**01 封装**

**student.h**

#import <Foundation/Foundation.h>

@interface Student : NSObject

{

//成员变量尽量不要用@public

//@public

/\*成员变量名的命名规范：一定要以\_开头

作用：

1、让成员变量和get方法的名称区分开来

2、可以跟局部变量区分开，只要看到\_开头的变量，一般都是成员变量

\*/

int \_age;

//只读（readonly）：只允许外界访问，不允许外界修改

//只需要提供get方法

int no;

}

/\*set方法

1、作用：提供一个方法给外界设置age的属性值,可以在方法里面对参数进行相应过滤

2、命名规范：

1>方法必须以set开头

2>set后面跟上成员变量的名称，成员变量的首字母必须大写

3>返回值一定是void

4>一定要接收一个参数，而且参数类型与成员变量一致

5>形参的名称不能跟成员变量名一样

\*/

-(void)setAge:(int)age;

/\*

get方法

1、作用：返回对象内部的成员变量

2、命名规范：

1>肯定有返回值，返回值类型肯定与成员变量类型一致

2>方法名与成员变量名一致

3>不需要接收任何参数

\*/

-(int)age;

-(void)study;

@end

**student.m**

#import "Student.h"

@implementation Student

-(void)setAge:(int)age

{

if(age<=0)

{

age = 1;

}

\_age = age;

}

-(int)age

{

return \_age;

}

-(void)study

{

NSLog(@"%d岁的学生在学习",\_age);

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Student.h"

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

Student \*stu = [Student new];

[stu setAge:-15];

[stu study];

NSLog(@"学生的年龄是%d岁",[stu age]);

return 0;

}

**02 类方法**

**person.h**

#import <Foundation/Foundation.h>

/\*

对象方法

1>减号 - 开头

2>只能由对象来调用

3>对象方法中能访问当前对象的成员变量（实例变量）

类方法

1>加号 + 开头

2>只能由类（名）来调用

3>类方法中不能访问成员变量

类方法的好处和使用场合

1>不依赖于对象，执行效率高

2>能用类方法尽量用类方法

3>场合：当方法内部不需要使用成员变量时，就可以改为类方法

可以允许类方法和对象方法同名

\*/

@interface Person : NSObject

{

int age;

}

//类方法都是以+开头

+(void)printClassName;

-(void)test;

+(void)test;

@end

**person.m**

#import "Person.h"

@implementation Person

+(void)printClassName

{

NSLog(@"这个类叫做Person");

}

-(void)test

{

NSLog(@"调用了对象方法test");

}

+(void)test

{

NSLog(@"调用了类方法test");

//实例变量age不能在类方法中访问

//NSLog(@"这个类叫做Person-%d",age);

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Person.h"

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

Person \*p = [Person new];

[p test];

[Person printClassName];

[Person test];

return 0;

}

**03 计算器类**

**calculator.h**

#import <Foundation/Foundation.h>

/\*

设计一个计算器类

1、求和

2、求平均值

\*/

//工具类：基本没有任何成员变量，里面的方法基本都是类方法

@interface Calculator : NSObject

-(int)sumofNum1:(int)num1 andNum2:(int)num2;

-(int)averageofNum1:(int)num1 andNum2:(int)num2;

+(int)sumofNum1:(int)num1 andNum2:(int)num2;

+(int)averageofNum1:(int)num1 andNum2:(int)num2;

@end

**calculator.m**

#import "Calculator.h"

@implementation Calculator

-(int)sumofNum1:(int)num1 andNum2:(int)num2

{

return num1 + num2;

}

-(int)averageofNum1:(int)num1 andNum2:(int)num2

{

return (num1 + num2)/2;

}

+(int)sumofNum1:(int)num1 andNum2:(int)num2

{

return num1 + num2;

}

+(int)averageofNum1:(int)num1 andNum2:(int)num2

{

//return (num1 + num2)/2;

int sum = [Calculator sumofNum1:num1 andNum2:num2];

return sum/2;

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Calculator.h"

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

// Calculator \*cal = [Calculator new];

// [cal sumofNum1:10 andNum2:5];

int sum = [Calculator sumofNum1:10 andNum2:5];

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

return 0;

}

**04 property**

**Dog.h**

#import <Foundation/Foundation.h>

@interface Dog : NSObject

/\*property

1、用在@interface中

2、用来自动生成setter和getter的声明

3、@property int age;

（1）如果成员变量\_age不存在，就会自动成成一个@private的成员变量\_age

（2）在类的声明文件中替代 -(void)setAge:(int)age; -(int)age; 语句

（3）在类的实现文件中替代

-(void)setAge:(int)age

{

\_age = age;

}

-(int)age

{

return \_age;

}

即@property可以同时生成setter和getter的声明和实现，默认情况下，setter和getter方法中的实现，会

去访问下划线 \_ 开头的成员变量

若手动实现了setter方法，编译器就只会自动生成getter方法和成员变量

若手动实现了getter方法，编译器就只会自动生成setter方法和成员变量

若同时手动实现了setter和getter方法，编译器就不会自动生成不存在的成员变量

\*/

@property int age;

@end

**Dog.m**

#import "Dog.h"

@implementation Dog

-(void)setAge:(int)age

{

}

//-(int)age

//{

// return 10;

//}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Dog.h"

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

Dog \*d = [Dog new];

d.age = 5;

NSLog(@"狗的年龄是%d",d.age);

return 0;

}

**05 点语法**

**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;

}

-(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;

int a = p.age;

// int a = [p age];

return 0;

}

**06 成员变量的作用域**

**person.h**

/\*

@public:在任何地方都可以直接访问对象的成员变量

@private:只能在当前类的对象方法中访问成员变量（@implementation中默认是@private）

@protected:可以在当前类及其子类的对象方法中访问成员变量（@interface中默认就是@protected）

@package:只要处在同一个框架里，就能直接访问对象的成员变量

\*/

#import <Foundation/Foundation.h>

@interface Person : NSObject

{

@public //在任何地方都可以直接访问对象的成员变量

int \_age;

@private //只能在当前类的对象方法中直接访问

int \_height;

@protected //能在当前类和子类的对象方法中直接访问

int \_weight;

}

-(void)test;

-(void)setHeight:(int)height;

-(int)height;

@end

**person.m**

#import "Person.h"

@implementation Person

-(void)test

{

\_age = 10;

}

-(void)setHeight:(int)height

{

\_height = height;

}

-(int)height

{

return \_height;

}

@end

**student.h**

#import "Person.h"

@interface Student : Person

-(void)study;

@end

**student.m**

#import "Student.h"

@implementation Student

-(void)study

{

[self setHeight:10];

int h = [self height];

//\_height = 10;

\_weight = 68;

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Person.h"

#import "Student.h"

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

Person \*p = [Person new];

p->\_age = 58;

//p->\_height = 178;

Student \*s = [Student new];

[s setHeight:167];

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

return 0;

}

**07 self**

**person.h**

#import <Foundation/Foundation.h>

@interface Person : NSObject

@property int age;

-(void)test;

@end

**person.m**

#import "Person.h"

@implementation Person

-(void)test

{

//self:指向了方法的调用者，代表了当前对象

int \_age = 20;

NSLog(@"person的年龄是%d",self->\_age);

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Person.h"

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

Person \*p = [Person new];

p.age = 10;

[p test];

return 0;

}

**08 self**

**Dog.h**

#import <Foundation/Foundation.h>

@interface Dog : NSObject

-(void)bark;

-(void)run;

-(void)test;

+(void)test;

-(void)test1;

+(void)test2;

@end

**Dog.m**

#import "Dog.h"

@implementation Dog

-(void)bark

{

NSLog(@"汪汪汪");

}

//让狗跑之前先叫三声

-(void)run

{

[self bark];

NSLog(@"跑跑跑");

}

-(void)test

{

NSLog(@"调用了-test方法");

//会引发死循环

//[self test];

}

+(void)test

{

NSLog(@"调用了+test方法");

//会引发死循环

//[self test];

}

-(void)test1

{

[self test];

}

+(void)test2

{

[self test];

}

@end

**main.m**

#import <Foundation/Foundation.h>

#import "Dog.h"

/\*

self的用途：

1、谁调用了当前方法，self就代表谁（方法调用者）

\*self出现在对象方法中，self就代表对象

\*self出现在类方法中，self就代表类

2、在对象方法中，可以利用“self->成员变量名”访问当前对象的成员变量

3、[self 方法名]可以调用其他对象方法或类方法

\*/

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

Dog \*d = [Dog new];

[d run];

[d test];

[Dog test];

[d test1];

[Dog test2];

return 0;

}

**09 继承**

**Animal.h**

#import <Foundation/Foundation.h>

@interface Animal : NSObject

@property int age;

@property double weight;

-(void)bark;

+(void)test;

@end

**Animal.m**

#import "Animal.h"

@implementation Animal

-(void)bark

{

NSLog(@"动物叫");

}

+(void)test

{

NSLog(@"Animal +test");

}

@end

**Dog.h**

#import "Animal.h"

@interface Dog : Animal

@property int age;

-(void)bark;

@end

**Dog.m**

#import "Dog.h"

@implementation Dog

//重写：子类重新实现父类中的某个方法，覆盖父类以前的做法

-(void)bark

{

NSLog(@"汪汪汪");

}

+(void)test

{

NSLog(@"dog +test");

}

@end

**Cat.h**

#import "Animal.h"

@interface Cat : Animal

@end

**Cat.m**

#import "Cat.h"

@implementation Cat

@end

**Main.m**

#import <Foundation/Foundation.h>

#import "Dog.h"

//:Animal 继承了Animal，相当于继承了Animal里面的所有成员变量和方法

//Animal称为Dog的父类，Dog称为Animal的子类

/\*

继承的好处：

1、抽取重复的代码

2、建立了类之间的关系

3、子类可以拥有父类中的所有成员变量和方法

注意点：

1、基本上所有类的根类都是继承NSobject

2、重写：子类重新实现父类中的某个方法，覆盖父类以前的做法

3、调用某个对象的方法时，优先去当前类中找，如果找不到再去父类中找

\*/

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

Dog \*d = [Dog new];

d.age = 10;

NSLog(@"dog's age is %d",d.age);

[d bark];

//[Animal test];

[Dog test];

return 0;

}