

冰凌天 Lv2

2018年08月06日 阅读 126

[关注](#)

## 小码哥iOS学习笔记第三天: isa和superclass

- Objective-C中的对象, 主要可以分为3种
  - instance: 实例对象, 包含 isa和其他成员变量的值, ...
  - class: 类对象, 包含, isa、superclass、属性、对象方法、协议、成员变量的描述, ...
  - meta-class: 元类对象, 包含 isa、superclass、类方法, ...
- 可以用下图表示每种对象中包含的内容



### 一、准备代码

- 准备两个类, `Person` 类继承自 `NSObject`, `Student` 类继承自 `Person`, 具体如下:

```
@interface Person : NSObject <NSCopying> {
    int _age;
}

@property (nonatomic, assign) double height;
- (void)personInstanceMethod;
+ (void)personClassMethod;
@end

@implementation Person
```

ObjectiveC

```

- (void)personInstanceMethod {}
+ (void)personClassMethod {}

- (id)copyWithZone:(nullable NSZone *)zone {
    return nil;
}
@end

@interface Student : Person {
    int _no;
}
@property (nonatomic, assign) double weight;
- (void)studentInstanceMethod;
+ (void)studentClassMethod;
@end

@implementation Student
- (void)studentInstanceMethod {}
+ (void)studentClassMethod {}
@end

```

- **Person** 类中, 主要包含以下内容
  - **int** 类型的成员变量 **\_age**
  - **double** 类型的属性 **height**
  - 一个实例方法 **- (void)personInstanceMethod**
  - 一个类方法 **+ (void)personClassMethod**
  - 遵守 **NSCopying** 协议, 并实现 **- (id)copyWithZone:(nullable NSZone \*)zone** 方法
- **Student** 类中, 主要包含以下内容
  - **int** 类型的成员变量 **\_no**
  - **double** 类型的成员变量 **weight**
  - 一个对象方法 **- (void)studentInstanceMethod**
  - 一个类方法 **+ (void)studentClassMethod**

## 二、验证OC调用方法是发送消息机制

- OC中方法的调用, 是通过发送消息机制实现的, 我们可以查看底层代码来验证
- 创建 **Person** 类的对象, 调用方法 **personInstanceMethod** 方法

```
Person *person = [[Person alloc] init];  
[person personInstanceMethod];
```

ObjectiveC

- 当前的代码如下图:

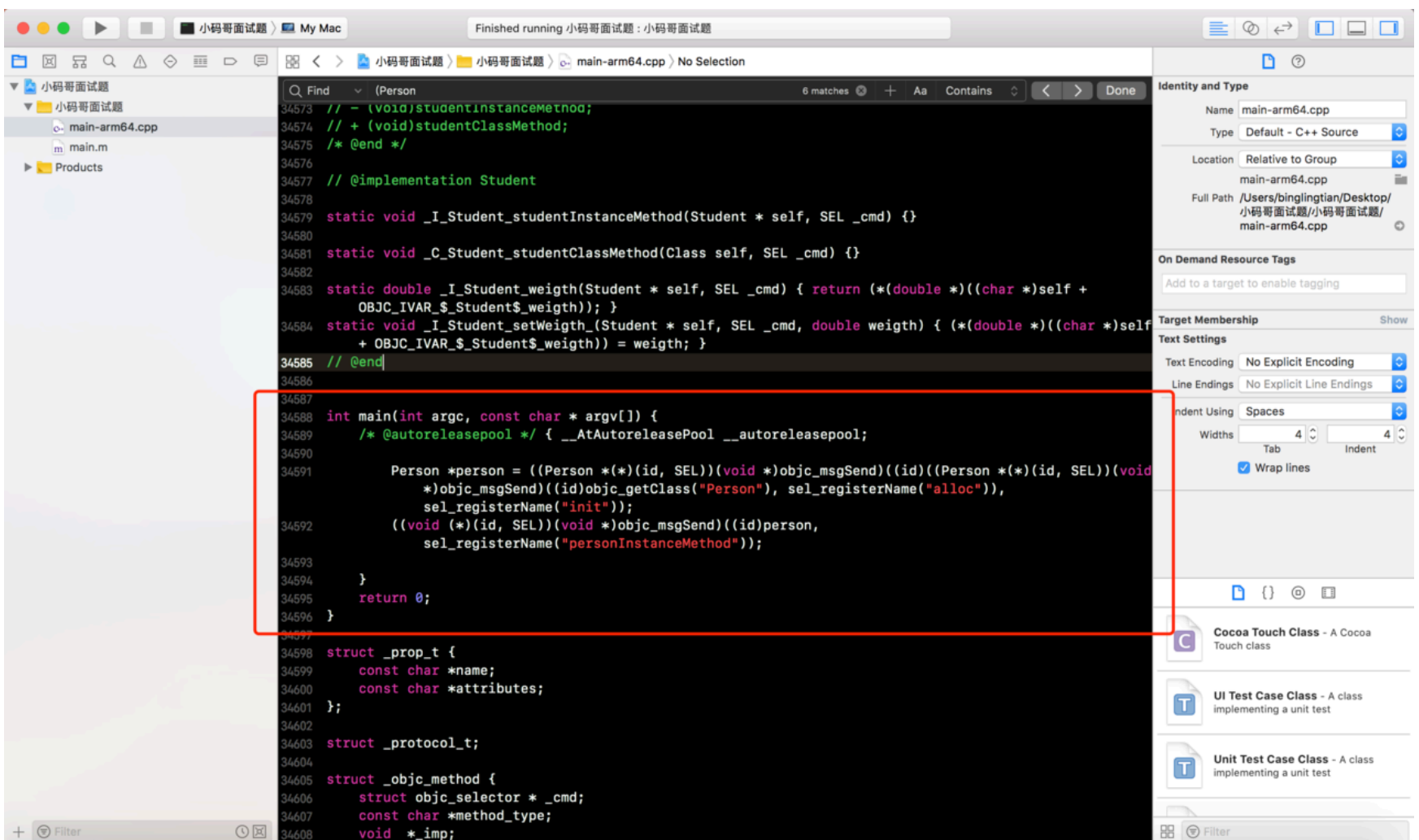
- 我们使用终端 `cd` 到 `main.m` 的文件中, 并执行命令:

```
xcrun -sdk iphoneos clang -arch arm64 -rewrite-objc main.m -o main-arm64.cpp
```

ObjectiveC

- 将生成的 `main-arm64.cpp` 文件拖到当前工程中:

- 可以在 `main-arm64.cpp` 文件中, 找到如下代码:



- 可以看到有一句代码:

CPP  
`((void (*)(id, SEL))(void *)objc_msgSend)((id)person, sel_registerName("personInstanceM`

- 去掉类型转换后:

ObjectiveC

```
objc_msgSend(person, sel_registerName("personInstanceMethod"));
```

- 即: OC 中调用方法, 在底层就是发送一条消息

### 三、对象的isa指针是指向哪里?

问: 对象的isa指针是指向哪里?

答: instance的isa, 指向class类对象, 类对象的isa指向meta-class元类对象



- 以 `NSObject` 对象为例
  - `NSObject` 的实例对象: `isa` 指向 `NSObject` 的 `class`类对象
  - `NSObject` 类对象: `isa` 指向 `NSObject` 的 `meta-class` 元类对象
  - `NSObject` 的 `meta-class` 元类对象: `isa` 指向 `NSObject` 的 `meta-class` 元类对象
- 以 `Person` 对象为例
  - `Person` 的实例对象: `isa` 指向 `Person` 的 `class`类对象
  - `Person` 类对象: `isa` 指向 `Person` 的 `meta-class` 元类对象
  - `Person` 的 `meta-class` 元类对象: `isa` 指向 `NSObject` 的 `meta-class` 元类对象
- 以 `Student` 对象为例
  - `Student` 的实例对象: `isa` 指向 `Student` 的 `class`类对象
  - `Student` 类对象: `isa` 指向 `Student` 的 `meta-class` 元类对象
  - `Student` 的 `meta-class` 元类对象: `isa` 指向 `NSObject` 的 `meta-class` 元类对象

- `instance` 的 `isa` 指向 `class`
  - 当调用对象方法时, 通过 `instance` 的 `isa` 找到 `class`, 最后找到对象方法的实现进行调用
- `class` 的 `isa` 指向 `meta-class`
  - 当调用类方法时, 通过 `class` 的 `isa` 找到 `meta-class`, 最后找到类方法的实现进行调用
- `meta-class` 的 `isa` 指向基类的 `meta-class`

#### 四、class对象的 `superclass` 指针

- 类对象的 `superclass` 指针, 指向父类的类对象



那么 `class` 中的 `superclass` 指针的作用是什么呢?

- 现有如下代码:

```
Student *student = [[Student alloc] init];
[student studentInstanceMethod];
[student personInstanceMethod];
[student init];
```

ObjectiveC

- 当 `student` 调用对象方法 `studentInstanceMethod` 时, 会有以下查找方式
  - 通过 `instance` 的 `isa` 找到 `Student` 的 `class` 类对象, 查看是否有 `studentInstanceMethod`
  - 发现 `Student` 的 `class` 对象中有 `studentInstanceMethod` 方法, 停止继续查找, 通过消息机制调用方法

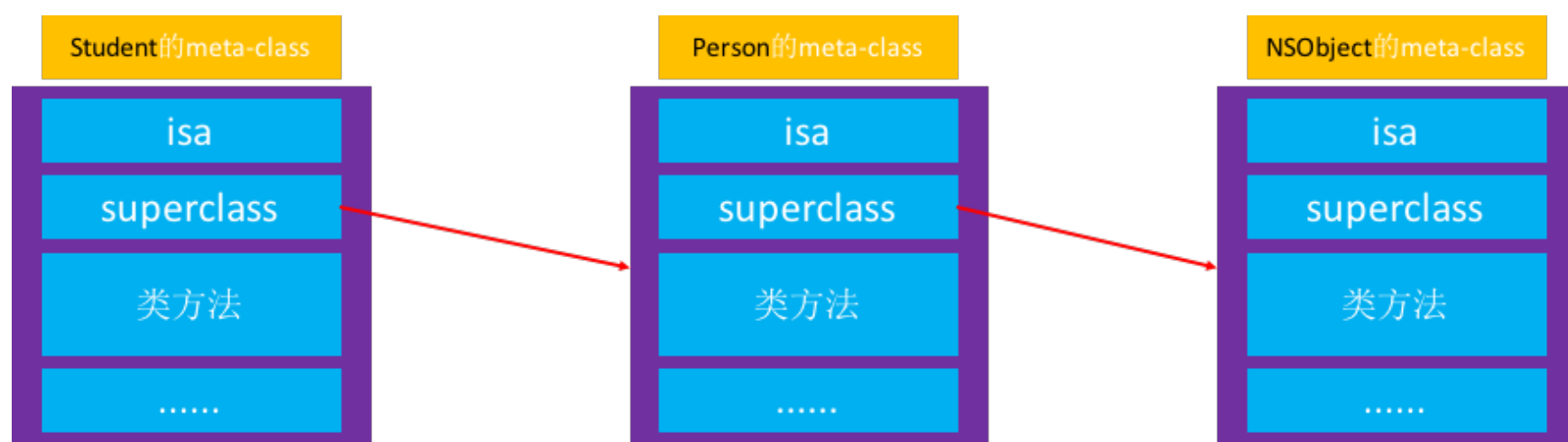
- 当 `student` 调用父类 `Person` 的对象方法 `personInstanceMethod` , 会有以下查找方式
  - 通过 `student` 的 `isa` 找到 `Student` 的 `class` 类方法, 查看是否有 `personInstanceMethod`
  - 发现 `Student` 的 `class` 对象中没有 `personInstanceMethod` , 于是通过 `superclass` 指针找到 `Person` 的 `class` 对象, 查看是否有 `personInstanceMethod`
  - 在 `Person` 的 `class` 对象中发现 `personInstanceMethod` 方法, 停止继续查找, 通过消息机制调用方法
- 当 `student` 调用基类 `NSObject` 的 `init` 方法时, 会有以下查找方式
  - 首先, 通过 `isa` 指针找到 `Student` 的 `class` 对象, 查看是否有 `init` 方法
  - 在 `Student` 的 `class` 对象中没有发现 `init` 方法, 于是通过 `superclass` 指针找到 `Person` 的 `class` 对象
  - 在 `Person` 的 `class` 中查找 `init` 方法, 结果发现 `Person` 的 `class` 对象中也没有 `init` 方法
  - 此时, 就会通过 `Person` 的 `class` 对象中的 `superclass` 指针, 找到 `NSObject` 的 `class` 对象中, 查找 `init` 方法
  - 在 `NSObject` 的 `class` 对象中, 找到了 `init` 方法, 停止继续查找, 通过消息机制调用方法

注意: `class` 的 `superclass` 会指向父类的 `class` 对象, 最后指向的是 `NSObject` 的 `class` 对象, 而 `NSObject` 的 `class` 对象中的 `superclass` 指针, 会指向 `nil`

如果在发现 `NSObject` 的 `class` 中也没有找到要调用的方法时, 就会报错 `unrecognized selector sent to instance`

## 五、`meta-class` 中的 `superclass` 指针

- 与类对象的 `superclass` 指针类似, `meta-class` 中的 `superclass` 指针指向父类的 `meta-class`



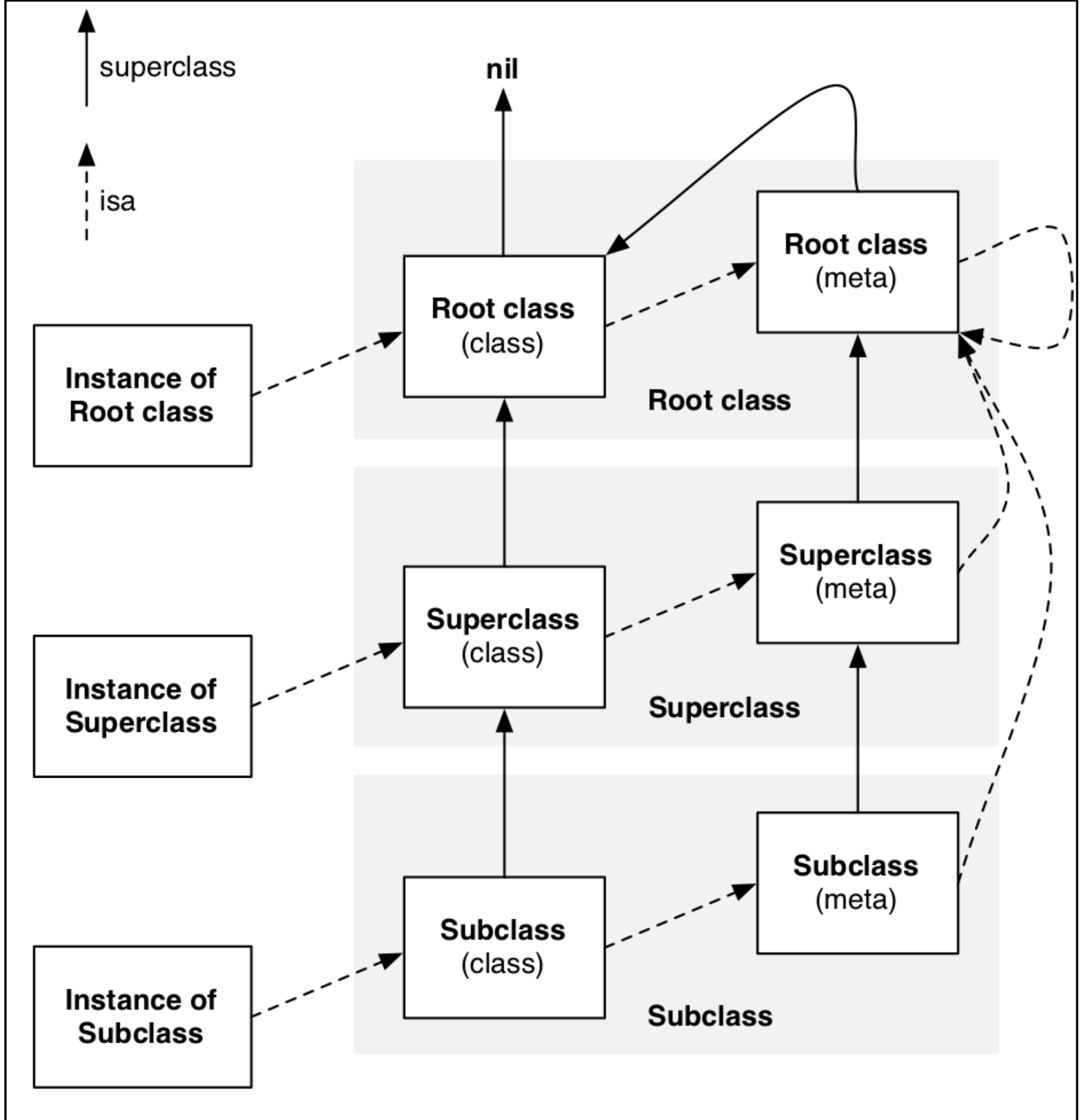
- 在类方法的调用上, 与实例方法调用类似
- 当 `Student` 的 `class` 要调用 `Person` 的类方法时, 会先通过 `isa` 找到 `Student` 的 `meta-class`, 然后通过 `superclass` 找到 `Person` 的 `meta-class`, 最后找到类方法的实现进行调用

注意: 基类 `NSObject` 的 `meta-class` 对象的 `superclass` 最终指向的是 `NSObject` 的 `class` 对象, 而不是指向 `nil`

## 六、isa、superclass总结

- isa、superclass的作用如下图:





- `instance` 的 `isa` 指向 `class`
- `class` 的 `isa` 指向 `meta-class`
- `meta-class` 的 `isa` 指向基类的 `meta-class`
- `class` 的 `superclass` 指向父类的 `class`
  - 如果没有父类, `superclass` 指针为 `nil`
- `meta-class` 的 `superclass` 指向父类的 `meta-class`

- 基类的 `meta-class` 的 `superclass` 指向基类的 `class`
- `instance` 调用对象方法的轨迹
  - `isa` 找到 `class`, 方法不存在, 就通过 `superclass` 找父类
- `class` 调用类方法的轨迹
  - `isa` 找 `meta-class`, 方法不存在, 就通过 `superclass` 找父类

## 七、验证NSObject的Meta-class对象中的superclass指向自身的Class对象

- 上面提到过: 基类的 `meta-class` 的 `superclass` 指向基类的 `class`
- 下面通过代码来进行验证

ObjectiveC

```
@interface Person: NSObject
+ (void)test;
@end
@implementation Person
+ (void)test {
    NSLog(@"+ [Person test] - %p", self);
}
@end
```

- `Person` 类继承自 `NSObject`, 有一个类方法 `+ (void)test`, 并给出了方法的实现

ObjectiveC

```
NSLog(@"[Person class] - %p", [Person class]);
[Person test];

// 控制台打印:
[Person class] - 0x100001170
+ [Person test] - 0x100001170
```

- 根据打印可以知道, 调用方法的正是 `Person` 类的 `Class` 对象
- 现在删除 `test` 方法的实现部分, 只保留声明部分

ObjectiveC

```
@interface Person: NSObject
+ (void)test;
@end
```

```
@implementation Person
@end
```

- 再次调用该方法, 会报出运行时错误, `test` 方法不存在

ObjectiveC

```
[Person test];
// 报错: '+[Person test]: unrecognized selector sent to class 0x100001130'
```

- 我们在给 `NSObject` 添加一个分类, 实现 `+(void)test` 方法

ObjectiveC

```
@interface NSObject (Test)
+ (void)test;
@end
@implementation NSObject (Test)
+ (void)test
{
    NSLog(@"+ [NSObject test] - %p", self);
}
@end
```

- 再次使用 `Person` 的类对象调用 `test` 方法

ObjectiveC

```
NSLog(@"[Person class] - %p", [Person class]);
[Person test];
// 控制台打印:
[Person class] - 0x100001220
+ [NSObject test] - 0x100001220
```

- 此时的调用顺序是:
  - 1、`Person` 的类对象, 通过 `isa` 找到了 `Person` 的元类对象, 并查找有没有 `test` 方法
  - 2、由于 `Person` 的元类对象中没有 `test` 方法, 于是通过 `superclass` 找到了父类 `NSObject` 的元类对象
  - 3、在 `NSObject` 的元类对象中, 发现了 `test` 方法, 发送消息, 调用方法
- 接着我们移除掉 `NSObject` 分类中的 `+(void)test` 方法的实现

ObjectiveC

```
@interface NSObject (Test)
+ (void)test;
@end
```

```
@implementation NSObject (Test)
@end
```

- 此时再次使用 `Person` 调用 `+(void)test` 方法, 就会报运行时错误

```
[Person test];
reason: '+[Person test]: unrecognized selector sent to class 0x100001178'
```

- 接着在 `NSObject` 的分类中, 给出一个对象方法 `-(void)test` 的方法实现

```
@interface NSObject (Test)
+ (void)test;
@end
@implementation NSObject (Test)
- (void)test {
    NSLog(@"- [NSObject test] - %p", self);
}
@end
```

ObjectiveC

- 再次使用 `Person` 调用类方法 `+(void)test`

```
NSLog(@"[Person class] - %p", [Person class]);
[Person test];
// 控制台打印:
[Person class] - 0x1000011b8
- [NSObject test] - 0x1000011b8
```

ObjectiveC

- 此时调用成功, 说明当 `NSObject` 的元类对象中没有 `test` 方法时, 就会通过 `superclass` 指针找到 `NSObject` 的类对象, 并查找有没有 `test` 方法
- 由于在 `NSObject` 中找到了 `test` 方法, 所以会直接调用

关注下面的标签, 发现更多相似文章

iOS

Objective-C



冰凌天 Lv2 iOS

获得点赞 189 · 获得阅读 9,709

关注

## 安装掘金浏览器插件

打开新标签页发现好内容，掘金、GitHub、Dribbble、ProductHunt 等站点内容轻松获取。快来安装掘金浏览器插件获取高质量内容吧！

### 评论

输入评论...

### 相关推荐

**荐** · 知识小集 · 1小时前 · iOS  
**专栏**

iOS 13 正式发布，来看看有哪些 API 变动

 6 

**专栏**iOS桃子 · 13小时前 · Objective-C

iOS | 面试知识整理 - OC底层 (三)

 4 

**专栏**CainLuo · 1天前 · iOS / Swift

UISplitViewController简单入门

 3 

**专栏**QiShare · 17小时前 · iOS / Swift

iOS 高德SDK应用实践（三）—— 自定义气泡CalloutView

 2 

**专栏**凉介 · 1天前 · iOS

iOS卡顿监测方案总结

 7 

**专栏**littleliang · 18小时前 · iOS

WCCgiMock——客户端模拟网络回包工具介绍

 2 

专栏 Henry\_Jeannie · 12小时前 · Objective-C


## iOS -- Autorelease & AutoreleasePool

 1



专栏 Binboy\_王兴彬 · 4天前 · iOS / Swift

## 一份 iOS 开发实践检查清单

 54

 6

专栏 OneAlon · 1天前 · iOS

## 从YYImage源码中学习如何处理图片显示

 7



专栏 QiShare · 9天前 · iOS / Apple

## 2019苹果秋季新品发布会速览

 22

 18