Day12

达内纪老师 2016年02月16日
微信公众号:iOS_yingxin 名称:小新的iOS之旅
我的: GitHub CSDN博客

```
程序，源于生活.
好的程序员要有能够从生活中提炼知识的能力.
```

通过NSUserDefaults做欢迎页显示机制

欢迎页:通常是几张滚动的图片,给用户介绍应用的功能和亮点.

欢迎页的机制:
1.首次运行时,显示欢迎页. 看过以后,再次运行则直接进入主页面,跳过欢迎页.
2.升级之后,首次运行,显示欢迎页. 再次运行则直接进入主页面,跳过欢迎页.

解决方案: 通过NSUserDefaults保存当前的版本号, 每次运行时, 检查这个保存的版本号与当前系统版本号是否一致.
1.首次运行时, 获取plist中的版本号, 一定是nil的. 那么显示欢迎页, 进入到主页面以后, 把当前系统版本号存入.
2.当升级时, 获取plist的版本号与当前版本号对比,如果不一样,则表示版本有变动, 则显示欢迎页. 进入主页以后, 同步当前版本号
因为window的根视图控制器,需要根据实际情况发生变化, 通过storyboard启动无法满足. 必须使用代码操作.

版本号

1.version 发布版本号,是给用户看的
2.build 测试版本号,用于发布之前的版本迭代区分的, 上传APPStore使用

从info.plist中获取版本号

```
//获取info.plist文件
NSDictionary *infoDic = [[NSBundle mainBundle] infoDictionary];
//通过打印, 查看key-value组合, 获取版本号
NSString *versionNumber = infoDic[@"CFBundleShortVersionString"];
```

通过对比已运行版本号和当前系统版本号决定展示的界面

```
//2.获取已经运行过的版本号
NSString *runVersionNumber = [[NSUserDefaults standardUserDefaults] objectForKey:
kRunVersion];
id vc = nil;
UIStoryboard *sb = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
if (runVersionNumber == nil || ![runVersionNumber isEqualToString:versionNumber])
 {
    //显示欢迎页
    vc = [sb instantiateInitialViewController];
}else{
    //主页面
    vc = [sb instantiateViewControllerWithIdentifier:@"MainViewController"];
}
self.window.rootViewController = vc;
```

获取当前 `AppDelegate` 实例对象指针

`AppDelegate` 对象, 就是 `UIApplication` 单例的代理对象.

```
AppDelegate *delegate = [UIApplication sharedApplication].delegate;
```

进入主页以后, 同步已运行版本号

```
//设置当前版本号为已运行版本号
NSString *version = [[[NSBundle mainBundle] infoDictionary] objectForKey:versionK
ey];
[[NSUserDefaults standardUserDefaults] setObject:version forKey:kRunVersion];
```

归档 `Archive` 与解档 `UnArchive`

归档和解档可以理解为压缩和解压缩, 作用是可以把任意对象类型存储到磁盘中, 进行持久化操作.

系统类型的归档与解档

OC中的系统类型, 自带归档解档功能. 例如 NSString, NSArray, NSDictionary...
系统类型通常带有writeToFile功能, 所以归档不是很常用.

归档

```
//归档到的文件路径, ~/Documents/archiveFile
NSString *docPath = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES).firstObject;
_archivePath =[docPath stringByAppendingPathComponent:@"archiveFile"];

NSArray *arr = @[@"腾讯", @"阿里", @"百度", @"谷歌"];
BOOL success = [NSKeyedArchiver archiveRootObject:arr toFile:@"归档到的文件路径"];
NSLog(@"%@", success?@"归档成功":@"归档失败");
```

解档

```
NSArray *arr = [NSKeyedUnarchiver unarchiveObjectWithFile:@"解档文件的路径"];
```

自定义类型的归档与解档

归档常用于自定义类型的持久化操作,
自定义类型,要实现归档和解档功能,必须遵循NSCoding协议, 并且实现协议的两个方法.

遵循协议

```
@interface Student : NSObject <NSCoding>

@property (nonatomic) NSString *name;
@property (nonatomic) NSString *sex;
@property (nonatomic) NSUInteger age;
@property (nonatomic) BOOL marry;
@property (nonatomic) NSString *school;
@property (nonatomic) NSString *className;
@property (nonatomic) NSString *favor;
@property (nonatomic) NSString *skill;
@property (nonatomic) NSString *score;

@end
```

解档协议方法实现

```
- (nullable instancetype)initWithCoder:(NSCoder *)aDecoder{
    if (self = [super init]) {
        self.name = [aDecoder decodeObjectForKey:@"name"];
        self.sex = [aDecoder decodeObjectForKey:@"sex"];
        self.age = [aDecoder decodeIntegerForKey:@"age"];
        self.skill = [aDecoder decodeObjectForKey:@"skill"];
        self.favor = [aDecoder decodeObjectForKey:@"favor"];
        self.className = [aDecoder decodeObjectForKey:@"className"];
        self.school = [aDecoder decodeObjectForKey:@"school"];
        self.score = [aDecoder decodeObjectForKey:@"score"];
        self.marry = [aDecoder decodeBoolForKey:@"marry"];
    }
    return self;
}
```

归档协议方法实现

```
- (void)encodeWithCoder:(NSCoder *)aCoder{
    [aCoder encodeObject:self.name forKey:@"name"];
    [aCoder encodeObject:self.sex forKey:@"sex"];
    [aCoder encodeInteger:self.age forKey:@"age"];
    [aCoder encodeBool:self.marry forKey:@"marry"];
    [aCoder encodeObject:self.skill forKey:@"skill"];
    [aCoder encodeObject:self.score forKey:@"score"];
    [aCoder encodeObject:self.school forKey:@"school"];
    [aCoder encodeObject:self.className forKey:@"className"];
    [aCoder encodeObject:self.favor forKey:@"favor"];
}
```

使用 `runtime` 简化自定义类型的归档与解档

归档和解档协议方法的实现, 是非常有规律的. 都是把属性名字和属性值进行相关的操作.
所以我们可以考虑获取当前类的属性列表, 通过循环方式, 让代码自动完成遍历归档解档操作. 要获取一个类的属性列表, 需要使用到运行时 `runtime` 类

引入 `runtime` 头文件

```
#import <objc/runtime.h>

所有OC代码在程序运行起来以后， 会被编译成runtime代码.
runtime代码在工作中不常用， 但是用的时候都是为了解决比较大的问题， OC代码无法解决的问题.
```

通过 `runtime` 获取属性列表

```
//属性数量, 通过地址传递
unsigned int outCount = 0;
//获取参数1类的属性列表, 属性数量存入参数2地址, 返回值是一个包含Ivar参数的数组
Ivar *varList = class_copyIvarList(self.class, &outCount);
//2.通过for循环,对每个属性进行设置
for (int i = 0; i< outCount; i++) {
    //Ivar是结构体,包含属性名等..
    Ivar tmpIvar = varList[i];
    //从Ivar构造体中获取属性名字符串
    const char *name = ivar_getName(tmpIvar);
    //C字符串->OC字符串
    NSString *propertyName = [NSString stringWithUTF8String:name];
    //打印属性名字
    NSLog(@"propertyName %@", propertyName);
}
//用完以后必须释放
free(varList);
```

通过KVC机制实现归档解档自动化代码

```objc
- (void)encodeWithCoder:(NSCoder *)aCoder{
    unsigned int outCount = 0;
    Ivar *varList = class_copyIvarList(self.class, &outCount);
    for (int i = 0; i< outCount; i++) {
        Ivar tmpIvar = varList[i];
        const char *name = ivar_getName(tmpIvar);
        NSString *propertyName = [NSString stringWithUTF8String:name];
        id obj = [self valueForKey:propertyName];
        [aCoder encodeObject:obj forKey:propertyName];
    }
    free(varList);
}

- (instancetype)initWithCoder:(NSCoder *)aDecoder{
    if (self = [super init]) {
        unsigned int outCount = 0;
        Ivar *varList = class_copyIvarList(self.class, &outCount);
        for (int i = 0; i < outCount; i++) {
            Ivar tmpVar = varList[i];
            const char *name = ivar_getName(tmpVar);
            NSString *propertyName = [NSString stringWithUTF8String:name];
            id obj = [aDecoder decodeObjectForKey:propertyName];
            //self.name = @"Faker";
            //->[self setValue:@"Faker" forKey:@"name"];
            [self setValue:obj forKey:propertyName];
        }
        free(varList);
    }
    return self;
}
```

通过宏定义简化代码

通过runtime+KVC实现的归档解档操作代码, 是通用的. 所以可以用宏保存起来, 使用更加简洁
通过在宏中使用 `\` 符号, 可以实现宏的换行排版功能.

```
#define kTeduCoding \
- (void)encodeWithCoder:(NSCoder *)aCoder{\
    unsigned int outCount = 0;\
    Ivar *varList = class_copyIvarList(self.class, &outCount);\
    for (int i = 0; i< outCount; i++) {\
        Ivar tmpIvar = varList[i];\
        const char *name = ivar_getName(tmpIvar);\
        NSString *propertyName = [NSString stringWithUTF8String:name];\
        id obj = [self valueForKey:propertyName];\
        [aCoder encodeObject:obj forKey:propertyName];\
    }\
    free(varList);\
}\
\
- (instancetype)initWithCoder:(NSCoder *)aDecoder{\
    if (self = [super init]) {\
        unsigned int outCount = 0;\
        Ivar *varList = class_copyIvarList(self.class, &outCount);\
        for (int i = 0; i < outCount; i++) {\
            Ivar tmpVar = varList[i];\
            const char *name = ivar_getName(tmpVar);\
            NSString *propertyName = [NSString stringWithUTF8String:name];\
            id obj = [aDecoder decodeObjectForKey:propertyName];\
            [self setValue:obj forKey:propertyName];\
        }\
        free(varList);\
    }\
    return self;\
}\
```

NSURLSession

下载大文件 `NSURLSessionDownloadTask`

下载下来的文件会被存放在临时文件夹下,如果不做处理,会被自动删除以节省空间, 我们需要复制到其他位置.

```
NSURL *imageURL = [NSURL URLWithString:kImagePath];
//~/Documents/image.png
NSString *path = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUser
DomainMask, YES).firstObject stringByAppendingPathComponent:@"image.png"];

NSURLSessionDownloadTask *task = [[NSURLSession sharedSession] downloadTaskWithUR
L:imageURL completionHandler:^(NSURL * _Nullable location, NSURLResponse * _Nulla
ble response, NSError * _Nullable error) {
    //location下载的文件的存储位置
    NSLog(@"location %@", location.path);
    //tmp->复制->document下
    [[NSFileManager defaultManager] copyItemAtPath:location.path toPath:path erro
r:nil];
    NSData *data = [NSData dataWithContentsOfFile:path];
    UIImage *image = [UIImage imageWithData:data];
    //回到主线程中,设置图片
    [[NSOperationQueue mainQueue] addOperationWithBlock:^{
        _imageView.image = image;
    }];
    NSLog(@"path : %@", path);
}];
[task resume];
```

上传大文件 `NSURLSessionUploadTask`

```
NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL URLWithString:@"填写服
务器地址"]];
NSURL *fileURL = [NSURL URLWithString:@"本地要上传的文件地址"];
NSURLSessionUploadTask *task = [[NSURLSession sharedSession] uploadTaskWithReques
t:request fromFile:fileURL completionHandler:^(NSData * _Nullable data, NSURLResp
onse * _Nullable response, NSError * _Nullable error) {

}];
[task resume];
```

任务的 `暂停` , `取消` , `继续` , `开始`

暂停

```
[dataTask suspend];
```

取消

```
[dataTask cancel];
```

继续和开始

```
[dataTask resume];
```

通过代理方式监听网络下载任务进度

可以通过代理的方式,进行网络请求. 这样可以实时监听网络请求的相关状态. 例如上传的进度, 下载的进度, 任务完成情况等.

带有代理的请求任务

```
@interface ViewController ()<NSURLSessionDataDelegate>

//实时监听下载进度,需要通过代理方法实现
/*
 defaultSessionConfiguration:下载的数据存放在tmp临时文件夹下
 ephemeralSessionConfiguration:缓存不存放在磁盘,而是在内存中(不推荐)
 backgroundSessionConfiguration:网络操作支持后台模式
 */
NSURLSessionConfiguration *config = [NSURLSessionConfiguration defaultSessionConf
iguration];
//参数3:代理方法运行的线程,填nil,是在子线程中
NSURLSession *session = [NSURLSession sessionWithConfiguration:config delegate:se
lf delegateQueue:nil];
_dataTask = [session dataTaskWithURL:[NSURL URLWithString:kImagePath]];
```

代理方法监听下载任务状态

下载的数据并不是一次性传递的, 而是分批次. 每次的数据大小根据网络情况而定.
需要使用可变的数据类型把这些零散的数据拼接起来.

```
//当收到数据时触发的,数据是分批次传递,所以此方法会多次触发
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask didReceiveData:(NSData *)data{
    [_fileData appendData:data];
    UIImage *image = [UIImage imageWithData:_fileData];
    dispatch_async(dispatch_get_main_queue(), ^{
    //已有数据长度除以总长度, 得到下载百分比
        _progressView.progress = dataTask.countOfBytesReceived *1.0/dataTask.countOfBytesExpectedToReceive;
        _imageView.image = image;
    });
}
//数据请求完毕时
- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task didCompleteWithError:(NSError *)error{
    //下载完毕以后, 保存下载下来的数据到磁盘中
    if (!error) {
        [_fileData writeToFile:self.filePath atomically:YES];
    }
}
```