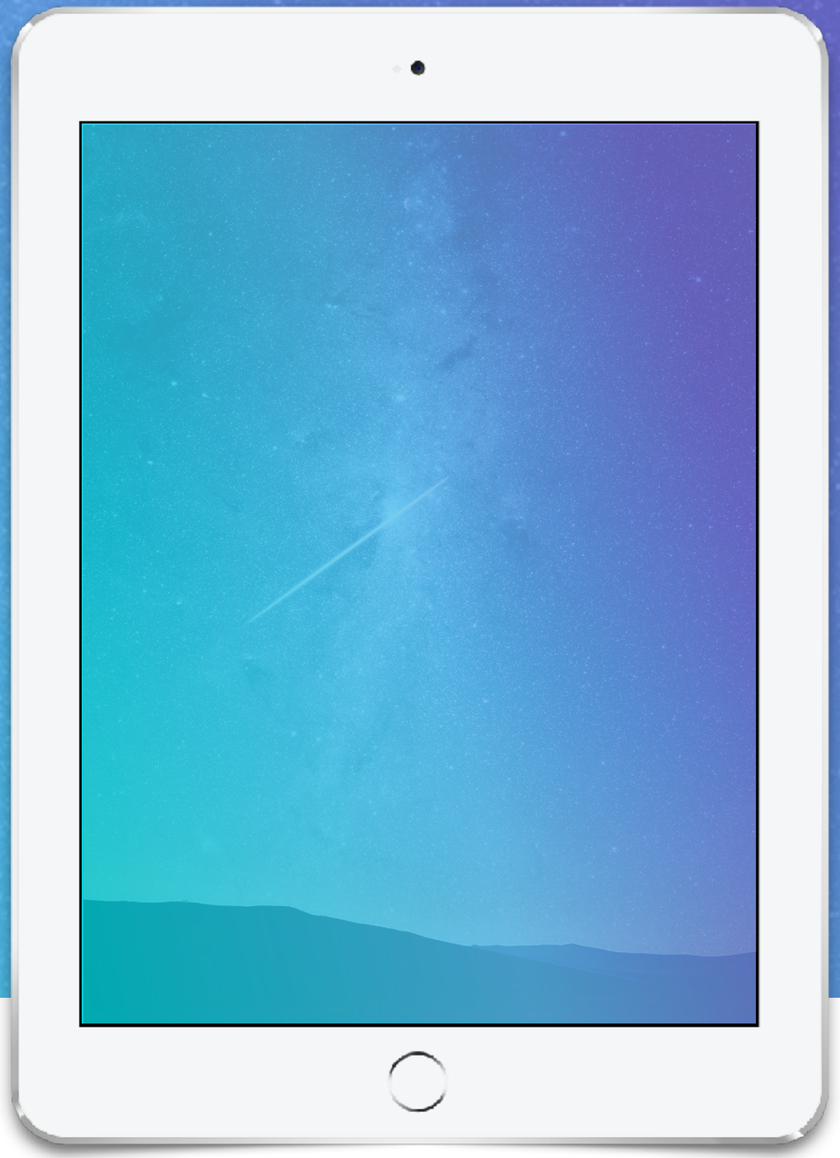
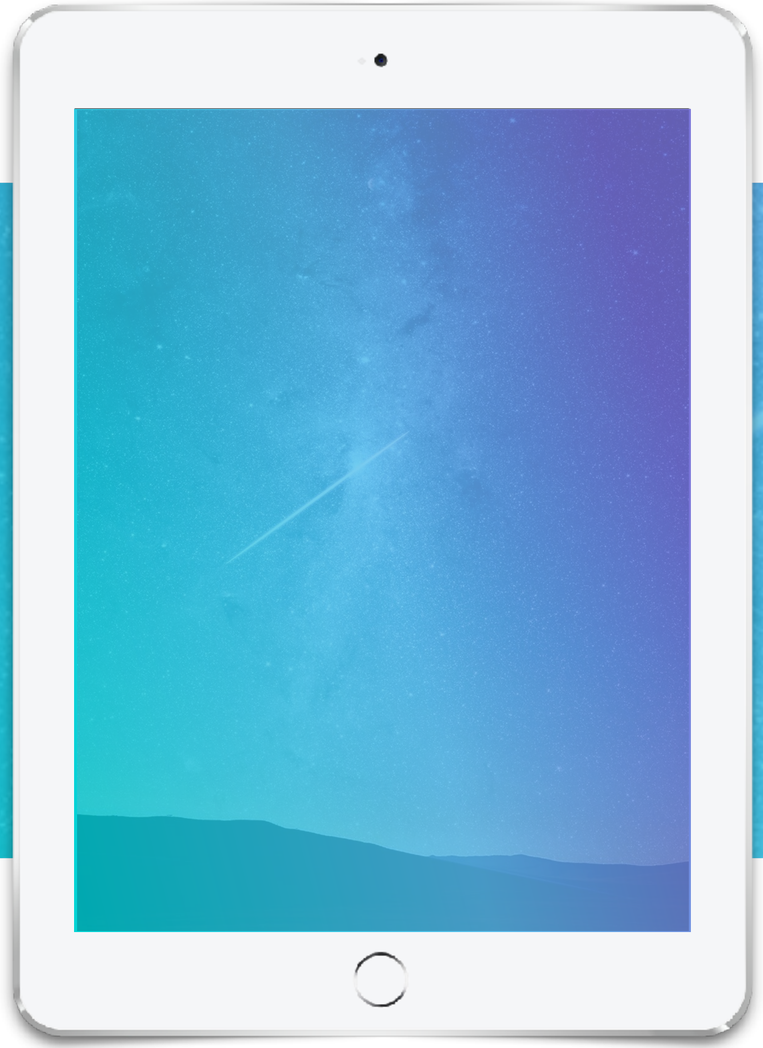
iOS 黑魔法课程

第三课
OC语言魔法（下）



StuQueue

一个新的学习方式



本课内容

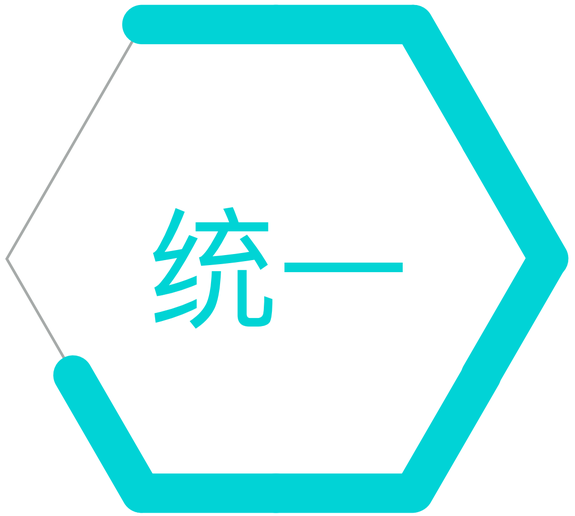
- OC OOP 之封装特性
- OC OOP 之继承特性
- OC OOP 之多态特性

OC OOP 之封装特性

StuQ

OC OOP 之封装性

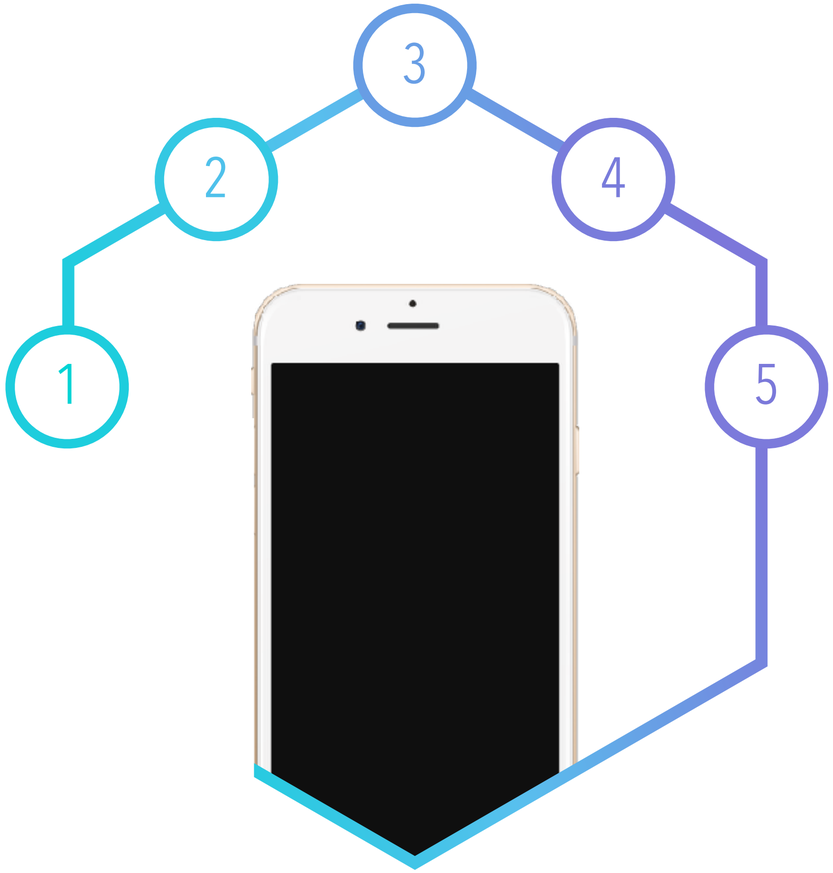
特点



OC OOP 之封装性

统一性

- 实例操作统一到一起
- 实例的定义放在类上
- 成员变量、属性、方法



OC OOP 之封装性

隐藏细节

1. 成员变量的隐藏
2. 方法的隐藏
3. 属性的隐藏

StuQ

OC OOP 之封装性

隐藏成员变量

- 在interface中利用private定义变量

```
@interface PrivateVariables : NSObject {
    @private
    NSString *privateIvar1;
}
@end
```

- 在extension中定义变量

```
@interface PrivateVariables () {
    float privateIvar2;
}
@end
```

- 在implementation中定义变量

```
@implementation PrivateVariables {
    NSDictionary *privateIvar3;
}
@end
```

OC OOP 之封装性

打破私有变量的隐藏

```
@interface PrivateVariables : NSObject {
    @private
    NSString *privateIvar1;
}
@end
```

```
@interface PrivateVariables () {
    float privateIvar2;
}
@end
```

```
@implementation PrivateVariables {
    NSDictionary *privateIvar3;
}
@end
```

利用KVC

利用成员变量相关API

```
NSString *ivar1 = [object
        valueForKey:@"privateIvar1"];
```

```
unsigned int ivarCount = 0;
Ivar *ivars = class_copyIvarList(object.class, &ivarCount);
NSMutableDictionary *testIvar = [@{@"f": @NO, @"struct": @NO, @"id": @NO} mutableCopy];
for (int i = 0; i < ivarCount; ++i) {
    Ivar ivar = ivars[i];
    const char *type = ivar_getTypeEncoding(ivar);
    NSString *typeString = [NSString stringWithUTF8String:type];
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF like $TYPE"];
    if ([predicate evaluateWithObject:typeString substitutionVariables:@{@"TYPE": @"f"} ]) {
        float f = 0.0f;
        ptrdiff_t offset = ivar_getOffset(ivar);
        f = *(float *)((__bridge void *)object + offset);

        expect(f).to.equal(1.05f);
        testIvar[@"f"] = @YES;
    }
    if ([predicate evaluateWithObject:typeString substitutionVariables:@{@"TYPE": @"{CGRect*"} ]) {
        CGRect rect = CGRectZero;
        ptrdiff_t offset = ivar_getOffset(ivar);
        rect = *(CGRect *)((__bridge void *)object + offset);

        expect(rect).to.equal(CGRectMake(15, 15, 20, 20));

        testIvar[@"struct"] = @YES;
    }
    if ([predicate evaluateWithObject:typeString substitutionVariables:@{@"TYPE": @"@\"NSDictionary\""} ])
{
        NSString *str = object_getIvar(object, ivar);
        expect(str).to.equal(@{@"key": @"value"});
        testIvar[@"id"] = @YES;
    }
}
expect(testIvar).to.equal(@{@"f": @YES, @"struct": @YES, @"id": @YES});
free(ivars);
```

```
unsigned int ivarCount = 0;
Ivar *ivars = class_copyIvarList(object.class, &ivarCount);
NSMutableDictionary *testIvar = [@{@"f": @NO, @"struct": @NO, @"id": @NO} mutableCopy];
for (int i = 0; i < ivarCount; ++i) {
    Ivar ivar = ivars[i];
    const char *type = ivar_getTypeEncoding(ivar);
    NSString *typeString = [NSString stringWithUTF8String:type];
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF like $TYPE"];
    if ([predicate evaluateWithObject:typeString substitutionVariables:@{@"TYPE": @"f"} ]) {
        float f = 0.0f;
        ptrdiff_t offset = ivar_getOffset(ivar);
        f = *(float *)((__bridge void *)object + offset);

        expect(f).to.equal(1.05f);
        testIvar[@"f"] = @YES;
    }
    if ([predicate evaluateWithObject:typeString substitutionVariables:@{@"TYPE": @"{CGRect*"} ]) {
        CGRect rect = CGRectZero;
        ptrdiff_t offset = ivar_getOffset(ivar);
        rect = *(CGRect *)((__bridge void *)object + offset);

        expect(rect).to.equal(CGRectMake(15, 15, 20, 20));

        testIvar[@"struct"] = @YES;
    }
    if ([predicate evaluateWithObject:typeString substitutionVariables:@{@"TYPE": @"@\"NSDictionary\""} ]) {
        NSString *str = object_getIvar(object, ivar);
        expect(str).to.equal(@{@"key": @"value"});
        testIvar[@"id"] = @YES;
    }
}
expect(testIvar).to.equal(@{@"f": @YES, @"struct": @YES, @"id": @YES});
free(ivars);
```

OC OOP 之封装性

P.S. ivar runtime api bug

```
OBJC_EXPORT Ivar object_getInstanceVariable(id obj,
                                            const char *name,
                                             void **outValue)
OBJC_AVAILABLE(10.0, 2.0, 9.0, 1.0)
OBJC_ARC_UNAVAILABLE;
```

OC OOP 之封装性

隐藏方法

- .h 文件中暴露公开方法接口
- .m 文件中实现隐藏方法

```
@interface HidingMethods : NSObject

- (void)publicMethod1;

@end
```

```
@implementation HidingMethods

- (void)publicMethod1
{
    NSLog(@"This is A Public method!");
}

- (void)privateMethod2
{
    NSLog(@"This is A Private method!");
}

@end
```

OC OOP 之封装性

打破私有方法的隐藏

已经确认了方法 → 利用Category

尚未确认了方法名 → 利用方法相关API

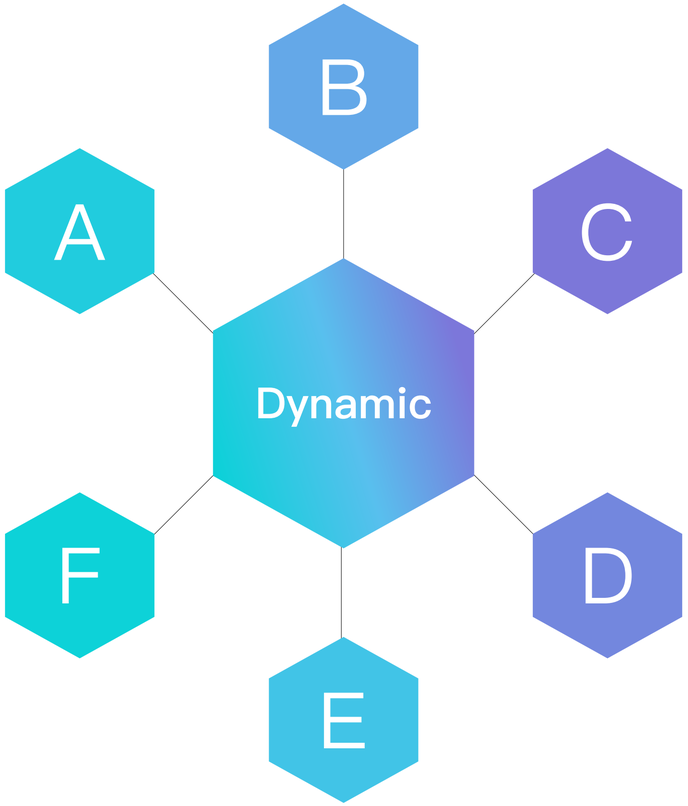
```
#import "Hidingmethods.h"

@interface HidingMethods(BreakEncapsulation)

- (void)privatemethod2;

@end
```

```
void breakMethodEncapsulation3(void){
    HidingMethods *hidingMethods = [[HidingMethods alloc]
init];
    //利用方法API
    unsigned int methodsCount;
    method *methodList = class_copymethodList([hidingMethods
class], &methodsCount);
    for (int i = 0; i < methodsCount; i++) {
        SEL cmd = method_getName(methodList[i]);
        //invocation执行方法
        NSmethodSignature *signature = [hidingMethods
methodSignatureForSelector:cmd];
        NSInvocation *invocation = [NSInvocation
invocationWithmethodSignature:signature];
        [invocation setTarget:hidingMethods];
        [invocation setSelector:cmd];
        [invocation invoke];
    }
}
```

OC OOP 之封装性

动态调用方法的手段

- performSelector
- objc_msgSend
- NSInvocation
- libffi



OC OOP 之封装性

隐藏属性

- 属性与方法
- .h 文件暴露公开属性
- .m interface extension隐藏属性

```
@interface HidingProperty : NSObject

@property (nonatomic, strong) NSString *publicProperty;

@end
```

```
@interface HidingProperty ()

@property (nonatomic, strong) NSString *privateProperty;

@end
```

OC OOP 之封装性

打破私有属性的隐藏

已经确认了属性 → 利用Category

尚未确认了属性名 → 利用属性相关API

```
#import "HidingProperty.h"

@interface HidingProperty(BreakEncapsulation)

@property (nonatomic, strong) NSString *privateProperty;

@end
```

```
void breakPropertyEncapsulation(NSObject *object){
    //利用propertyAPI
    unsigned int propertysCount;
    objc_property_t *propertyList =
class_copyPropertyList(object.class,
&propertysCount);
    for (int i = 0; i < propertysCount; i++) {
        objc_property_t property =
propertyList[i];
        //取得属性名
        NSString *propertyName = [NSString
stringWithUTF8String:property_getName(property)];
        //利用kvc获取成员变量
        id ivar = [object
valueForKey:propertyName];
    }
}
```

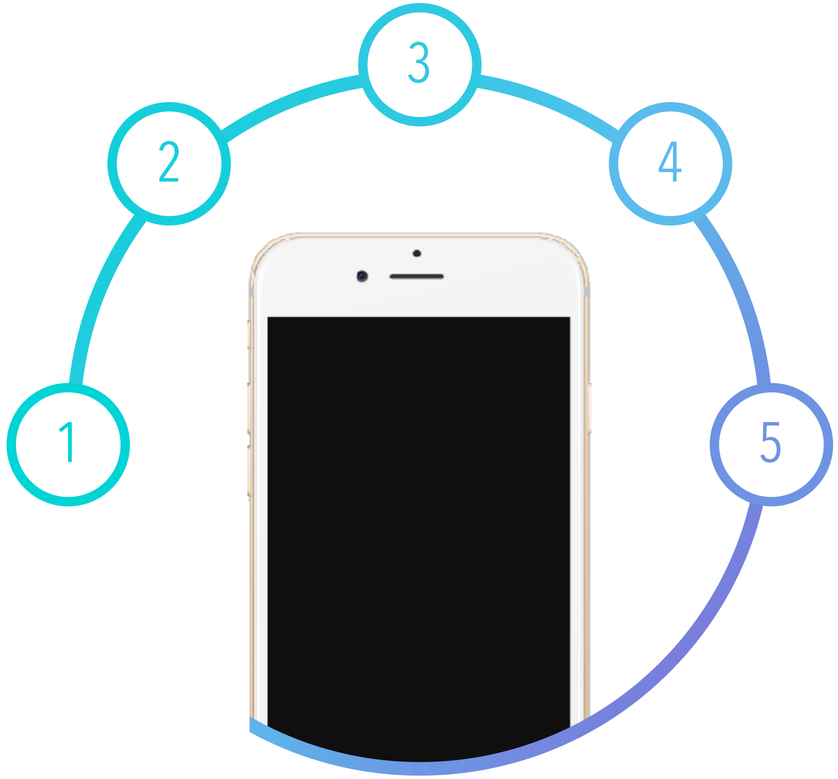
OC OOP 之继承特性

StuQ

OC OOP 之继承性

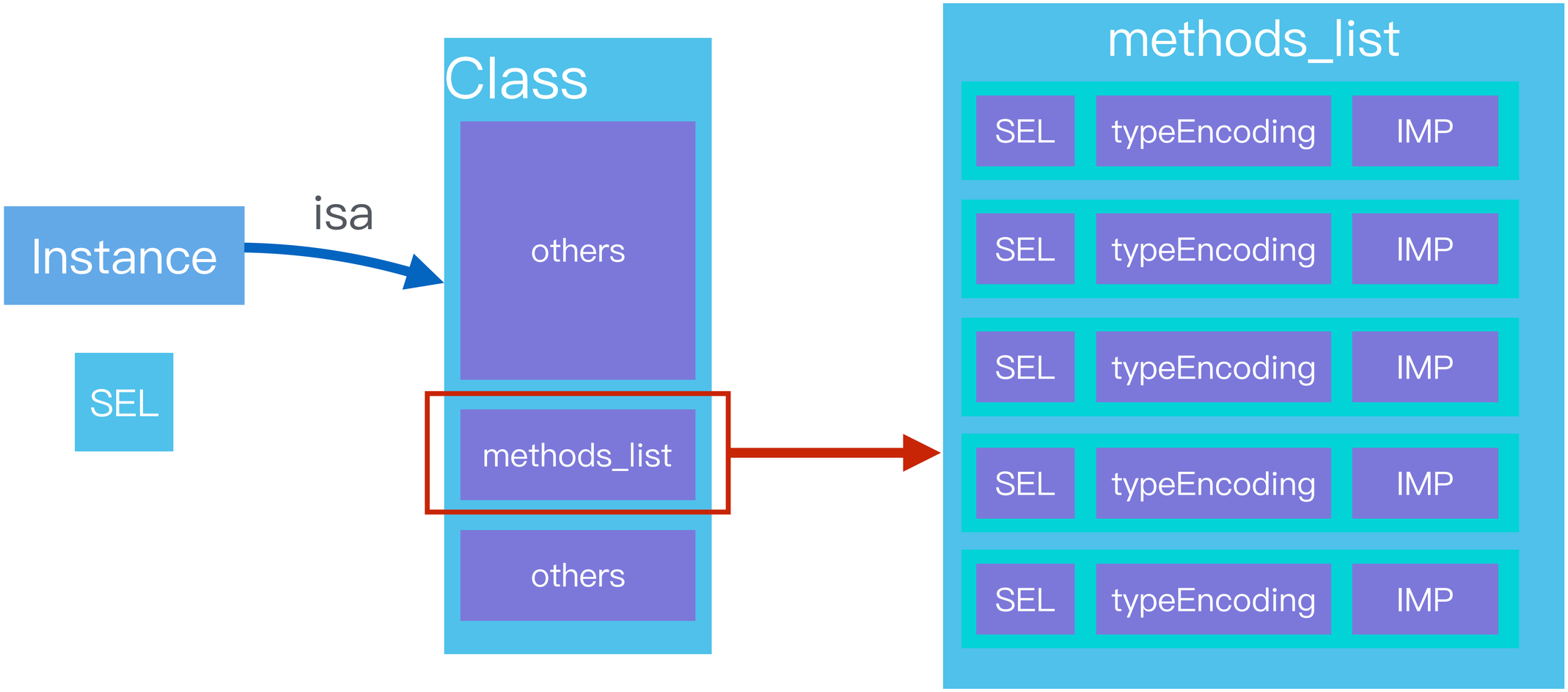
内容

- 行为的继承
- 状态的扩展



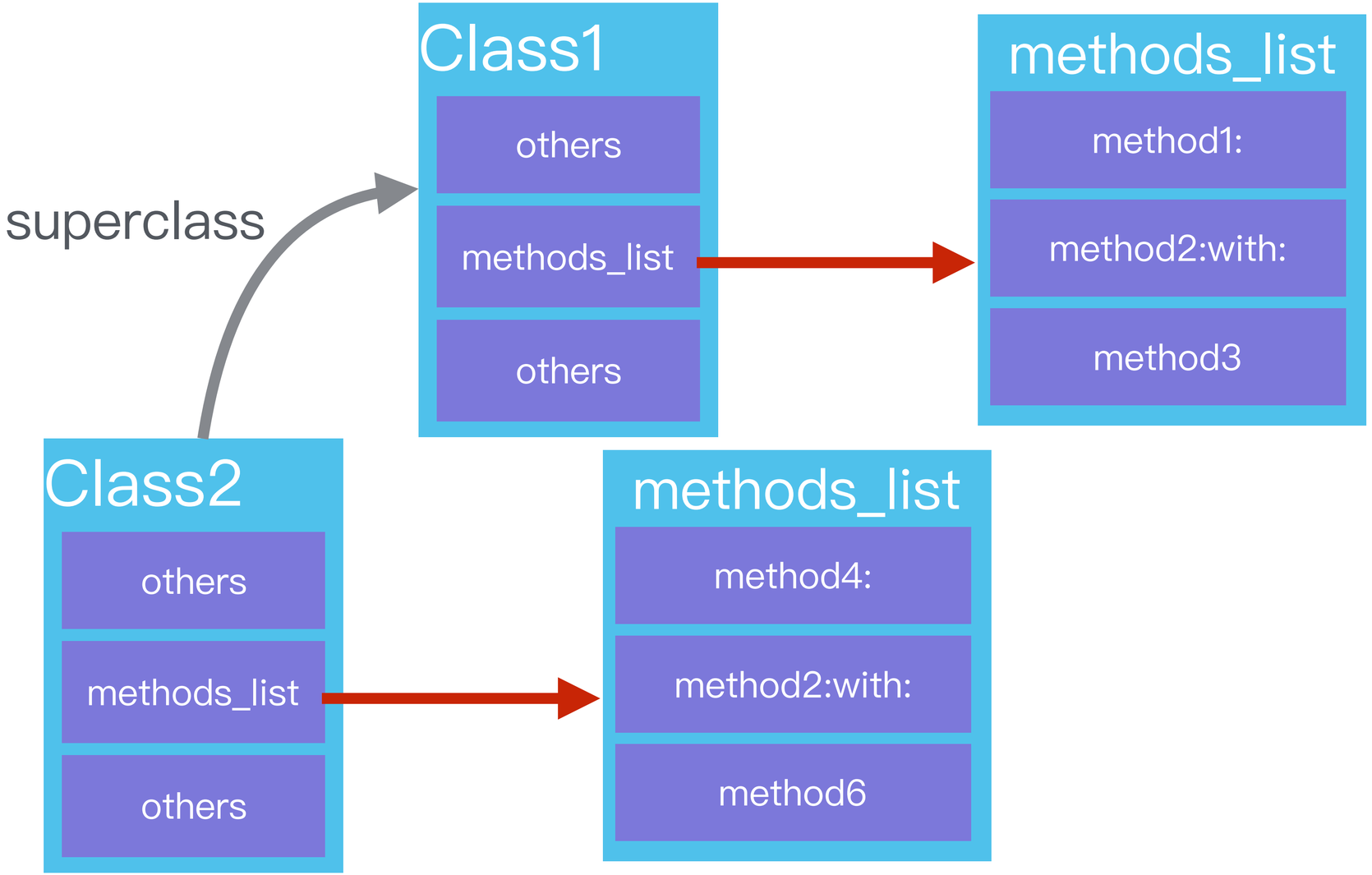
OC OOP 之继承性

回忆：实例如何找到方法



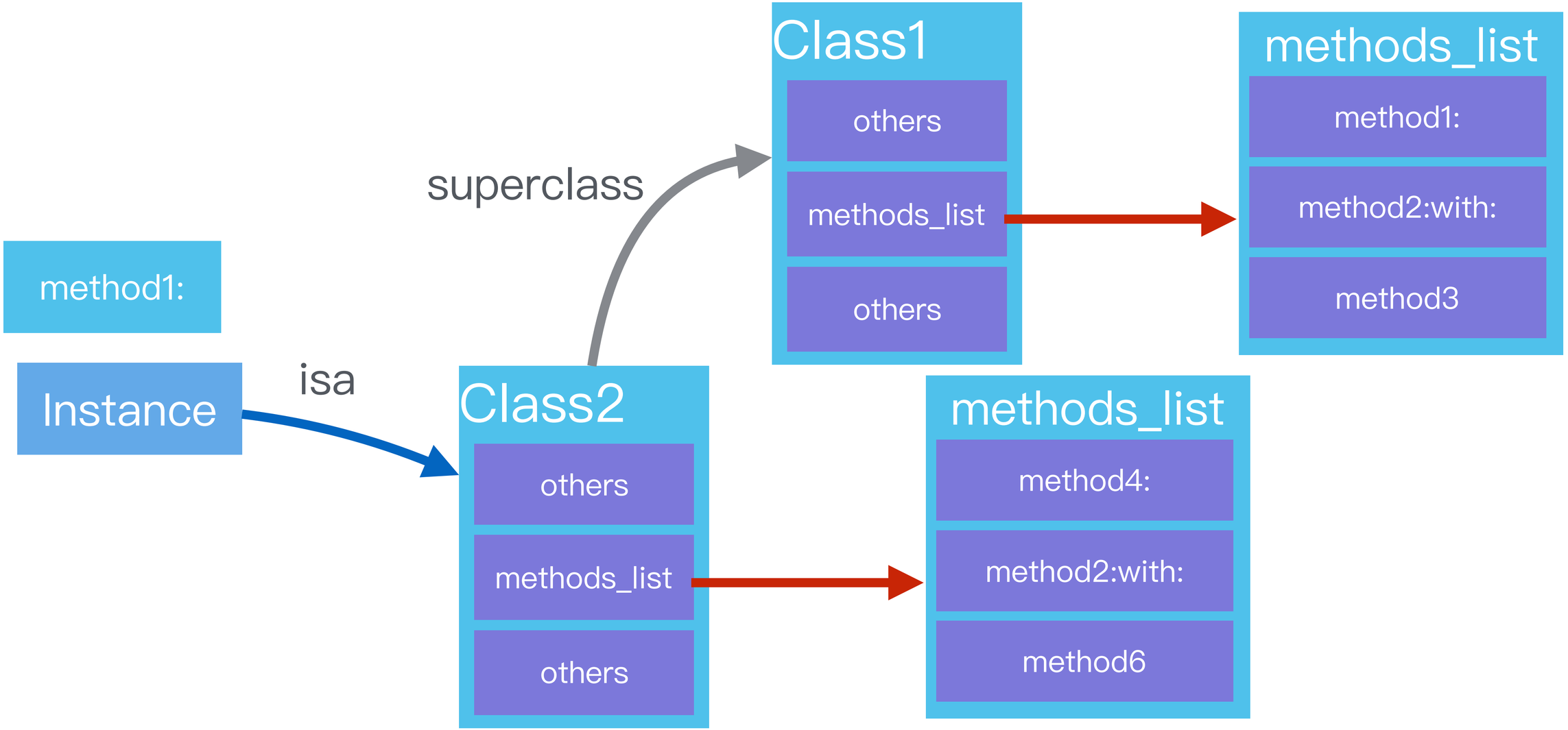
OC OOP 之继承性

扩展方法



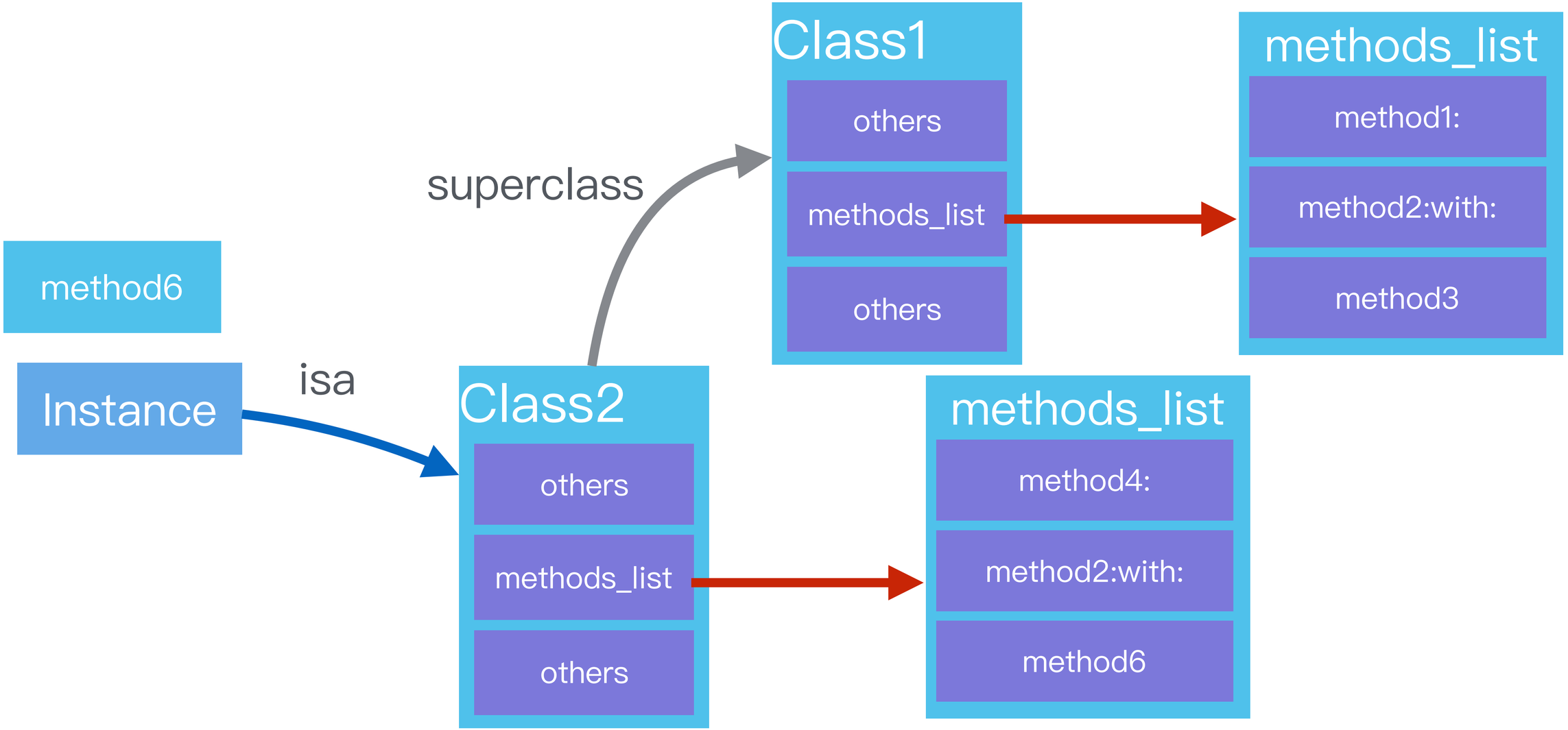
OC OOP 之继承性

调用方法



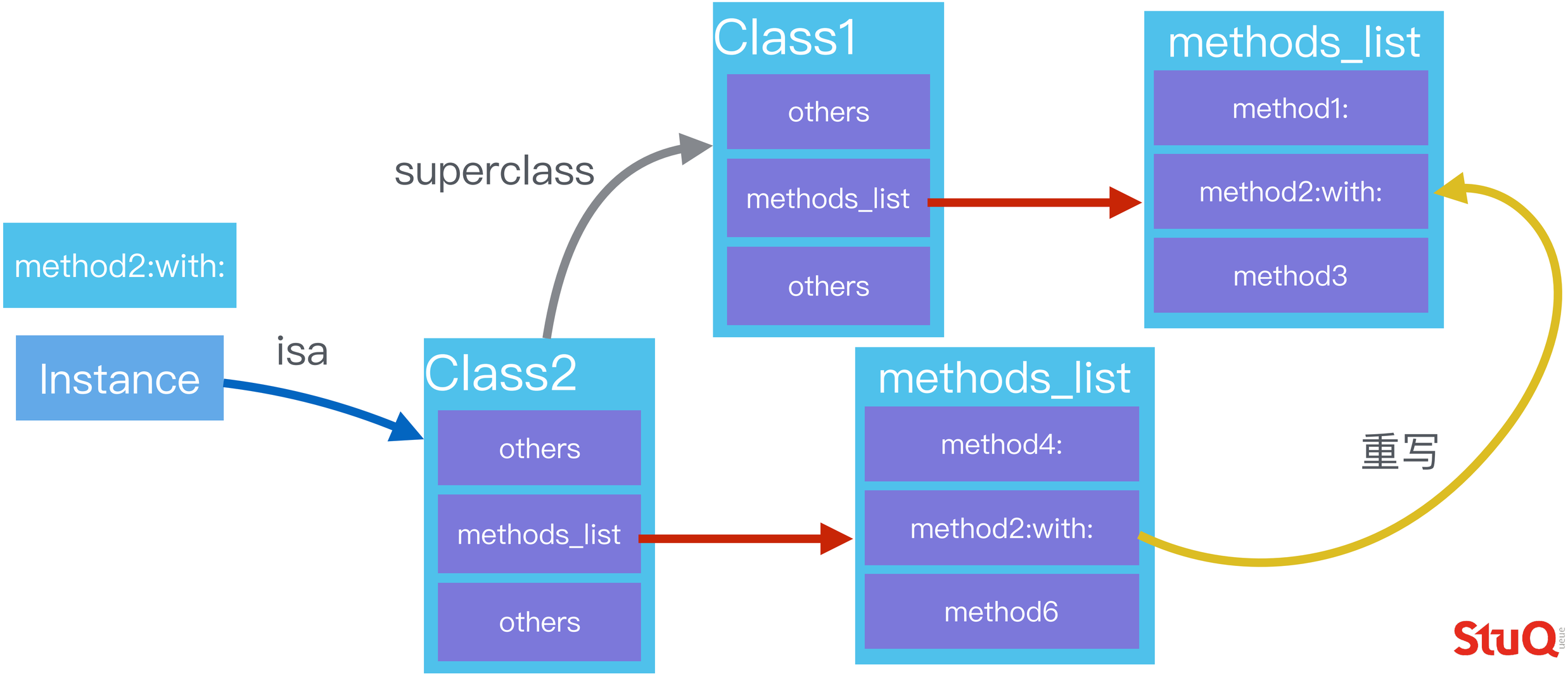
OC OOP 之继承性

调用方法



OC OOP 之继承性

调用方法



OC OOP 之继承性

重写

```
@interface Class1 : NSObject
- (void)method1:(NSString *)param1;
- (int)method2:(int)param1 with:(int)param2;
- (id)method3;
@end

@interface Class2 : Class1
- (void)method4:(NSString *)param1;
- (int)method2:(int)param1 with:(int)param2;
- (id)method6;
@end
```

OC OOP 之继承性

重写

```
@interface Class1 : NSObject
- (void)method1:(NSString *)param1;
- (int)method2:(int)param1 with:(int)param2;
- (id)method3;
@end

@interface Class2 : Class1
- (void)method4:(NSString *)param1;
- (int)method2:(int)param1 with:(int)param2;
- (id)method6;
@end
```

```
@implementation Class1
- (int)method2:(int)param1 with:(int)param2;
{
    NSLog(@"%s", __FUNCTION__);
    return param1 + param2;
}
@end

typedef int (*Function)(id self, SEL sel, int param1, int param2);

@implementation Class2

- (int)method2:(int)param1 with:(int)param2;
{
    NSLog(@"%s", __FUNCTION__);
    unsigned int count;
    Method *methodsList = class_copyMethodList([Class1 class], &count);
    for (int i = 0; i < count; i++) {
        IMP imp = method_getImplementation(methodsList[i]);
        SEL sel = method_getName(methodsList[i]);
        if (strcmp(sel_getName(sel), "method2:with:") == 0) {
            Function func = (Function)imp;
            return func(self, sel, param1, param2);
        }
    }
    return -1;
}

@end
```

OC OOP 之继承性

Super api

```
- (int)method2:(int)param1 with:(int)param2;
{
    NSLog(@"%s", __FUNCTION__);
    unsigned int count;
    Method *methodsList = class_copyMethodList([Class1 class], &count);
    for (int i = 0; i < count; i++) {
        IMP imp = method_getImplementation(methodsList[i]);
        SEL sel = method_getName(methodsList[i]);
        if (strcmp(sel_getName(sel), "method2:with:") == 0) {
            Function func = (Function)imp;
            return func(self,sel, param1, param2);
        }
    }
    return -1;
}
```

```
- (int)method2:(int)param1 with:(int)param2
{
    struct objc_super super_struct = {
        .receiver = self,
        .super_class = [Class1 class]
    };
    typedef int *MyObjc_msgSendSuper(struct objc_super *, SEL, int, int);
    MyObjc_msgSendSuper *func = &objc_msgSendSuper;
    return func(&super_struct, @selector(method2:with:), param1, param2);
}
```

OC OOP 之继承性

Super 关键字

```
- (int)method2:(int)param1 with:(int)param2;
{
    NSLog(@"%s", __FUNCTION__);
    unsigned int count;
    Method *methodsList = class_copyMethodList([Class1 class], &count);
    for (int i = 0; i < count; i++) {
        IMP imp = method_getImplementation(methodsList[i]);
        SEL sel = method_getName(methodsList[i]);
        if (strcmp(sel_getName(sel), "method2:with:") == 0) {
            Function func = (Function)imp;
            return func(self,sel, param1, param2);
        }
    }
    return -1;
}
```

```
- (int)method2:(int)param1 with:(int)param2
{
    return [super method2:param1 with:param2];
}
```

self

Class1

OC OOP 之继承性

动态查找与静态查找

```
class CPPClassA {
public:
    int a(int a);
};

class CPPClassB : public CPPClassA {

};

class CPPClassC : public CPPClassB {
public:
    int a(int a);
};
```

```
int CPPClassA::a(int a) {
    return a * a;
}

int CPPClassC::a(int a) {
    return CPPClassA::a(a);
}
```

```
@interface OCClassA : NSObject
- (int)a:(int)a;
@end

@interface OCClassB : OCClassA
@end

@interface OCClassC : OCClassB
- (int)a:(int)a;
@end
```

```
@implementation OCClassA

- (int)a:(int)a { return a * a; }

@end

@implementation OCClassB
@end

@implementation OCClassC

- (int)a:(int)a { return [super a:a]; }

@end
```

OCClassA

?

OCClassB

OC OOP 之继承性

OC super vs Java super

```
class JavaClassA {
    public int a(int a) {
        return a * a;
    }
}

class JavaClassB extends JavaClassA {

}

class JavaClassC extends JavaClassB {
    @override
    public int a(int a) {
        return super.a(a);
    }
}
```

```
@interface OCClassA : NSObject
- (int)a:(int)a;
@end

@interface OCClassB : OCClassA
@end

@interface OCClassC : OCClassB
- (int)a:(int)a;
@end
```

```
@implementation OCClassA

- (int)a:(int)a { return a * a; }

@end

@implementation OCClassB
@end

@implementation OCClassC

- (int)a:(int)a { return [super a:a]; }

@end
```

OCClassB

OC OOP 之继承性

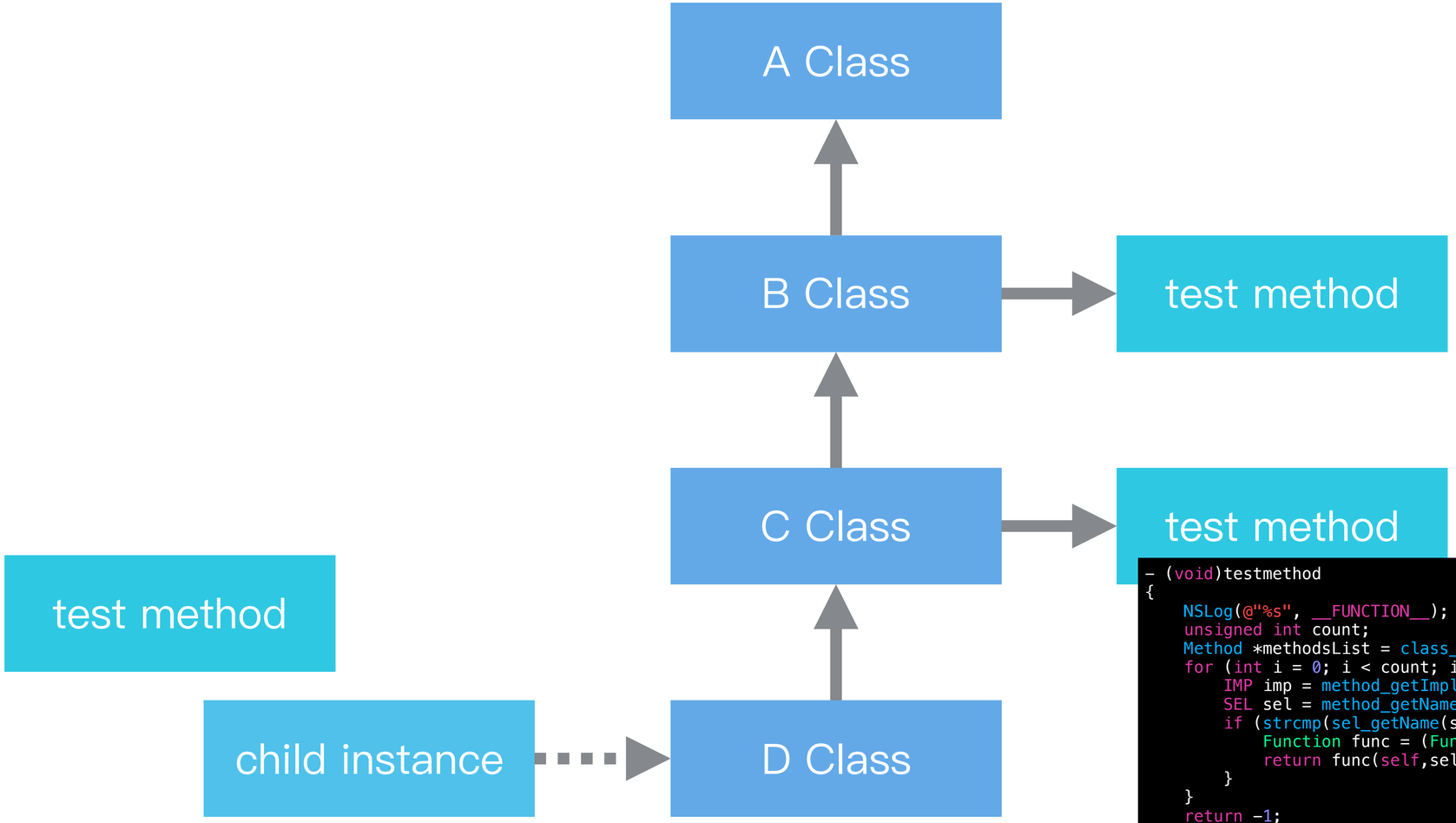
编译时 vs 运行时

```
- (int)method2:(int)param1 with:(int)param2;
{
    NSLog(@"%s", __FUNCTION__);
    unsigned int count;
    Method *methodsList = class_copyMethodList([Class1 class], &count);
    for (int i = 0; i < count; i++) {
        IMP imp = method_getImplementation(methodsList[i]);
        SEL sel = method_getName(methodsList[i]);
        if (strcmp(sel_getName(sel), "method2:with:") == 0) {
            Function func = (Function)imp;
            return func(self,sel, param1, param2);
        }
    }
    return -1;
}
```

```
- (int)method2:(int)param1 with:(int)param2;
{
    NSLog(@"%s", __FUNCTION__);
    unsigned int count;
    Method *methodsList = class_copyMethodList(self.class.superclass, &count);
    for (int i = 0; i < count; i++) {
        IMP imp = method_getImplementation(methodsList[i]);
        SEL sel = method_getName(methodsList[i]);
        if (strcmp(sel_getName(sel), "method2:with:") == 0) {
            Function func = (Function)imp;
            return func(self,sel, param1, param2);
        }
    }
    return -1;
}
```

OC OOP 之继承性

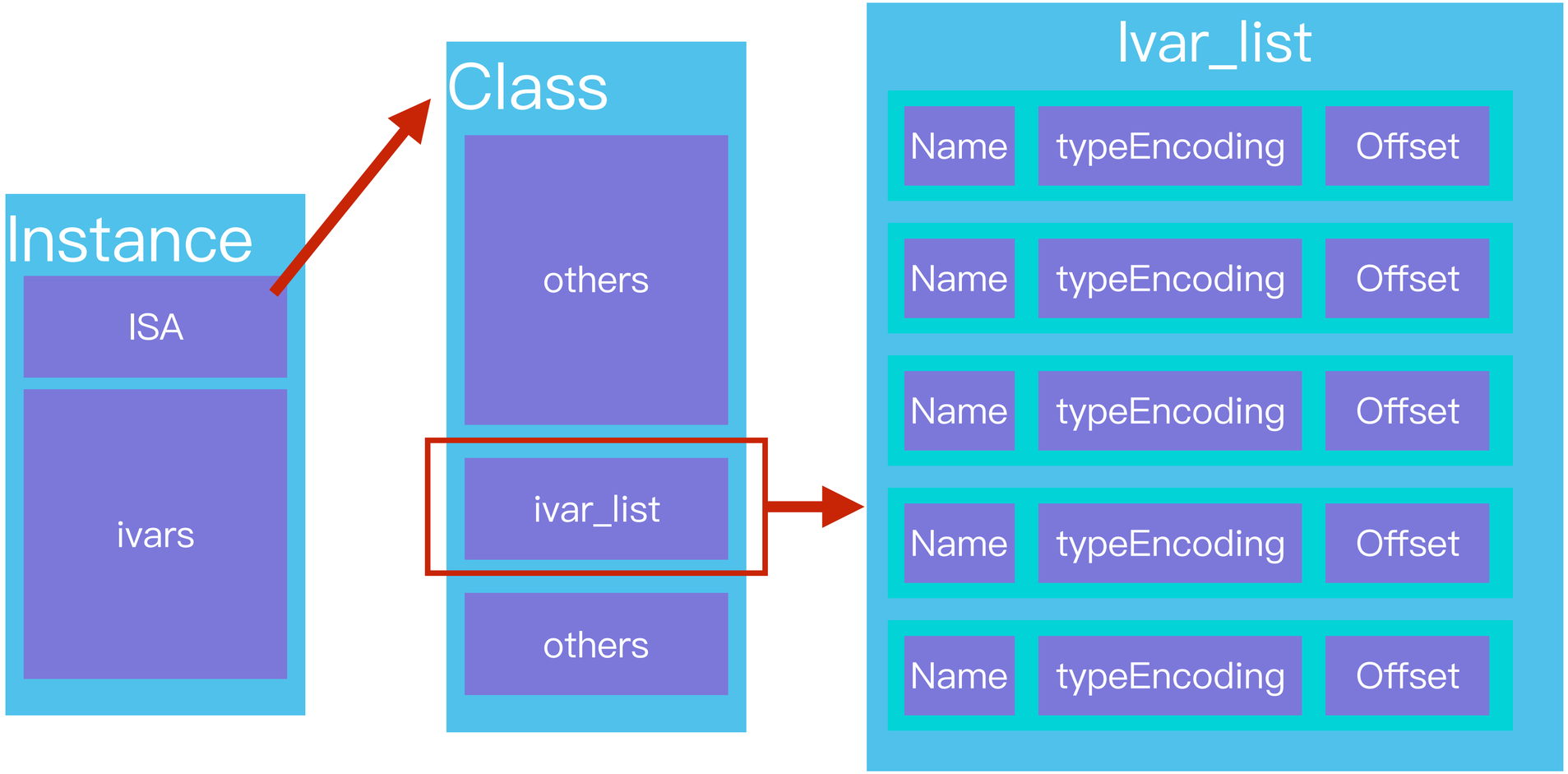
运行时找superclass的情况




```
- (void)testmethod
{
    NSLog(@"%s", __FUNCTION__);
    unsigned int count;
    Method *methodsList = class_copyMethodList(self.class.superclass, &count);
    for (int i = 0; i < count; i++) {
        IMP imp = method_getImplementation(methodsList[i]);
        SEL sel = method_getName(methodsList[i]);
        if (strcmp(sel_getName(sel), "testmethod") == 0) {
            Function func = (Function)imp;
            return func(self,sel);
        }
    }
    return -1;
}
```

OC OOP 之继承性

回忆：实例变量的存储

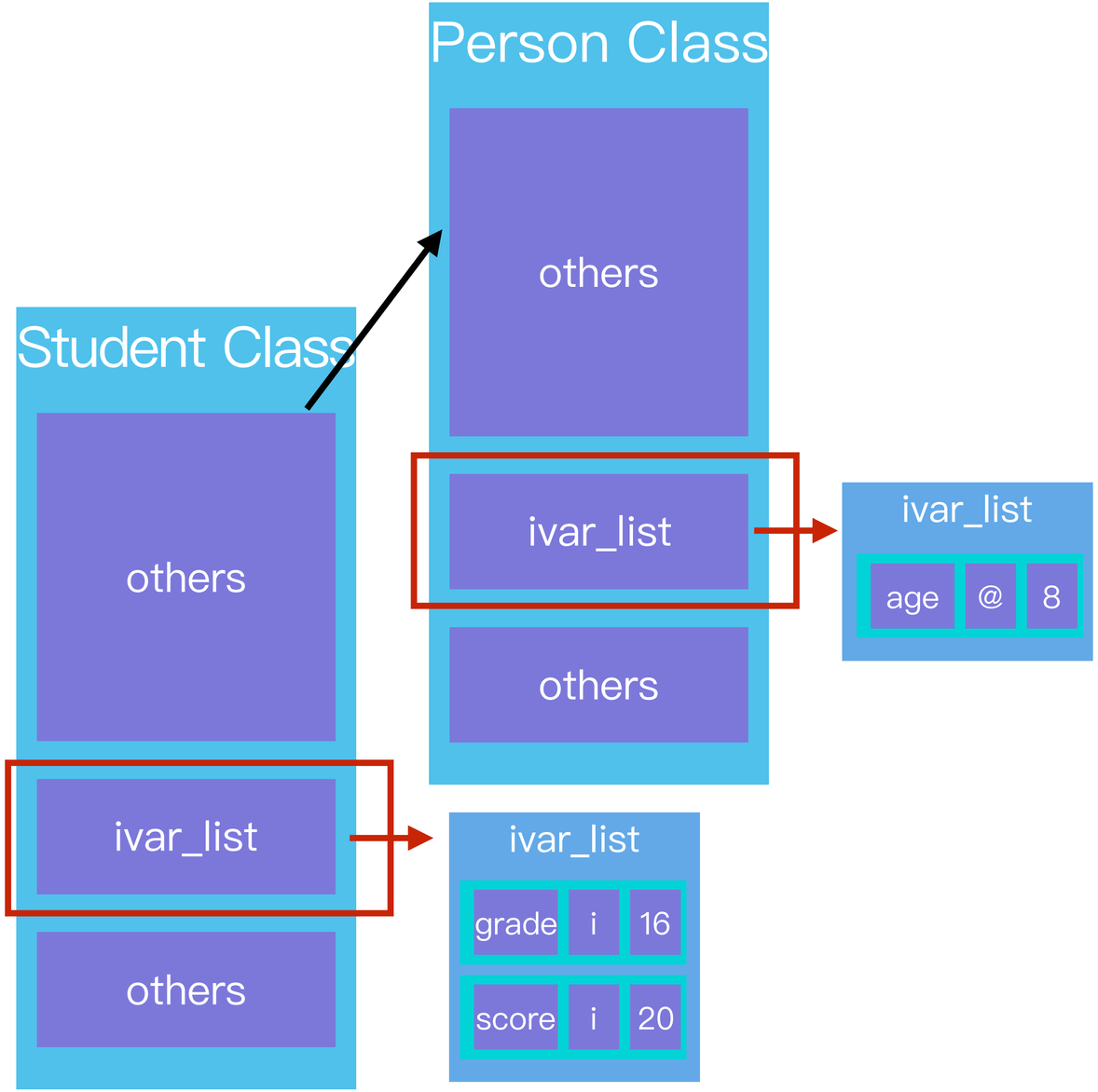


OC OOP 之继承性

继承模型中状态的描述

```
@interface Person : NSObject

@property (strong) NSNumber *age;

@end
```

```
@interface Student : Person

@property (assign) int grade;
@property (assign) int score;

@end
```

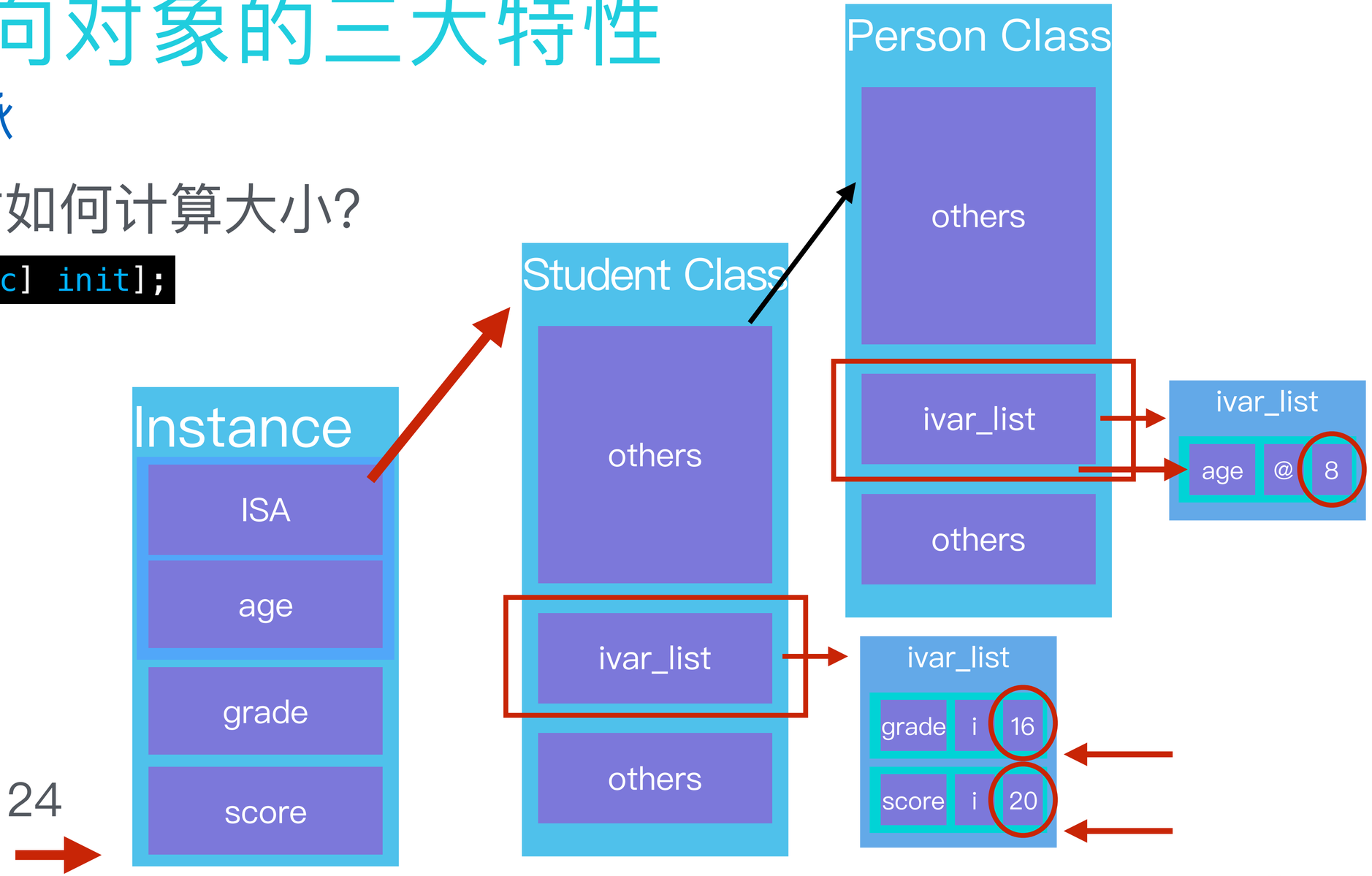


OC 面向对象的三大特性

状态的继承

子类创建时如何计算大小?

```
[[Student alloc] init];
```

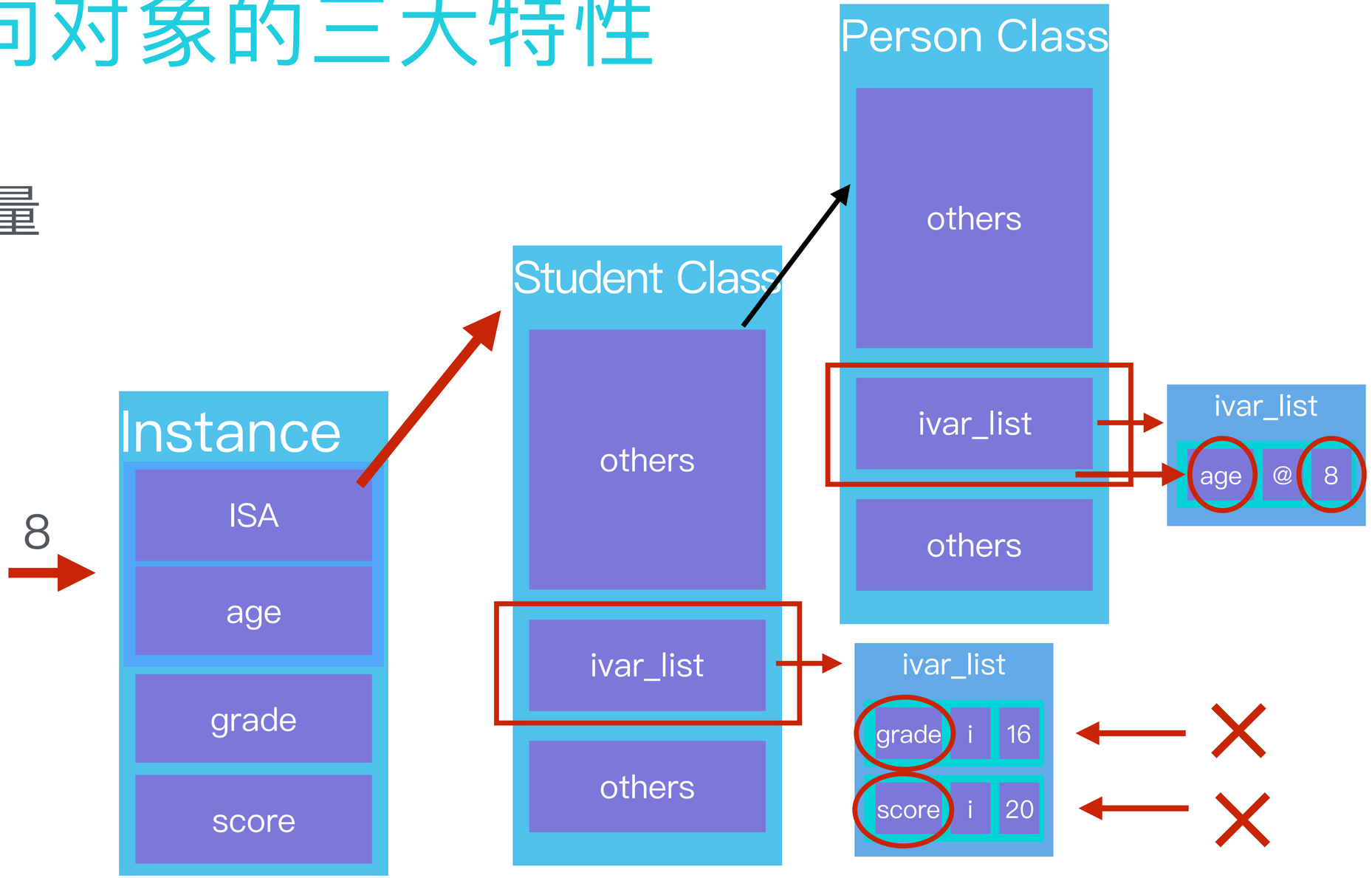


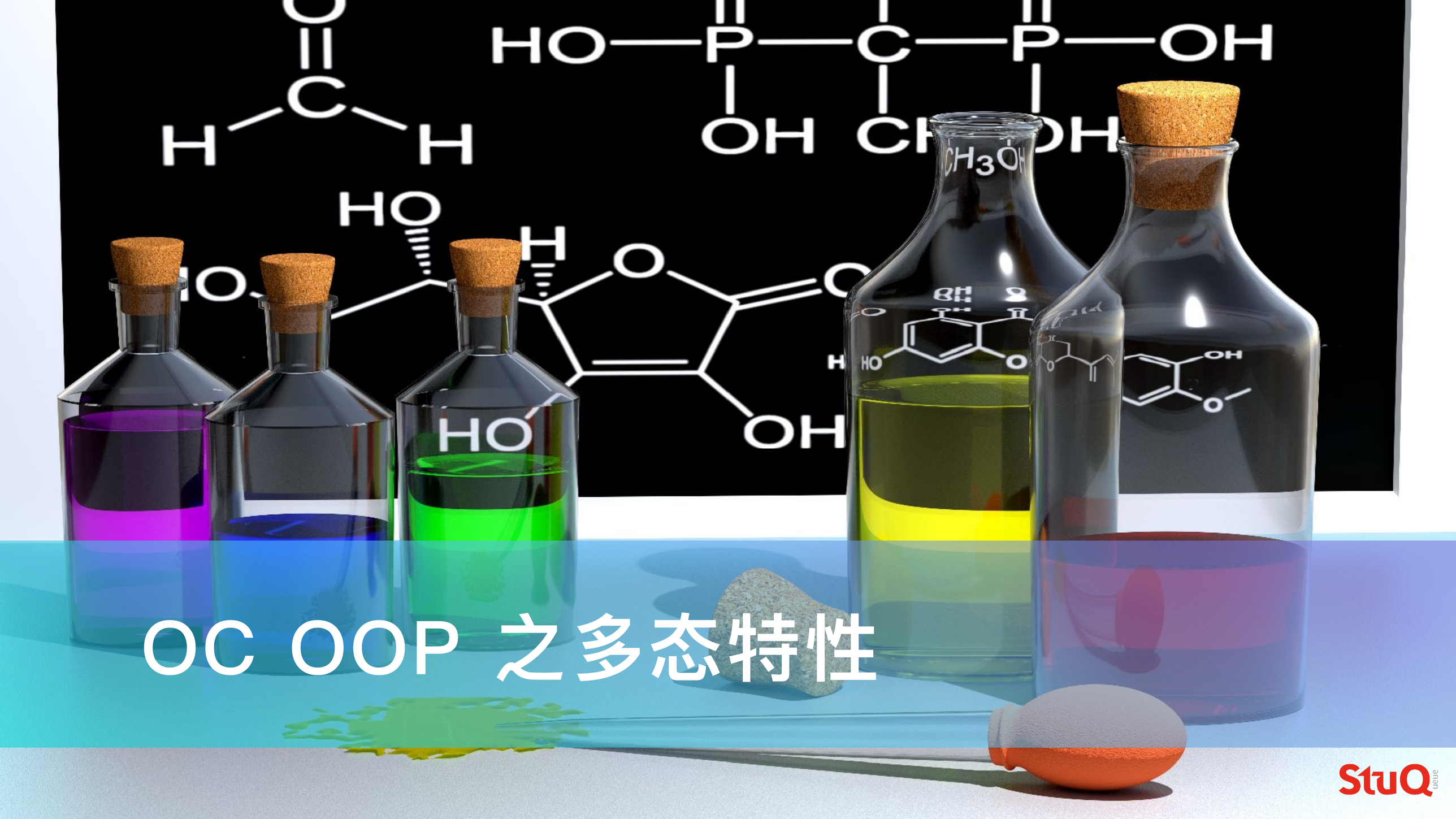
OC 面向对象的三大特性

状态的继承

如何访问变量

```
int var1 = obj->age;
```





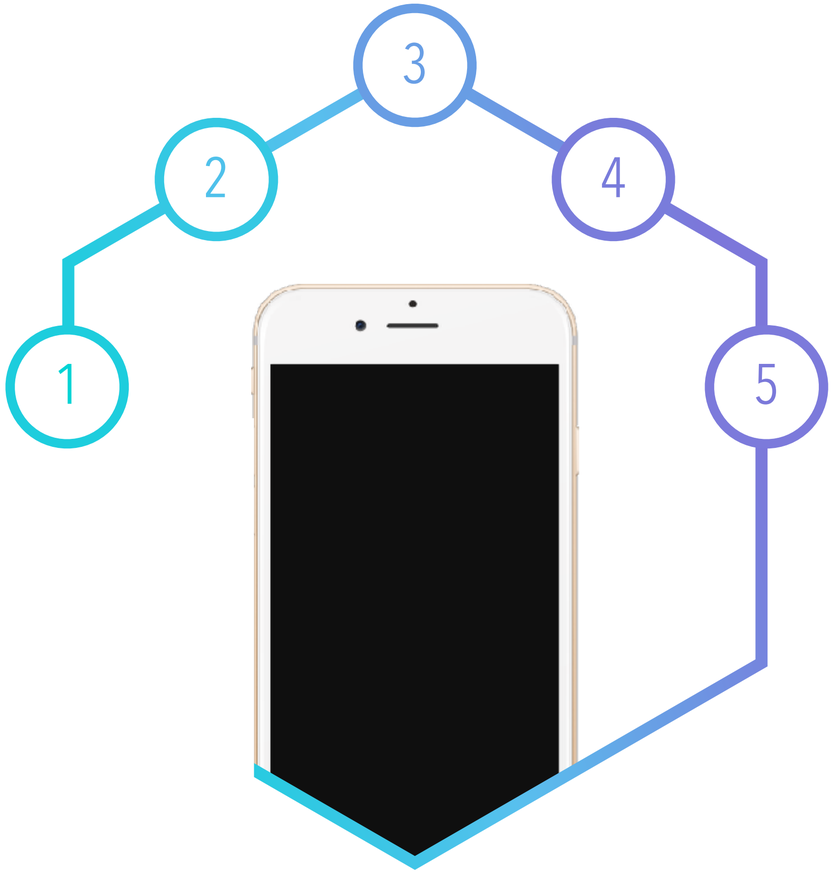
OC OOP 之多态特性

StuQ

OC OOP 之多态性

回忆：多态性

- 相同行为子类多种实现
- 通过父类抽象访问行为
- 不同子类给与不同响应



OC OOP 之多态性

运行时 vs 编译时

多态一定是利用运行时的而非编译时

```
@interface Car : NSObject
- (void)drive;
@end

@interface Taxi : Car
- (void)drive;
@end
```

```
+ (void)testMethodWithClass:(Car *)car
{
    [car drive];
}

+ (void)testPolymorphism
{
    Car *car = [[Taxi alloc] init];
    [TestPolymorphism testMethodWithClass:car];
}
```

```
class Car {
public:
    int gas;
    virtual void drive();
};

class Taxi : public Car {
public:
    int money;
    virtual void drive();
};
```

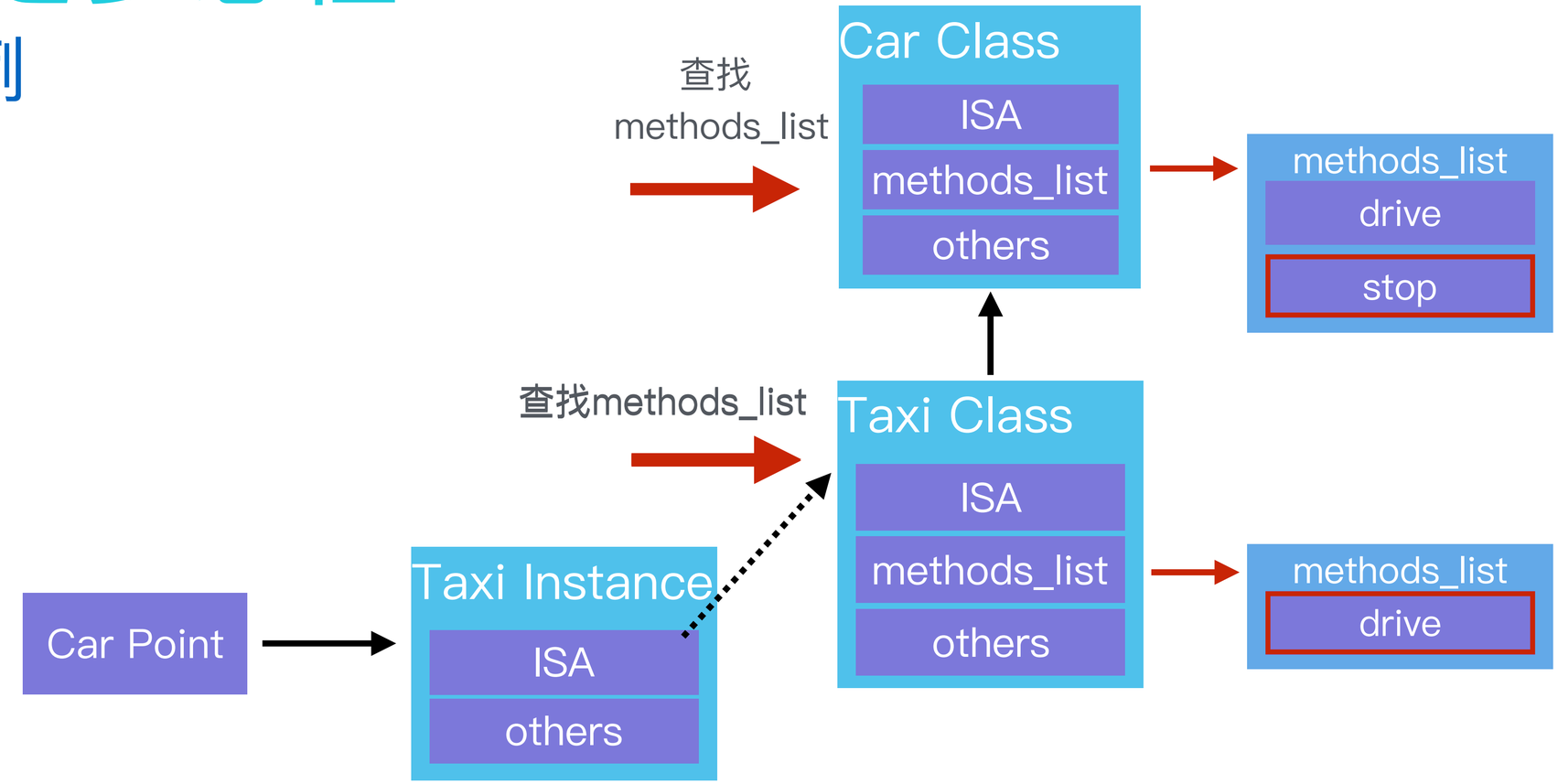
```
void testPolymorphism(Car *car){
    car->drive();
}
void test(void){
    Taxi taxi;
    Car *car = &taxi;
    testPolymorphism(car);
}
```

OC OOP 之多态性

OC多态过程示例

```
//定义新类Car，有驾驶和停止两个方法
@interface Car : NSObject
- (void)drive;
- (void)stop;
@end

@implementation Car
- (void)drive
{
    NSLog(@"Car Drive");
}

- (void)stop
{
    NSLog(@"Car Stop");
}
@end
```

```
//定义Taxi继承自Car
@interface Taxi : Car
@property (nonatomic, assign) int money;

- (void)drive;
- (void)stop;
@end

@implementation Taxi
- (void)drive
{
    self.money++;
    NSLog(@"Taxi Drive");
}
@end
```




```
Car *car = [[Taxi alloc] init];
//测试OC多态类，car的型在运行时才确定
[car drive];
[car stop];
```

- ISA永远指向最近的类
- 天然支持多态

OC OOP 之多态性

C++的多态–无虚函数

- 早期绑定
- 编译成指定方法

```
class Car {
public:
    int gas;
    void drive(){
        std::cout << "Car Drive" << std::endl;
    }
};
```

```
class Taxi : public Car {
public:
    int money;
    void drive(){
        money++;
        std::cout << "Taxi Drive" << std::endl;
    }
};
```

```
void testPolymorphism(Car *car){
    car->drive();
}
```

```
Taxi taxi;
Car *car = &taxi;
testPolymorphism(car); //执行Car的drive
```

OC OOP 之多态性

C++的多态–有虚函数

```
class Car {
public:
    int gas;
    virtual void drive(){
        std::cout << "Car Drive" << std::endl;
    }
    void stop(){
        std::cout << "Car Stop" << std::endl;
    }
};
```

```
void testPolymorphism(Car *car){
    car->drive();
    car->stop();
}
```

```
class Taxi : public Car {
public:
    int money;
    virtual void drive(){
        money++;
        std::cout << "Taxi Drive" << std::endl;
    }
    virtual void stop(){
        std::cout << "Taxi Stop" << std::endl;
    }
};
```

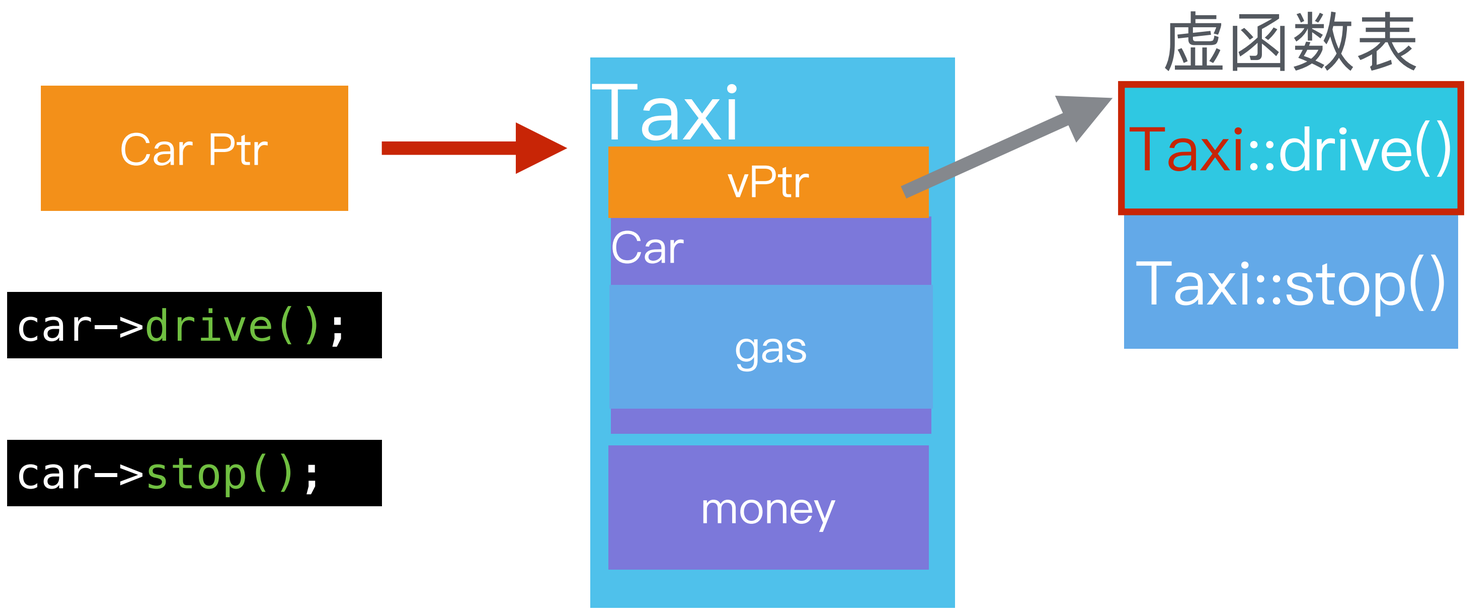
```
Taxi taxi;
Car *car = &taxi;
testPolymorphism(car);
```

OC OOP 之多态性

C++的多态细节

```
class Car {
public:
    int gas;
    virtual void drive(){
        std::cout << "Car Drive" << std::endl;
    }
    void stop(){
        std::cout << "Car Stop" << std::endl;
    }
};
```

```
class Taxi : public Car {
public:
    int money;
    virtual void drive(){
        money++;
        std::cout << "Taxi Drive" << std::endl;
    }
    virtual void stop(){
        std::cout << "Taxi Stop" << std::endl;
    }
};
```




```
car->drive();
```

```
car->stop();
```

```
void testPolymorphism(Car *car){
    car->drive();
    car->stop();
}
```

```
Taxi taxi;
Car *car = &taxi;
testPolymorphism(car);
```

OC OOP 之多态性

有虚函数表的类型强转

```
class SomeClassA {
    int var1;
public:
    virtual void func1();
};

class SomeClassB {
    int var2;
public:
    virtual void func2();
};
```

```
void SomeClassA::func1() {
    std::cout << __FUNCTION__ << std::endl;
}

void SomeClassB::func2() {
    std::cout << __FUNCTION__ << std::endl;
}
```

?

```
void testCpp2() {
    SomeClassA a = SomeClassA();
    SomeClassB *b = reinterpret_cast<SomeClassB *>(&a);
    b->func2();
}
```

Q&A

感谢聆听