31.new方法和单例模式

1.__new__方法

`__new__`和`__init__`的作用

```
class A(object):
    def __init__(self):
        print("这是 init 方法")

    def __new__(cls):
        print("这是 new 方法")
        return object.__new__(cls)

A()
```

总结

- `__new__`至少要有一个参数cls，代表要实例化的类，此参数在实例化时由Python解释器自动提供
- `__new__`必须要有返回值，返回实例化出来的实例，这点在自己实现`__new__`时要特别注意，可以return父类`__new__`出来的实例，或者直接是object的`__new__`出来的实例
- `__init__`有一个参数self，就是这个`__new__`返回的实例，`__init__`在`__new__`的基础上可以完成一些其它初始化的动作，`__init__`不需要返回值
- 我们可以将类比作制造商，`__new__`方法就是前期的原材料购买环节，`__init__`方法就是在有原材料的基础上，加工，初始化商品环节

2.设计模式:单例模式

1. 单例是什么

举个常见的单例模式例子，我们日常使用的电脑上都有一个回收站，在整个操作系统中，回收站只能有一个实例，整个系统都使用这个唯一的实例，而且回收站自行提供自己的实例。因此回收站是单例模式的应用。

确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，单例模式是一种对象创建型模式。

2. 创建单例-保证只有1个对象

```
# 实例化一个单例
class Singleton(object):
    __instance = None

    def __new__(cls, age, name):
```

```
        #如果类属性__instance的值为None，
        #那么就创建一个对象，并且赋值为这个对象的引用，保证下次调用这个方法时
        #能够知道之前已经创建过对象了，这样就保证了只有1个对象
        if not cls.__instance:
            cls.__instance = object.__new__(cls)
        return cls.__instance

a = Singleton(18, "zuoge")
b = Singleton(8, "zuoge")

print(id(a))
print(id(b))

a.age = 19 #给a指向的对象添加一个属性
print(b.age)#获取b指向的对象的age属性
```

运行结果：

```
4391023224
4391023224
19
```

3. 创建单例时，只执行1次init方法

```
# 实例化一个单例
class Singleton(object):
    __instance = None
    __is_first = True

    def __new__(cls, age, name):
        if not cls.__instance:
            cls.__instance = object.__new__(cls)
        return cls.__instance

    def __init__(self, age, name):
        if self. __is_first:
            self.age = age
            self.name = name
            Singleton. __is_first = False


a = Singleton(18, "习大大")
b = Singleton(28, "习大大")

print(id(a))
print(id(b))
```

```
print(a.age)
print(b.age)

a.age = 19
print(b.age)
```