iOS基础/计算机基础

- 字节/字符
- int
  - int的取值范围
    - 32位和64位
      - int = 4byte = 4*8bit = 32bit
      - -2^(32-1) ~ 2^(32-1)-1
        - -2147483648－2147483647
    - 16位
      - int = 2byte = 2*8bit = 16bit
      - -2^(16-1) ~ 2^(16-1)-1
  - 计算
- load
  - load方法：对于运行期系统的每个类（class）及分类（category），都必调用且仅调用一次。
    基本调用顺序就是，执行当前类的load方法，必定先调用super的load方法，如果代码依赖其他程序库，那么程序库中得相关类的load方法也会先执行。类load方法执行完成后，会执行相应的分类的load的方法。
    但是如果在一个程序库中得若干个class，却没法确定load顺序，所以load方法不建议使用其他class。
    load方法在调用的时候，app会阻塞，所以一定要轻！
    应用程序会阻塞并把所有类的load方法都执行完，才能继续。
    load方法直接使用函数内存地址的方式(*load_method)(cls, SEL_load)调用的，而不是使用发送消息 objc_msgSend 的方式。
    因此：load方法不遵循继承规则：如果一个class本身没有实现load方法，那么无论其各级super是否实现load，都不会调用；
    同时：当一个类和它的分类都实现了 +load 方法时，两个方法都会被调用。
    所以：像method swizzling这种操作会放在load里面进行。
    load方法调用时，程序甚至没有autoreleasepool
    initialize方法：首次使用该class前调用，且只有1次。
    和load方法相比，他是“lazy”调用的，也就是类或它的子类收到第一条消息之前被调用的，这里所指的消息包括实例方法和类方法的调用。。
    可以安全使用任何class，因为它们都已经load过了。
    initialize方法执行一定在thread-safe environment，也就是说会阻塞所有线程。
    initialize会走和objc_msgSend一样的消息发送原则，也就是会有覆盖。
    实际编译器调用时，父类的会优先于子类调用。
- 警告
  - 消除方法弃用警告

    ```
    #pragma clang diagnostic push
    #pragma clang diagnostic ignored "-Wdeprecated-declarations"
    // 这里写出现警告的代码
    #pragma clang diagnostic pop
    ```

- int 和 NSInteger
  - 应使用NSUInteger，而非int。这样做的是基于64-bit 适配考虑
- 规范

```
// .h文件
// http://weibo.com/luohanchenyilong/
// https://github.com/ChenYilong
// 第二种修改方法（基于第一种修改方法的基础上）

typedef NS_ENUM(NSInteger, CYLSex) {
    CYLSexMan,
    CYLSexWoman
};

@interface CYLUser : NSObject<NSCopying>

@property (nonatomic, copy, readonly) NSString *name;
@property (nonatomic, assign, readonly) NSUInteger age;
@property (nonatomic, assign, readwrite) CYLSex sex;

- (instancetype)initWithName:(NSString *)name age:(int)age sex:(CYLSex)sex;
- (instancetype)initWithName:(NSString *)name age:(int)age;
+ (instancetype)userWithName:(NSString *)name age:(int)age sex:(CYLSex)sex;

@end
```

- NSArray使用copy，NSMutableArray 使用strong
  - 浅复制和深复制
- #import不会重复导入，
- #include 导入 c，c++头文件
- @class运行时导入,保证编译可以通过,避免循环引用
- Objective-C中堆
  - 存放oc对象，需要手动申请和释放内存(ARC无需手动释放)
- Objective-C中栈
  - 栈由系统自动分配，一般存放非oc对象的基本类型数据，例如int，float，不需要手动管理
- self
  - self调用方法时，会优先在当前类的方法列表中寻找方法，没有再去父类(.h声明)找
- super
  - 直接去父类找方法
    - 注意 [self class]; 和 [super class]; 返回都是当前子类，区别是寻找方法时是否在当前子类寻找
- const
  - 确定后值不能修改，定义该 const 变量时，通常需要对它进行初始化，因为以后就没有机会再去改变它了
  - //声明一个int类型的变量a，设置后不能再修改(下面两种方式等价)
    int const a = 10;
    const int a = 10;
  - 判断p 和p是只读还是变量，关键是看const在谁前面。如果只在p前面，那么p只读，p还是变量；如果在p前面,那么p只读，p变量 ：
  - int const *p // *p只读 ;p变量
    int * const p // *p变量 ; p只读
    const int * const p //p和*p都只读
    int const * const p //p和*p都只读
- volatile
  - 可能被意想不到的修改，因此优化器在用到这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份
- static
  - static标记的变量会存储到全局变量区，生命周期和程序相同
  - 节省内存。静态变量只存储一处，但供所有该类的对象使用
  - 它的值是可以更新的（修改后所有访问都是修改后的）
  - .m内声明：只在该类内部使用，.h声明:所有子类或者导入类可使用
- extern
  - 在.h声明，作用域是整个程序：比如
    - //.h中声明一个字符串常量
      extern NSString * const testName;
    - //.m中实现
      NSString * const testName = @"小明";
- 属性
  - @property (nonatomic,strong) UITextField   *username;
  - 可以使用“.”语法访问变量
  - 类属性：
    - 编译器会自动生成set和get方法
  - 分类属性：
    - 默认没有set和get方法
      - 需要runtime进行方法绑定,手动实现set,get方法，但是其他方法也要实现,不然不要调用
- id/instancetype类型区别？
  - id：编译时无法确认对象真实类型，只有等到运行时才能确认
  - instancetype：
    - 编译时就能确认对象真实类型时使用
    - 比如 类内部自身对象初始化返回
- 实例变量
  - @interface BaseViewController : UIViewController {
        UITextField   *username2;
    }
  - 只能在类内部访问
  - 不能使用“.”语法
  - 需要手动实现set和get方法
- atomic 和 nonatomic
  - atomic原子性，对set方法加锁，不是绝对安全的
  - nonatomic 无锁，效率高
    - 资源竞争需要手动加锁
- 读写控制
  - readonly
    - 只读，编译器只提供get方法
  - readwrite
    - 读写，编译器提供set和get方法
- 内存管理
  - assign
    - 修饰oc对象的基础类型
      - 指针弱引用(引用计数不变)
    - 如果修饰oc非基础对象，不会自动释放指针，从而导致野指针问题，不建议
  - weak
    - 弱引用修饰oc非基础对象
    - weak修饰的变量在销毁后，自动将指针置为nil，避免野指针
    - 避免循环引用时需要weak
      - 常见“delegate，block”
  - retain
    - 修饰oc非基础对象
      - 强引用，引用计数+1
  - strong
    - ARC替代retain
    - storyboard 使用IBOutlet修饰的控件
      - 以前默认是weak,现在是strong
  - copy
    - 只是创建内存地址指针，不会重新创建新对象
      - 新对象引用计数=1
  - mutableCopy
    - 创建新内存地址
      - 新对象引用计数=1
- 类工厂方法
  - 快速创建对象的的类方法，可以直接返回一个初始化好的对象
  - UIButton类中的 + (instancetype) buttonWithType:类工厂方法
- 方法重载
  - 方法重载：方法名称一样,参数名称不一样
  - OC没有方法重载，不允许相同方法,不同参数
    - – (void) useName:(NSString *)name;
    - – (void) useName:(NSString *)name2;
    - 上面两种方法不能存在一个类里
    - OC是消息转发机制，使用useName:来区分方法，无法区分参数名称
- OC继承
  - OC是单继承
    - 多继承需要通过组合，协议，分类实现类似多继承
  - 不能实现多继承?
    - OC的消息机制，名字查找发生在运行时，而不是编译时，不能解决多个基类的二义性
- 分类Category
  - 不破坏已有类的情况下，为该类添加新方法的一种方式
  - 会创建新的类文件，xxx (xxx)形式
  -
    ```
    //分类的定义，结构体
    typedef struct category_t {
        const char *name;//分类名
        classref_t cls;//扩展的类
        struct method_list_t *instanceMethods;//实例方法列表
        struct method_list_t *classMethods;//类方法列表
        struct protocol_list_t *protocols;//协议列表
        struct property_list_t *instanceProperties;//属性列表
    } category_t;
    ```
  - 默认只支持类方法，默认不支持属性方法和属性对象(不会自动创建set和get等方法)
  - Category新建的类方法和系统方法重名,会覆盖系统方法
  - 加载步骤：
    - （1）_objc_init runtime入口函数，初始化
      （2）map_images 加锁
      （3）map_images_nolock 完成类的注册，初始化，及load方法加载
      （4）_read_images 完成类的加载，协议的加载，类别的加载等工作
      （5）remethodizeClass 这一步非常关键，它将类别绑定到目标类上
      （6）attachCategories 这是最重要的一步，将类别中的方法，属性绑定到目标类
      （7）attachLists 将目标类中的方法和分类中的方法放到一个列表中
  - +load 加载
    - 不是继承，所有类都会实现+load方法
    - 加载顺序：父类，子类，分类
    - 通过函数指针调用，runtime运行时调用，较早
    - initialize方法可以继承，是通过消息传递调用的，第一次收到消息时调用，较晚
- 扩展
  - 破坏类结构,直接添加方法和属性
- APP生命周期
  - 1、程序启动：状态由Not running -> Inactive -> Active
    willFinishLaunchingWithOptions
    didFinishLaunchingWithOptions
    applicationDidBecomeActive
    2、点击home键|锁屏：由Active -> Inactive -> Backgroud
    applicationWillResignActive
    applicationDidEnterBackground
    3、重新进入前台：Backgroud -> Inactive -> Active applicationWillEnterForeground
    applicationDidBecomeActive
    4、在前台，双击home键，手动杀掉APP：Active -> Inactive -> Backgroud -> end
    applicationWillResignActive
    applicationDidEnterBackground
    applicationWillTerminate
    5、当URL到达时，如果你的应用没在正在运行，则会被启动并且移到前台运行以打开URL
    application:didFinishLaunchingWithOptions:
    application:openURL:sourceApplication:
    applicationDidBecomeActive
    6、当URL到达时，如果你的应用正在background运行或被suspended，它将会被移到前台以打开URL
    applicationWillEnterForeground
    application:openURL:sourceApplication:
    applicationDidBecomeActive
- UIViewController的生命周期
  - 1、 alloc
    创建对象，分配空间
    2、init (initWithNibName|initWithCoder)
    初始化对象，初始化数据
    3、awakeFromNib
    所有视图的outlet和action已经连接，但还没有被确定。
    4、loadView
    完成一些关键view的初始化工作，加载view。
    5、viewDidLoad
    载入完成，可以进行自定义数据以及动态创建其他控件
    6、viewWillAppear
    视图将出现在屏幕之前
    7、viewWillLayoutSubviews
    将要对子视图进行调整
    8、viewDidLayoutSubviews
    对子视图进行调整完毕
    9、viewDidAppear
    视图已在屏幕上渲染完成
    10、viewWillDisappear
    视图将被从屏幕上移除
    11、viewDidDisappear
    视图已经被从屏幕上移除
    12、dealloc
    视图被销毁，此处需要对你在init和viewDidLoad中创建的对象进行释放