架构分级：

1、最上层业务：

```
#import "UIImageView+WebCache.h"
#import "UIButton+WebCache.h"
#import "UIImageView+HighlightedWebCache.h"
#import "UIView+WebCacheOperation.h"
#import "MKAnnotationView+WebCache.h"
```

2、逻辑层：

```
#import "SDWebImageManager.h"
```

3、具体业务实现

```
//缓存&&磁盘操作
#import "SDImageCache.h"
//下载操作
#import "SDWebImageDownloader.h"
```

等

最上层业务实现：

```
#import "UIImageView+WebCache.h"
#import "UIButton+WebCache.h"
#import "UIImageView+HighlightedWebCache.h"
#import "MKAnnotationView+WebCache.h"
```

这些类所做的操作都是一样的，分别是为UIImageView，UIButton，MKAnnotationView提供下载显示图片的入口。

以UIImageView为例：

```
//下载图片的核心方法
```

第一步

```
/*
 * url            图片的二进制数据
 * placeholder  UIImageView的占位图片
 * options      图片下载选项（策略）
 * progressBlock    进度回调
 * completedBlock   完成回调
 */
- (void)sd_setImageWithURL:(NSURL *)url placeholderImage:(UIImage
```

*)placeholder options:(SDWebImageOptions)options progress:(SDWebImageDownloaderProgressBlock)progressBlock completed:(SDWebImageCompletionBlock)completedBlock;

这里的逻辑如下：

1、先取消当前正在下载的操作。

2、保存将要下载图片的URL

3、如果不延迟显示占位图片则先显示占位图片

4、判断URL是否为空，如果为空则抛出一个错误

5、如果URL不为空则生成一个下载操作实例，并保存当前的操作。

6、如果图片下载成功，并且设置手动显示图片则直接返回图片，否则将图片设置到UIImageView并且刷新视图

7、如果图片下载失败，则显示占位图，并且返回一个错误。

加载图片的核心方法：

第二步

```
/*
 * 如果URL对应的图像在缓存中不存在，那么就下载指定的图片 ， 否则返回缓存的图像
 *
 * @param url 图片的URL地址
 * @param options 指定此次请求策略的选项
 * @param progressBlock 图片下载进度的回调
 * @param completedBlock 操作完成后的回调
 *     此参数是必须的，此block没有返回值
 *     Image：请求的 UIImage，如果出现错误，image参数是nil
 *     error：如果出现错误，则error有值
 *     cacheType：`SDImageCacheType`枚举，标示该图像的加载方式
 *         SDImageCacheTypeNone：从网络下载
 *         SDImageCacheTypeDisk：从本地缓存加载
 *         SDImageCacheTypeMemory：从内存缓存加载
 *         finished：如果图像下载完成则为YES，如果使用SDWebImageProgressiveDownload 选项，同时只获取到部分图片时，返回 NO
 *         imageURL：图片的URL地址
 *
```

```
 * @return SDWebImageOperation对象，应该是
SDWebimageDownloaderOperation实例
 */
- (id <SDWebImageOperation>)downloadImageWithURL:(NSURL *)url
                                         options:(SDWebImageOptions)options
                                        progress:
(SDWebImageDownloaderProgressBlock)progressBlock
                                       completed:
(SDWebImageCompletionWithFinishedBlock)completedBlock;
```

这里的逻辑如下：

1、检查URL是否正确，加互斥锁@synchronized判断URL是否在下载失败的黑名单中

2、查询缓存：（1）先检查是否有内存缓存（2）如果没有内存缓存则检查沙盒缓存（3）如果有沙盒缓存，则把图片放到内存缓存，最后返回图片。

3、使用下载器SDWebImageDownloader下载图片，下载失败，将URL加入到黑名单

4、如果下载成功则将图片缓存下来。

第三步

```
//SDWebImageDownloader核心方法：下载图片的操作
- (id <SDWebImageOperation>)downloadImageWithURL:(NSURL *)url
options:(SDWebImageDownloaderOptions)options progress:
(SDWebImageDownloaderProgressBlock)progressBlock completed:
(SDWebImageDownloaderCompletedBlock)completedBlock;
```

这里的逻辑如下：

1、设置图片下载超时时间，默认是15秒

2、生成NSMutableURLRequest实例设置超时时间，缓存策略，是否使用cookies，设置请求头信息

3、创建下载图片操作，设置是否需要解码，https身份认证，将下载操作添加到NSOperationQueue队列中

第四步

SDWebImageDownloaderOperation继承NSOperation，重写-(void)start方法在线程内下载

start逻辑如下：

1、判断是否需要在后台下载，在程序进入后台时通过

UIBackgroundTaskIdentifier实现在后台下载
2、创建NSURLConnection对象发送下载网络请求
3、开启runloop

第五步

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data

NSURLConnection的回调方法，接收图片数据
1、将data图片数据转为CGImageSourceRef
2、当第一次接收到图片数据，即确定图片的方向。然后后面则接着前面的数据就行了
3、图片是否需要解码，需要解码的图片就进行解码
4、返回图片
图片下载主要流程就走完了。

现在搜集一下里面的一些细节面试用

1、@autoreleasepool的使用，在创建大量临时对象的时候，内存会暴增，短时间内得不到释放。如果把这些临时对象放到@autoreleasepool块中，则系统会在这个块的末尾将内存回收，且不会影响主线程的pool。
2、支持WebP和gif图片。分别有UIImage分类进行处理
对于gif的处理，首先将data图片转为CGImageSourceRef，然后获取图片的帧数，如果帧数大于1，则创建一个数组，把每一帧的图片放到数组中，最后返回可动画的图片。
3、如何区分图片格式。
NSData+ImageContentType分类进行图片格式的区分，根据图片的二进制数据data转换为十六进制的第一个字节来区分图片的格式。

```
uint8_t c;
    //获得传入的图片二进制数据的第一个字节
    [data getBytes:&c length:1];
switch (c) {
case 0xFF:
            return @"image/jpeg";
        case 0x89:
            return @"image/png";
        case 0x47:
            return @"image/gif";
```

```
        case 0x49:
        case 0x4D:
            return @"image/tiff";
        case 0x52:
            //WEBP :是一种同时提供了有损压缩与无损压缩的图片文件格式
}
```

4、SDWebImage 缓存图片的名称如何避免重名。通过对图片URL的绝对路径进行MD5加密后的密文作为缓存图片的文件名和缓存的key。

5、宏定义

SDWebImage使用了FOUNDATION_EXPORT和UIKIT_EXTERN

UIKIT_EXTERN简单来说，就是将函数修饰为兼容以往C编译方式的、具有extern属性(文件外可见性)、public修饰的方法或变量库外仍可见的属性；

6、SDWebImage 如何保证UI操作放在主线程中执行?

最简单的方法

```
if ([NSThread isMainThread]) {
    block();
} else {
    dispatch_async(dispatch_get_main_queue(), block);
}
```

但是有的时候你需要在主队列上执行任务会出现异常

虽然每个应用程序都只有一个主线程，但是在这个主线程上执行许多不同的队列是可能的。如果在主线程执行非主队列调度的API，而这个API需要检查是否由主队列上调度，那么将会出现问题。

SDWebImage使用的方法：（判断是否在主线程执行改为判断是否在主队列上执行，因为主队列调度的任务肯定在主线程执行，而在主线程执行的任务不一定是由主队列调度的）

```
//取得当前队列的队列名
dispatch_queue_get_label(DISPATCH_CURRENT_QUEUE_LABEL)
//取得主队列的队列名
dispatch_queue_get_label(dispatch_get_main_queue())
```

然后通过 strcmp函数进行比较，如果为0则证明当前队列就是主队列。

就是下面这个宏定义

```
// SDWebImageCompat.h 中
#ifndef
```

dispatch_main_async_safe#definedispatch_main_async_safe(block)\if(strcmp(dispatch_queue_get_label(DISPATCH_CURRENT_QUEUE_LABEL), dispatch_queue_get_label(dispatch_get_main_queue())) == 0) {\ block();\ }else{\ dispatch_async(dispatch_get_main_queue(), block);\ }
#endif
7、SDWebImage 的最大并发数 和 超时时长
默认并发数为6，超时时长是15秒
_downloadQueue.maxConcurrentOperationCount = 6;
_downloadTimeout = 15.0;
8、SDWebImage 的Memory缓存和Disk缓存是用什么实现的
Memory的实现是通过继承自NSCache的AutoPurgeCache来实现的，AutoPurgeCache只是增加了一个监听到UIApplicationDidReceiveMemoryWarningNotification（应用程序发生内存警告）通知后，调用removeAllObjects方法的功能。
NSCache是苹果自己的内存缓存类，是一个线程安全的类，NSCache在系统内存很低时会自动释放对象，NSCache的Key只是对对象进行了Strong引用，而非拷贝
Disk的实现是通过NSFileManager文件管理类将数据存储到沙盒中Library的caches目录下
9、读取Memory和Disk的时候如何保证线程安全?
由于NSCache是线程安全的，所以Memeory缓存的读取是线程安全的，不需要加锁
对所有的Disk操作，SDWebImage创建了一个IOQueue的串行队列，然后创建一个异步任务
10、SDWebImage使用@synchronized来保证线程安全
11、SDWebImage 的Memory警告是如何处理的
UIApplicationDidReceiveMemoryWarningNotification 接收到内存警告的通知执行 clearMemory 方法，清理内存缓存！
12、SDWebImage Disk缓存时长？ Disk清理操作时间点？ Disk清理原则?
//默认的最大缓存时间为1周
static const NSInteger kDefaultCacheMaxCacheAge = 60 * 60 * 24 * 7; // 1 week
清理时机（1）监听到内存警告时清理内存（2）程序终止时清理过期的磁盘缓存（3）APP进入后台时

```
//当监听到UIApplicationDidReceiveMemoryWarningNotification（系统级内存警告）调用clearMemory方法
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(clearMemory)

name:UIApplicationDidReceiveMemoryWarningNotification
                                               object:nil];
    //当监听到UIApplicationWillTerminateNotification（程序将终止）调用cleanDisk方法
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(cleanDisk)
                                                 name:UIApplicationWillTerminateNotification
                                               object:nil];
//当监听到UIApplicationDidEnterBackgroundNotification（进入后台），调用backgroundCleanDisk方法
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(backgroundCleanDisk)

name:UIApplicationDidEnterBackgroundNotification
                                               object:nil];
```

清理缓存的规则分两步进行。 第一步先清除掉过期的缓存文件。 如果清除掉过期的缓存之后，空间还不够。 那么就继续按文件时间从早到晚排序，先清除最早的缓存文件，直到剩余空间达到要求。

13、SDWebImage 的回调设计?

使用次数不多或者有很强对象绑定，使用Delegate分散的回调不方便时使用Block回调。

需要多次调用时使用Delegate

同理苹果自身的回调设计

UITableView的使用Delegate、是用为在滚动途中、代理方法需要被不断的执行。

UIButton也是将会被多次点击。

UIView的动画/GCD则可以使用Block、因为只执行一次、用完释放。

所以、在日常使用中、我们也可以参考上述原则进行设计。

14、SDWebImage 中的工具类介绍

工具类深入研读

NSData+ImageContentType: 根据图片数据获取图片的类型，比如GIF、PNG等。

SDWebImageCompat: 根据屏幕的分辨倍数成倍放大或者缩小图片大小。

SDImageCacheConfig: 图片缓存策略记录。比如是否解压缩、是否允许iCloud、是否允许内存缓存、缓存时间等。默认的缓存时间是一周。

UIImage+MultiFormat: 获取UIImage对象对应的data、或者根据data生成指定格式的UIImage，其实就是UIImage和NSData之间的转换处理。

UIImage+GIF: 对于一张图片是否GIF做判断。可以根据NSData返回一张GIF的UIImage对象，并且只返回GIF的第一张图片生成的GIF。如果要显示多张GIF，使用FLAnimatedImageView。

SDWebImageDecoder: 根据图片的情况，做图片的解压缩处理。并且根据图片的情况决定如何处理解压缩。