Ice Object-C Client

首先根据https://zeroc.com/downloads/icetouch#osx下载Ice Touch，默认安装路径是/usr/local/Cellar/icetouch36
然后根据https://github.com/zeroc-ice/ice-builder-xcode安装Ice Builder for Xcode

创建iOS项目
在Build Settings的Additional SDKs设置Ice Touch的安装目录，设置的值为

Language	Location
Objective-C SDK	Ice Touch路径/lib/IceTouch/ObjC/$(PLATFORM_NAME).sdk
C++ SDK	Ice Touch路径/lib/IceTouch/Cpp/$(PLATFORM_NAME).sdk

，设置Ice Builder - General Options的Ice Home值是Ice的路径，在添加Security.framework和CFNetwork.framework，最后添加或创建.ice文件并将其添加到Build Phases的Compile Sources。然后编译，编译器就会根据.ice文件自动编译出相应的文件。

使用，参考https://github.com/zeroc-ice/ice-demos/tree/master/objective-c的例子。按照以上操作后，可以在需要调用的文件中添加一下代码。
ice文件

```
module ticket{
	struct Order{
		long orderId;
		string phone;
		string orderNum;
		int orderDate;
		int ticketType;
		double amount;
		int orderStatus;
	};
	sequence<Order> OrderSeq;
	interface TicketService{
		bool createOrder(Order myOrder);
		OrderSeq queryMyOrders(string phone);
		bool cancleOrder(long orderId);
	};
};
```

```
#import <objc/Ice.h>
#import <ticket.h> //我的ice文件名是ticket.ice
@protocol ICECommunicator; //Ice连接器
@protocol ticketTicketServicePrx; //根据定义的ticket.ice生成的服务代理类，命名规则是“module+interface+Prx”，module是ticket，interface是TicketService，所以服务代理类名就是ticketTicketServicePrx
```

初始化ice连接器

```
ICEInitializationData* initData = [ICEInitializationData initializationData];
initData.properties = [ICEUtil createProperties];
[initData.properties load:[[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@"config.client"]];
//[initData.properties setProperty:@"Ice.Default.Locator" value:@"IceGrid/Locator:tcp -h 192.168.0.112 -p
4061"];
initData.dispatcher = ^(id<ICEDispatcherCall> call, id<ICEConnection> con) {
    dispatch_sync(dispatch_get_main_queue(), ^{ [call run]; });
};
NSAssert(communicator == nil, @"communicator == nil");
communicator = [ICEUtil createCommunicator:initData];
```

获取服务代理实例

```
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
        @try {
            ICEObjectPrx* base = [communicator stringToProxy:@"TicketService"];
            base = [base ice_invocationTimeout:5000]; //调用超时时间单位毫秒
            NSLog(@"%@",[base ice_toString]);
            ticketServicePrx = [ticketTicketServicePrx checkedCast:base];
//            NSLog(@"%@",[NSThread callStackSymbols]);
            ticketMutableOrderSeq* orders = [ticketServicePrx queryMyOrders:@"13631276694"];
        } @catch(ICEEndpointParseException* ex) {
            NSString* s = [NSString stringWithFormat:@"Invalid router: %@", ex.reason];
            dispatch_async(dispatch_get_main_queue(), ^{
                [self exception:s];
            });
        } @catch(ICEException* ex) {
            dispatch_async(dispatch_get_main_queue(), ^{
                [self exception:[ex description]];
            });
        }
    });
```

释放ice连接器

```
    id<ICECommunicator> c = communicator;
    communicator = nil;

    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
            @try{
                [c shutdown];
            }@catch(ICEException* ex){
            }@finally{
                [c destroy];
                c = nil;
            }
    });
```

优化

iceClient.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>idleTimeOutSeconds</key>
	<integer>300</integer>
	<key>--Ice.Default.Locator</key>
	<string>IceGrid/Locator:tcp -h 192.168.0.112 -p 4061</string>
</dict>
</plist>
```

IceClientUtil.h

```
#import <Foundation/Foundation.h>
#import <objc/Ice.h>

@protocol ICECommunicator;

typedef void (^dispatch_block)(void);

@interface IceClientUtil : NSObject
+ (id<ICECommunicator>) getICECommunicator;
+ (void) closeCommunicator:(BOOL)removeServiceCache;
+ (id<ICEObjectPrx>) getServicePrx:(NSString*)serviceName class:(Class)serviceCls ;
+ (void)serviceAsync:(dispatch_block)block;
@end
```

IceClientUtil.m

```
#import "IceClientUtil.h"

const static NSString * locatorKey = @"--Ice.Default.Locator";
static NSString * iceLocatorValue;
static id<ICECommunicator> communicator;
static NSMutableDictionary * cls2PrxMap;
static UInt64 lastAccessTimestamp;
static int idleTimeOutSeconds;
static NSThread *monitorThread;
static BOOL stoped;

@interface IceClientUtil ()
@end

@implementation IceClientUtil
+ (id<ICECommunicator>) getICECommunicator{
    if (communicator == nil) {
        @synchronized(@"communicator"){
            if (iceLocatorValue == nil) {
                NSString * plistPath = [[ NSBundle mainBundle] pathForResource:@"iceClient" ofType:@"plist"];
                NSDictionary * plist = [[NSDictionary alloc] initWithContentsOfFile:plistPath];
                iceLocatorValue = [plist objectForKey:locatorKey];
                idleTimeOutSeconds = [[plist objectForKey:@"idleTimeOutSeconds"] intValue];
                NSLog(@"Ice client`s locator is %@ proxy cache time out seconds :
%d",iceLocatorValue,idleTimeOutSeconds);
            }
            ICEInitializationData* initData = [ICEInitializationData initializationData];
            initData.properties = [ICEUtil createProperties];
            [initData.properties setProperty:@"Ice.Default.Locator" value:iceLocatorValue];
            initData.dispatcher = ^(id<ICEDispatcherCall> call, id<ICEConnection> con) {
                dispatch_sync(dispatch_get_main_queue(), ^ { [call run]; });
            };
            communicator = [ICEUtil createCommunicator:initData];
```

```
            [IceClientUtil createMonitoerThread];
        }
    }
    lastAccessTimestamp = [[NSDate date] timeIntervalSince1970]*1000;
    return communicator;
}

+ (void) closeCommunicator:(BOOL)removeServiceCache{
    @synchronized(@"communicator"){
        if(communicator != nil){
            dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
                if (communicator != nil) {
                    [IceClientUtil safeShutdown];
                    stoped = YES;
                    if(removeServiceCache && cls2PrxMap!=nil && [cls2PrxMap count]!=0){
                        [cls2PrxMap removeAllObjects];
                    }
                }
            });
        }
    }
}

+(void) safeShutdown{
    @try{
        [communicator shutdown];
    }@catch(ICEException* ex){
    }@finally{
        [communicator destroy];
        communicator = nil;
    }
}

+ (void)serviceAsync:(dispatch_block)block{
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
        block();
    });
}

+ (id<ICEObjectPrx>) getServicePrx:(NSString*)serviceName class:(Class)serviceCls{
    id<ICEObjectPrx> proxy = [cls2PrxMap objectForKey:serviceName];
    if(proxy != nil){
        lastAccessTimestamp = [[NSDate date] timeIntervalSince1970]*1000;
        return proxy;
    }
    proxy = [IceClientUtil createIceProxy:[IceClientUtil getICECommunicator] serviceName:serviceName class:serviceCls];
    [cls2PrxMap setObject:proxy forKey:serviceName];
    lastAccessTimestamp = [[NSDate date] timeIntervalSince1970]*1000;
    return proxy;
}

+ (id<ICEObjectPrx>) createIceProxy:(id<ICECommunicator>)c serviceName:(NSString*)serviceName class:
(Class)serviceCls {
    id<ICEObjectPrx> proxy = nil;
    @try{
        ICEObjectPrx* base = [communicator stringToProxy:serviceName];
        base = [base ice_invocationTimeout:5000];
        SEL selector = NSSelectorFromString(@"checkedCast:");
        proxy = [serviceCls performSelector:selector withObject:base];
        return proxy;
    } @catch(ICEEndpointParseException* ex) {
        return ex;
    } @catch(ICEException* ex) {
        return ex;
    }@catch(NSException* ex){
        return ex;
```

```
    }
}

+ (void)createMonitoerThread{
    stoped = NO;
    monitorThread = [[NSThread alloc]initWithTarget:self selector:@selector(monitor) object:nil];
    [monitorThread start];
}

+ (void) monitor{
    while(!stoped){
        @try{
            [NSThread sleepForTimeInterval:5.0f]; // 让当前线程睡眠 5.0 秒
            UInt64 nowTime = [[NSDate date] timeIntervalSince1970]*1000;
            if(lastAccessTimestamp + idleTimeOutSeconds * 1000L < nowTime){
                [IceClientUtil closeCommunicator:true];
            }
        }@catch(NSException * e){
            NSLog(@"%@",[e description]);
        }
    }
}
@end
```

调用

```
    [IceClientUtil serviceAsync:^{
        ticketServicePrx = [IceClientUtil getServicePrx:@"TicketService" class:[ticketTicketServicePrx class]];
        if ([ticketServicePrx isKindOfClass:[ticketTicketServicePrx class]]) {
            ticketMutableOrderSeq* orders = [ticketServicePrx queryMyOrders:@"13631276694"];
            datas = orders;
            [self.tableView reloadData];
        }else if ([ticketServicePrx isKindOfClass:[ICEEndpointParseException class]]) {
            ICEException * ex = (ICEException *)ticketServicePrx;
            NSString* s = [NSString stringWithFormat:@"Invalid router: %@", ex.reason];
            dispatch_async(dispatch_get_main_queue(), ^{
                [self exception:s];
            });
        }else if ([ticketServicePrx isKindOfClass:[ICEException class]]) {
            NSException * ex = (NSException *)ticketServicePrx;
            dispatch_async(dispatch_get_main_queue(), ^{
                [self exception:[ex description]];
            });
        }else if ([ticketServicePrx isKindOfClass:[NSException class]]) {
            NSException * ex = (NSException *)ticketServicePrx;
            dispatch_async(dispatch_get_main_queue(), ^{
                [self exception:[ex description]];
            });
        }
    }];
```

停止

```
[IceClientUtil closeCommunicator:YES];
```