Kingfisher

Grey

作者简介

王巍，微博ID:onevcat，圈儿内人称喵神，活跃的知名iOS/Unity 开发者，目前就职于Line。

objc中国发起人。

VVDocumenter作者,Kingfisher作者。

著有《Swifter-100个Swift必备Tips》。

主导翻译《函数式Swift》，独立翻译《Swift进阶》。

Usage

UIImageView extension:

Basic

```
import Kingfisher

imageView.kf_setImageWithURL(NSURL(string: “http://your_image_url.png")!)
imageView.kf_setImageWithURL(NSURL(string: "http://your_image_url.png")!, placeholderImage: nil)
```

```
let URL = NSURL(string: "http://your_image_url.png")!
let resource = Resource(downloadURL: URL, cacheKey: "your_customized_key")

imageView.kf_setImageWithResource(resource)
```

Options

```
imageView.kf_setImageWithURL(NSURL(string: "your_image_url")!,
                             placeholderImage: nil,
                             optionsInfo: [.ForceRefresh])
```

```
imageView.kf_setImageWithURL(NSURL(string: "your_image_url")!,
                             placeholderImage: nil,
                             optionsInfo: [.Transition(ImageTransition.Fade(1))])
```

KingfisherOptionsInfoItem

Options

```
//定义参数数组
public typealias KingfisherOptionsInfo = [KingfisherOptionsInfoItem]

//参数为枚举类型:
public enum KingfisherOptionsInfoItem {
    case TargetCache(ImageCache?)//自定义Cache
    case Downloader(ImageDownloader?)//自定义ImageDownloader
    case Transition(ImageTransition)//指定动画
    case DownloadPriority(Float)//任务优先级
    case ForceRefresh//强制刷新，忽略cache
    case ForceTransition//从cache获取image后set也强制使用动画
    case CacheMemoryOnly//cache只保存于内存中，不保存到disk
    case BackgroundDecode//后台解码
    case CallbackDispatchQueue(dispatch_queue_t?)//callback所处的队列,默认是main queue
    case ScaleFactor(CGFloat)//缩放比例
    case PreloadAllGIFData//预先加载所有gif数据
}
```

以上这些Options可以任意组合

```
let queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0)
let optionInfo: KingfisherOptionsInfo = [
    .ForceRefresh,
    .DownloadPriority(0.5),
    .CallbackDispatchQueue(queue),
    .Transition(ImageTransition.Fade(1))
]
```

Usage

UIImageView extension:

Callbacks

```
imageView.kf_setImageWithURL(NSURL(string: "your_image_url")!,
                             placeholderImage: nil,
                             optionsInfo: nil,
                             progressBlock: { (receivedSize, totalSize) -> () in
                                print("Download Progress: \(receivedSize)/\(totalSize)")
    },
                             completionHandler: { (image, error, cacheType, imageURL) -> () in
                                print("Downloaded and set!")
    }
)
```

Usage

UIImageView extension:

Cancel Task

方法一：通过imageview实例取消

```
imageView.kf_setImageWithURL(NSURL(string: "http://your_image_url.png")!)

// The image retrieving will stop.
imageView.kf_cancelDownloadTask()
```

方法二：通过RetrieveImageTask取消

```
let task = imageView.kf_setImageWithURL(NSURL(string: "http://your_image_url.png")!)

let urlShouldNotBeCancelled: URL = ...

if task.downloadTask?.URL != urlShouldNotBeCancelled {
    task.cancel()
}
```

Usage

UIImageView extension:

函数定义

```
    public func kf_setImageWithURL(URL: NSURL?,
                                   placeholderImage: Image? = nil,
                                   optionsInfo: KingfisherOptionsInfo? = nil,
                                   progressBlock: DownloadProgressBlock? = nil,
                                   completionHandler: CompletionHandler? = nil) -> RetrieveImageTask
```

↓

```
    public func kf_setImageWithResource(resource: Resource?,
                                placeholderImage: Image? = nil,
                                     optionsInfo: KingfisherOptionsInfo? = nil,
                                   progressBlock: DownloadProgressBlock? = nil,
                               completionHandler: CompletionHandler? = nil) -> RetrieveImageTask
```

除了第一个参数外，其它参数都有默认值，所以都是可选参数

RetrieveImageTask

返回对象RetrieveImageTask

```swift
/// RetrieveImageTask 代表图片的获取任务，这个任务被传递到最上层，供调用者控制.
/// 含有2个Task，并且可以对图片获取任务进行取消
public class RetrieveImageTask {
    static let emptyTask = RetrieveImageTask()
    var cancelledBeforeDownloadStarting: Bool = false
    public var diskRetrieveTask: RetrieveImageDiskTask?//从磁盘获取图片的任务
    public var downloadTask: RetrieveImageDownloadTask?//从网络获取图片的任务

    public func cancel() {
        if let diskRetrieveTask = diskRetrieveTask {
            dispatch_block_cancel(diskRetrieveTask)
        }
        if let downloadTask = downloadTask {
            downloadTask.cancel()
        } else {
            cancelledBeforeDownloadStarting = true
        }
    }
}
```

KingfisherManager

```swift
public class KingfisherManager {

    /// Shared manager used by the extensions across Kingfisher.
    public class var sharedManager: KingfisherManager {
        return instance
    }

    /// Cache used by this manager
    public var cache: ImageCache

    /// Downloader used by this manager
    public var downloader: ImageDownloader
}
```

ImageDownloader

instance

方法1、默认downloader

```
let downloader = KingfisherManager.sharedManager.downloader
```

方法2、自定义downloader

```
let customDownloader = ImageDownloader(name: "Custom_ImageDownloader")
```

downloadTimeout

```
//默认downloader
/// The duration before the download is timeout. Default is 15 seconds.
public var downloadTimeout: NSTimeInterval = 15.0

//自定义downloader
let timeout = self.downloadTimeout == 0.0 ? 15.0 : self.downloadTimeout
```

ImageDownloader

trustedHosts

```
downloader.trustedHosts = Set(["your_self_signed_host"])
```

ImageDownloader提供了对http响应的认证要求，向服务器发起get请求后，服务器会先返回要求认证的challenge，询问请求发起方是谁，此时发起方需要提供正确的认证信息，服务器才会返回真正的http响应。

收到认证要求时，NSURLSession的delegate会收到一个NSURLAuthenticationChallenge实例，该实例遵守NSURLAuthenticationChallengeSender协议，此时需要向服务器回复一个NSURLCredential实例。

当然，客户端认证流程需要服务器支持

加入信任集合trustedHosts中的服务器，客户端会忽略认证过程，直接返回NSURLCredential实例

ImageDownloader

requestModifier

```
public class ImageDownloader: NSObject {
    //……
    public var requestModifier: (NSMutableURLRequest -> Void)?
    //……
}
```

图片下载前会先调用此方法,所以可以在此修改request请求, eg:修改header认证等.

```
downloader.requestModifier = {
    (request: NSMutableURLRequest) in
    // Do what you need to modify the download request. Maybe add your HTTP basic
authentication for example.
}
```

ImageDownloader

ImageFetchLoad,用于数据处理传递的一个嵌套类

```
class ImageFetchLoad {
    var callbacks = [CallbackPair]()
    var responseData = NSMutableData()

    var options: KingfisherOptionsInfo?

    var downloadTaskCount = 0
    var downloadTask: RetrieveImageDownloadTask?
}
```

```
typealias CallbackPair = (progressBlock: ImageDownloaderProgressBlock?, completionHander:
ImageDownloaderCompletionHandler?)
```

一个URL请求可能会有多个callback

```
    internal func setupProgressBlock(progressBlock: ImageDownloaderProgressBlock?, completionHandler:
ImageDownloaderCompletionHandler?, forURL URL: NSURL, started: ((NSURLSession, ImageFetchLoad) -> Void)) {

        dispatch_barrier_sync(barrierQueue, { () -> Void in

            let loadObjectForURL = self.fetchLoads[URL] ?? ImageFetchLoad()
            let callbackPair = (progressBlock: progressBlock, completionHander: completionHandler)

            loadObjectForURL.callbacks.append(callbackPair)
            self.fetchLoads[URL] = loadObjectForURL

            if let session = self.session {
                started(session, loadObjectForURL)
            }
        })
    }
```

ImageDownloader

```
setupProgressBlock(progressBlock, completionHandler: completionHandler, forURL: request.URL!) {(session, fetchLoad) -> Void
in
    if fetchLoad.downloadTask == nil {
        let dataTask = session.dataTaskWithRequest(request)

        fetchLoad.downloadTask = RetrieveImageDownloadTask(internalTask: dataTask, ownerDownloader: self)
        fetchLoad.options = options

        dataTask.priority = options?.downloadPriority ?? NSURLSessionTaskPriorityDefault
        dataTask.resume()

        // Hold self while the task is executing.
        self.sessionHandler.downloadHolder = self
    }
```

ImageDownloader

sessionHandler

session

Strong

```
private var session: NSURLSession?
```

```
session = NSURLSession(configuration: sessionConfiguration, delegate: self,
delegateQueue: NSOperationQueue.mainQueue())
```

```
public var sessionConfiguration =
NSURLSessionConfiguration.ephemeralSessionConfiguration() {
        didSet {
            session = NSURLSession(configuration: sessionConfiguration, delegate: self,
delegateQueue: NSOperationQueue.mainQueue())
        }
    }
```

```
IMPORTANT
The session object keeps a strong reference to the delegate until your app exits or
explicitly invalidates the session. If you do not invalidate the session by calling the
invalidateAndCancel or finishTasksAndInvalidate method, your app leaks memory until it
exits.
```

defaultDownloader采用单例，正常生命周期内不会deinit，但是自定义ImageDownloader可能就会内存泄露

```
添加额外的一个sessionHandler,打破循环引用
private let sessionHandler: ImageDownloaderSessionHandler
```

```
if downloader.fetchLoads.isEmpty {
    downloadHolder = nil
}
```

ImageDownloader

NSData->图片

```
// MARK: - Create images from data
extension Image {
    static func kf_imageWithData(data: NSData, scale: CGFloat, preloadAllGIFData: Bool) -> Image? {
        var image: Image?
           switch data.kf_imageFormat {
            case .JPEG: image = Image(data: data, scale: scale)
            case .PNG: image = Image(data: data, scale: scale)
            case .GIF: image = Image.kf_animatedImageWithGIFData(gifData: data, scale: scale, duration: 0.0,
preloadAll: preloadAllGIFData)
            case .Unknown: image = Image(data: data, scale: scale)
            }
        return image
    }
}
```

backgroundDecode

主线程 加载图片数据->解码->渲染显示
UIImage数据从RAM 拷贝到 VRAM，效率不高，
详见：https://www.objccn.io/issue-3-1/

子线程中调用CGContextDrawImage将UIImage数据绘制到VRAM中，提高效率

ImagePrefetcher

初始化

URLS/Rources
manager
downloader
progressBlock
completionHandle
optionsInfo(过滤非主线程的callbackdispatchQueue)

Start

```
for i in 0 ..< initialConcurentDownloads {//同时下载
    self.startPrefetchingResource(self.prefetchResources[i])
}
```

```
func startPrefetchingResource(resource: Resource)
{
    requestedCount += 1
    if optionsInfo.forceRefresh {
        downloadAndCacheResource(resource)
    } else {
        let alreadyInCache = manager.cache.isImageCachedForKey(resource.cacheKey).cached
        if alreadyInCache {
            appendCachedResource(resource)
        } else {
            downloadAndCacheResource(resource)//就与前面的下载是一样的了
        }
    }
}
```

ImagePrefetcher

Stop

```
public func stop() {
    dispatch_async_safely_to_main_queue {

        if self.finished {
            return
        }

        self.stopped = true
        self.tasks.forEach { (_, task) -> () in
            task.cancel()
        }
    }
}
```

ImagePrefetcher

确保在指定线程执行

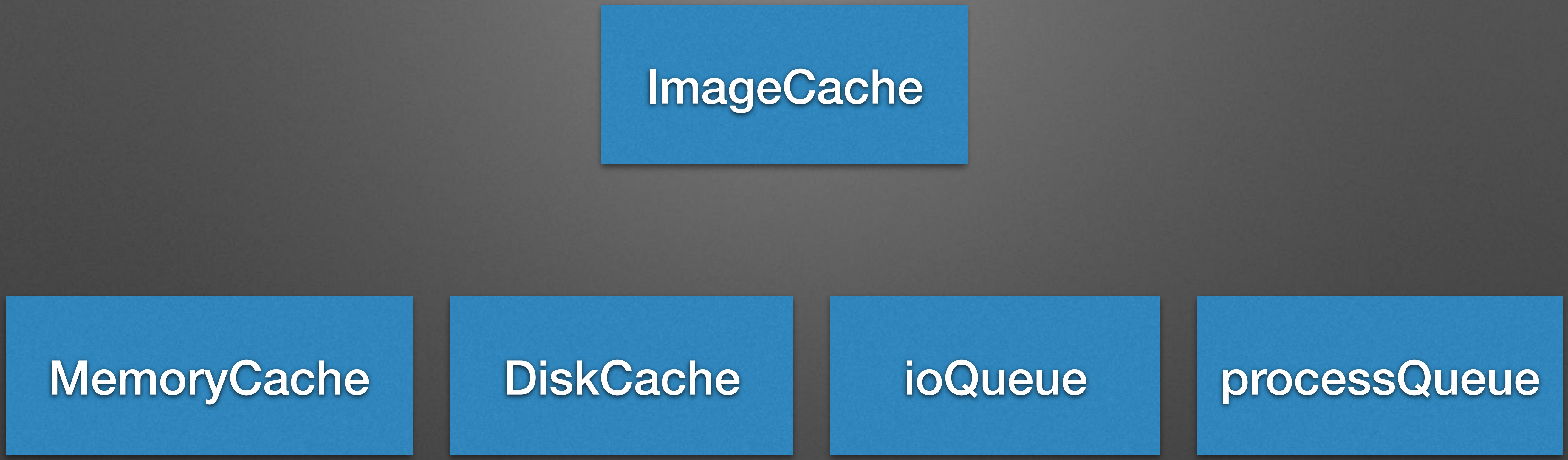
```
//在主线程执行，prefetcher时回调只支持主线程，初始化时就过滤了非主线程的回调
func dispatch_async_safely_to_main_queue(block: ()->()) {
    dispatch_async_safely_to_queue(dispatch_get_main_queue(), block)
}

// This method will dispatch the `block` to a specified `queue`.
// If the `queue` is the main queue, and current thread is main thread, the block
// will be invoked immediately instead of being dispatched.
func dispatch_async_safely_to_queue(queue: dispatch_queue_t, _ block: ()->()) {
    if queue === dispatch_get_main_queue() && NSThread.isMainThread() {
        block()
    } else {
        dispatch_async(queue) {
            block()
        }
    }
}
```

ImageDownloader回调时，确保在指定的线程中

```
for callbackPair in callbackPairs {
    dispatch_async_safely_to_queue(options.callbackDispatchQueue, { () -> Void in
        callbackPair.completionHander?(image: image, error: error, imageURL: imageURL,
originalData: originalData)
    })
}
```

ImageCache

主要的属性



ImageCache

主要的操作

Store & Remove

Get data from cache

Clear & Clean

Check cache status

End