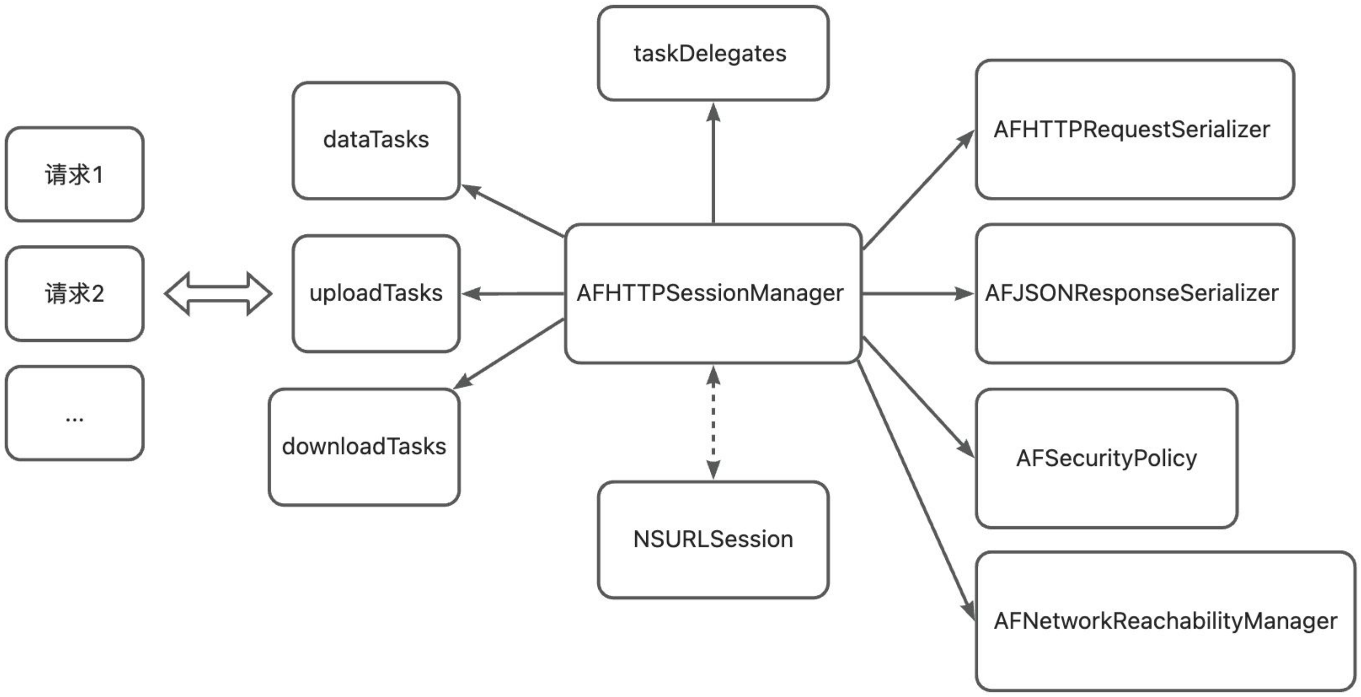
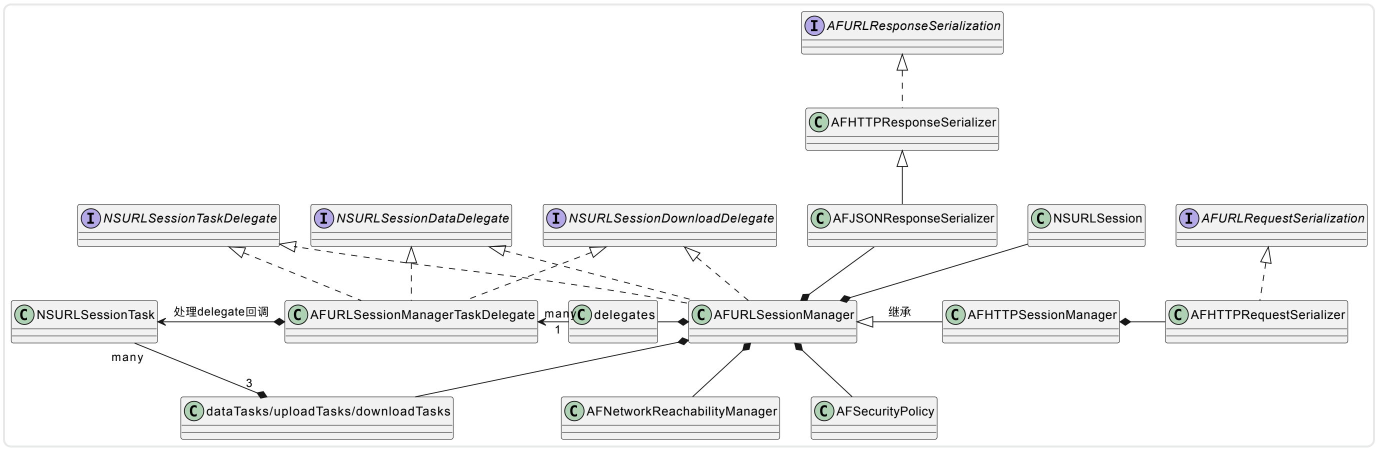
AFNetworking





delegate处理

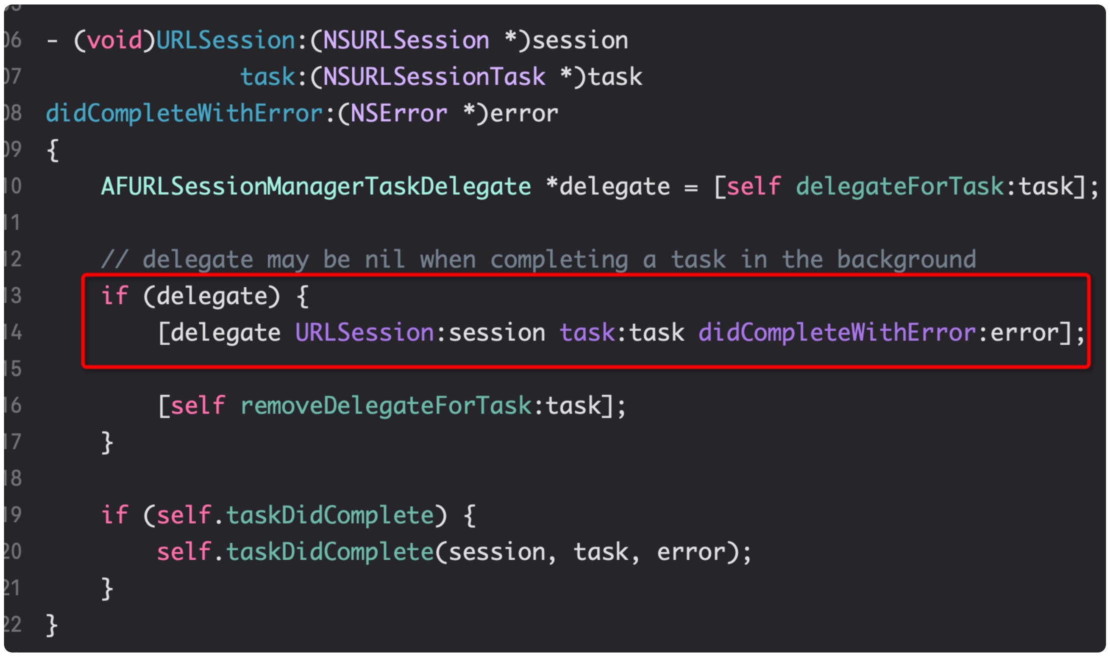
根据URLSessionTask的delegate属性的注释

> /* Sets a task-specific delegate. Methods not implemented on this delegate will
>
> * still be forwarded to the session delegate.
>
> *

> * Cannot be modified after task resumes. Not supported on background session.
>
> *
>
> * Delegate is strongly referenced until the task completes, after which it is
>
> * reset to `nil`.
>
> */

URLSessionTask的delegate回调会优先找自身的delegate，如果不存在，就会去调用URLSession的delegate

因此，框架中所有task的回调都会先在sessionManager层的delegate触发，再由sessionManager层从mutableTaskDelegatesKeyedByTaskIdentifier 找到 task 对应的 taskDelegate，然后触发taskDelegate 的回调方法。



```
- (void)URLSession:(NSURLSession *)session
              task:(NSURLSessionTask *)task
didCompleteWithError:(NSError *)error
{
    AFURLSessionManagerTaskDelegate *delegate = [self delegateForTask:task];

    // delegate may be nil when completing a task in the background
    if (delegate) {
        [delegate URLSession:session task:task didCompleteWithError:error];

        [self removeDelegateForTask:task];
    }

    if (self.taskDidComplete) {
        self.taskDidComplete(session, task, error);
    }
}
```

task进度管理

delegate => update NSProgress => block call

进度关联

taskDelegate 维护了两个 NSProgress，用来管理下载和上传的进度

```
    __weak __typeof__(task) weakTask = task;
    for (NSProgress *progress in @[ _uploadProgress, _downloadProgress ])
    {
        progress.totalUnitCount = NSURLSessionTransferSizeUnknown;
        progress.cancellable = YES;
        progress.cancellationHandler = ^{
            [weakTask cancel];
        };
        progress.pausable = YES;
        progress.pausingHandler = ^{
            [weakTask suspend];
        };
#if AF_CAN_USE_AT_AVAILABLE
        if (@available(macOS 10.11, *))
#else
        if ([progress respondsToSelector:@selector(setResumingHandler:)])
#endif
        {
            progress.resumingHandler = ^{
                [weakTask resume];
            };
        }

        [progress addObserver:self
                   forKeyPath:NSStringFromSelector(@selector(fractionCompleted))
                      options:NSKeyValueObservingOptionNew
                      context:NULL];
    }
```

进度更新

在各个delegate回调方法中更新NSProgress进度，例如

```
#pragma mark - NSURLSessionDataDelegate

- (void)URLSession:(__unused NSURLSession *)session
          dataTask:(__unused NSURLSessionDataTask *)dataTask
    didReceiveData:(NSData *)data
{
    self.downloadProgress.totalUnitCount = dataTask.countOfBytesExpectedToReceive;
    self.downloadProgress.completedUnitCount = dataTask.countOfBytesReceived;

    [self.mutableData appendData:data];
}

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
   didSendBodyData:(int64_t)bytesSent
    totalBytesSent:(int64_t)totalBytesSent
totalBytesExpectedToSend:(int64_t)totalBytesExpectedToSend{

    self.uploadProgress.totalUnitCount = task.countOfBytesExpectedToSend;
    self.uploadProgress.completedUnitCount = task.countOfBytesSent;
}
```

KVO监听

并且通过KVO监听NSProgress的进度更新，从而触发调用对应block

```
#pragma mark - NSProgress Tracking

- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary<NSString *,id> *)change
    context:(void *)context {
   if ([object isEqual:self.downloadProgress]) {
        if (self.downloadProgressBlock) {
            self.downloadProgressBlock(object);
        }
    }
    else if ([object isEqual:self.uploadProgress]) {
        if (self.uploadProgressBlock) {
            self.uploadProgressBlock(object);
        }
    }
}
```

其他

多读单写

```
- (NSDictionary *)HTTPRequestHeaders {
    // 多读单写 - 同步读，读完再退出
    NSDictionary __block *value;
    dispatch_sync(self.requestHeaderModificationQueue, ^{
        value = [NSDictionary dictionaryWithDictionary:self.mutableHTTPRequestHeaders];
    });
    return value;
}

- (void)setValue:(NSString *)value
forHTTPHeaderField:(NSString *)field
{
    // 多读单写 - 同步栅栏写，确保之前的所有读写都完成了，才能开始写
    // 理论上可以异步栅栏写，不需要写完才返回，不知道为什么这里用了同步栅栏
    dispatch_barrier_sync(self.requestHeaderModificationQueue, ^{
        [self.mutableHTTPRequestHeaders setValue:value forKey:field];
    });
}
```