整个AFNetworking框架由四部分构成：分别是UIKit相关的扩展，参数的序列化，网络监听，网络请求管理，https证书安全。
其中网络请求管理是最核心的，面向用户使用的类是AFHTTPSessionManager，提供用户调用的API，其继承于AFURLSessionManager。
AFHTTPSessionManager中可以知道AFNetworking默认是AFHTTPRequestSerializer形式去序列化请求链接，返回数据是AFJSONResponseSerializer。
这里主要就是介绍AFURLSessionManager。
1、从AFURLSessionManager初始化方法可以知道session的delegate是放在队列中的，并且使用的队列是NSOperationQueue，并不是直接使用GCD。默认的session会话类型是defaultSessionConfiguration，默认类型的。

```
- (instancetype)initWithSessionConfiguration:(NSURLSessionConfiguration *)configuration {
    self = [super init];
    if (!self) {
        return nil;
    }
    // 1 创建session的配置类，有三种会话模式可以设置
    //defaultSessionConfiguration;默认会话模式，使用磁盘缓存数据
    //ephemeralSessionConfiguration;私有会话模式，使用内存缓存数据
    //+ (NSURLSessionConfiguration *)backgroundSessionConfigurationWithIdentifier:(NSString *)identifier后台下载上传会话模式
    if (!configuration) {
        ///如果没有设置，将使用默认的会话模式
        configuration = [NSURLSessionConfiguration defaultSessionConfiguration];
    }

    self.sessionConfiguration = configuration;
    //创建session的delegate运行的队列
```

```
    self.operationQueue = [[NSOperationQueue alloc] init];
    self.operationQueue.maxConcurrentOperationCount = 1;
    //创建session实例,并且强引用,delegate 是自己
    self.session = [NSURLSession
sessionWithConfiguration:self.sessionConfiguration delegate:self
delegateQueue:self.operationQueue];
    //创建response 接受的数据序列化的辅助类,默认接受JSON数据
    self.responseSerializer = [AFJSONResponseSerializer serializer];
    // 配置安全策略 -> 主要在delegate中收到鉴权请求时候使用
    self.securityPolicy = [AFSecurityPolicy defaultPolicy];

#if !TARGET_OS_WATCH
    self.reachabilityManager = [AFNetworkReachabilityManager
sharedManager];
#endif
    //创建dict,用来管理task对应的delegate,因为有部分内容交给 delegate helper
类去完成的
    self.mutableTaskDelegatesKeyedByTaskIdentifier = [[NSMutableDictionary
alloc] init];

    //[task:delegateHelper] 访问时候的锁
    self.lock = [[NSLock alloc] init];
    self.lock.name = AFURLSessionManagerLockName;

    // 方法的主要作用是获取session中所有的真该执行的tasks
    [self.session getTasksWithCompletionHandler:^(NSArray *dataTasks,
NSArray *uploadTasks, NSArray *downloadTasks) {
        for (NSURLSessionDataTask *task in dataTasks) {
            [self addDelegateForDataTask:task uploadProgress:nil
downloadProgress:nil completionHandler:nil];
        }

        for (NSURLSessionUploadTask *uploadTask in uploadTasks) {
            [self addDelegateForUploadTask:uploadTask progress:nil
```

```
completionHandler:nil];
        }

        for (NSURLSessionDownloadTask *downloadTask in downloadTasks) {
            [self addDelegateForDownloadTask:downloadTask progress:nil
destination:nil completionHandler:nil];
        }
    }];
    return self;
}
```

2、每生成一个任务task，其都会与一个AFURLSessionManagerTaskDelegate通过task的内存地址去绑定，通过AFURLSessionManagerTaskDelegate来管理那个task的回调。部分session的delegate也会在AFURLSessionManagerTaskDelegate来处理。

```
- (NSURLSessionDataTask *)dataTaskWithRequest:(NSURLRequest *)request
                             uploadProgress:(nullable void (^)(NSProgress
*uploadProgress)) uploadProgressBlock
                           downloadProgress:(nullable void (^)(NSProgress
*downloadProgress)) downloadProgressBlock
                          completionHandler:(nullable void (^)(NSURLResponse
*response, id _Nullable responseObject,  NSError * _Nullable
error))completionHandler {

    __block NSURLSessionDataTask *dataTask = nil;
    //通过一个生成任务的队列
    url_session_manager_create_task_safely(^{
        ///生成一个任务
        dataTask = [self.session dataTaskWithRequest:request];
    });

    [self addDelegateForDataTask:dataTask
uploadProgress:uploadProgressBlock
```

```
downloadProgress:downloadProgressBlock
completionHandler:completionHandler];

    return dataTask;
}
/**
 *  将dataTask和一个 AFURLSessionManagerTaskDelegate 关联起来,将其中部分progressBlock交给delegate去管理
 */
- (void)addDelegateForDataTask:(NSURLSessionDataTask *)dataTask
             uploadProgress:(nullable void (^)(NSProgress *uploadProgress)) uploadProgressBlock
           downloadProgress:(nullable void (^)(NSProgress *downloadProgress)) downloadProgressBlock
          completionHandler:(void (^)(NSURLResponse *response, id responseObject, NSError *error))completionHandler
{
    AFURLSessionManagerTaskDelegate *delegate = [[AFURLSessionManagerTaskDelegate alloc] init];
    delegate.manager = self;
    delegate.completionHandler = completionHandler;
    //用来标识task唯一性!!!!用的task的内存地址,将dataTask <--> delegate helper关联起来
    dataTask.taskDescription = self.taskDescriptionForSessionTasks;
    [self setDelegate:delegate forTask:dataTask];

    // NSProgress相关的Block都是由delegate helper去更新的
    delegate.uploadProgressBlock = uploadProgressBlock;
    delegate.downloadProgressBlock = downloadProgressBlock;
}
```

3、共使用了2种锁来。
第一种NSLock，这个锁用于mutableTaskDelegatesKeyedByTaskIdentifier的存取操作。mutableTaskDelegatesKeyedByTaskIdentifier保存了task和delegate

helper的健值对。因为task是在后台进行的。而mutableTaskDelegatesKeyedByTaskIdentifier是manager的一个全局对象。所以当在task回调中操作mutableTaskDelegatesKeyedByTaskIdentifier时要对其进行加锁。

第二种是信号量，信号量是用在获取正在运行的人task任务列表。

///这里使用了信号量，因为这些任务都在后台进行。要在后台拿到这些任务，而拿到后台任务是一个getTasksWithCompletionHandler回调，所以通过信号量去控制等待那个回调返回的结果

```
- (NSArray *)tasksForKeyPath:(NSString *)keyPath {
    __block NSArray *tasks = nil;
    ///创建信号量
    dispatch_semaphore_t semaphore = dispatch_semaphore_create(0);
    [self.session getTasksWithCompletionHandler:^(NSArray *dataTasks,
NSArray *uploadTasks, NSArray *downloadTasks) {
        if ([keyPath
isEqualToString:NSStringFromSelector(@selector(dataTasks))]) {
            tasks = dataTasks;
        } else if ([keyPath
isEqualToString:NSStringFromSelector(@selector(uploadTasks))]) {
            tasks = uploadTasks;
        } else if ([keyPath
isEqualToString:NSStringFromSelector(@selector(downloadTasks))]) {
            tasks = downloadTasks;
        } else if ([keyPath
isEqualToString:NSStringFromSelector(@selector(tasks))]) {
            tasks = [@[dataTasks, uploadTasks, downloadTasks]
valueForKeyPath:@"@unionOfArrays.self"];
        }
        ///回调返回过来了，释放信号量
        dispatch_semaphore_signal(semaphore);
    }];
    ///信号量-1，由于创建时信号量为0，所以等待
    dispatch_semaphore_wait(semaphore, DISPATCH_TIME_FOREVER);
```

```
    return tasks;
}
```