**泛型：**iOS9新特性，限制类型／迎合swift

使用情景：1.在集合中（数组、字典、nsset）

2.在声明一个类，类里面的某些属性或方法的类型不确定，在使用这个类的时候才确定,就可以采用泛型

eg：@interface Person<ObjectType> :NSObject

@property (nonatomic) ObjectType language;

@end

作用：1.提高开发规范，减少程序员之间的交流

2.通过集合取出来的对象，直接当作泛型对象使用，可以直接使用点语法，而id类型不可以用点语法

eg: NSMutableArray<NSString \*> \*array = [[NSMutableArray alloc] init];

array[0].length 不使用泛型，不能用.length

注意：泛型只能修饰方法的调用

\_\_covariant(协变):子类转父类

\_\_contravariant(逆变):父类转子类

**\_\_kindof：**iOS9新特性，表示当前类或则它的子类

+(\_\_kindof Person \*)person;---表示返回值可以是person类或它的子类

返回值用id坏处：不能在编译的时候检查真实类型，而且返回值没有提示

返回值使用instancetype会自动识别当前对象的类，但也没有提示

**runtime运行时**

**#import <objc/message.h>**

oc就是运行时机制，对于oc的函数，属于动态调用过程，在编译的时候并不能决定真正调用哪个函数，只有在真正运行的时候才会根据函数的名称找到对应的函数来调用。

作用：

1.发送消息(方法调用的本质，就是让对象发消息)

objc\_msgSend(p,@selector(eat));

用类名调用类方法，底层会自动把类名转换成成员变量调用

objc\_msgSend([Person class],@selector(eat));

消息机制原理：对象根据方法编号SEL去映射表查找对应的方法实现。

2.交换方法的实现（在分类的+load方法交换）

class\_getClassMethod(获取类方法)

class\_getInstanceMethod(获取对象方法)

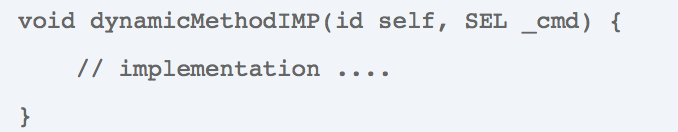
method\_exchangeImplementations (交换方法实现)

3.动态添加方法

使用场景：如果一个类方法非常多，加载类到内存的时候也比较耗费资源，需要给每个方法生成映射，因此可以使用动态给某个类添加方法解决

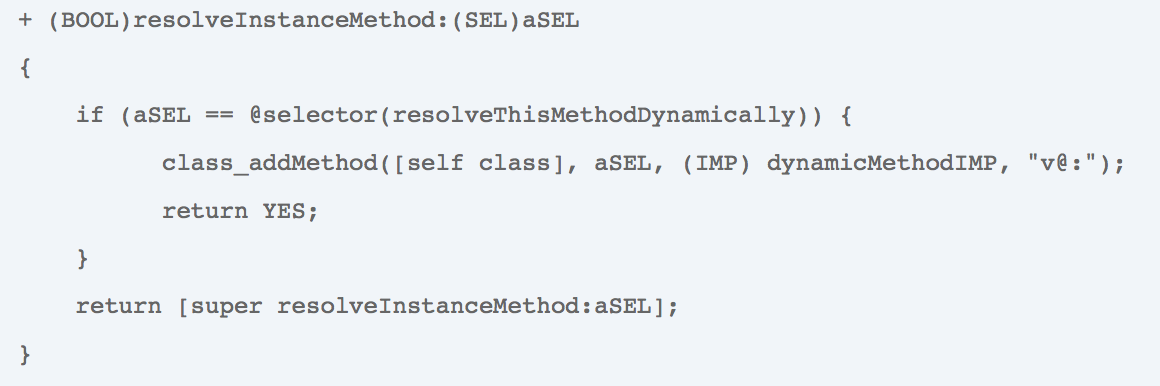
IMP:方法实现或函数入口或函数名

a C function that take at least two arguments—self and \_cmd



you can dynamically add it to a class as a method

(called resolveThisMethodDynamically) using resolveInstanceMethod: like this:



使用performselector动态添加方法，如果调用没有实现的方法，就会调用+resolveInstanceMethod或+resolveClassMethod，从而知道哪些方法没有实现

4.动态添加属性(实质是跟对象进行关联)

objc\_setAssociatedObject(<#id object#>, <#const void \*key#>, <#id value#>, <#objc\_AssociationPolicy policy#>)

object:给哪个对象添加属性

key：属性名，根据key去获取关联的对象，void \* == id

value:关联的值

policy：策略

objc\_getAssociatedObject(<#id object#>, <#const void \*key#>) --- 根据key值取关联值

5.获取所有属性---字典转模型

unsigned int count = 0;

//获取模型所有成员变量

Ivar \*ivarList = class\_copyIvarList(self, &count);

//获取属性名

Ivar ivar = ivarList[i];

NSString \*name =

[NSString stringWithUTF8String:ivar\_getName(ivar)];

// 处理成员属性名->字典中的key,从第一个角标开始截取,去掉下划线 \_

NSString \*key = [name substringFromIndex:1];

// 根据成员属性名去字典中查找对应的value

id value = dict[key];

//获取属性类型

NSString \*name =

[NSString stringWithUTF8String:ivar\_getName(ivar)];

runtime字典转模型：遍历模型中所有属性，去字典中查找,

都少不了[xx setValue:value forKey:key]这句代码的

kvc应用：字典转模型，遍历字典中所有key，去模型中查找对应的属性名，当字典中的key，在对象属性中找不到，就会报错，只需要重写setValue: forUndefinedKey:即可

kvc

ivar

const(常量)与宏的区别：

1. 编译时刻：宏是预编译（#：预编译），const：编译时刻（com+b）
2. 编译检查：宏不会检查错误，const会检查错误
3. 宏的好处：可以定义代码，
4. 宏的坏处：宏过多，会造成编译时间过长（每次都会编译一次），因此常用的字符串通常用const修饰（只会编译一次）

宏：

1.命名规范：以项目前缀开头，以key结尾

eg:#define XMGAccountKey @”account”

2.常见用法：常用的字符串抽成宏，常用的代码抽成宏

**const：**

1.当有字符串常量的时候，苹果推荐使用const

2.作用：仅仅用来修饰右边的变量（只能修饰变量：基本变量、指针变量、对象变量），而且const修饰的变量，表示只读。

3.使用场景：定义一个全局只读变量，或在方法中定义只读参数

**static：**

1. 作用：

修饰局部变量：延长这个局部变量的生命周期，只要程序运行，局部变量就会一直存在，而且只分配一次内存。

修饰全局变量：只会修改全局变量的作用域，表示只能是当前文件内使用

**extern：**

1. 作用：声明一个变量，不能定义变量

注意：extern修饰的变量不能初始化

2.使用场景：一般用于声明全局变量

static与const联合使用：

static NSString \* const CellID = @"cell";(表示当前字符串只能在本文件夹使用，并且只读)

extern与const联合使用：声明全局变量。

Class:获取方法调用的类名

Superclass:获取方法调用者的父类类名

Super：编译修饰符，不是指针，是指向父类标志，本质还是拿到**当前对象**去调用父类的方法

[super class] --- sonPerson

[slef class] --- sonPerson

[self superclass] --- Person

多控制器：当有很多控制器，交给一个大控制管理

父子控制器：UINavigationCont,UITabBarCont,

本质：搞一个控制器容器，管理很多子控制器

控制器不存在，但是控制器的view也是可以存在的，比如控制器1的view addsubview 控制器2的view上，那么1的view就对2的view有强引用，即使2的控制器销毁了，但2但view还没有销毁，这样会导致view2 上的所有事件都不能处理，因为界面事件有控制器决定的

注意：1、如果一个控制器的view显示，那么这个控制器也必须要存在，否则事件不会响应。2、dismissViewControllerAnimated:首先会判断自己是不是modal出来的，如果是就modal自己，如果不是再去找父控制器有没有modal，如果modal，就会dismiss父控制器。

kvo:ios11前添加的观察者要移除，否则会崩溃，ios11后可以不用移除，但建议移除

NSobject分类不能重写dealloc方法，但是其他的分类可以，因为NSobject没有super，它是基类

Storyboard或xib报错：setvalueforundefinedkey是因为属性删除了，而连线还在（显示为！号）

Block开发中的使用场景

1. 把block保存到对象中，恰当时候才调用
2. 把block当作方法的参数使用，外界不调用，都是方法内部去调用，block实现交给外界决定
3. 把block当作方法的返回值，目的就是为了代替方法，block交给内部实现，外界不需要知道block怎么实现，只调用，重写setter方法

ReactiveCocoa简介

简称RAC，有github开源的一个应用于ios和os开发的新框架，Cocoa是苹果整套框架的简称，因此很多苹果框架喜欢以cocoa结尾。

RAC作用：RAC为事件提供了很多处理方法，而且利用RAC处理事件很方便，可以把要处理的事情和监听的事情的代码放在一起，这样非常方便我们管理，就不需要跳到对应的方法里，非常符合我门开发的“高内聚、低耦合”的思想。

RAC常见类：

1. RACSignal(信号类): 表示当数据改变时，信号内部会发出数据，它本身不具备发送信号的能力，而是交给内部一个订阅者去发出，默认一个信号都是冷信号，也就是值改变了，也不会触发，只有订阅了这个信号，这个信号才会变为热信号，值改变了才会触发。

执行流程：创建信号（创建信号，保存didsubscribe block）---订阅信号（创建订阅者，保存nextblock，执行didsubscribe）---发送信号（sendNext发送数据，拿到订阅者执行nextblock）【只要订阅者调用sendnext，就会执行nextblock；只要RACDynamicSingal 信号被订阅，就会执行didsubscribe】

1. RACSubscriber：表示订阅者的意思，用于发送信号，这是一个协议，不是一个类，只要遵守这个协议，并且实现方法才能成为订阅者
2. RACDisposable`:用于 理资源，当信号发送完成或者发送错误的时候，就会自动触发它。【使用场景:不想监听某个信号时，可以通过它主动取消订阅信号】
3. RACSubject:信号提供者，自己可以充当信号，又能发送信号。【使用场景:通常用来代替代理，有了它，就不必要定义代理了】

执行流程：创建信号（创建信号，创建空数组）---订阅信号（创建订阅者，把订阅者保存到数组中，而且保存nextblock）---发送信号（遍历数组，获取所有订阅者，执行他们对应的nextblock）【RACSubject被订阅，仅仅是保存订阅者；RACSubject发送数据，遍历所有的订阅者，调用他们的nextBlock】

1. RACReplaySubject`:重复提供信号类，RACSubject的子类【使用场景一:如果一个信号每被订阅一次，就需要把之前的值重复发送一遍，使用重复提供信号类。使用场景二:可以设置capacity数量来限制缓存的value的数量,即只缓充最新的几个值。】

执行流程：创建信号（创建信号，创建2个空数组）---订阅信号（创建订阅者，遍历valuesReceived数组，如果有订阅者就执行sendnext，同时把订阅者保存到另外一个数组中，而且保存nextblock）---发送信号（保存值到valuesReceived，同时调用父类的sendnext发送数据（遍历数组，获取所有订阅者，执行他们对应的nextblock））

`RACReplaySubject`与`RACSubject`区别:

RACReplaySubject可以先发送信号，再订阅信号

RACSubject就不可以。

6.`RACSequence`:RAC中的集合类，用于代替NSArray,NSDictionary,可以使用它来快速遍历数组和字典。

1）遍历数组/字典

[dict.rac\_sequence.signal subscribeNext:^(id x) {

NSLog(@"%@",x);

（字典的时候x为RACTuple元祖对象）

// 解包元组，用专用的宏RACTupleUnpack（）

//会把元组的值，按顺序给参数里面的变量赋值

RACTupleUnpack(NSString \*key,NSString \*value) = x;

}];

2）字典转模型

NSMutableArray \*flags = [NSMutableArray array];

[dictArr.rac\_sequence.signal subscribeNext:^(id x) {

// 运用RAC遍历字典，x：字典

FlagItem \*item = [FlagItem flagWithDict:x];

[flags addObject:item];

}];

高级写法：

NSArray \*flags =

[[dictArr.rac\_sequence map:^id(id value) {

// map:映射的意思，目的：把原始值value映射成一个新值

//array: 把集合转换成数组

// 底层实现：当信号被订阅，会遍历集合中的原始值，映射成新值，并且保存到新的数组里。

return [FlagItem flagWithDict:value];

}] array];

7. `RACMulticastConnection`:用于当一个信号，被多次订阅时，为了保证创创建信号时，避免多次调用创建信号中的block，造成副作用，可以使用这个类处理

执行流程：创建信号（创建信号RACDynamicSingal,保存didsubscribe block）---创建连接（）---订阅连接信号（RACSubject）---保存订阅者（RACSubscriber及对应的nextblock）---连接---订阅源信号---执行didsubscribeblock---发送数据---遍历RACSubject里面所有订阅者，执行nextblock

8. `RACCommand`:RAC中用于处理事件的类，可以把事件如何处理,事件中的数据如何传递，包装到这个类中，他可以很方便的监控事件的执行过程。

`使用场景`:监听按钮点击，网络请求

执行流程：创建信

9. ReactiveCocoa操作方法之组合。

1)`concat`:按一定顺序拼接信号，当多个信号发出的时候，有顺序的接收信号

注意：第一个信号必须发送完成([subscriber sendCompleted];)，第二个信号才会被激活

2) `then`:用于连接两个信号，当第一个信号完成，才会连接then返回的信号,

注意：使用then之前信号的值（第一个信号的值）会被忽略掉

3）`merge`:把多个信号合并为一个信号，任何一个信号有新值的时候就会调用.

4）`zipWith`:把两个信号压缩成一个信号，只有当两个信号同时发出信号内容时，并且把两个信号的内容合并成一个元组，才会触发压缩流的next事件

5）`combineLatest`:将多个信号合并起来，并且拿到各个信号的最新的值,必须每个合并的signal至少都有过一次sendNext，才会触发合并的信号

RAC开发中常见的用法：

1. `RACSubject替换代理`：在view中懒加载创建信号（\_subject = [RACSubject subject];），在view点击事件发送信号（[self.customView.subject sendNext:@"1"];），在控制器中订阅信号（[subject subscribeNext:^(id x) {}]）

`rac\_signalForSelector`：监听对象有没有调用某方法，用于代理时传不了值，只能用第一种方法

eg:[[customView rac\_signalForSelector:@selector(btnClick:)] subscribeNext:^(id x) {

//监听customview到btnclick方法是否调用

}]

1. `代替kvo`：

方法1:

[[ customview rac\_valuesForKeyPath:@”” observer：nil] subscribNext:^(id x) {

//x为修改后的值，

//oberver：nil，因为不需要再调用方法

}]；

方法2:

[customview rac\_observeKeyPath:@”” option:……{

//监听和处理代码写在一块，而且有多个监听时候，少了判断

}]

1. `监听事件`：rac\_signalForControlEvents
2. `代替通知`：rac\_addObserverForName
3. `监听文本框文字改变`：rac\_textSignal
4. 处理当界面有多次请求时，需要都获取到数据时，才能展示界面：

`rac\_liftSelector:withSignalsFromArray:Signals`:当传入的Signals(信号数组)，每一个signal都至少sendNext过一次，就会去触发第一个selector参数的方法

使用注意：几个信号，参数一的方法就几个参数，每个参数对应信号发出的数据。

RAC常用的宏：

1. `RAC(TARGET, [KEYPATH, [NIL\_VALUE]])`:用于给某个对象的某个属性绑定信号，只要产生信号内容，就会把内容给属性复制。

// 只要文本框文字改变，就会修改label的文字

（响应式编程）

RAC(self.labelView,text) = \_textField.rac\_textSignal;

1. `RACObserve(self, name) `:监听某个对象的某个属性,返回的是信号

[RACObserve(self.view, center) subscribeNext:^(id x) {

NSLog(@"%@",x);

}];

1. `@weakify(Obj)和@strongify(Obj)`,一般两个都是配套使用,解决循环引用问题.
2. `RACTuplePack`：把数据包装成RACTuple（元组类）

// 把参数中的数据包装成元组

RACTuple \*tuple = RACTuplePack(@10,@20);

1. `RACTupleUnpack`：把RACTuple（元组类）解包成对应的数据。

RACTupleUnpack(NSString \*name,NSNumber \*age) = tuple; //把元组的值，按顺序给参数里面的变量赋值

链式编程特点： 方法的放回值是block，block必须有返回值（本身对象），block参数（需要操作的值）

响应式编程：（Reactive Programing）

KVO底层实现：

1.addOber时会自动创建一个Person类的子类NSKVONotifying\_ Person

2. 使当前对象的isa指向新的类，就会调用新类的set方法

3. 重写NSKVONotifying\_Person的setAge方法，每次调用，就调用观察者的observeValueForKeyPath方法4. 使用运行时让当前对象关联观察者这个属性。

kvo的本质就是监听一个对象有没有调用set方法

函数式编程（Functional Programing）

函数式编程思想：是把操作尽量写成一系列嵌套的函数或方法调用。

本质：就是往方法传入block，方法中嵌套block调用，把代码聚合起来管理

特点：每个方法必须有返回值（本身对象），把函数或则bolck当做参数block参数（需要操作的值）block放回值（操作的结果）

代表：ReactiveCocoa

RAC结合了函数式编程和响应式编程，所有被描述为函数响应式编程（FRP）框架

以后你使用RAC解决问题，就不需要考虑调用顺序，直接考虑结果（响应式编程），把每一次操作都写成一系列嵌套的方法中，使代码高聚合，方便管理（函数式编程）。

get和post请求区别。1。数据在head ，限制2k

面向协议思想：要想让一个类有多种功能，可以让这个类继承某个类，然后遵循另外一个类的协议。

MVVM---M：模型 V：视图+控制器 VM：视图模型（处理界面上所有业务逻辑，每个控制器对应一个vm模型， ）

MVC---M：模型 V：视图 C：控制器

MVCS---M：模型 V：视图 C：控制器 S：服务类

VIPER---V：视图 I：交互器 P：展示器 E：实体（模型） R：路由