1. SDWebImage内部实现原理步骤

1.1 实现步骤

1\. 入口setImageWithUrl:placeHolderImage:options:会把placeHolderImage显示,然后SDWebImageManager根据URL开始处理图片.

2\. 进入SDWebImageManager-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:didFineImage: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NSInvocationOperation添加到队列开始从硬盘查找图片是否已经缓存

6\. 根据URLKey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:didFinishImage: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

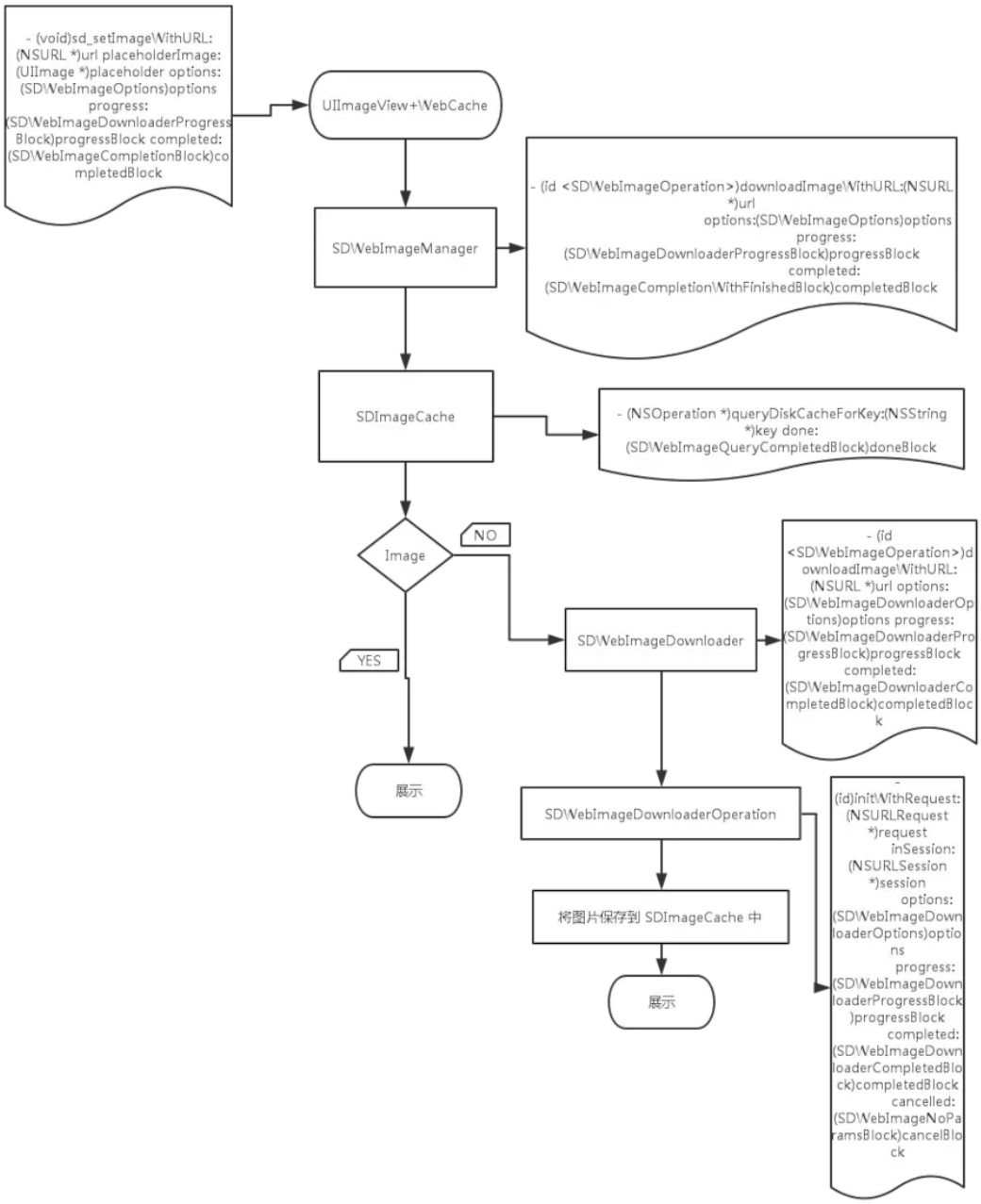
17\. 将图片保存到SDImageCache中内存缓存和硬盘缓存同时保存,写文件到硬盘也在以单独NSInvocationOperation完成,避免拖慢主线程

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

19\. SDWI也提供UIButton + WebCache和MKAnnptation + WebCache方便使用

20\. SDWebImagePrefetcher 可以预先下载图片,方便后续使用

再用一张图说明:



1.2 API中参数枚举类型

例如，设置图片的两个例子

```
[self.image2 sd_setImageWithURL:imagePath2 completed:^(UIImage
*image, NSError *error, SDImageCacheType cacheType, NSURL
*imageURL) {
        NSLog(@"这里可以在图片加载完成之后做些事情");
```

```
}];
```

```
//使用默认图片，而且用block 在完成后做一些事情
[self.image1 sd_setImageWithURL:imagePath1 placeholderImage:
[UIImage imageNamed:@"default"] completed:^(UIImage *image, NSError
*error, SDImageCacheType cacheType, NSURL *imageURL) {
        NSLog(@"图片加载完成后做的事情");
}];
```

还有获取下载进度的例子

```
//使用默认图片，而且用block 在完成后做一些事情
[self.image1 sd_setImageWithURL:imagePath1 placeholderImage:
[UIImage imageNamed:@"default"] completed:^(UIImage *image, NSError
*error, SDImageCacheType cacheType, NSURL *imageURL) {
        NSLog(@"图片加载完成后做的事情");
}];
```

上面的例子，都有个`SDImageCacheTyp`的参数，你记得这个参数有哪些?

- SDImageCacheType

```
//定义Cache类型
typedef NS_ENUM(NSInteger, SDImageCacheType) {
//不使用cache获得图片,依然会从web下载图片
    SDImageCacheTypeNone,
//图片从disk获得
    SDImageCacheTypeDisk,
//图片从Memory中获得
    SDImageCacheTypeMemory
};
```

2. 最大缓存和时间设置

- SDImageCache类的源码

```
//这个变量默认值为YES，显示比较高质量的图片，但是会浪费比较多的内存，可以通过设置NO来缓解内存
@property (assign, nonatomic) BOOL shouldDecompressImages;
//总共的内存允许图片的消耗值
@property (assign, nonatomic) NSUInteger maxMemoryCost;
//图片存活于内存的时间初始化的时候默认为一周
@property (assign, nonatomic) NSInteger maxCacheAge;
//每次存储图片大小的限制
@property (assign, nonatomic) NSUInteger maxCacheSize;
```

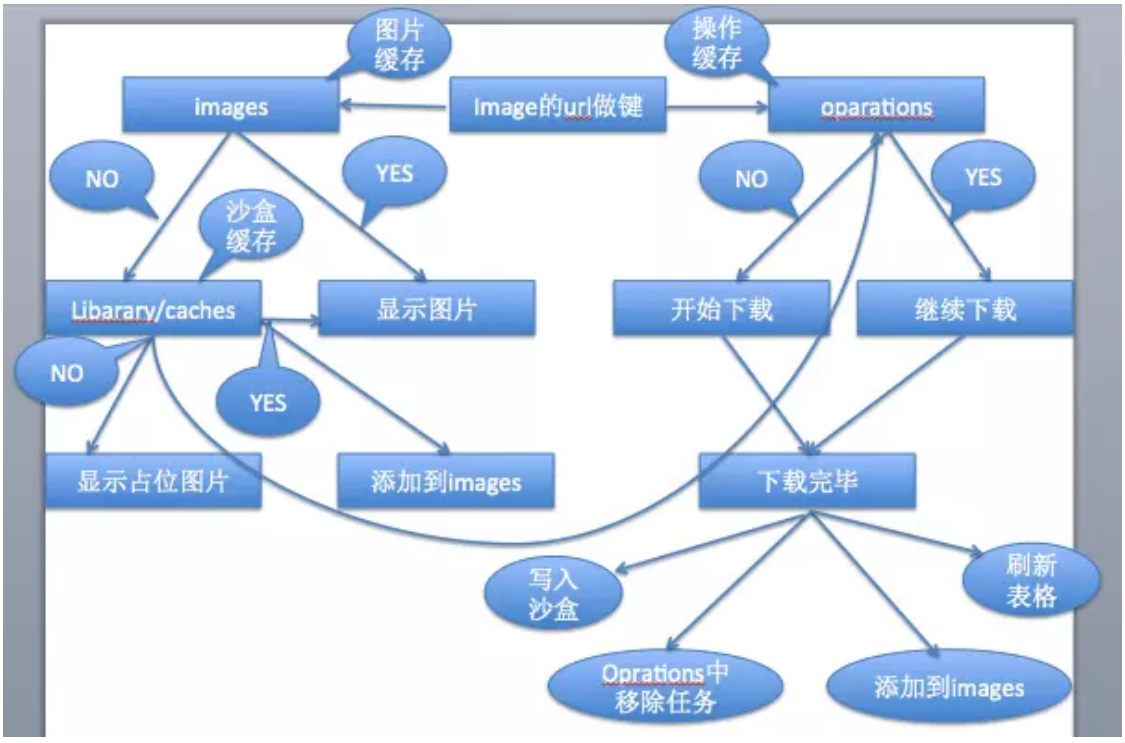
- 设置maxCacheSize的例子

```
SDWebImageManager *manager = [SDWebImageManager sharedManager];
[manager.imageCache setMaxMemoryCost:1000000];//设置总缓存大小，默认为0没有限制
[manager.imageCache setMaxCacheSize:640000];//设置单个图片限制大小
[manager.imageDownloader setMaxConcurrentDownloads:1];//设置同时下载线程数，这是下载器的内容，下面将会介绍
[manager downloadImageWithURL:[NSURL URLWithString:@"http://p9.qhimg.com/t01eb74a44c2eb43193.jpg"]
                      options:SDWebImageProgressiveDownload progress:^(NSInteger receivedSize, NSInteger expectedSize) {
                          NSLog(@"%lu", receivedSize);
                      } completed:^(UIImage *image, NSError *error, SDImageCacheType cacheType, BOOL finished, NSURL *imageURL) {
                          self.imageView1.image = image;

                      }];
[manager downloadImageWithURL:[NSURL URLWithString:@"http://img.article.pchome.net/00/28/33/87/pic_lib/wm/kuanpin12.jpg"]
                      options:SDWebImageProgressiveDownload progress:^(NSInteger receivedSize, NSInteger expectedSize) {
                          NSLog(@"%lu", receivedSize);
                      } completed:^(UIImage *image, NSError *error, SDImageCacheType cacheType, BOOL finished, NSURL *imageURL) {
                          self.imageView2.image = image;

                      }];
NSUInteger size = [manager.imageCache getSize];
NSUInteger count = [manager.imageCache getDiskCount];
NSLog(@"size = %lu", size); // 644621（两张测试图片）
NSLog(@"count = %lu", count); // 2
[manager.imageCache clearDisk];
```

```
size = [manager.imageCache getSize];
count = [manager.imageCache getDiskCount];
NSLog(@"sizeClean = %lu", size);  //  0
NSLog(@"countClean = %lu", count);     //  0   这里使用的是clear
```

3. 区分：三种缓存（内存图片缓存，磁盘图片缓存，内存操作缓存）

- 先查看内存图片缓存，内存图片缓存没有，后生成操作，查看磁盘图片缓存
- 磁盘图片缓存有，就加载到内存缓存，没有就下载图片
- 在建立下载操作之前，判断下载操作是否存在
- 默认情况下，下载的图片数据会同时缓存到内存和磁盘中



> 关于缓存位置

- 内存缓存是通过 NSCache的子类AutoPurgeCache来实现的；
- 磁盘缓存是通过 NSFileManager 来实现文件的存储(默认路径为/Library/Caches/default/com.hackemist.SDWebImageCache.default)，是异步实现的。

> 关于图片下载操作

SDWebImage的大部分工作是由缓存对象SDImageCache和异步下载器管理对象SDWebImageManager来完成的。

SDWebImage的图片下载是由SDWebImageDownloader这个类来实现的，它是一个异步下载管理器，下载过程中增加了对图片加载做了优化的处理。而真正实现图片下载的是自定义的一个Operation操作，将该操作加入到下载管理器的操作队列downloadQueue中，Operation操作依赖系统提供的NSURLConnection类实现图片的下载。

4. 高清和低清图片与网络环境的问题

- 网络判断的问题--利用AFNetworking的API 首先，启用监控

```
// AppDelegate.m 文件中
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 监控网络状态
    [[AFNetworkReachabilityManager sharedManager] startMonitoring];
}
```

然后，在需要的地方获取监控管理

```
    // 以下代码在需要监听网络状态的方法中使用
    AFNetworkReachabilityManager *mgr =
[AFNetworkReachabilityManager sharedManager];
    if (mgr.isReachableViaWiFi)     { // 在使用Wifi，下载原图

    } else     { // 其他，下载小图

    }
```

- 并不能简单的这样：WIFI就下载高清图，蜂窝网络就下载缩略图。要考虑和利用缓存的因素。
- 一个典型例子

```objc
- setItem:(CustomItem *)item
{
    _item = item;

     // 占位图片
    UIImage *placeholder = [UIImage
imageNamed:@"placeholderImage"];

    // 从内存\沙盒缓存中获得原图,
    UIImage *originalImage = [[SDImageCache sharedImageCache]
imageFromDiskCacheForKey:item.originalImage];
    if (originalImage) { // 如果内存\沙盒缓存有原图，那么就直接显示原图（不
管现在是什么网络状态）
        self.imageView.image = originalImage;
    } else { // 内存\沙盒缓存没有原图
        AFNetworkReachabilityManager *mgr =
[AFNetworkReachabilityManager sharedManager];
        if (mgr.isReachableViaWiFi) { // 在使用Wifi，下载原图
            [self.imageView sd_setImageWithURL:[NSURL
URLWithString:item.originalImage] placeholderImage:placeholder];
        } else if (mgr.isReachableViaWWAN) { // 在使用手机自带网络
            //     用户的配置项假设利用NSUserDefaults存储到了沙盒中
            //    [[NSUserDefaults standardUserDefaults] setBool:NO
forKey:@"alwaysDownloadOriginalImage"];
            //    [[NSUserDefaults standardUserDefaults]
synchronize];
#warning 从沙盒中读取用户的配置项：在3G\4G环境是否仍然下载原图
            BOOL alwaysDownloadOriginalImage = [[NSUserDefaults
standardUserDefaults] boolForKey:@"alwaysDownloadOriginalImage"];
            if (alwaysDownloadOriginalImage) { // 下载原图
                [self.imageView sd_setImageWithURL:[NSURL
URLWithString:item.originalImage] placeholderImage:placeholder];
            } else { // 下载小图
                [self.imageView sd_setImageWithURL:[NSURL
URLWithString:item.thumbnailImage] placeholderImage:placeholder];
            }
        } else { // 没有网络
            UIImage *thumbnailImage = [[SDImageCache
sharedImageCache] imageFromDiskCacheForKey:item.thumbnailImage];
            if (thumbnailImage) { // 内存\沙盒缓存中有小图
                self.imageView.image = thumbnailImage;
            } else { // 处理离线状态，而且有没有缓存时的情况
                self.imageView.image = placeholder;
            }
        }
    }
```

```
}
```

5. 可能的问题

- 后台没有处理的高清大图导致APP内存过大而奔溃
  - 思路，改写sd_imageWithData方法的源代码，可参考 https://blog.csdn.net/benyoulai5/article/details/50462586