**1 Super**

**Zombie.h**

#import <Foundation/Foundation.h>

@interface zombie : NSObject

-(void)walk;

+(void)test;

-(void)test;

@end

**Zombie.m**

#import "zombie.h"

@implementation zombie

-(void)walk

{

NSLog(@"往前挪两步");

}

-(void)test

{

NSLog(@"Zoombie-test");

}

+(void)test

{

NSLog(@"Zoombie+test");

}

@end

**JumpZombie.h**

#import "zombie.h"

@interface JumpZombie : zombie

+(void)test1;

@end

**JumpZombie.m**

#import "JumpZombie.h"

@implementation JumpZombie

-(void)walk

{

//跳两下

NSLog(@"跳两下");

//往前挪两下

//NSLog(@"往前挪两步");

[super walk];

}

+(void)test1

{

[super test];

}

@end

**Main.m**

#import <Foundation/Foundation.h>

#import "JumpZombie.h"

/\*

super的作用

1、直接调用父类中的某个方法

2、super处在对象方法中，就会调用父类的对象方法

super处在类方法中，就会调用父类的类方法

使用场合：子类重写父类的方法时想保留父类的一些行为

\*/

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

JumpZombie \*jz = [JumpZombie new];

[jz walk];

[JumpZombie test1];

}

**2 多态**

**Animal.h**

#import <Foundation/Foundation.h>

@interface Animals : NSObject

-(void)eat;

@end

**Animal.m**

#import "Animals.h"

@implementation Animals

-(void)eat

{

NSLog(@"吃东西");

}

@end

**Dog.h**

#import "Animals.h"

@interface Dog : Animals

-(void)run;

@end

**Dog.m**

#import "Dog.h"

@implementation Dog

-(void)eat

{

NSLog(@"狗吃东西");

}

-(void)run

{

NSLog(@"狗跑起来了");

}

@end

**Cat.h**

#import "Animals.h"

@interface Cat : Animals

@end

**Cat.m**

#import "Cat.h"

@implementation Cat

-(void)eat

{

NSLog(@"猫吃东西");

}

@end

**Main.m**

/\*

多态：多种形态

（1）没有继承就没有多态

（2）代码的体现：父类类型的指针指向子类对象

（3）好处：如果函数\方法中参数中使用的是父类类型，可以传入父类、子类对象

（4）局限性：父类类型的变量 不能 直接调用子类特有的方法。

\*/

#import <Foundation/Foundation.h>

#import "Dog.h"

#import "Cat.h"

//这个函数是专门用来喂动物的

//void feed(Dog \*d)

//{

// [d eat];

//}

//如果参数中使用的是父类类型，可以传入父类、子类对象

void feed(Animals \*a)

{

[a eat];

}

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

NSObject \*n = [Dog new];

Dog \*d = [Dog new]; //Dog类型

feed(d);

Cat \*c = [Cat new];

feed(c);

//父类指针指向子类对象，狗是一个动物

Animals \*a = [Dog new];

//父类类型的变量 不能 直接调用子类特有的方法。

//[a run];

//调用方法时会检测对象的真实类型

//[a eat];

return 0;

}

**03 字符串**

#import <Foundation/Foundation.h>

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

NSString \*str = @"cumt";

NSLog(@"string is %@",str);

int age = 10;

int no = 5;

NSString \*name = @"你好Jack";

//创建OC字符串的另一种方式

NSString \*newStr = [NSString stringWithFormat:@"My age is %d ,no is %d and name is %@",age,no,name];

NSLog(@"%@",newStr);

int size = [name length];

NSLog(@"size is %d",size);

return 0;

}

**04 id**

**Person.h**

#import <Foundation/Foundation.h>

@interface Person : NSObject

@property int age;

@property id obj;

@end

**Person.m**

#import "Person.h"

@implementation Person

@end

**main.m**

void test(id d)

{

}

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

Person \*p = [Person new];

NSObject \*o = [Person new];

//万能指针，能指向\操作任何对象

// id相当于NSObject \*

id d = [Person new];

[d setAge:10];

//可以传递任何类型的参数

[d setObj:@"jujkljklio"];

NSLog(@"age is %d",[d age]);

return 0;

}

**05 代码段保存等**

**Person.h**

#import <Foundation/Foundation.h>

@interface Person : NSObject

{

int \_age;

NSString \*\_name;

}

-(void)setAge:(int)age;

-(int)age;

-(void)setName:(NSString \*)name;

-(NSString \*)name;

@end

**Person.m**

#import "Person.h"

@implementation Person

-(void)setAge:(int)age

{

NSLog(@"setAge");

\_age = age;

}

-(int)age

{

NSLog(@"age");

return \_age;

}

-(void)setName:(NSString \*)name

{

\_name = name;

}

#pragma mark - 姓名的set方法

-(NSString \*)name

{

//return \_name;

return @"Rose";

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Person.h"

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

Person \*p = [Person new];

//点语法的本质还是方法调用

p.age = 10;

// [p setAge:10];

p.name = @"Jack";

NSString \*s = p.name;

NSLog(@"name is %@",s);

int a = p.age;

// int a = [p age];

return 0;

}

**06 构造方法**

**Person.h**

#import <Foundation/Foundation.h>

@interface Person : NSObject

@property int age;

@end

**Person.m**

#import "Person.h"

@implementation Person

-(id)init

{

// //一定要调用回super的init方法：初始化父类中声明的一些成员变量和其他属性

// self = [super init]; //返回当前对象

//

// //如果对象初始化成功，才有必要进行接下来的初始化

// if (self != nil) {

// //初始化成功

// \_age = 18;

// }

if (self = [super init]) {

\_age = 18;

}

//返回一个已经初始化完毕的对象

return self;

}

@end

**Student.h**

#import "Person.h"

@interface Student : Person

@property int no;

@end

**Student.m**

#import "Student.h"

@implementation Student

//学生对象初始化完毕后，年龄是18，学号就是1

-(instancetype)init

{

if (self = [super init]) {

\_no = 1;

}

return self;

}

@end

**main.m**

#import <Foundation/Foundation.h>

#include "Person.h"

#import "Student.h"

//构造方法：用来初始化对象的方法，是的对象方法，-开头

//重写构造方法的目的：为了让对象创建出来，成员变量就会有一些固定的值

/\*

重写构造方法的注意点：

1、先调用父类的构造方法[super init]

2、再进行子类内部成员变量的初始化

\*/

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

//[Person new];

/\*

完整地创建一个可用的对象

1、分配存储空间 +alloc

2、初始化 -init

\*/

//1、调用alloc分配存储空间

//Person \*p1 = [Person alloc];

//2、调用init方法进行初始化

//Person \*p2 = [p1 init];

//要求每个Person对象创建出来，\_age都是18

//Person \*p3 = [Person new];

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

NSLog(@"age is %d",p.age);

Student \*s = [[Student alloc] init];

return 0;

}

**07 自定义构造方法**

**Person.h**

#import <Foundation/Foundation.h>

@interface Person : NSObject

@property NSString \*name;

@property int age;

/\*

自定义构造方法的规范

1、一定是对象方法，以-开头

2、返回值一定是id类型

3、方法名以initWith开头

\*/

-(id)initWithName:(NSString \*)name;

-(id)initWithName:(NSString \*)name andAge:(int)age;

@end

**Person.m**

#import "Person.h"

@implementation Person

-(id)init

{

if (self = [super init]) {

\_name = @"Jack";

}

return self;

}

-(id)initWithName:(NSString \*)name

{

if (self = [super init]) {

\_name = name;

}

return self;

}

-(id)initWithName:(NSString \*)name andAge:(int)age

{

if (self = [super init]) {

\_name = name;

\_age = age;

}

return self;

}

@end

**Student.h**

#import "Person.h"

@interface Student : Person

@property int no;

-(id)initWithNo:(int)no;

-(id)initWithName:(NSString \*)name andAge:(int)age andNo:(int)no;

@end

**Student.m**

#import "Student.h"

@implementation Student

-(id)initWithNo:(int)no

{

if (self = [super init]) {

\_no = no;

}

return self;

}

//父类的属性交给父类方法去处理，子类方法处理子类自己的属性

-(id)initWithName:(NSString \*)name andAge:(int)age andNo:(int)no

{

//将name 和 age 传到父类方法中进行初始化

if (self = [super initWithName:name andAge:age]) {

\_no = no;

//[self setName:name];

//[self setAge:age];

//self.name = name;

//self.age = age;

}

return self;

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Person.h"

#import "Student.h"

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

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

Person \*p1 = [[Person alloc] initWithName:@"Rose"];

Person \*p2 = [[Person alloc] initWithName:@"Mary" andAge:18];

Student \*s = [[Student alloc] initWithNo:9];

Student \*s1 = [[Student alloc] initWithName:@"Jim" andAge:22 andNo:1234];

return 0;

}