**ios 图片操作**

简而言之，图片优化,所需要做的事情是尽可能少地通过网络从服务端下载图片;

图片缓存一般包括两部分：URL缓存和存储缓存，或曰“内存缓存”和“本地缓存”。前者是将图片存储到内存中，后者是将图片存储到本地磁盘中,因此，当UIImageView控件需要加载一张图片，图片来源可以由三个地方：内存、本地磁盘、网络[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)；消耗成本排序也是：内存 < 本地磁盘 < 网络服务器。

**SDWebImage 概论**

1. 提供了一个 UIImageView 的 category 用来加载网络图片并且对网络图片的缓存进行管理  
   2.采用异步方式来下载网络图片  
   3.采用异步方式，使用 memory＋disk 来缓存网络图片，自动管理缓存。  
   4.支持 GIF 动画  
   5.支持 WebP 格式  
   6.同一个 URL 的网络图片不会被重复下载  
   7.失效的 URL 不会被无限重试  
   8.耗时操作都在子线程，确保不会阻塞主线程  
   9.使用 GCD 和 ARC  
   10.支持 Arm64

SDWebImage提供一个UIImageView的类别以支持加载来自互联网的远程图片。具有缓存管理、异步下载，同一个URL下载次数控制和优化等特征。

###### 独立的异步图像下载

可能会用到单独的异步图片下载，则一定要用 - (id <SDWebImageOperation>)downloadImageWithURL:(NSURL \*)url options:(SDWebImageOptions)options progress:(SDWebImageDownloaderProgressBlock)progressBlock completed:(SDWebImageCompletionWithFinishedBlock)completedBlock; 来建立一个SDWebImageDownLoader 的实例。这样就可以有下载进度的回调和下载完成的回调，可以在回调完成进度条相关的操作和显示图片相关的操作。

###### 独立的异步图像缓存

SDImageCache类提供一个管理缓存的单例类。  
SDImageCache \*imageCache = [SDImageCache sharedImageCache]  
查找和缓存图片时以URL作为key。(先查找内存，如果内存不存在该图片，再查找硬盘；查找硬盘时，以URL的MD5值作为key).  
查找图片：  
UIImage \*cacheImage = [imageCache mageFromKey:myCacheKey];  
缓存图片：  
[ imageCache storeImage:myImage forKey:myCacheKey];  
默认情况下，图片是被存储到内存缓存和磁盘缓存中的。如果仅仅是想缓存到内存中，可以用下面方法：  
storeImage:forKey:toDisk: 第三个参数传NO即可。

###### 主要用到的对象：

* 1.UIImageView(WebCache)，入口封装，实现读取图片完成后的回调。
* 2.SDWebImagemanager,对图片进行管理的中转站，记录那些图片正在读取。向下层读取Cache（调用SDImageCache），或者向网络读取对象（调用SDWebImageDownloader）。实现SDImageCache和SDWebImageDownLoader的回调。
* 3.SDImageCache,根据URL作为key，对图片进行存储和读取（存在内存（以URL作为key）和存在硬盘两种（以URL的MD5值作为key））。实现图片和内存清理工作。

### SDWebImage加载图片的流程

* 1.入口 setImageWithURL:placeholderImage:options:会先把 placeholderImage显示，然后 SDWebImageManager根据 URL 开始处理图片。
* 2.进入SDWebImageManager 类中downloadWithURL:delegate:options:userInfo:，交给  
  SDImageCache从缓存查找图片是否已经下载  
  queryDiskCacheForKey:delegate:userInfo:.
* 3.先从内存图片缓存查找是否有图片，如果内存中已经有图片缓存，SDImageCacheDelegate回调 imageCache:didFindImage:forKey:userInfo:到  
  SDWebImageManager。
* 4.SDWebImageManagerDelegate 回调  
  webImageManager:didFinishWithImage: 到 UIImageView+WebCache,等前端展示图片。
* 5.如果内存缓存中没有，生成 ｀NSOperation ｀  
  添加到队列，开始从硬盘查找图片是否已经缓存。
* 6.根据 URL的MD5值Key在硬盘缓存目录下尝试读取图片文件。这一步是在 NSOperation 进行的操作，所以回主线程进行结果回调 notifyDelegate:。
* 7.如果上一操作从硬盘读取到了图片，将图片添加到内存缓存中（如果空闲内存过小， 会先清空内存缓存）。SDImageCacheDelegate'回调 imageCache:didFindImage:forKey:userInfo:`。进而回调展示图片。
* 8.如果从硬盘缓存目录读取不到图片，说明所有缓存都不存在该图片，需要下载图片， 回调 imageCache:didNotFindImageForKey:userInfo:。
* 9.共享或重新生成一个下载器 SDWebImageDownloader开始下载图片。
* 10.图片下载由 NSURLConnection来做，实现相关 delegate  
  来判断图片下载中、下载完成和下载失败。
* 11.connection:didReceiveData: 中利用 ImageIO做了按图片下载进度加载效果。
* 12.connectionDidFinishLoading: 数据下载完成后交给 SDWebImageDecoder做图片解码处理。
* 13.图片解码处理在一个 NSOperationQueue完成，不会拖慢主线程 UI.如果有需要 对下载的图片进行二次处理，最好也在这里完成，效率会好很多。
* 14.在主线程 notifyDelegateOnMainThreadWithInfo:  
  宣告解码完成 imageDecoder:didFinishDecodingImage:userInfo: 回调给 SDWebImageDownloader`。
* 15.imageDownloader:didFinishWithImage:回调给 SDWebImageManager告知图片 下载完成。  
  -16. 通知所有的 downloadDelegates下载完成，回调给需要的地方展示图片。
* 17.将图片保存到 SDImageCache中，内存缓存和硬盘缓存同时保存。写文件到硬盘 也在以单独 NSOperation 完成，避免拖慢主线程。
* 18.SDImageCache 在初始化的时候会注册一些消息通知，  
  在内存警告或退到后台的时 候清理内存图片缓存，应用结束的时候清理过期图片。