**01 category分类**

**Person.h**

#import <Foundation/Foundation.h>

@interface Person : NSObject

{

int \_age;

}

@property int age;

-(void)test;

@end

**Person.m**

#import "Person.h"

@implementation Person

-(void)test

{

NSLog(@"Person-test");

}

@end

**Person+xh.h**

#import "Person.h"

@interface Person (xh)

-(void)study;

@end

**Person+xh.m**

#import "Person+xh.h"

@implementation Person (xh)

-(void)test

{

NSLog(@"xh-test");

}

-(void)study

{

NSLog(@"%d岁的学生在学习",\_age);

}

@end

**Person+hx.h**

#import "Person.h"

@interface Person (hx)

@end

**Person+hx.m**

#import "Person+hx.h"

@implementation Person (hx)

-(void)test

{

NSLog(@"hx-test");

}

@end

**main.m**

/\*

分类的作用：在不改变原来类内容的基础上，可以为类增加一些方法

使用注意：

1、分类只能增加方法，不能增加成员变量

2、分类方法实现中可以访问原来类中声明的成员变量

3、分类可以重现实现原来类中的方法，但是会覆盖掉原来的方法，会导致原来的方法没法再使用

4、方法调用的优先级：分类(最后参与编译的分类优先)-->原来类-->父类

\*/

#import <Foundation/Foundation.h>

#import "Person.h"

#import "Person+xh.h"

int main(int argc, const char \* argv[]) {

Person \*p = [[Person alloc] init];

//优先去分类中查找，然后再去原来类中找，最后再去父类中找

[p test];

[p study];

return 0;

}

**02 分类的应用**

**NSString + Number.h**

/\*

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

@“abc3524673fds”

\*/

#import <Foundation/Foundation.h>

@interface NSString (Number)

+(int)numberCountOfString:(NSString \*)str;

-(int)numberCount;

@end

**NSString + Number.m**

#import "NSString+Number.h"

@implementation NSString (Number)

//@“abc3524673fds”

+(int)numberCountOfString:(NSString \*)str

{

// //1、定义变量计算数字的个数

// int count = 0;

// for (int i = 0; i<str.length; i++) {

// //取出i这个位置对应的字符

// unichar c = [str characterAtIndex:i];

// //如果这个字符是阿拉伯数字

// if (c>='0' && c<='9') {

// count++;

// }

// }

// return count;

return [str numberCount];

}

-(int)numberCount

{

int count = 0;

for (int i = 0; i<self.length; i++) {

unichar c = [self characterAtIndex:i];

if (c>='0' && c<='9') {

count++;

}

}

return count;

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "NSString+Number.h"

int main(int argc, const char \* argv[]) {

// int number = [NSString numberCountOfString:@"ksu243adiof2413kfdjsa"];

//

// NSString \*str = [NSString stringWithFormat:@"idsuaoi53jl54"];

// int number = [str numberCount];

int number = [@"jsdoaiu4325hl" numberCount];

NSLog(@"字符串中阿拉伯数字的个数是%d",number);

return 0;

}

**03 description**

**Person.h**

#import <Foundation/Foundation.h>

@interface Person : NSObject

@property int age;

@property NSString \*name;

@end

**Person.m**

#import "Person.h"

@implementation Person

-(NSString \*)description

{

return [NSString stringWithFormat:@"age=%d,name=%@",\_age,\_name];

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Person.h"

int main(int argc, const char \* argv[]) {

Person \*p = [[Person alloc] init];

p.age = 20;

p.name = @"Jack";

NSLog(@"name is %@",p.name);

//默认情况下，利用NSLog和%@输出对象时，结果是：<类名：内存地址>

//1、会调用对象的-description方法

//2、拿到-description方法的返回值（NSString \*）显示到屏幕上

//3、-description方法默认返回的是“类名+内存地址”

NSLog(@"%@",p);

//打印指针变量的地址

NSLog(@"%p",&p);

//对象地址

NSLog(@"%p",p);

//<类名：对象地址>

NSLog(@"%@",p);

return 0;

}

**04 SEL**

**Person.h**

#import <Foundation/Foundation.h>

@interface Person : NSObject

+(void)test;

-(void)test1;

-(void)test2:(NSString \*)str;

@end

**Person.m**

#import "Person.h"

@implementation Person

+(void)test

{

NSLog(@"+test");

}

-(void)test1

{

NSLog(@"-test1");

}

-(void)test2:(NSString \*)str

{

NSLog(@"test2--%@",str);

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Person.h"

int main(int argc, const char \* argv[]) {

Person \*p = [[Person alloc] init];

[p test1];

//间接调用test1方法

[p performSelector:@selector(test1)];

NSString \*name = @"test1";

SEL s1 = NSSelectorFromString(name);

[p performSelector:s1];

[p test2:@"123"];

[p performSelector:@selector(test2:) withObject:@"456"];

SEL s = @selector(test2:);

[p performSelector:s withObject:@"789"];

//1、把test1包装成SEL类型的数据

//2、根据SEL数据找到对应的方法地址

//3、根据方法地址调用对应的方法

return 0;

}

**05 ARC基本使用**

**Person.h**

#import <Foundation/Foundation.h>

#import "Dog.h"

@interface Person : NSObject

@property(nonatomic,strong) Dog \*dog;

//@property(nonatomic,weak) Dog \*dog;

@end

**Person.m**

#import "Person.h"

@implementation Person

-(void)dealloc

{

NSLog(@"person is dealloc");

}

@end

**Dog.h**

#import <Foundation/Foundation.h>

@interface Dog : NSObject

@end

**Dog.m**

@implementation Dog

-(void)dealloc

{

NSLog(@"dog is dealloc");

}

@end

main.m

/\*

ARC的判断准则：只要没有强指针指向对象，就会释放对象

1.ARC特点

1>不允许调用release、retain、retaincount

2>允许重写dealloc，但是不允许调用[super dealloc]

3>@property的参数

strong：成员变量是强指针（适用于OC对象类型）

weak：成员变量是弱指针（使用于OC对象类型）

assigh：适用于非OC对象类型

指针分2种:

1>强指针：默认情况下，所有的指针都是强指针 \_\_strong

2>弱指针：\_\_weak

\*/

#import <Foundation/Foundation.h>

#import "Person.h"

#import "Dog.h"

int main(int argc, const char \* argv[]) {

Dog \*d = [[Dog alloc] init];

Person \*p = [[Person alloc] init];

p.dog = d;

//错误的写法（没有意义）

//\_\_weak Person \*p = [[Person alloc] init];

//Person \*p2 = p;

//\_\_weak Person \*p2 = p;

p = nil;

// p = [[Person alloc] init];

return 0;

}

**06 类的循环引用**

**Person.h**

#import <Foundation/Foundation.h>

//#import "Dog.h"

@class Dog;

@interface Person : NSObject

@property(nonatomic,weak)Dog \*dog;

@end

**Person.m**

#import "Person.h"

@implementation Person

-(void)dealloc

{

NSLog(@"person is dealloc");

}

@end

**Dog.h**

#import <Foundation/Foundation.h>

//#import "Person.h"

@class Person;

@interface Dog : NSObject

@property(nonatomic,strong) Person \*person;

@end

**Dog.m**

#import "Dog.h"

@implementation Dog

-(void)dealloc

{

NSLog(@"dog is dealloc");

}

@end

**main.m**

/\*

当两端循环引用时，解决方案：

一端用strong，另一端用weak

\*/

#import <Foundation/Foundation.h>

#import "Person.h"

#import "Dog.h"

int main(int argc, const char \* argv[]) {

Person \*p = [[Person alloc] init];

Dog \*d = [[Dog alloc] init];

p.dog = d;

d.person = p;

return 0;

}