# 分类-Category

## 基本用途

* 如何在不改变原来类模型的前提下，给类扩充一些方法？有2种方式
* 继承
* 分类（Category）

## 格式

* 分类的声明

@interface 类名 (分类名称)

// 方法声明

@end

* 分类的实现

@implementation 类名 (分类名称)

// 方法实现

@end

## 好处

* 一个庞大的类可以分模块开发
* 一个庞大的类可以由多个人来编写，更有利于团队合作

## 给系统自带的类添加分类

* 给NSString增加一个类方法：计算某个字符串中阿拉伯数字的个数
* 给NSString增加一个对象方法：计算当前字符串中阿拉伯数字的个数

## 注意

* Category可以访问原始类的实例变量，但不能添加变量，只能添加方法。如果想添加变量，可以考虑通过继承创建子类
* Category可以实现原始类的方法，但不推荐这么做，因为它是直接替换掉原来的方法，这么做的后果是再也不能访问原来的方法
* 多个Category中如果实现了相同的方法，只有最后一个参与编译的才会有效

# 类的本质

## 类也是个对象

* 其实类也是一个对象，是Class类型的对象，简称“类对象”
* Class类型的定义

typedef struct objc\_class \*Class;

* 类名就代表着类对象，每个类只有一个类对象

## +load和+initialize

* +load
* 在程序启动的时候会加载所有的类和分类，并调用所有类和分类的+load方法
* 先加载父类，再加载子类；也就是先调用父类的+load，再调用子类的+load
* 先加载元原始类，再加载分类
* 不管程序运行过程有没有用到这个类，都会调用+load加载
* +initialize
* 在第一次使用某个类时（比如创建对象等），就会调用一次+initialize方法
* 一个类只会调用一次+initialize方法，先调用父类的，再调用子类的

## 获取类对象的2种方式

Class c = [Person class]; // 类方法

或者

Person \*p = [Person new];

Class c2 = [p class]; // 对象方法

## 类对象调用类方法

Class c = [Person class];

Person \*p2 = [c new];

# description方法

## -description方法

使用NSLog和%@输出某个对象时，会调用对象的-description方法，并拿到返回值进行输出

## + description方法

使用NSLog和%@输出某个类对象时，会调用类对象+description方法，并拿到返回值进行输出

## 修改NSLog的默认输出

* 重写-description或者+description方法即可

## 死循环陷阱

* 如果在-description方法中使用NSLog打印self

# SEL

## 方法的存储位置

* 每个类的方法列表都存储在类对象中
* 每个方法都有一个与之对应的SEL类型的对象
* 根据一个SEL对象就可以找到方法的地址，进而调用方法
* SEL类型的定义

typedef struct objc\_selector \*SEL;

## SEL对象的创建

SEL s = @selector(test);

SEL s2 = NSSelectorFromString(@"test");

## SEL对象的其他用法

// 将SEL对象转为NSString对象

NSString \*str = NSStringFromSelector(@selector(test));

Person \*p = [Person new];

// 调用对象p的test方法

[p performSelector:@selector(test)];

# NSLog输出增强

* \_\_FILE\_\_ ：源代码文件名
* \_\_LINE\_\_ ：NSLog代码在第几行
* \_cmd ：代表着当前方法的SEL

// 下面的代码会引发死循环

- (void)test {

[self performSelector:\_cmd];

}