OHHTTPStubs+OCMock分享

郁正杰

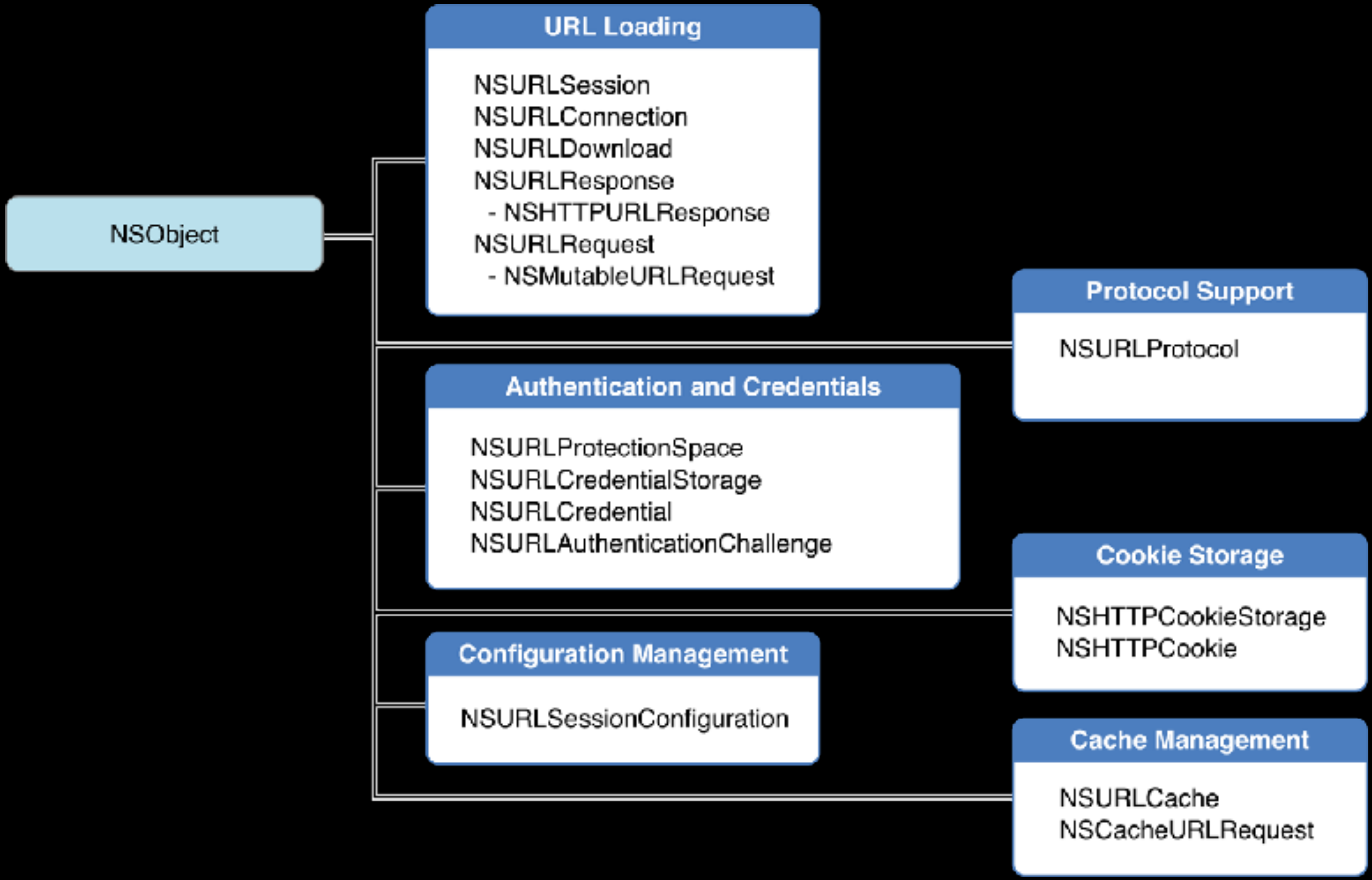
OHHTTPStubs

- OHHTTPStubs实现了在不需要后台Api的情况下，对本地HTTP请求进行拦截，返回想要的JSON数据。

OHHTTPStubs原理

- 使用NSURLProtocol拦截HTTP请求，伪造HTTP响应。
- 在一个 HTTP 请求开始时，URL 加载系统创建一个合适的 NSURLProtocol 对象处理对应的 URL 请求。我们需要做的就是写一个继承自 NSURLProtocol 的类，并且注册我们的协议类，然后 URL 加载系统就会在请求发出时使用我们创建的协议对象对该请求进行处理。

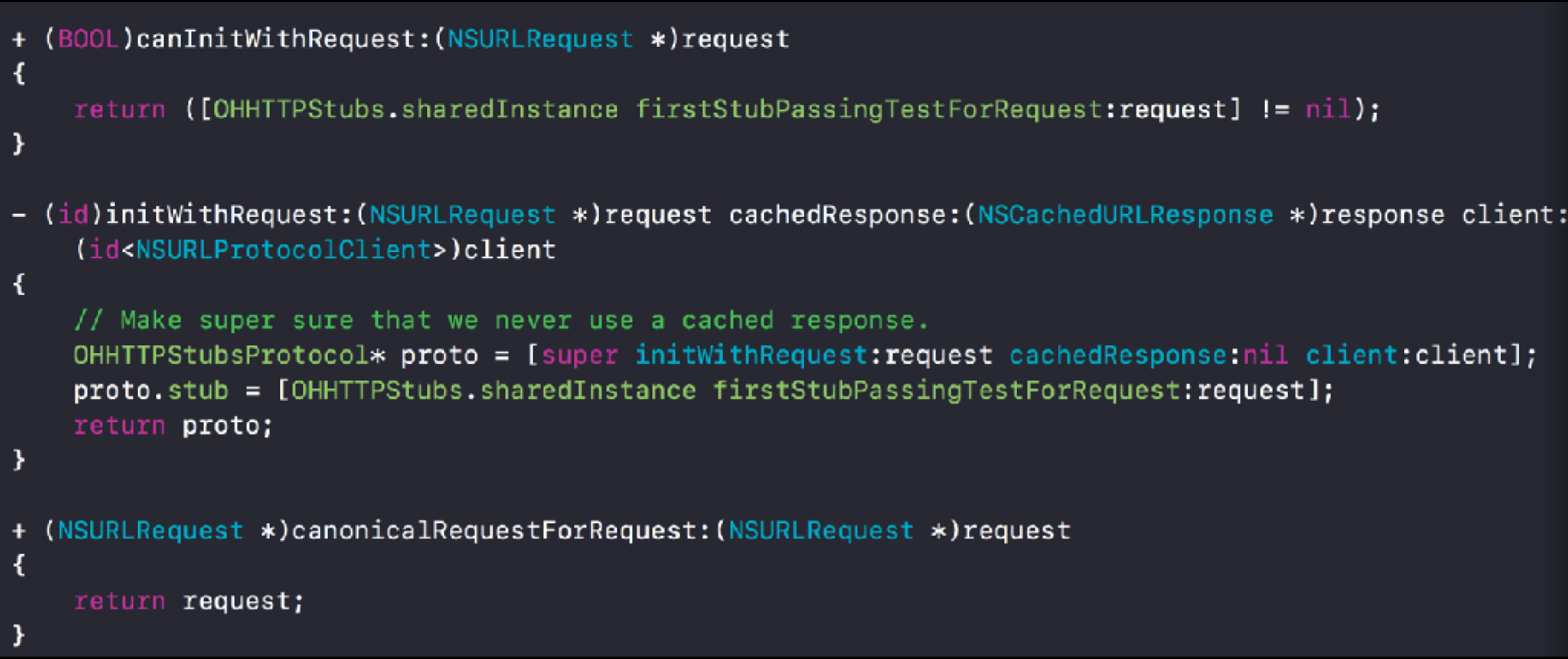
OHHTTPStubs实现



OHHTTPStubs实现

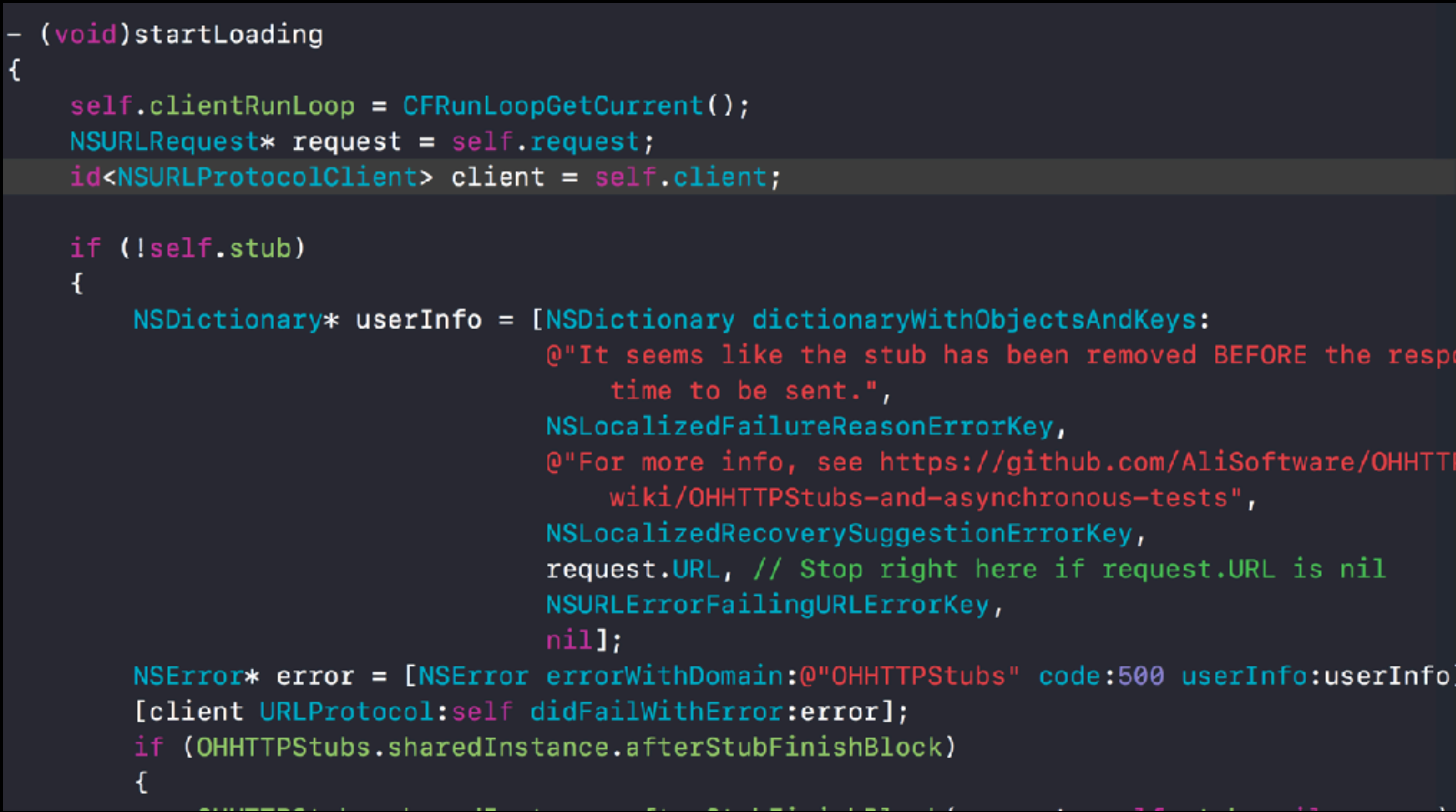
- OHHTTPStubsProtocol 拦截HTTP请求；
- OHHTTPStubs 单例管理OHHTTPStubsDescriptor实例；
- OHHTTPStubsResponse 伪造HTTP响应；
- 其它；

拦截HTTP请求



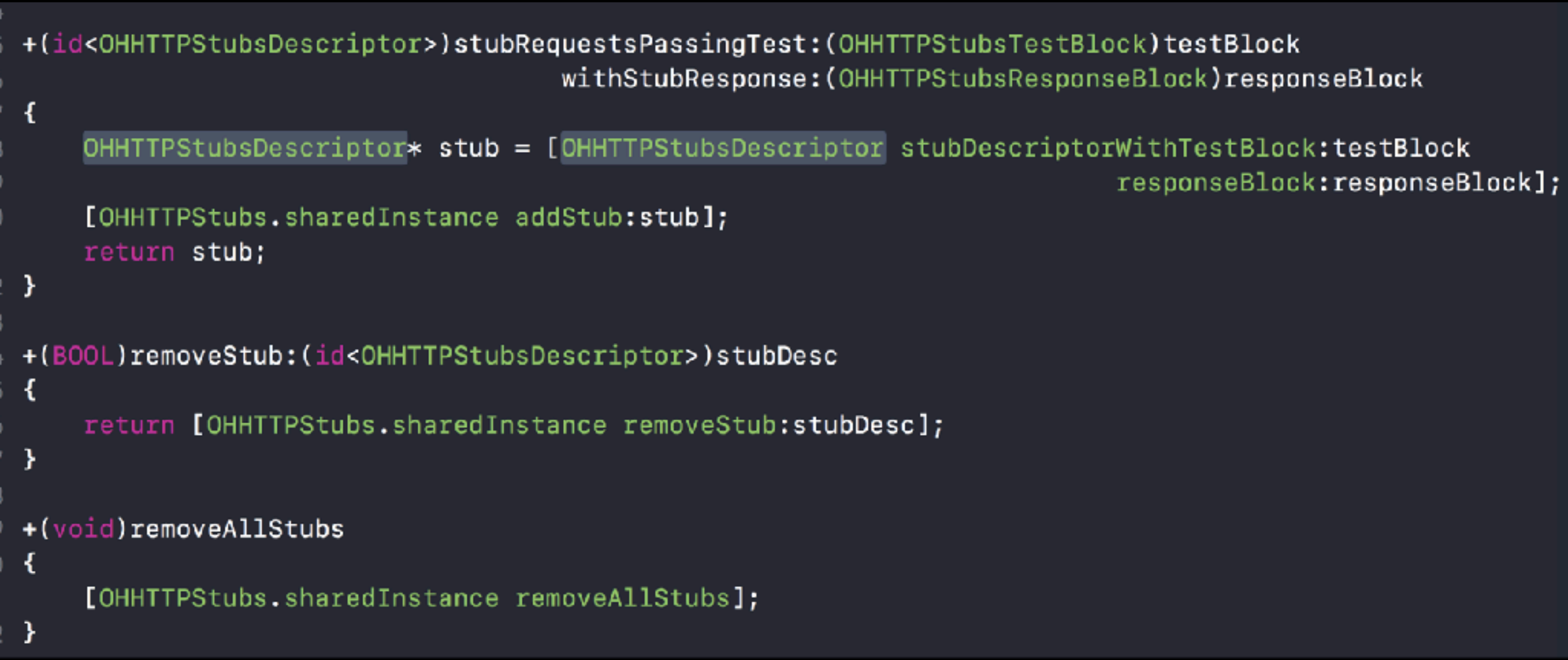
```
+ (BOOL)canInitWithRequest:(NSURLRequest *)request
{
    return ([OHHTTPStubs.sharedInstance firstStubPassingTestForRequest:request] != nil);
}

- (id)initWithRequest:(NSURLRequest *)request cachedResponse:(NSCachedURLResponse *)response client:
    (id<NSURLProtocolClient>)client
{
    // Make super sure that we never use a cached response.
    OHHTTPStubsProtocol* proto = [super initWithRequest:request cachedResponse:nil client:client];
    proto.stub = [OHHTTPStubs.sharedInstance firstStubPassingTestForRequest:request];
    return proto;
}

+ (NSURLRequest *)canonicalRequestForRequest:(NSURLRequest *)request
{
    return request;
}
```

处理HTTP请求



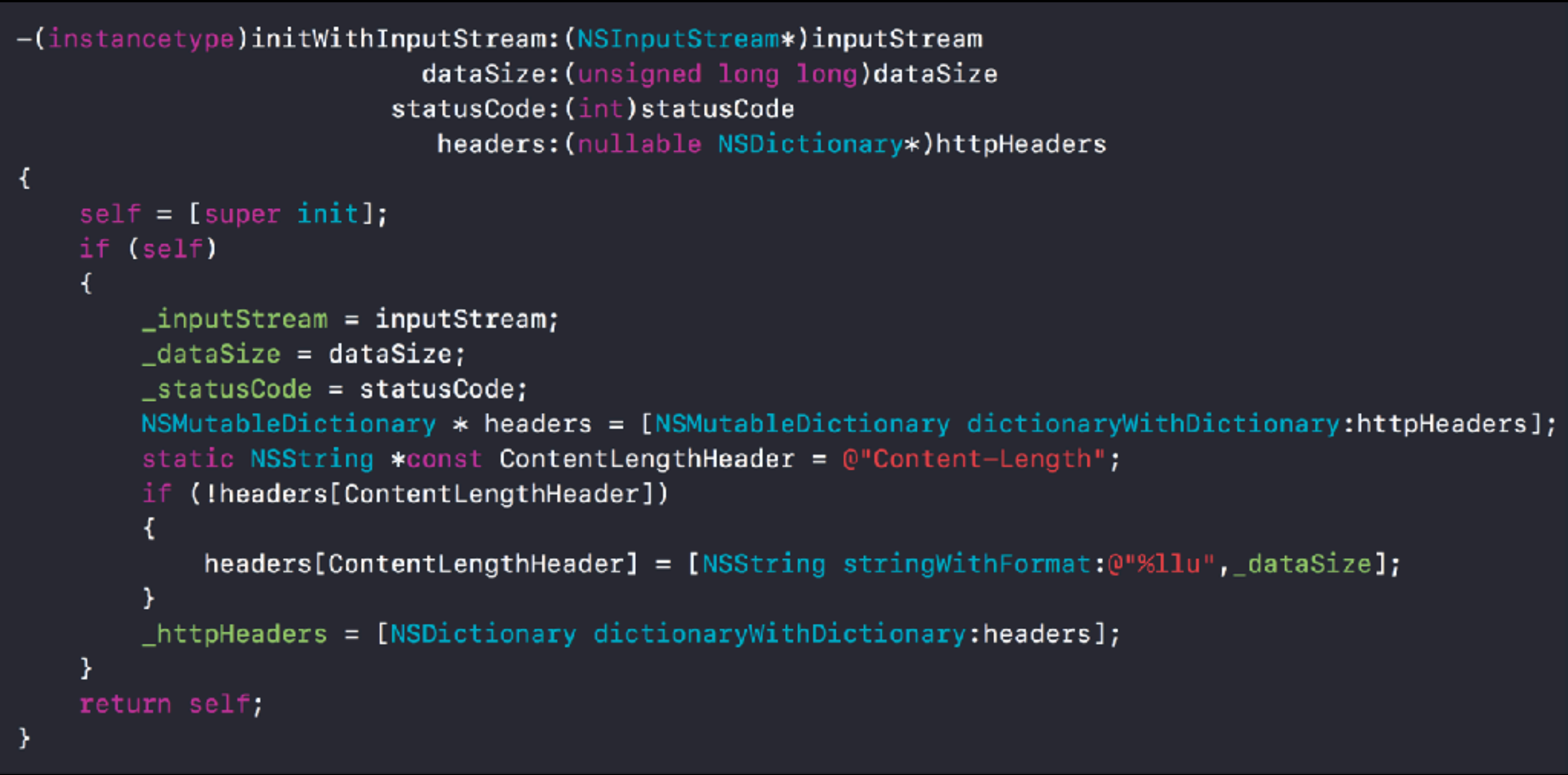
```
- (void)startLoading
{
    self.clientRunLoop = CFRunLoopGetCurrent();
    NSURLRequest* request = self.request;
    id<NSURLProtocolClient> client = self.client;

    if (!self.stub)
    {
        NSDictionary* userInfo = [NSDictionary dictionaryWithObjectsAndKeys:
                                  @"It seems like the stub has been removed BEFORE the resp
                                      time to be sent.",
                                  NSLocalizedFailureReasonErrorKey,
                                  @"For more info, see https://github.com/AliSoftware/OHHTT
                                      wiki/OHHTTPStubs-and-asynchronous-tests",
                                  NSLocalizedRecoverySuggestionErrorKey,
                                  request.URL, // Stop right here if request.URL is nil
                                  NSURLErrorFailingURLErrorKey,
                                  nil];
        NSError* error = [NSError errorWithDomain:@"OHHTTPStubs" code:500 userInfo:userInfo
        [client URLProtocol:self didFailWithError:error];
        if (OHHTTPStubs.sharedInstance.afterStubFinishBlock)
        {
```

stub管理



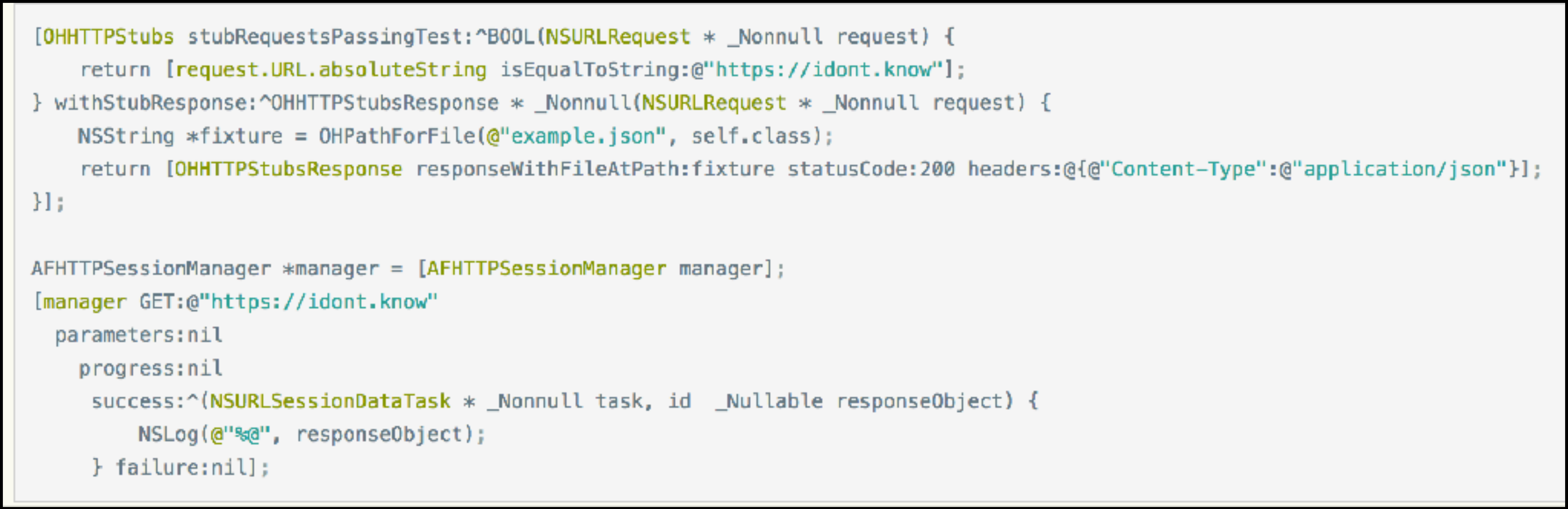
```
+(id<OHHTTPStubsDescriptor>)stubRequestsPassingTest:(OHHTTPStubsTestBlock)testBlock
                                   withStubResponse:(OHHTTPStubsResponseBlock)responseBlock
{
    OHHTTPStubsDescriptor* stub = [OHHTTPStubsDescriptor stubDescriptorWithTestBlock:testBlock
                                                                       responseBlock:responseBlock];
    [OHHTTPStubs.sharedInstance addStub:stub];
    return stub;
}

+(BOOL)removeStub:(id<OHHTTPStubsDescriptor>)stubDesc
{
    return [OHHTTPStubs.sharedInstance removeStub:stubDesc];
}

+(void)removeAllStubs
{
    [OHHTTPStubs.sharedInstance removeAllStubs];
}
```

OHHTTPStubsResponse



```
-(instancetype)initWithInputStream:(NSInputStream*)inputStream
                          dataSize:(unsigned long long)dataSize
                        statusCode:(int)statusCode
                           headers:(nullable NSDictionary*)httpHeaders
{
    self = [super init];
    if (self)
    {
        _inputStream = inputStream;
        _dataSize = dataSize;
        _statusCode = statusCode;
        NSMutableDictionary * headers = [NSMutableDictionary dictionaryWithDictionary:httpHeaders];
        static NSString *const ContentLengthHeader = @"Content-Length";
        if (!headers[ContentLengthHeader])
        {
            headers[ContentLengthHeader] = [NSString stringWithFormat:@"%llu",_dataSize];
        }
        _httpHeaders = [NSDictionary dictionaryWithDictionary:headers];
    }
    return self;
}
```

OHHTTPStubs使用



```
[OHHTTPStubs stubRequestsPassingTest:^BOOL(NSURLRequest * _Nonnull request) {
    return [request.URL.absoluteString isEqualToString:@"https://idont.know"];
} withStubResponse:^OHHTTPStubsResponse * _Nonnull(NSURLRequest * _Nonnull request) {
    NSString *fixture = OHPathForFile(@"example.json", self.class);
    return [OHHTTPStubsResponse responseWithFileAtPath:fixture statusCode:200 headers:@{@"Content-Type":@"application/json"}];
}];

AFHTTPSessionManager *manager = [AFHTTPSessionManager manager];
[manager GET:@"https://idont.know"
  parameters:nil
    progress:nil
     success:^(NSURLSessionDataTask * _Nonnull task, id  _Nullable responseObject) {
         NSLog(@"%@", responseObject);
     } failure:nil];
```

OCMock

- Mock测试：对于一些不容易构造或不容易获取的对象，创建一个虚拟的对象（mock object）来完成测试。

- OCMock：OCMock是一个用于iOS项目配置Mock测试的开源项目。

- 场景：网络请求等。

OCMock原理

其实现思想是根据要mock的对象的class来创建一个对应的对象，并且设置好该对象的属性和调用预定方法后的动作（例如返回一个值，调用代码块，发送消息，抛出异常等等），并将其记录到一个数组中，然后开发者主动调用该方法，最后做一个verify（验证），从而判断该方法是否被调用，或者调用过程中是否抛出异常等。

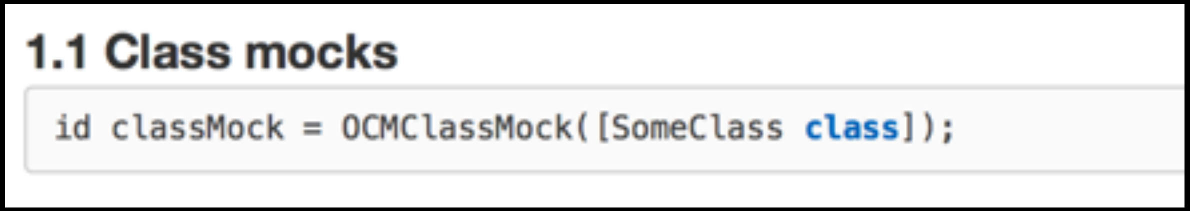
OCMock

- Mock测试：对于一些不容易构造或不容易获取的对象，创建一个虚拟的对象（mock object）来完成测试。

- OCMock：OCMock是一个用于iOS项目配置Mock测试的开源项目。

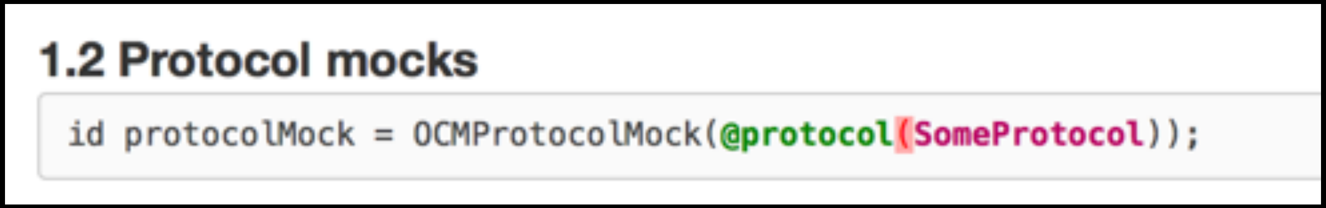
- 场景：网络请求等。

1、创建mock对象



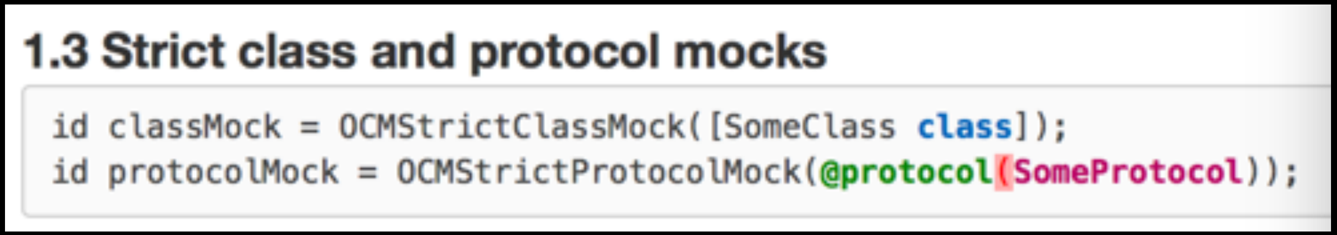
1.1 Class mocks

```
id classMock = OCMClassMock([SomeClass class]);
```



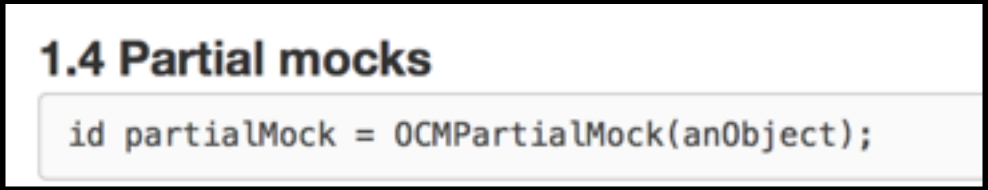
1.2 Protocol mocks

```
id protocolMock = OCMProtocolMock(@protocol(SomeProtocol));
```



1.3 Strict class and protocol mocks

```
id classMock = OCMStrictClassMock([SomeClass class]);
id protocolMock = OCMStrictProtocolMock(@protocol(SomeProtocol));
```



1.4 Partial mocks

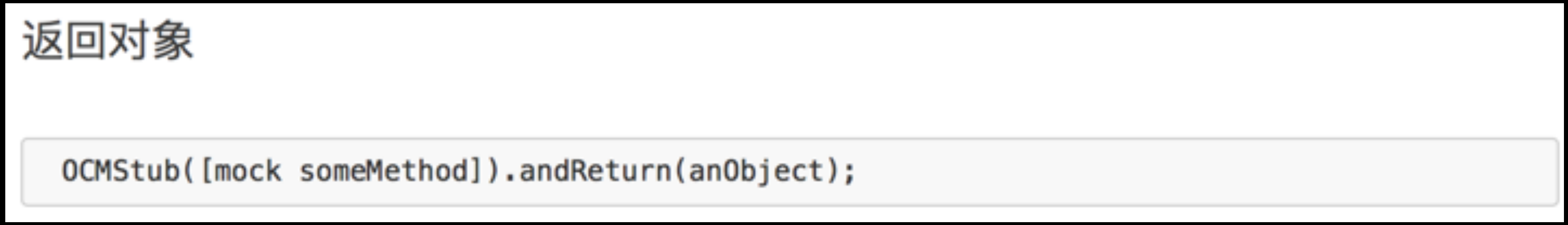
```
id partialMock = OCMPartialMock(anObject);
```

2、stubbing methods

- stub方法，即对方法进行预测，如设置返回值，设置调用逻辑，抛出异常等等。设置stub后，调用函数时，原函数不再执行,而以stub的逻辑来进行交互。

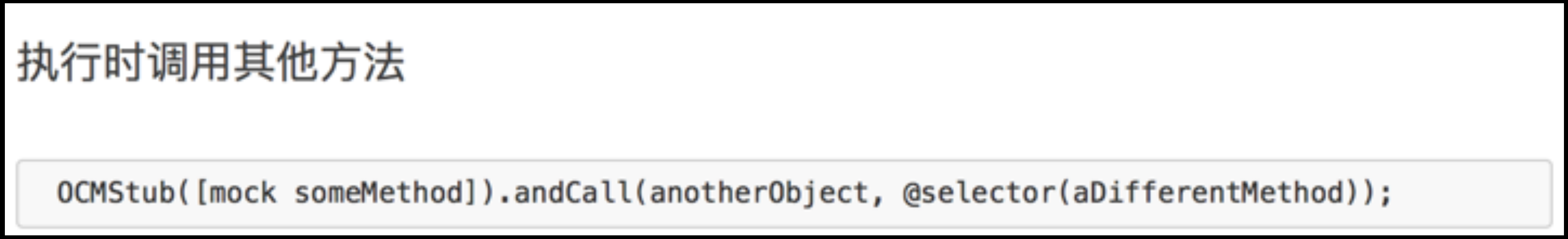
- 直接设置返回值、调用其它方法、调用block、抛出一个异常、发送一个消息等

2、stubbing methods



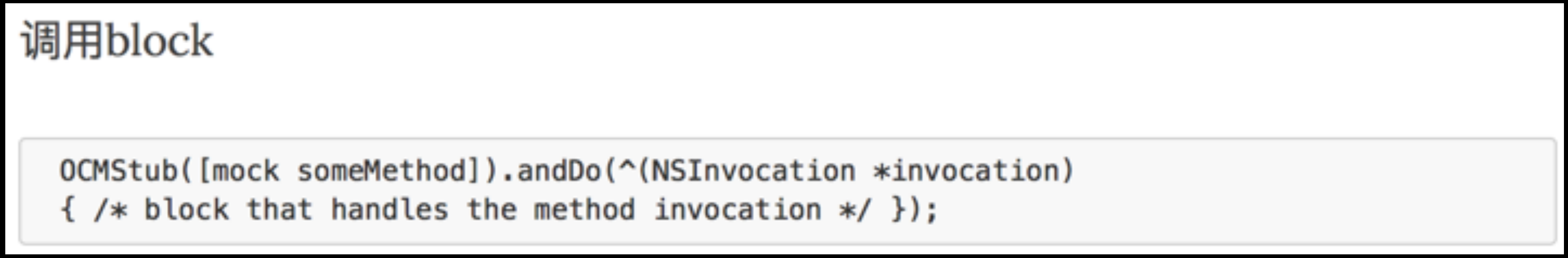
返回对象

```
OCMStub([mock someMethod]).andReturn(anObject);
```



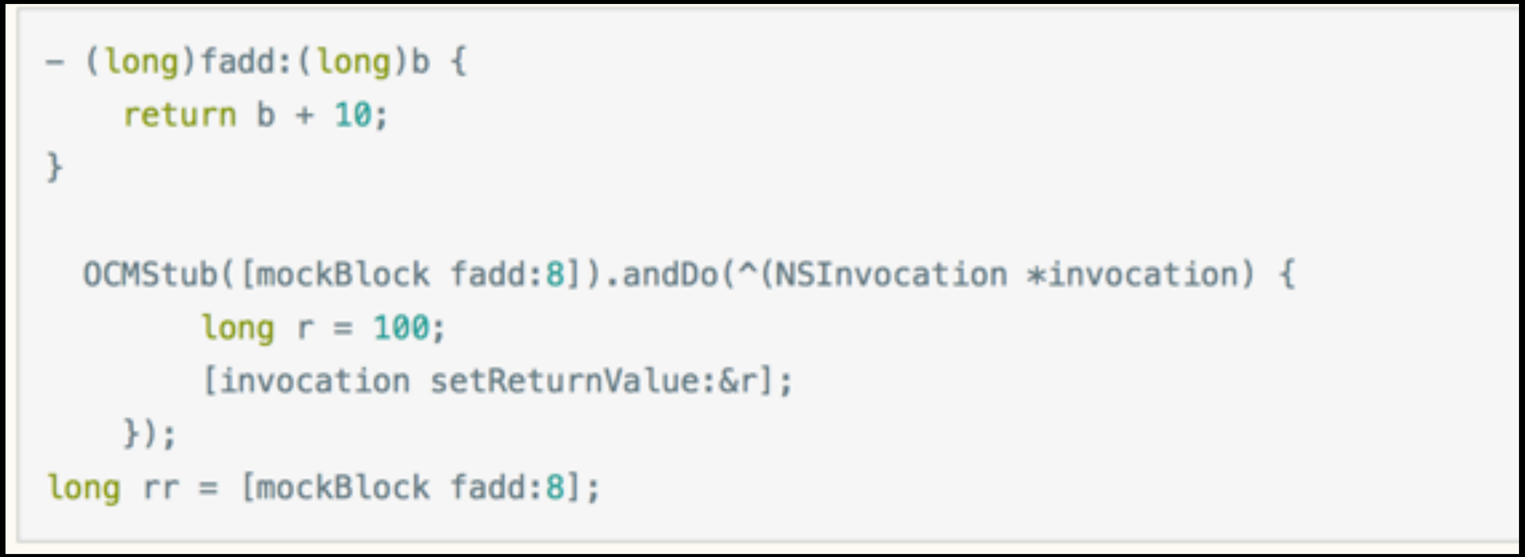
执行时调用其他方法

```
OCMStub([mock someMethod]).andCall(anotherObject, @selector(aDifferentMethod));
```



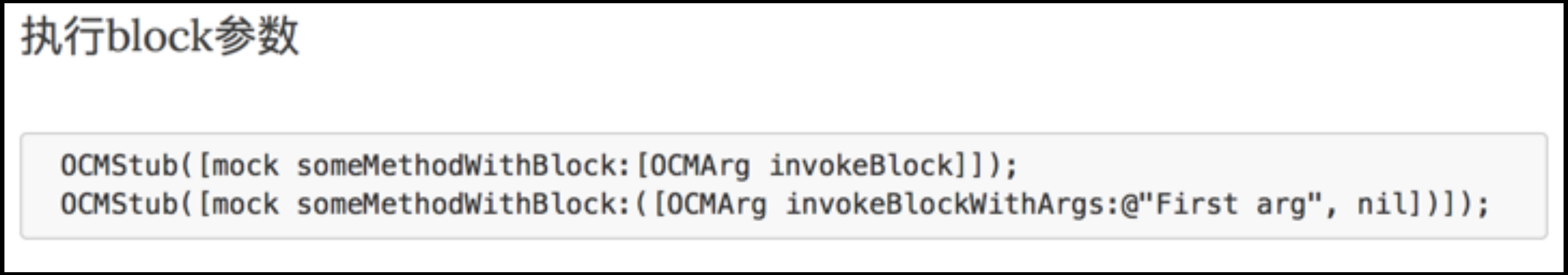
调用block

```
OCMStub([mock someMethod]).andDo(^(NSInvocation *invocation)
{ /* block that handles the method invocation */ });
```



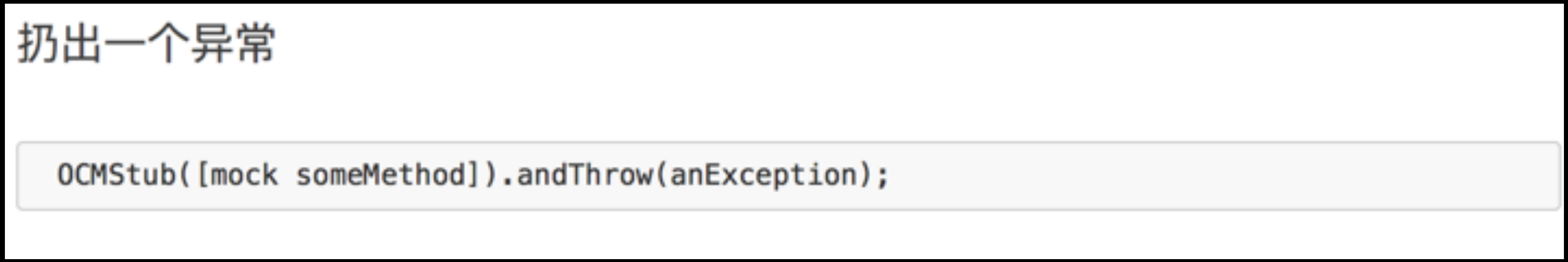
```
- (long)fadd:(long)b {
    return b + 10;
}

  OCMStub([mockBlock fadd:8]).andDo(^(NSInvocation *invocation) {
        long r = 100;
        [invocation setReturnValue:&r];
    });
long rr = [mockBlock fadd:8];
```

2、stubbing methods



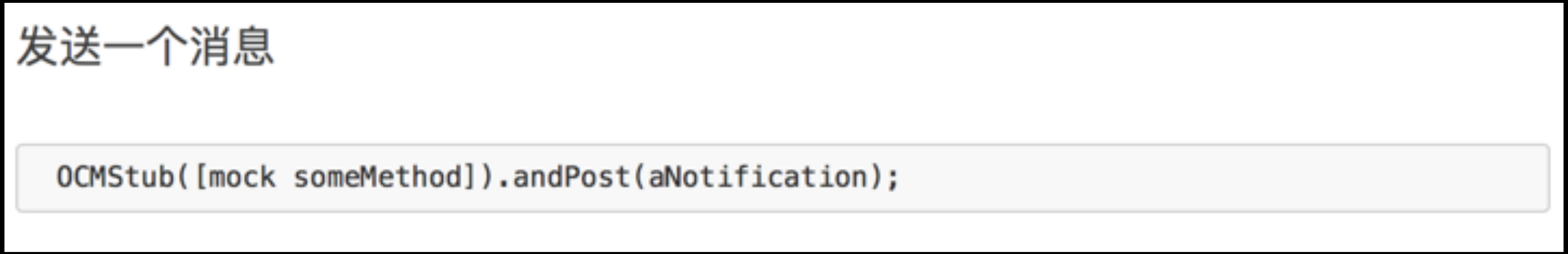
执行block参数

```
OCMStub([mock someMethodWithBlock:[OCMArg invokeBlock]]);
OCMStub([mock someMethodWithBlock:([OCMArg invokeBlockWithArgs:@"First arg", nil])]);
```



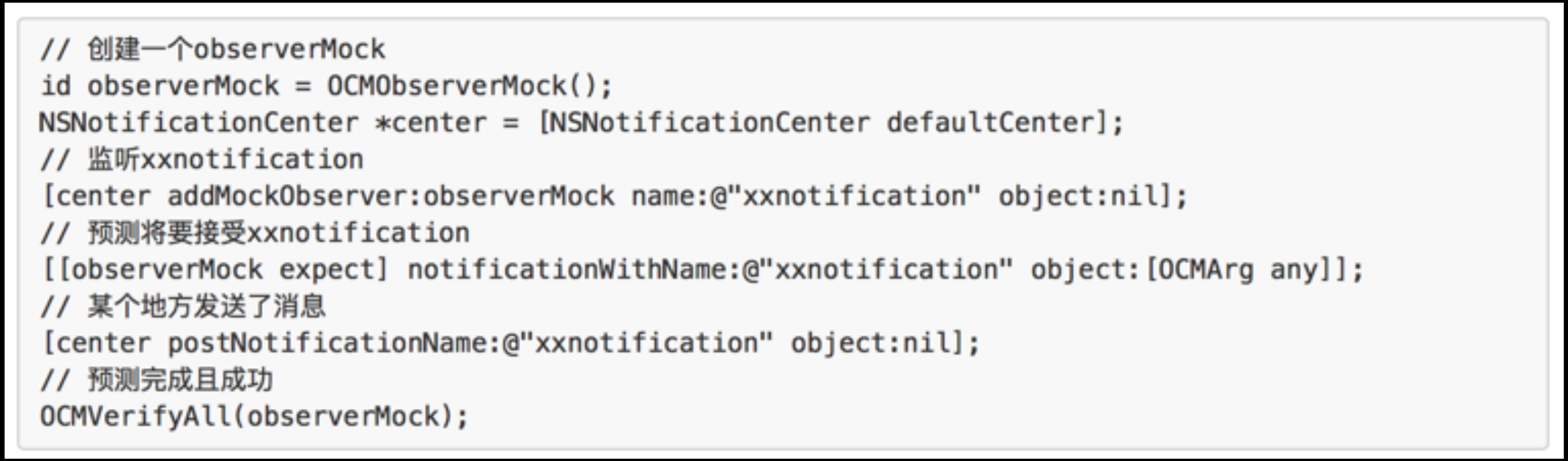
扔出一个异常

```
OCMStub([mock someMethod]).andThrow(anException);
```



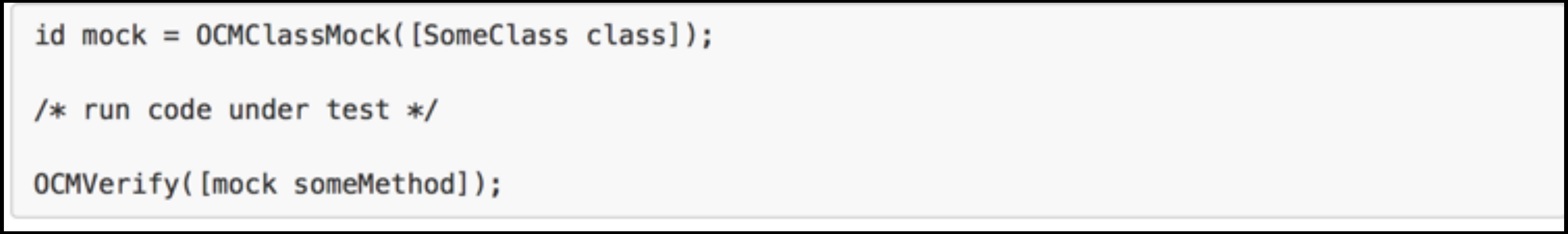
发送一个消息

```
OCMStub([mock someMethod]).andPost(aNotification);
```

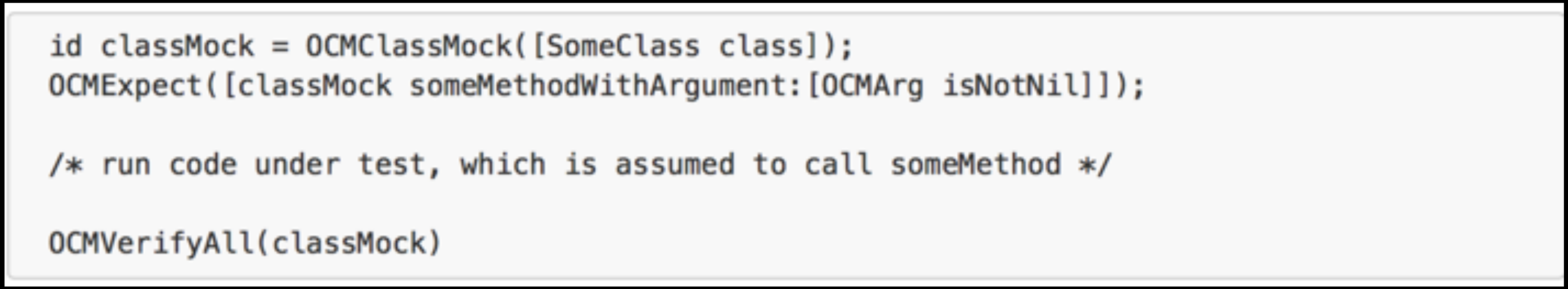


```
// 创建一个observerMock
id observerMock = OCMObserverMock();
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
// 监听xxnotification
[center addMockObserver:observerMock name:@"xxnotification" object:nil];
// 预测将要接受xxnotification
[[observerMock expect] notificationWithName:@"xxnotification" object:[OCMArg any]];
// 某个地方发送了消息
[center postNotificationName:@"xxnotification" object:nil];
// 预测完成且成功
OCMVerifyAll(observerMock);
```

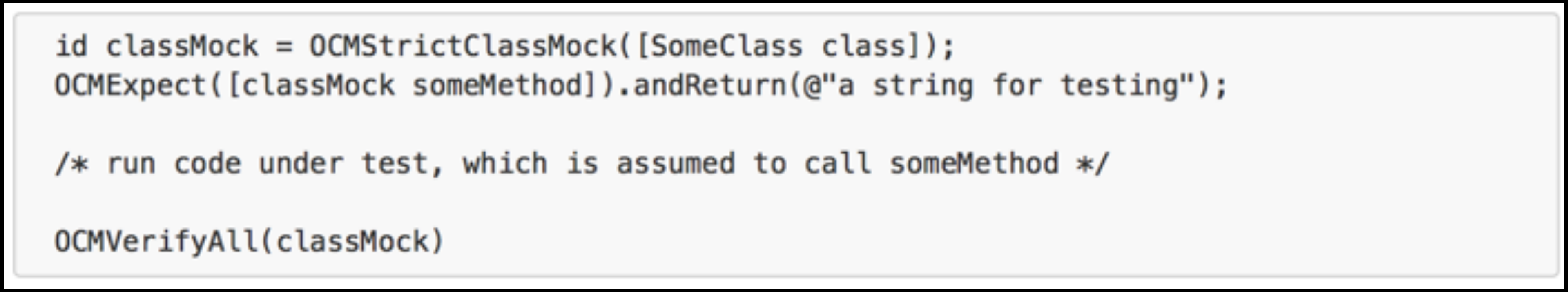
3、验证方法的调用



```
id mock = OCMClassMock([SomeClass class]);

/* run code under test */

OCMVerify([mock someMethod]);
```

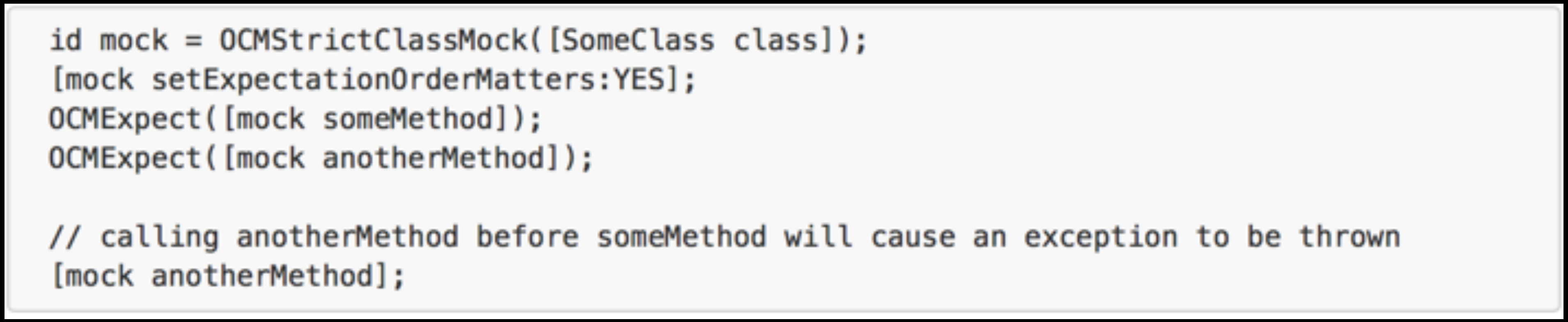


```
id classMock = OCMClassMock([SomeClass class]);
OCMExpect([classMock someMethodWithArgument:[OCMArg isNotNil]]);

/* run code under test, which is assumed to call someMethod */

OCMVerifyAll(classMock)
```

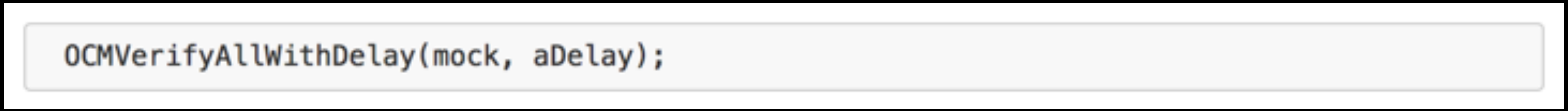


```
id classMock = OCMStrictClassMock([SomeClass class]);
OCMExpect([classMock someMethod]).andReturn(@"a string for testing");

/* run code under test, which is assumed to call someMethod */

OCMVerifyAll(classMock)
```

3、验证方法的调用



```
id mock = OCMStrictClassMock([SomeClass class]);
[mock setExpectationOrderMatters:YES];
OCMExpect([mock someMethod]);
OCMExpect([mock anotherMethod]);

// calling anotherMethod before someMethod will cause an exception to be thrown
[mock anotherMethod];
```

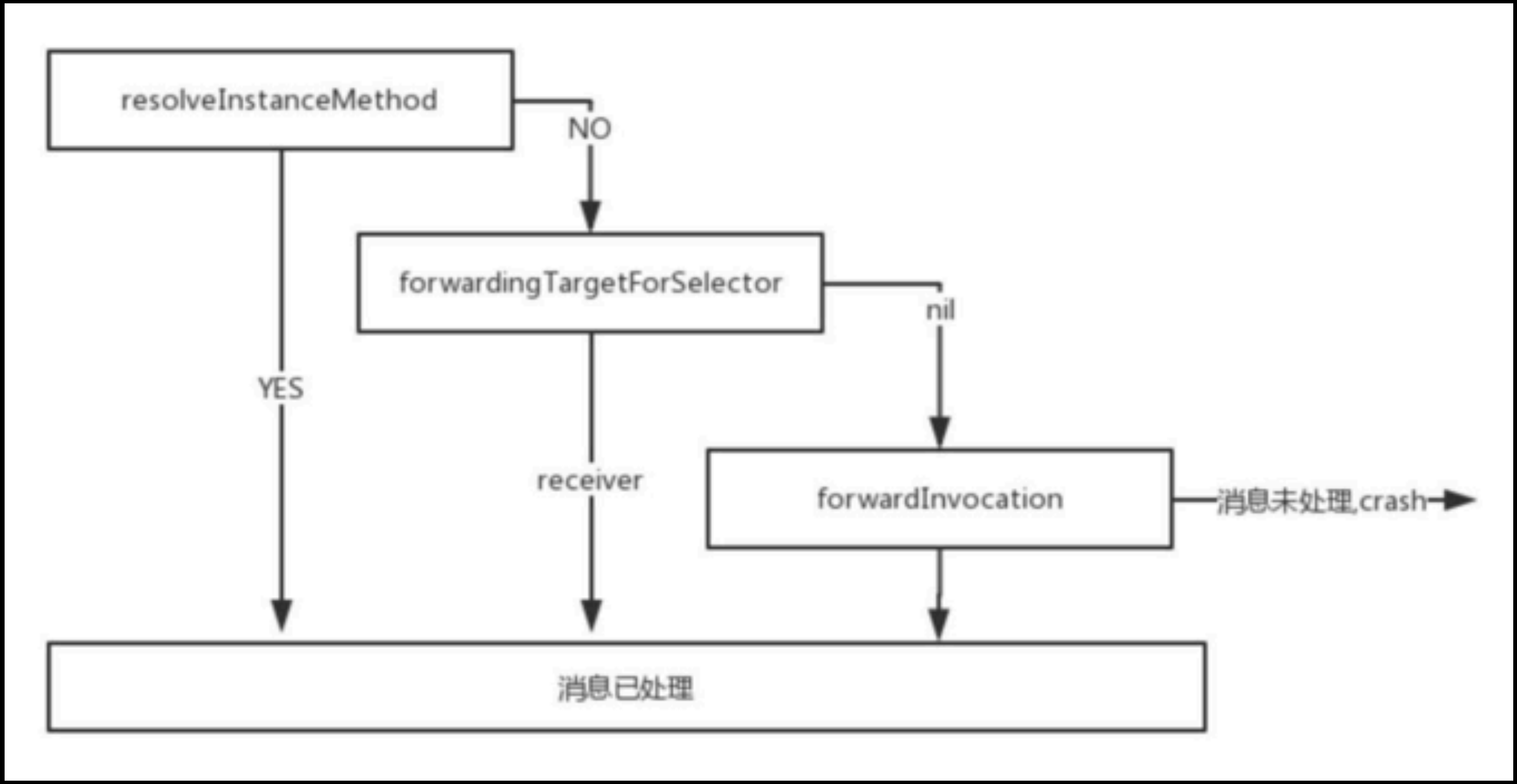


```
OCMVerifyAllWithDelay(mock, aDelay);
```

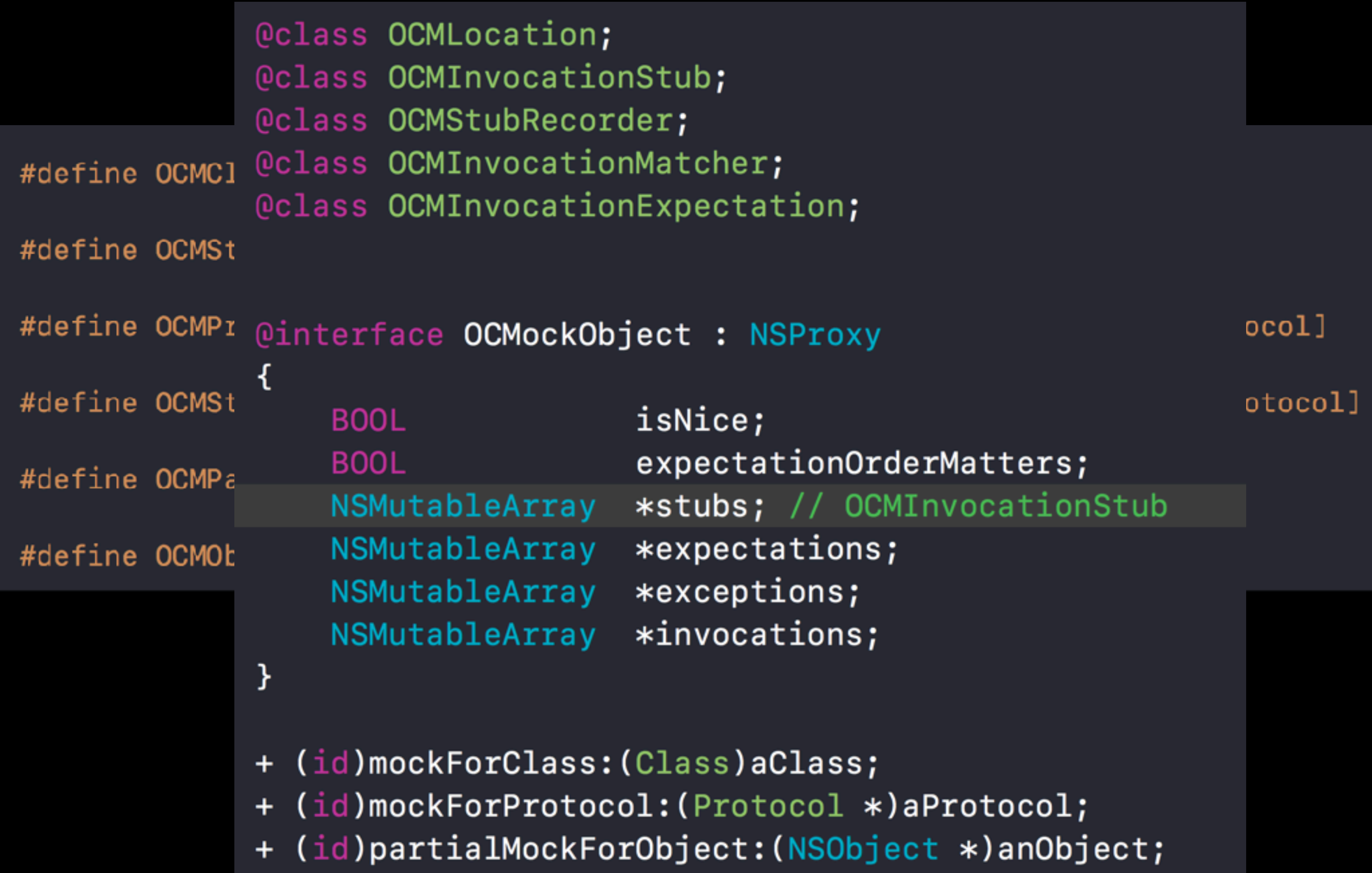
OCMock

- 消息转发机制
- NSProxy
- Runtime

OC消息转发机制


resolveInstanceMethod
NO
YES
forwardingTargetForSelector
nil
receiver
forwardInvocation
消息未处理,crash
消息已处理

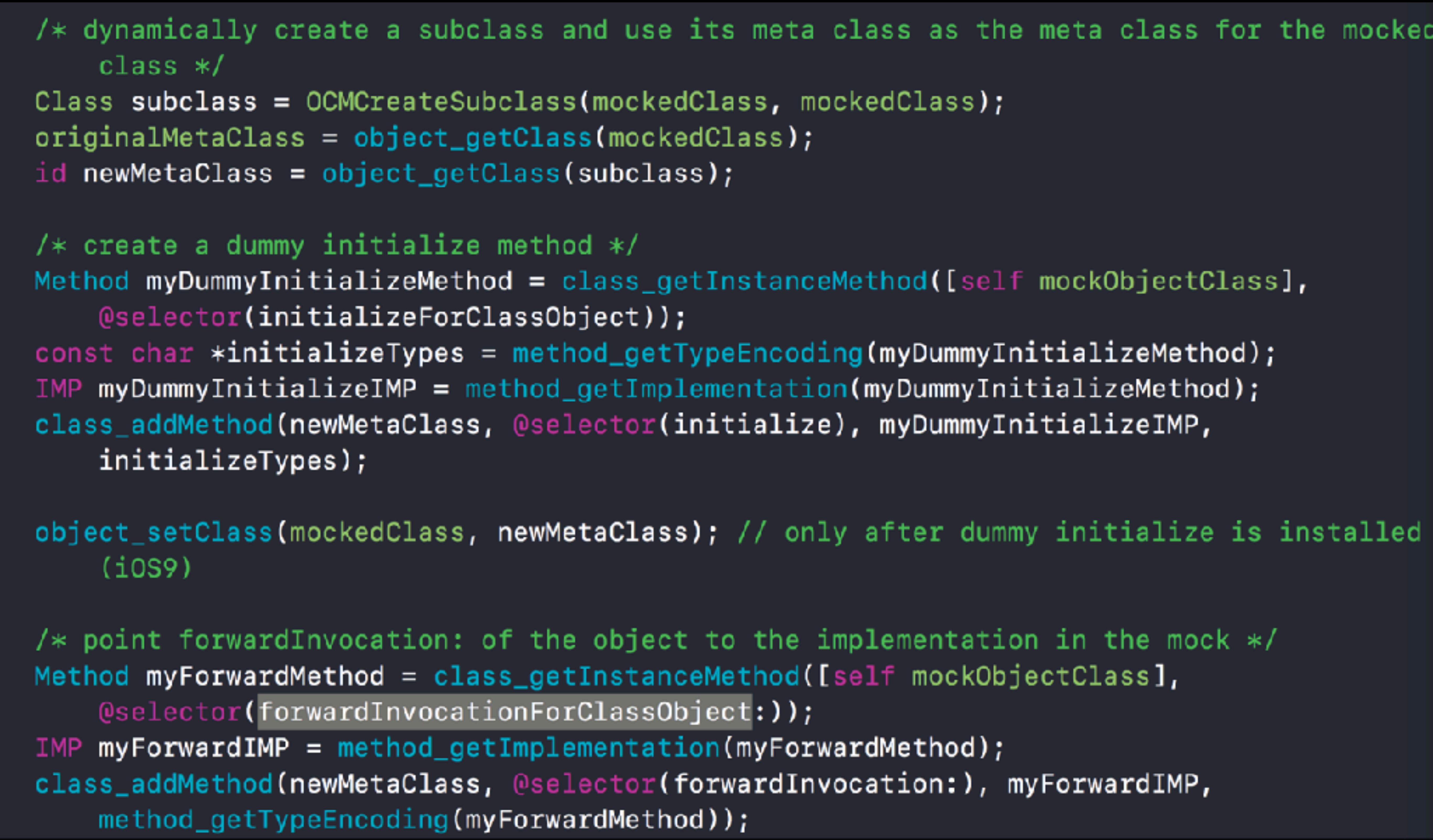
OCMock实现原理



```
#define OCMCl
#define OCMSt
#define OCMPr                                   ocol]
#define OCMSt                                   otocol]
#define OCMPa
#define OCMOb
```

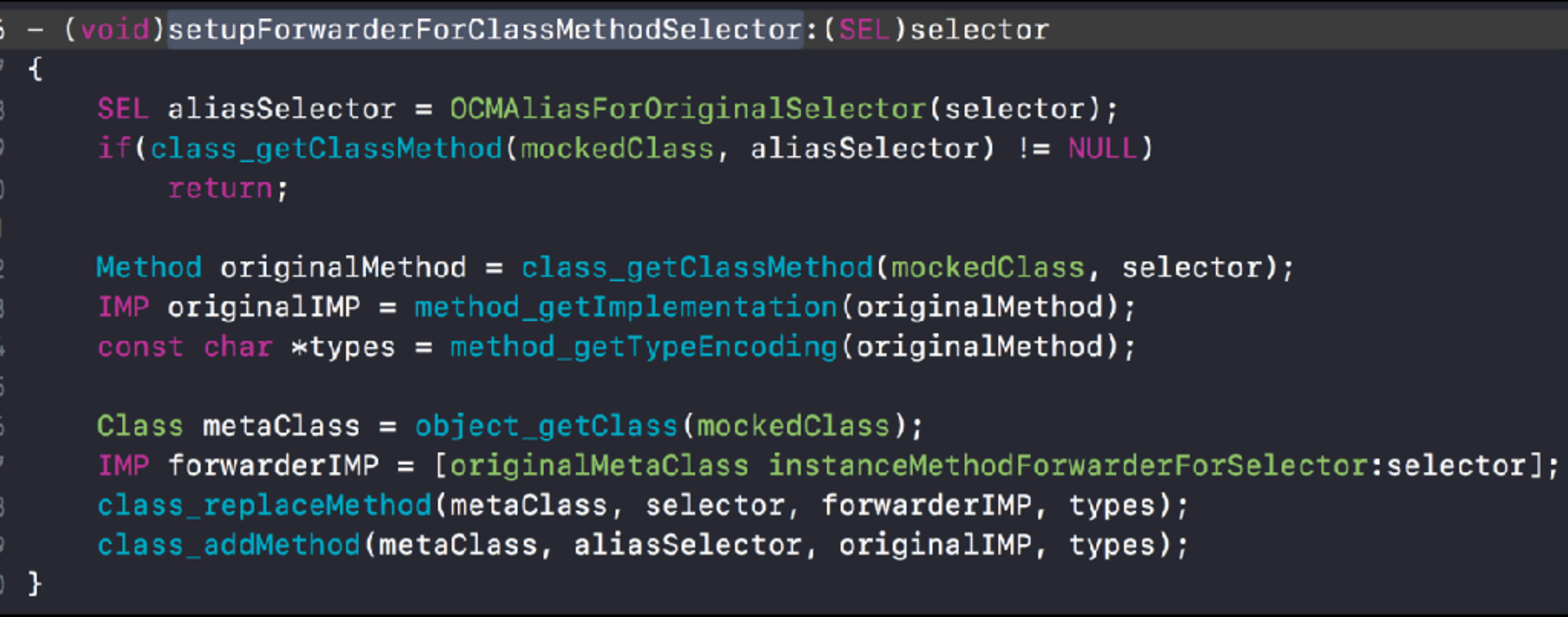
```
@class OCMLocation;
@class OCMInvocationStub;
@class OCMStubRecorder;
@class OCMInvocationMatcher;
@class OCMInvocationExpectation;


@interface OCMockObject : NSProxy
{
    BOOL            isNice;
    BOOL            expectationOrderMatters;
    NSMutableArray  *stubs; // OCMInvocationStub
    NSMutableArray  *expectations;
    NSMutableArray  *exceptions;
    NSMutableArray  *invocations;
}

+ (id)mockForClass:(Class)aClass;
+ (id)mockForProtocol:(Protocol *)aProtocol;
+ (id)partialMockForObject:(NSObject *)anObject;
```

OCMock实现原理



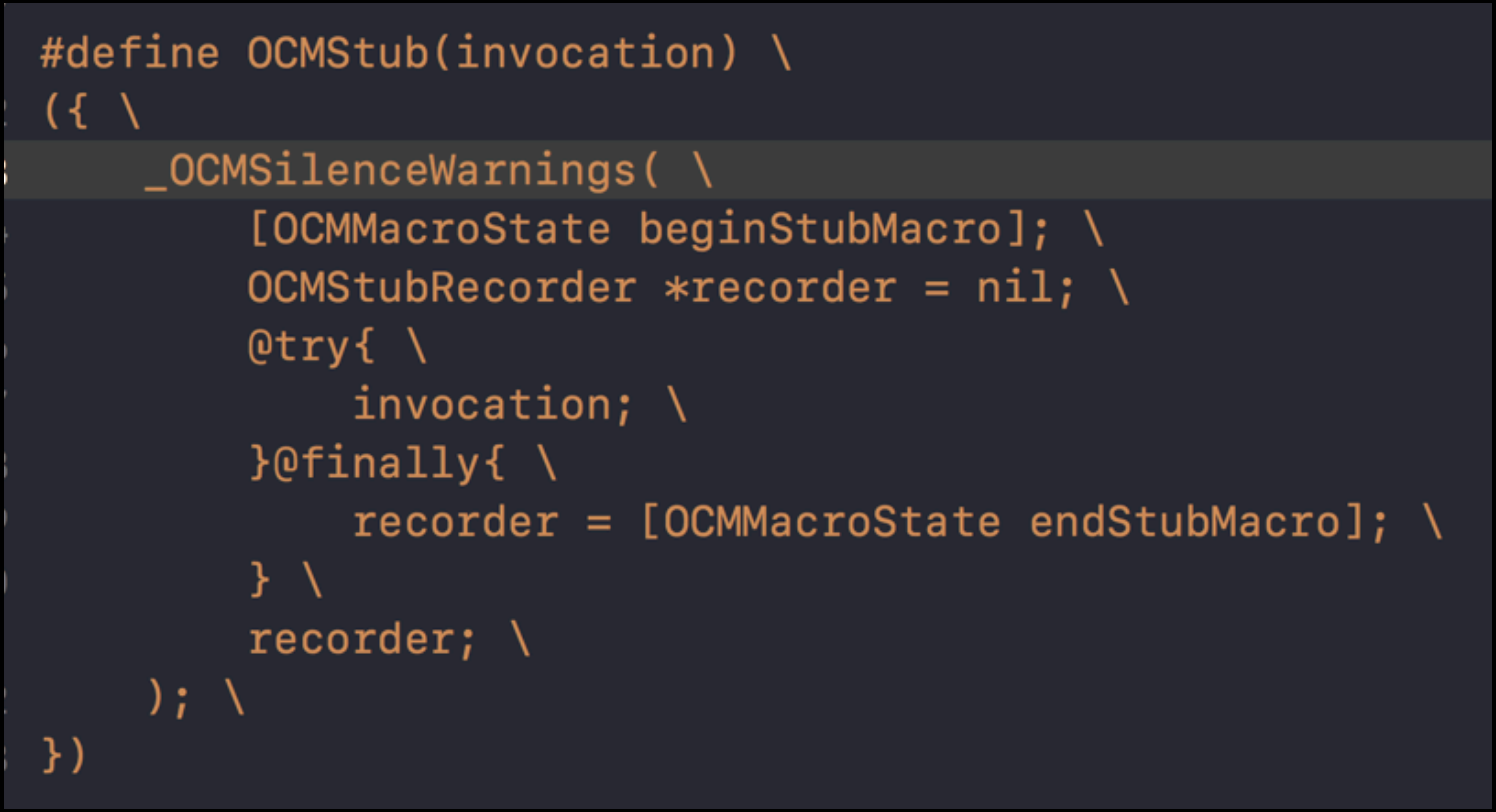
```
/* dynamically create a subclass and use its meta class as the meta class for the mocke
    class */
Class subclass = OCMCreateSubclass(mockedClass, mockedClass);
originalMetaClass = object_getClass(mockedClass);
id newMetaClass = object_getClass(subclass);

/* create a dummy initialize method */
Method myDummyInitializeMethod = class_getInstanceMethod([self mockObjectClass],
    @selector(initializeForClassObject));
const char *initializeTypes = method_getTypeEncoding(myDummyInitializeMethod);
IMP myDummyInitializeIMP = method_getImplementation(myDummyInitializeMethod);
class_addMethod(newMetaClass, @selector(initialize), myDummyInitializeIMP,
    initializeTypes);

object_setClass(mockedClass, newMetaClass); // only after dummy initialize is installed
    (iOS9)

/* point forwardInvocation: of the object to the implementation in the mock */
Method myForwardMethod = class_getInstanceMethod([self mockObjectClass],
    @selector(forwardInvocationForClassObject:));
IMP myForwardIMP = method_getImplementation(myForwardMethod);
class_addMethod(newMetaClass, @selector(forwardInvocation:), myForwardIMP,
    method_getTypeEncoding(myForwardMethod));
```

OCMock实现原理



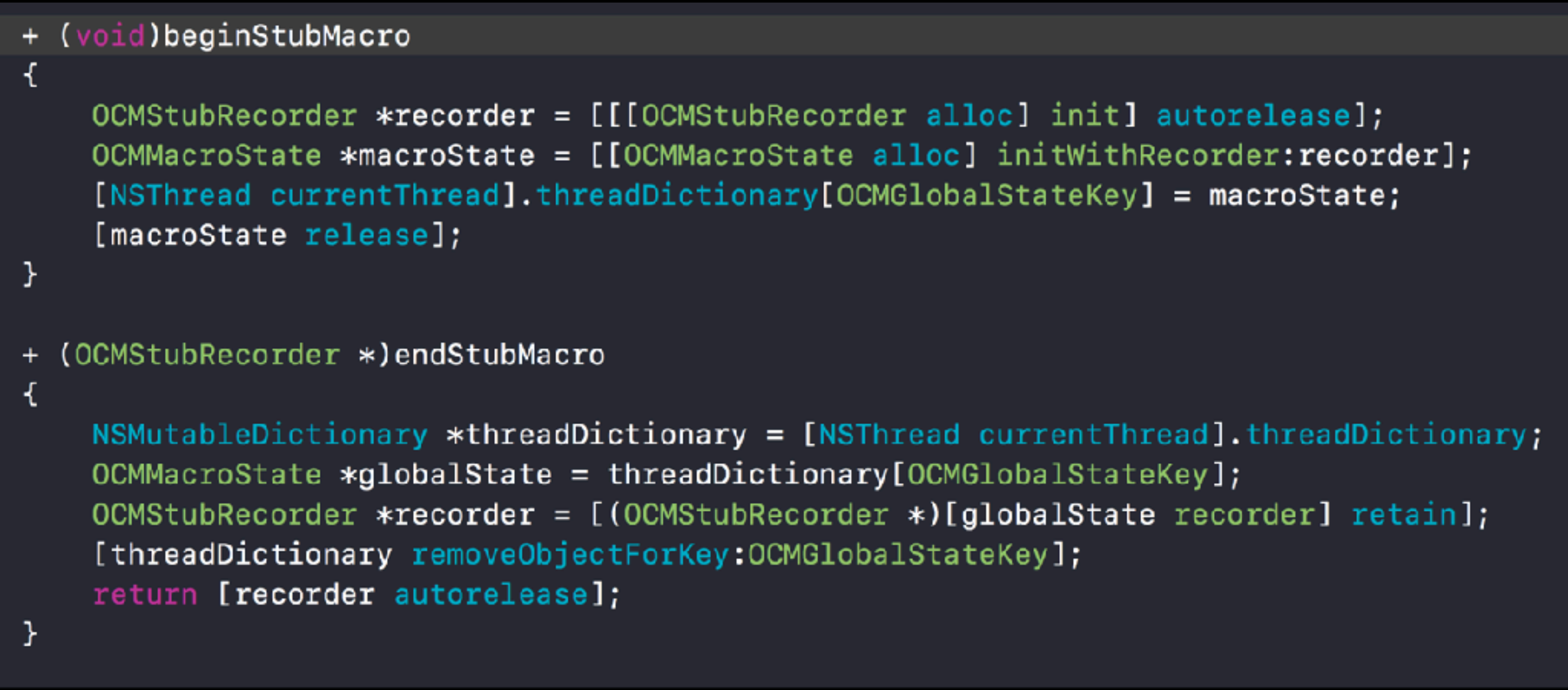
```
- (void)setupForwarderForClassMethodSelector:(SEL)selector
{
    SEL aliasSelector = OCMAliasForOriginalSelector(selector);
    if(class_getClassMethod(mockedClass, aliasSelector) != NULL)
        return;

    Method originalMethod = class_getClassMethod(mockedClass, selector);
    IMP originalIMP = method_getImplementation(originalMethod);
    const char *types = method_getTypeEncoding(originalMethod);

    Class metaClass = object_getClass(mockedClass);
    IMP forwarderIMP = [originalMetaClass instanceMethodForwarderForSelector:selector];
    class_replaceMethod(metaClass, selector, forwarderIMP, types);
    class_addMethod(metaClass, aliasSelector, originalIMP, types);
}
```

OCMock实现原理



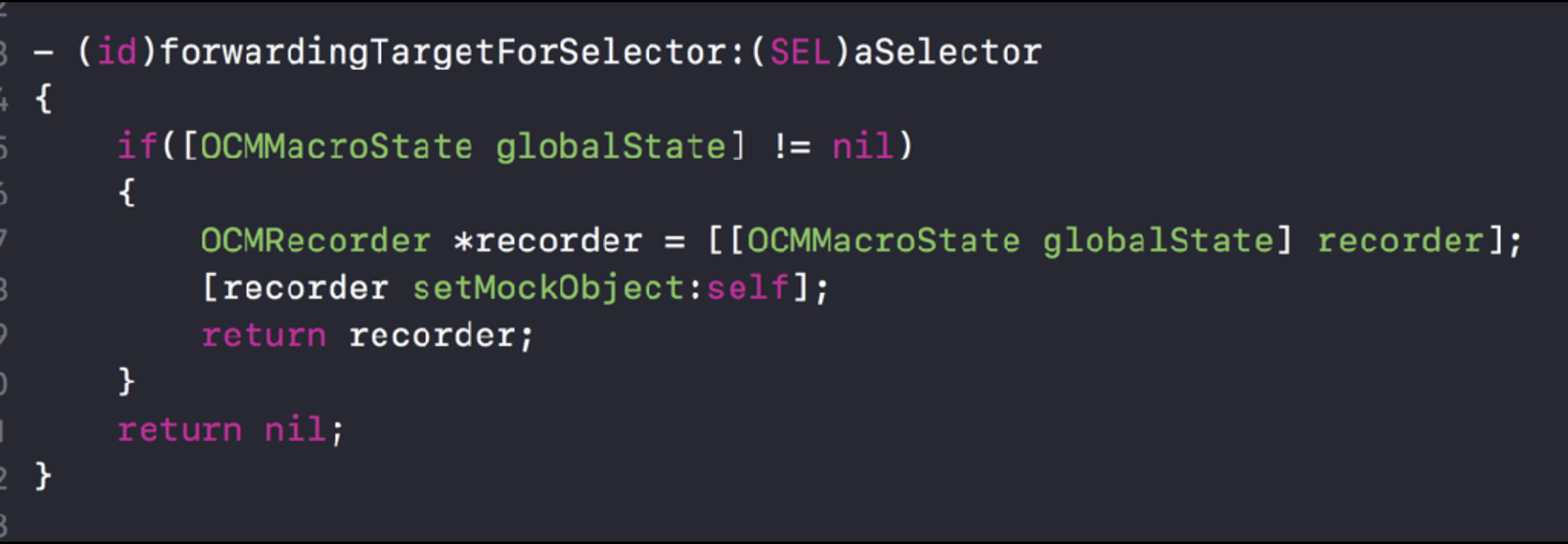
```
#define OCMStub(invocation) \
({ \
    _OCMSilenceWarnings( \
        [OCMMacroState beginStubMacro]; \
        OCMStubRecorder *recorder = nil; \
        @try{ \
            invocation; \
        }@finally{ \
            recorder = [OCMMacroState endStubMacro]; \
        } \
        recorder; \
    ); \
})
```

OCMock实现原理



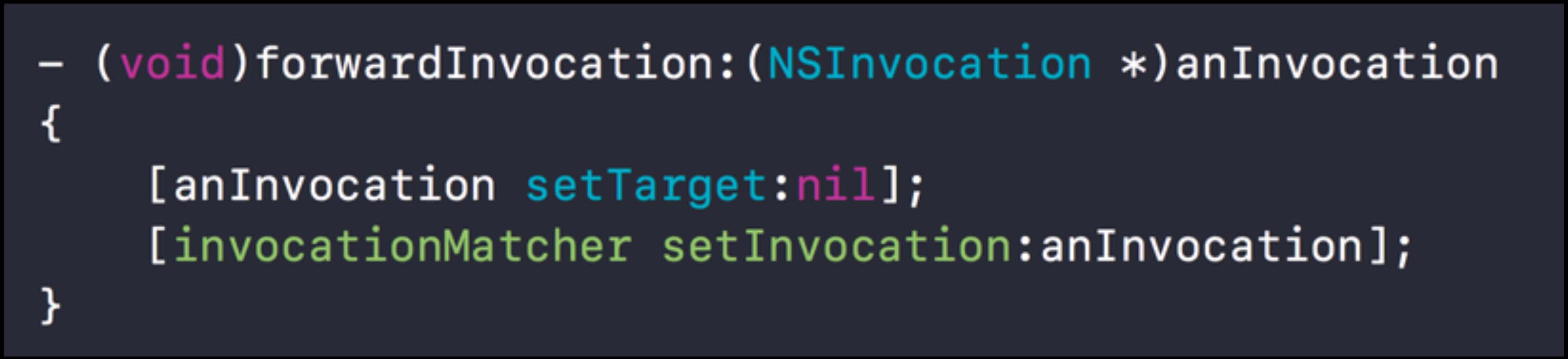
```objc
+ (void)beginStubMacro
{
    OCMStubRecorder *recorder = [[[OCMStubRecorder alloc] init] autorelease];
    OCMMacroState *macroState = [[OCMMacroState alloc] initWithRecorder:recorder];
    [NSThread currentThread].threadDictionary[OCMGlobalStateKey] = macroState;
    [macroState release];
}

+ (OCMStubRecorder *)endStubMacro
{
    NSMutableDictionary *threadDictionary = [NSThread currentThread].threadDictionary;
    OCMMacroState *globalState = threadDictionary[OCMGlobalStateKey];
    OCMStubRecorder *recorder = [(OCMStubRecorder *)[globalState recorder] retain];
    [threadDictionary removeObjectForKey:OCMGlobalStateKey];
    return [recorder autorelease];
}
```

OCMock实现原理



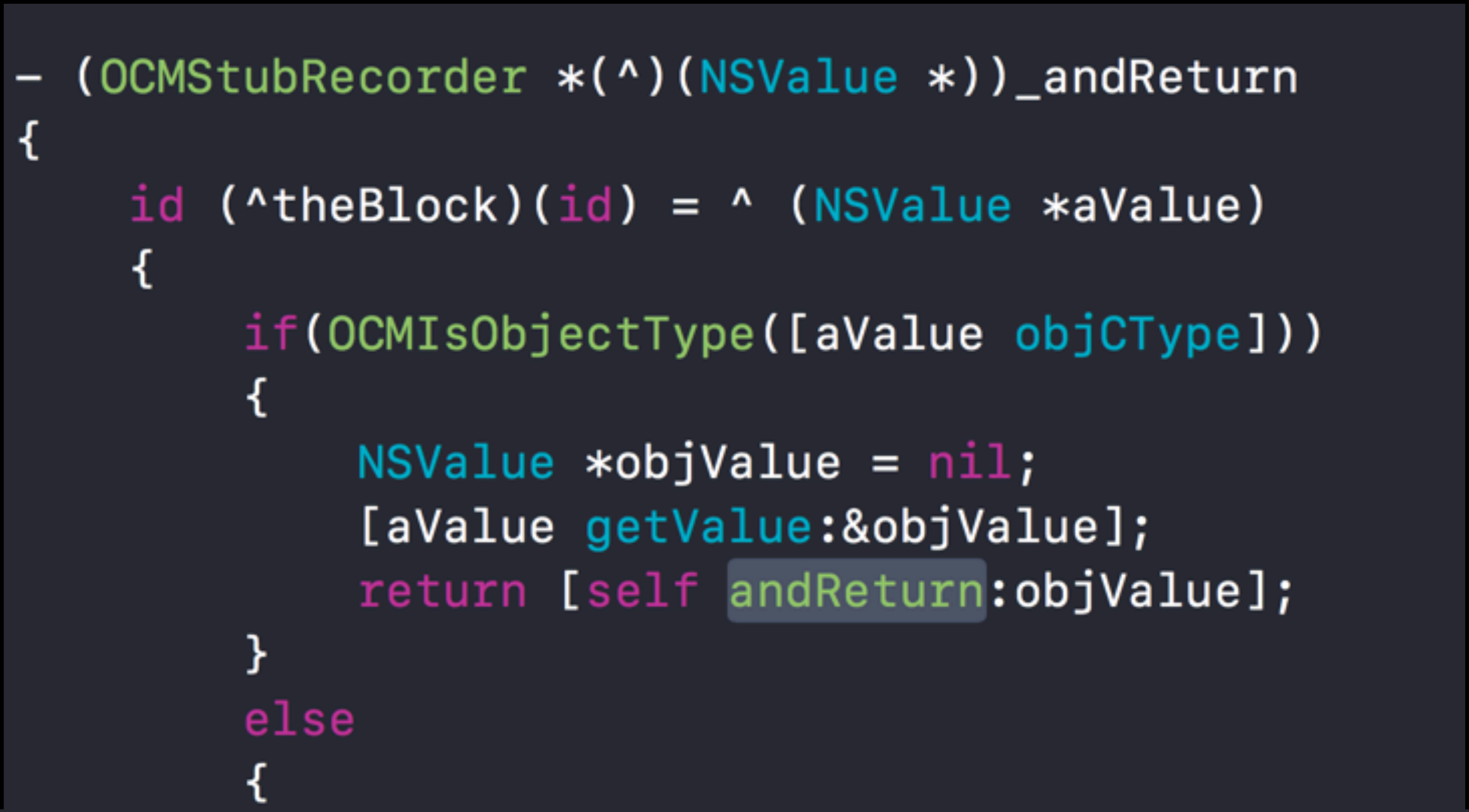
```
- (id)forwardingTargetForSelector:(SEL)aSelector
{
    if([OCMMacroState globalState] != nil)
    {
        OCMRecorder *recorder = [[OCMMacroState globalState] recorder];
        [recorder setMockObject:self];
        return recorder;
    }
    return nil;
}
```

OCMock实现原理

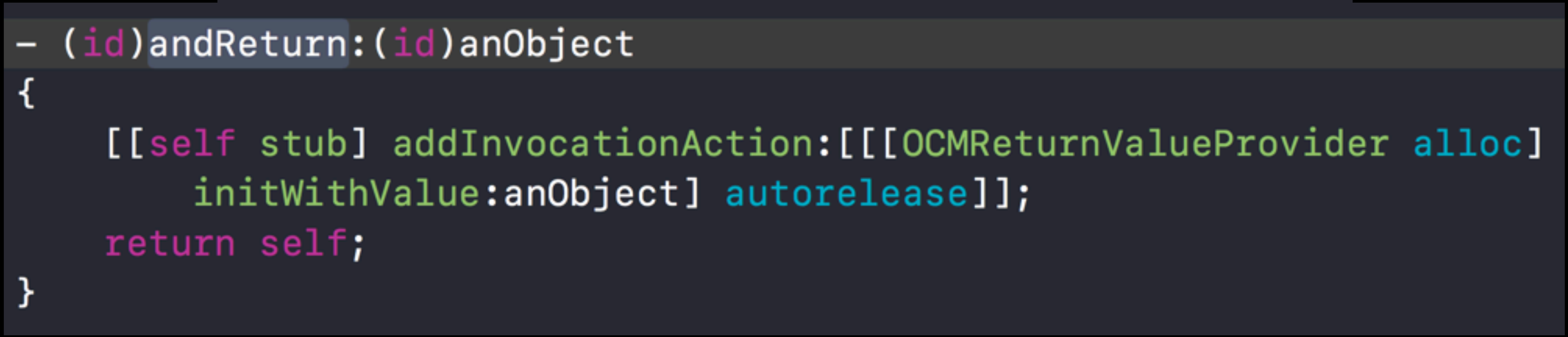


```
- (void)forwardInvocation:(NSInvocation *)anInvocation
{
    [anInvocation setTarget:nil];
    [invocationMatcher setInvocation:anInvocation];
}
```

OCMock实现原理

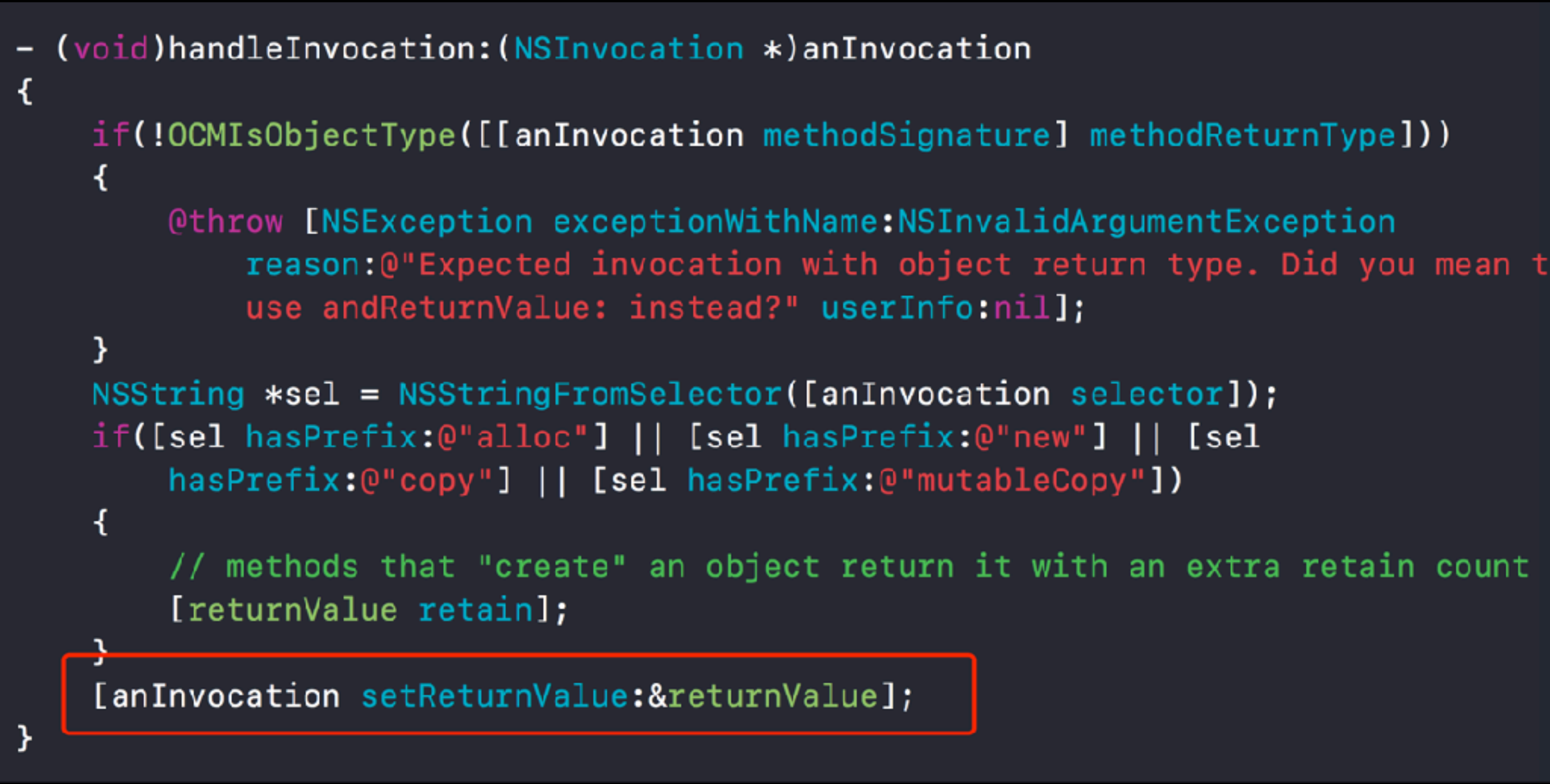


```
- (OCMStubRecorder *(^)(NSValue *))_andReturn
{
    id (^theBlock)(id) = ^ (NSValue *aValue)
    {
        if(OCMIsObjectType([aValue objCType]))
        {
            NSValue *objValue = nil;
            [aValue getValue:&objValue];
            return [self andReturn:objValue];
        }
        else
        {
```

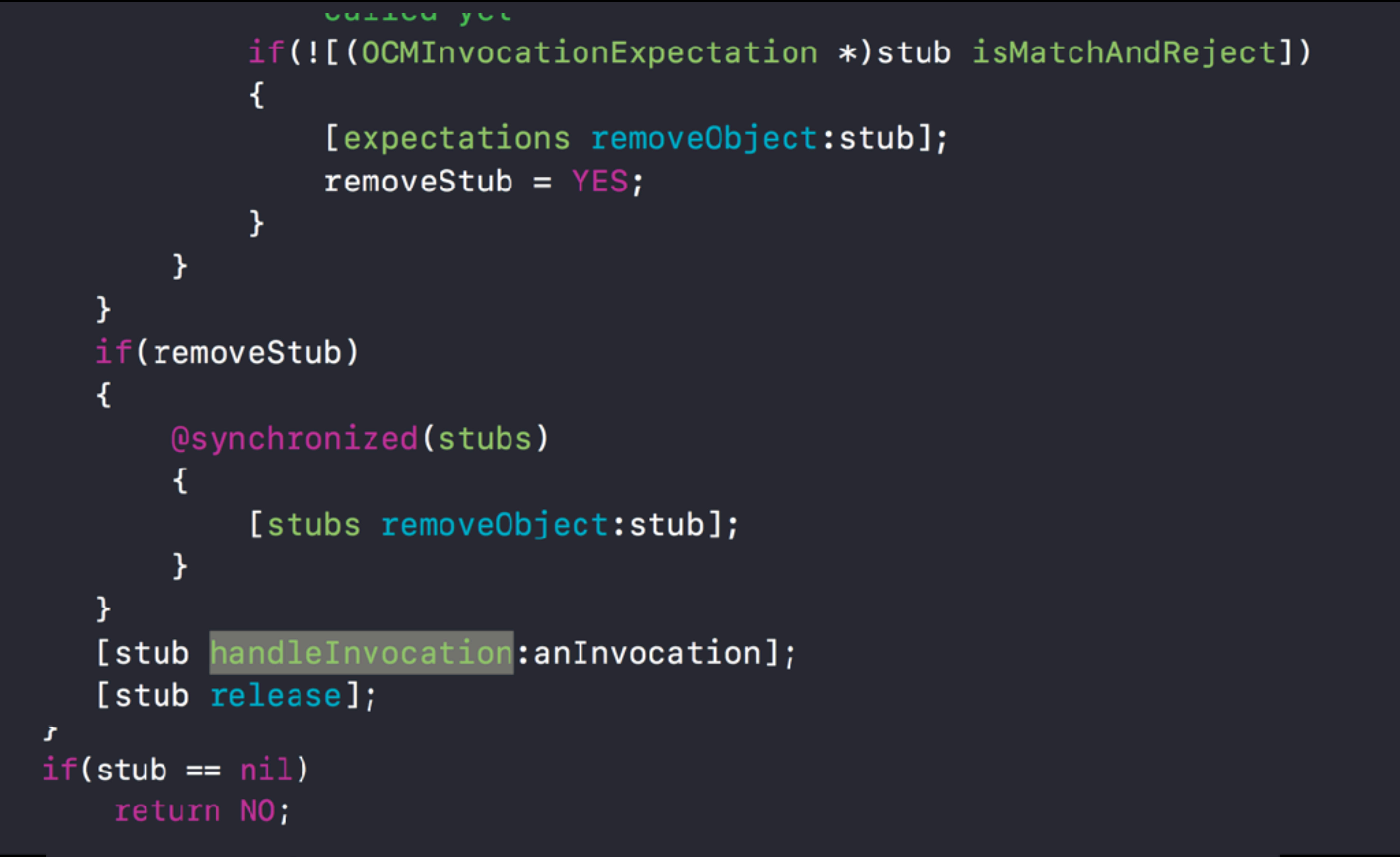


```
- (id)andReturn:(id)anObject
{
    [[self stub] addInvocationAction:[[[OCMReturnValueProvider alloc]
        initWithValue:anObject] autorelease]];
    return self;
}
```

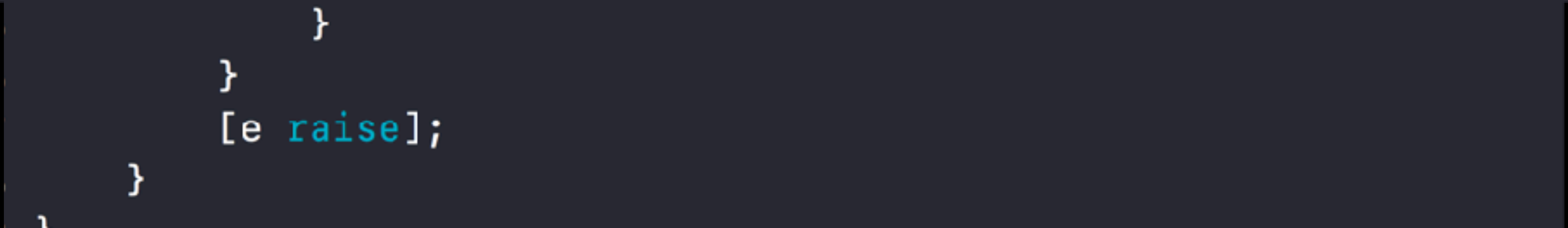
OCMock实现原理



```
- (void)handleInvocation:(NSInvocation *)anInvocation
{
    if(!OCMIsObjectType([[anInvocation methodSignature] methodReturnType]))
    {
        @throw [NSException exceptionWithName:NSInvalidArgumentException
            reason:@"Expected invocation with object return type. Did you mean t
            use andReturnValue: instead?" userInfo:nil];
    }
    NSString *sel = NSStringFromSelector([anInvocation selector]);
    if([sel hasPrefix:@"alloc"] || [sel hasPrefix:@"new"] || [sel
        hasPrefix:@"copy"] || [sel hasPrefix:@"mutableCopy"])
    {
        // methods that "create" an object return it with an extra retain count
        [returnValue retain];
    }
    [anInvocation setReturnValue:&returnValue];
}
```



```
                called yet
            if(![(OCMInvocationExpectation *)stub isMatchAndReject])
            {
                [expectations removeObject:stub];
                removeStub = YES;
            }
        }
    }
    if(removeStub)
    {
        @synchronized(stubs)
        {
            [stubs removeObject:stub];
        }
    }
    [stub handleInvocation:anInvocation];
    [stub release];
}
if(stub == nil)
    return NO;
```

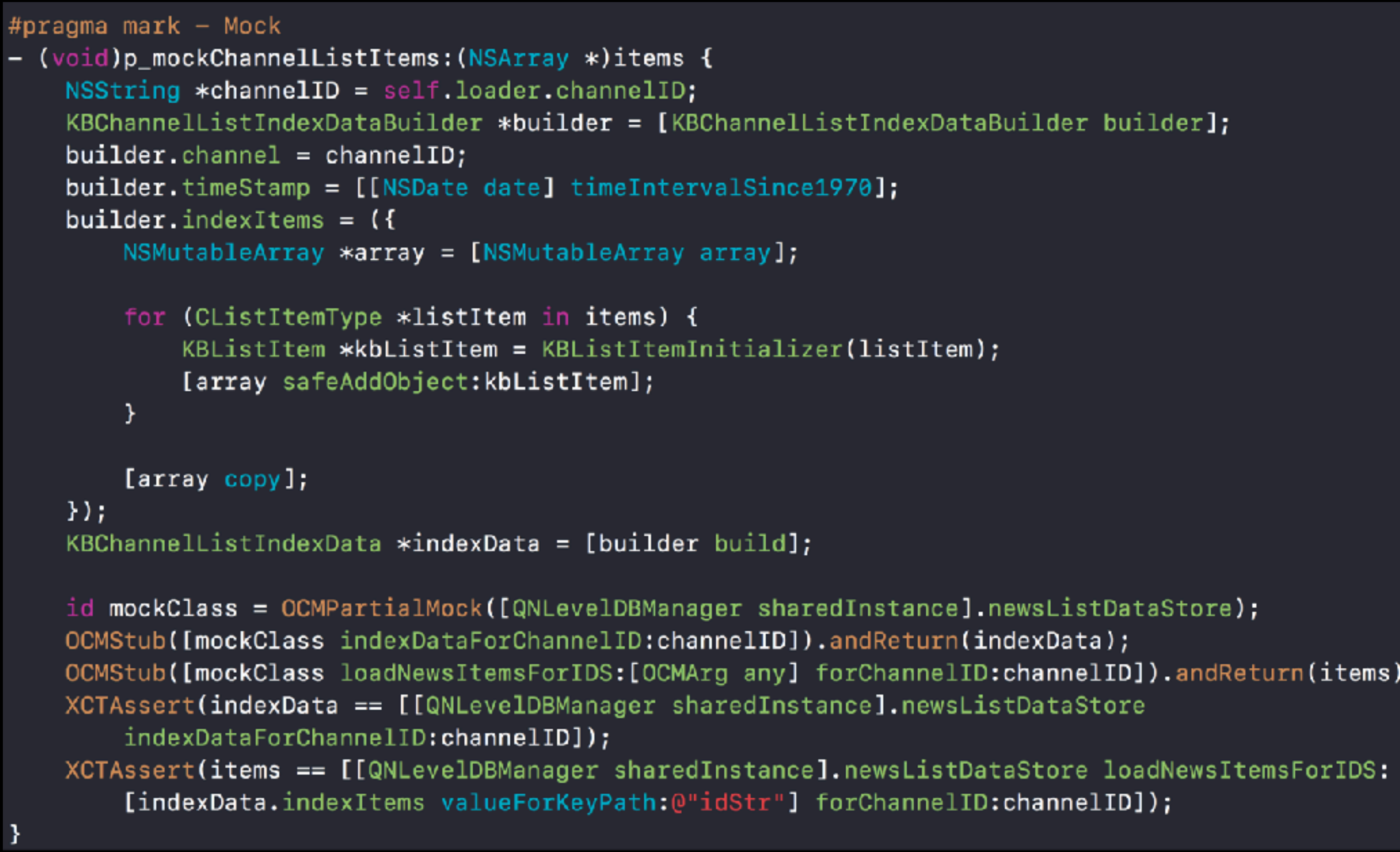


```
            }
        }
        [e raise];
    }
}
```

OHHTTPStubs+OCMock实例

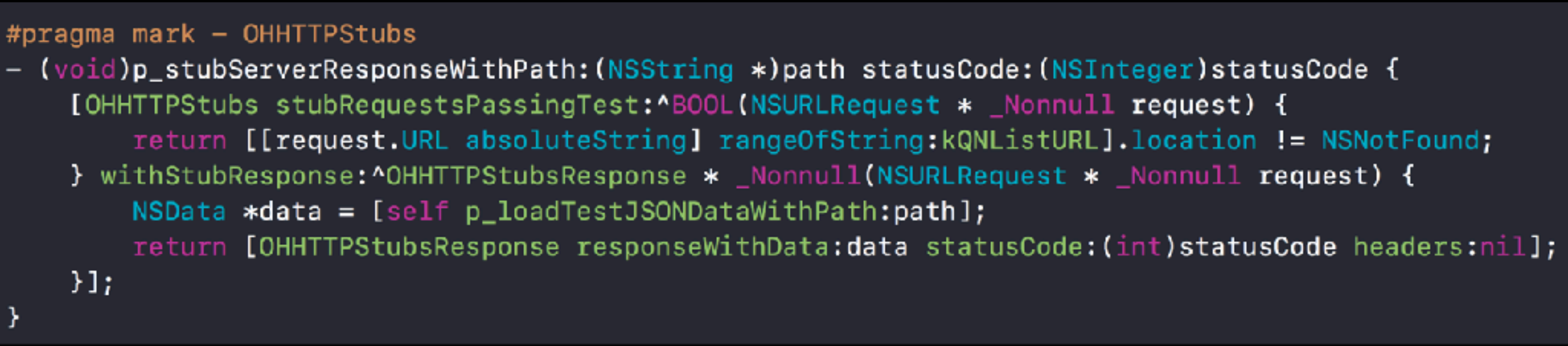
- 测试列表页的网络请求接口。
- 测试点：本地是否有缓存、数据加载策略、网络请求statusCode、请求结果等；

OHHTTPStubs+OCMock实例



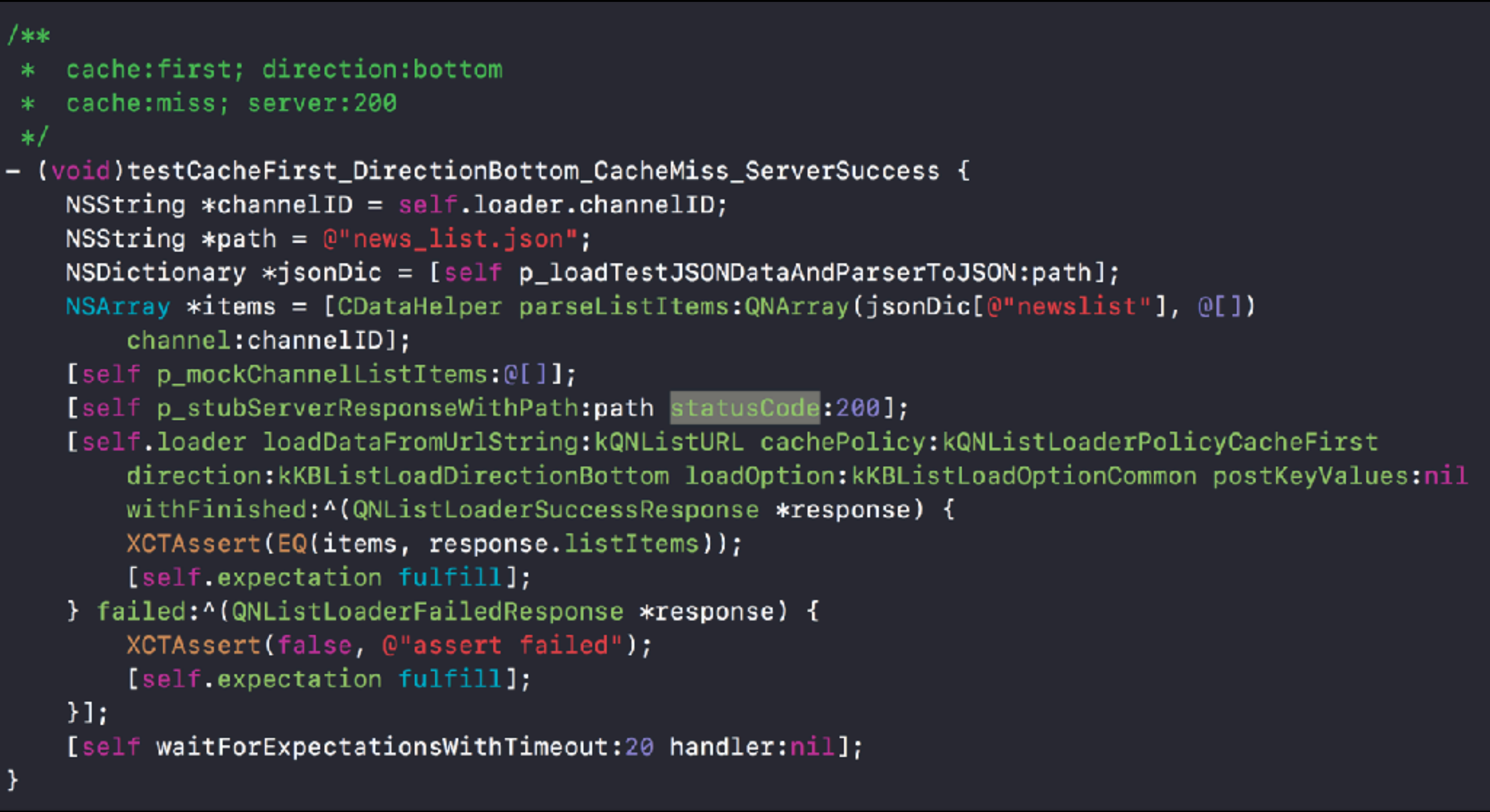
```
#pragma mark - Mock
- (void)p_mockChannelListItems:(NSArray *)items {
    NSString *channelID = self.loader.channelID;
    KBChannelListIndexDataBuilder *builder = [KBChannelListIndexDataBuilder builder];
    builder.channel = channelID;
    builder.timeStamp = [[NSDate date] timeIntervalSince1970];
    builder.indexItems = ({
        NSMutableArray *array = [NSMutableArray array];

        for (CListItemType *listItem in items) {
            KBListItem *kbListItem = KBListItemInitializer(listItem);
            [array safeAddObject:kbListItem];
        }

        [array copy];
    });
    KBChannelListIndexData *indexData = [builder build];

    id mockClass = OCMPartialMock([QNLevelDBManager sharedInstance].newsListDataStore);
    OCMStub([mockClass indexDataForChannelID:channelID]).andReturn(indexData);
    OCMStub([mockClass loadNewsItemsForIDS:[OCMArg any] forChannelID:channelID]).andReturn(items
    XCTAssert(indexData == [[QNLevelDBManager sharedInstance].newsListDataStore
        indexDataForChannelID:channelID]);
    XCTAssert(items == [[QNLevelDBManager sharedInstance].newsListDataStore loadNewsItemsForIDS:
        [indexData.indexItems valueForKeyPath:@"idStr"] forChannelID:channelID]);
}
```

OHHTTPStubs+OCMock实例



```
#pragma mark - OHHTTPStubs
- (void)p_stubServerResponseWithPath:(NSString *)path statusCode:(NSInteger)statusCode {
    [OHHTTPStubs stubRequestsPassingTest:^BOOL(NSURLRequest * _Nonnull request) {
        return [[request.URL absoluteString] rangeOfString:kQNListURL].location != NSNotFound;
    } withStubResponse:^OHHTTPStubsResponse * _Nonnull(NSURLRequest * _Nonnull request) {
        NSData *data = [self p_loadTestJSONDataWithPath:path];
        return [OHHTTPStubsResponse responseWithData:data statusCode:(int)statusCode headers:nil];
    }];
}
```

OHHTTPStubs+OCMock实例



```
/**
 *  cache:first; direction:bottom
 *  cache:miss; server:200
 */
- (void)testCacheFirst_DirectionBottom_CacheMiss_ServerSuccess {
    NSString *channelID = self.loader.channelID;
    NSString *path = @"news_list.json";
    NSDictionary *jsonDic = [self p_loadTestJSONDataAndParserToJSON:path];
    NSArray *items = [CDataHelper parseListItems:QNArray(jsonDic[@"newslist"], @[])
        channel:channelID];
    [self p_mockChannelListItems:@[]];
    [self p_stubServerResponseWithPath:path statusCode:200];
    [self.loader loadDataFromUrlString:kQNListURL cachePolicy:kQNListLoaderPolicyCacheFirst
        direction:kKBListLoadDirectionBottom loadOption:kKBListLoadOptionCommon postKeyValues:nil
        withFinished:^(QNListLoaderSuccessResponse *response) {
        XCTAssert(EQ(items, response.listItems));
        [self.expectation fulfill];
    } failed:^(QNListLoaderFailedResponse *response) {
        XCTAssert(false, @"assert failed");
        [self.expectation fulfill];
    }];
    [self waitForExpectationsWithTimeout:20 handler:nil];
}
```