(4) 类的属性重载

(5) 类的方法重载

(6) 类的多继承

Python类的继承

Python类的继承

继承的理解

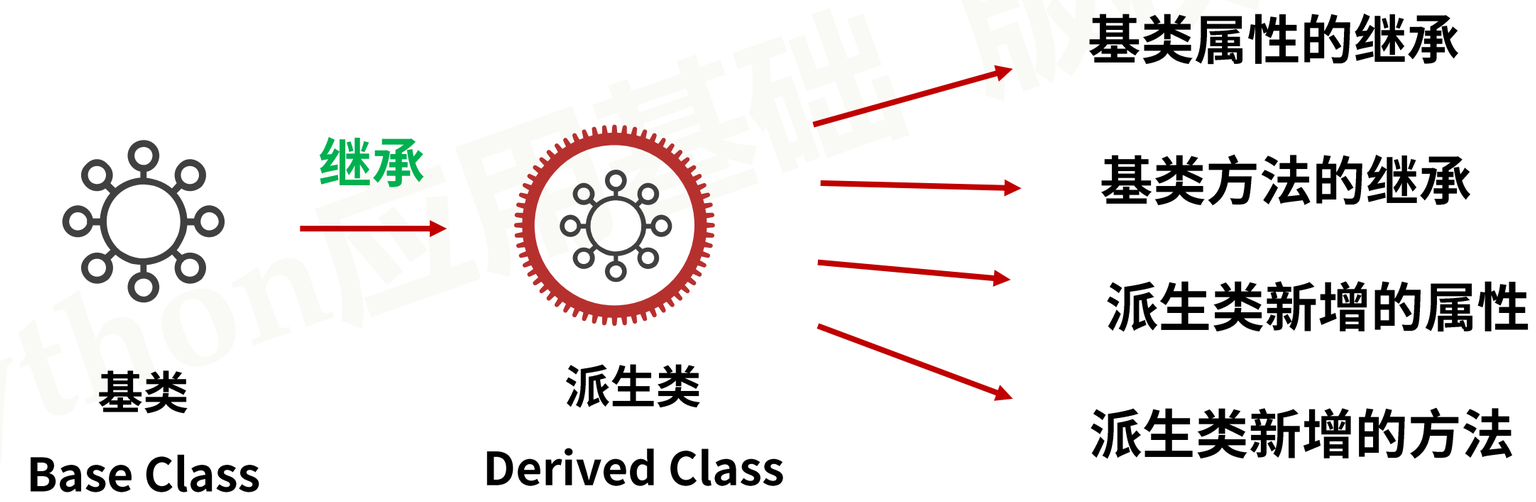
继承的理解

继承 Inheritance：代码复用的高级抽象

- 继承是面向对象程序设计的精髓之一
- 实现了以类为单位的的高抽象级别代码复用
- 继承是新定义类能够几乎完全使用原有类属性与方法的过程

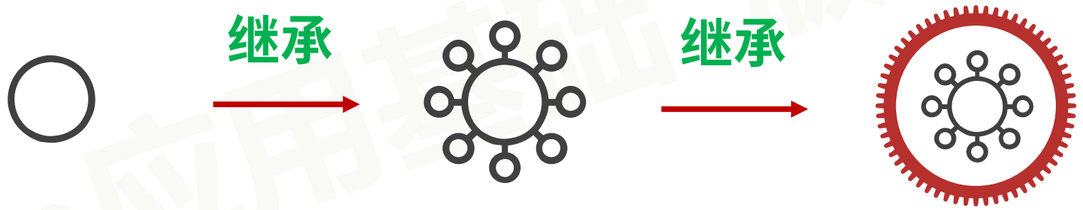
继承的理解

继承 Inheritance：代码复用的高级抽象



继承的理解

继承 Inheritance：代码复用的高级抽象



基类

Base Class

超类

基类 & 派生类

父类

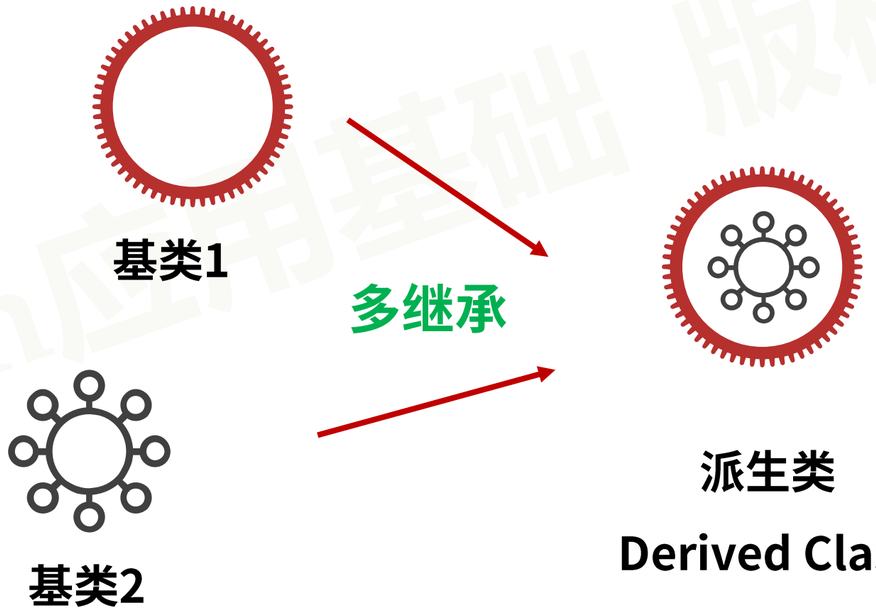
派生类

Derived Class

子类

继承的理解

继承 Inheritance：代码复用的高级抽象



Python类的继承

类继承的构建

类继承的构建

在定义类时声明继承关系

```
class <类名>(<基类名>):
    def __init__(self, <参数列表>)
        <语句块>
    ……
```

基类名可以带有路径：ModuleName.BaseClassName

类继承的使用

派生类可以直接使用基类的属性和方法

- 基类的属性 基本等同于 定义在派生类中
- 派生类可以直接使用基类的类属性、实例属性
- 派生类可以直接使用基类的各种方法
- 使用基类的类方法和类属性时，要用基类的类名调用

类继承的使用

```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1
    def getName(self):
        return self.name

class HumanNameClass(DemoClass):
    def printName(self):
        return str(DemoClass.count) + self.name + "同志"

dc1 = HumanNameClass("老王")
print(dc1.getName())
print(dc1.printName())
```

基类

派生类

派生类的实例对象

类继承的使用

```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1
    def getName(self):
        return self.name

class HumanNameClass(DemoClass):
    def printName(self):
        return str(DemoClass.count) + self.name + "同志"

dc1 = HumanNameClass("老王")
print(dc1.getName())
print(dc1.printName())
```

对基类类属性的使用

对基类实例方法的使用

对派生类实例方法的使用

继承关系的判断

2个与继承关系判断有关的Python内置函数

函数	描述
isinstance(obj, cls)	判断对象obj是否是类cls的实例或子类实例，返回True/False
issubclass(cls1, cls2)	判断类cls1是否是类cls2的子类，返回True/False

类继承的使用

```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1
    def getName(self):
        return self.name

class HumanNameClass(DemoClass):
    def printName(self):
        return str(DemoClass.count) + self.name + "同志"

dc1 = HumanNameClass("老王")
print(isinstance(dc1, DemoClass))
print(isinstance(dc1, HumanNameClass))
print(issubclass(HumanNameClass, DemoClass))
```

```
True
True
True
```

类继承的使用

派生类的约束

- 派生类只能继承基类的公开属性和方法
- 派生类不能继承基类的私有属性和私有方法

Python类的继承

Python最基础类

Python最基础类

object类是Python所有类的基类

- object是Python最基础类的名字，不建议翻译理解
- 所有类定义时默认继承object类
- 保留属性和保留方法本质上是object类的属性和方法

Python最基础类

```
print(object.__name__)
print(object.__doc__)
print(object.__bases__)
print(object.__class__)
print(object.__module__)
print(object.__dict__)
```

```
object
The most base type
()
<class 'type'>
builtins
{'__repr__': <slot wrapper(略)
```

Python最基础类

Python对象的三个要素：标识、类型和值

- 标识 identity：对象一旦构建不会改变，用id()获得，一般是内存地址
- 类型 type：对象的类型，用type()获得
- 值 value：分为可变mutable与不可变immutable两种

Python最基础类

2个与基础类有关的Python内置功能

函数/保留字	描述
id(x)	返回x的标识，Cpython用内存地址表示
x is y	判断x和y的标识是否相等，返回True/False，不判断值

Python最基础类

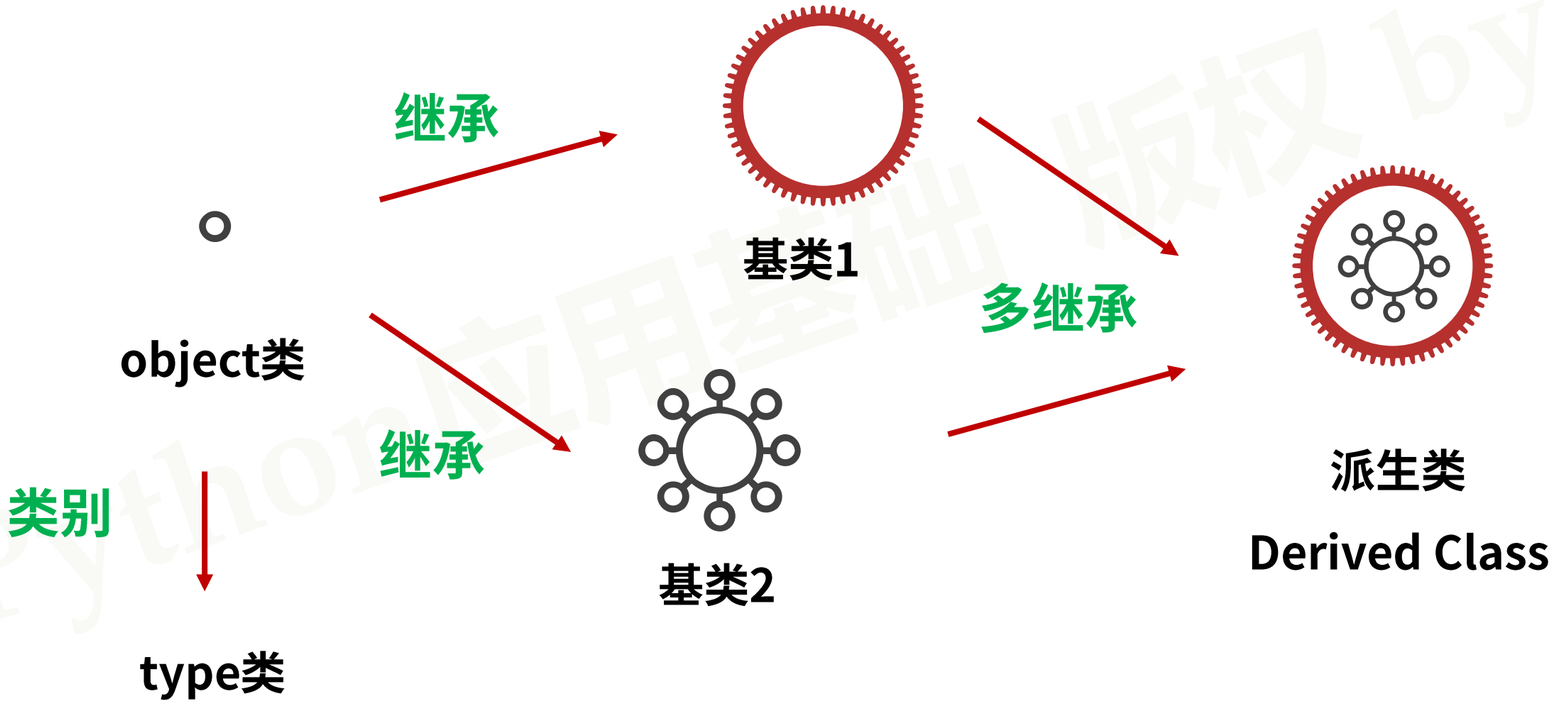
```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1
    def getName(self):
        return self.name

class HumanNameClass(DemoClass):
    def printName(self):
        return str(DemoClass.count) + self.name + "同志"

dc1 = HumanNameClass("老王")
print(id(dc1), type(dc1))
print(id(DemoClass), type(DemoClass))
print(dc1 is DemoClass)
print(type(object), type(type))
```

```
86222320 <class '__main__.HumanNameClass'>
86207768 <class 'type'>
False
<class 'type'> <class 'type'>
```

Python最基础类



Python类的继承

类的属性重载

重载

重载：派生类对基类属性或方法的再定义

- 属性重载：派生类定义并使用了与基类相同名称的属性
- 方法重载：派生类定义并使用了与基类相同名称的方法

类的属性重载

最近覆盖原则：重载无需特殊标记

- 步骤1：优先使用派生类重定义的属性和方法
- 步骤2：然后寻找基类的属性和方法
- 步骤3：再寻找超类的属性和方法

类的属性重载

```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1

class HumanNameClass(DemoClass):
    count = 99
    def __init__(self, name):
        self.name = name
        HumanNameClass.count -= 1
    def printCount(self):
        return str(HumanNameClass.count) + self.name

dc1 = HumanNameClass("老王")
print(dc1.printCount())
```

类属性 被重载

类属性 被重载

类属性用类名调用，不容易被误解

类的属性重载

```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1

class HumanNameClass(DemoClass):
    count = 99
    def __init__(self, name):
        self.name = name
        HumanNameClass.count -= 1
    def printCount(self):
        return str(HumanNameClass.count) + self.name

dc1 = HumanNameClass("老王")
print(dc1.printCount())
```

实例属性 被重载

实例属性 被重载

实例属性用对象名调用，容易被误解

类的属性重载

```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1

class HumanNameClass(DemoClass):
    count = 99
    def __init__(self, name):
        self.name = name
        HumanNameClass.count -= 1
    def printCount(self):
        return str(HumanNameClass.count) + self.name

dc1 = HumanNameClass("老王")
print(dc1.printCount())
```

构造方法 被重载

构造方法 被重载

类的属性重载

```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1

class HumanNameClass(DemoClass):
    count = 99
    def __init__(self, name):
        self.name = name
        HumanNameClass.count -= 1
    def printCount(self):
        return str(HumanNameClass.count) + self.name

dc1 = HumanNameClass("老王")
print(dc1.printCount())
```

```
98老王
```

Python类的继承

类的方法重载

类的方法重载

方法重载：派生类对基类方法的再定义

- **完全重载：派生类完全重定义与基类相同名称的方法**

  直接在派生类中定义同名方法即可

- **增量重载：派生类扩展定义与基类相同名称的方法**

类的方法重载

增量重载：使用super()方法

```
class <派生类名>(<基类名>):
    def <方法名>(self, <参数列表>)
        super().<基类方法名>(<参数列表>)
        ……
```

类的方法重载

```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1
    def printCount(self):
        return str(DemoClass.count) + self.name

class HumanNameClass(DemoClass):
    def __init__(self, name):
        self.name = name
    def printCount(self):
        return super().printCount() + "同志"

dc1 = HumanNameClass("老王")
print(dc1.printCount())
```

```
0老王同志
```

类的方法重载

```
class DemoClass:
    count = 0
    def __init__(self, name):
        self.name = name
        DemoClass.count  += 1
    def printCount(self):
        return str(DemoClass.count) + self.name

class HumanNameClass(DemoClass):
    def __init__(self, name):
        self.name = name
    def printCount(self):
        return super().printCount() + "同志"

dc1 = HumanNameClass("老王")
print(dc1.printCount())
```

通过super()找到了基类

super().printCount()调用了基类方法

Python类的继承

类的多继承

类的多继承

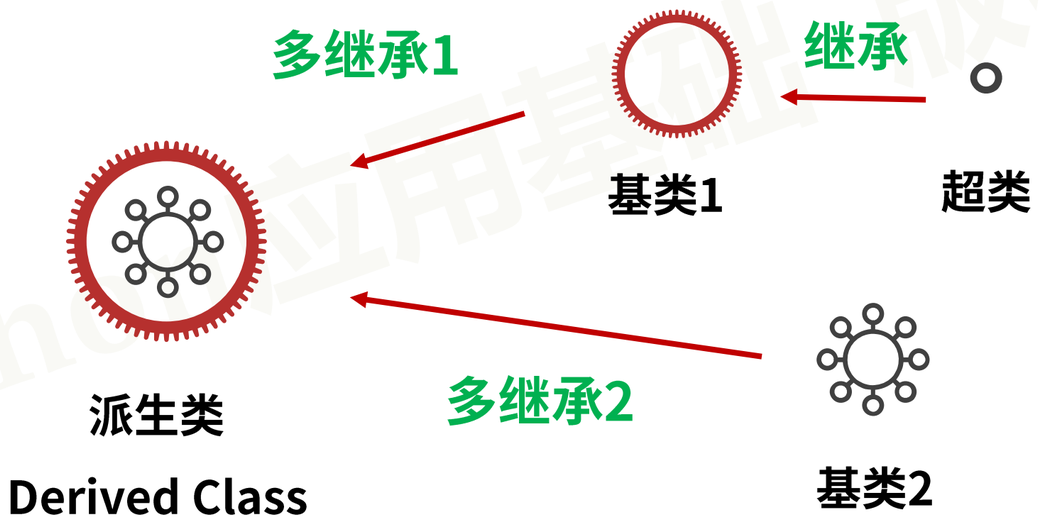
多继承的构建：在定义类时声明继承关系

```
class <类名>(<基类名1>, <基类名2>,…,<基类名N>):
    def __init__(self, <参数列表>)
        <语句块>
    ……
```

基类名可以带有路径：ModuleName.BaseClassName

类的多继承

Python采用深度优先、从左至右的方法实施多继承



类的多继承

```
class DemoClass:
    def __init__(self, name):
        self.name = name
    def printName(self):
        return self.name

class NameClass:
    def __init__(self, title):
        self.nick = title
    def printName(self):
        return self.nick + "同志"

class HumanNameClass(DemoClass, NameClass):
    pass

dc1 = HumanNameClass("老王")
print(dc1.printName())
```

老王

多继承，方法printName()按照深度优先、从左至右方式寻找

类的多继承

```
class DemoClass:
    def __init__(self, name):
        self.name = name
    def printName(self):
        return self.name

class NameClass:
    def __init__(self, title):
        self.nick = title
    def printName(self):
        return self.nick + "同志"

class HumanNameClass(NameClass, DemoClass):
    pass

dc1 = HumanNameClass("老王")
print(dc1.printName())
```

老王同志

多继承，方法printName()按照深度优先、从左至右方式寻找

类的多继承

类多继承的使用说明

- 所有属性和方法的使用按照“深度优先、从左到右”的方式选取
- 构造函数也参照上述原则，super()也参照上述原则
- 多个基类的顺序是关键

类的多继承

```
class DemoClass:
    def __init__(self, name):
        self.name = name
    def printName(self):
        return self.name

class NameClass:
    def __init__(self, title):
        self.nick = title
    def printName(self):
        return self.nick + "同志"

class HumanNameClass(DemoClass, NameClass):
    def printName(self):
        return super().printName() + "你好"

dc1 = HumanNameClass("老王")
print(dc1.printName())
```

老王你好

多继承，super()也按照深度优先、从左至右方式寻对应找基类方法

Python类的继承

单元小结

单元小结

(3) Python最基础类

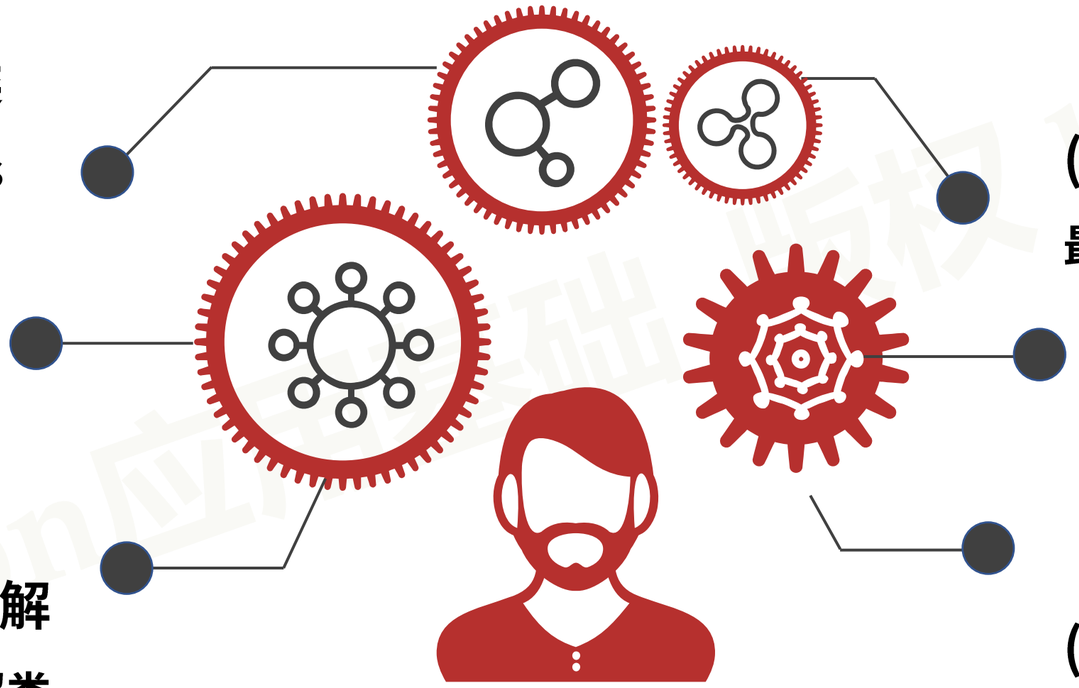
object类、id()、is

(2) 类继承的构建

派生类的构建、2个内置函数

(1) 继承的理解

基类、派生类、父类、子类、超类



(4) 类的属性重载

最近覆盖原则、实例属性重载

(5) 类的方法重载

增量重载、super()函数

(6) 类的多继承

深度优先、从左到右

Python类的继承


Python 123
Thank you