# Socket:

SocketConnection初始化

Connection.get()

* Creators.getDeferObjectCreator().getConnection() [单例]
  + new SocketConnection(socket, session)
    - socket = new SmartSocket
    - session = SocketSession.get(handler) [单例]
      * filterChain = Creators.getDeferObjectCreator().createIoFilterChain() [单例]
      * setHandler(ProtolHandlers. globalHandler())
        + filterChain.tailFilter = ProtolHandlers. globalHandler()

ProtocolHandlers.createGlobalHandler()

new JavaBehindXmlProtocolProcessor

向ServerPackageCollection(Map<class<?extends JavaBehindXmlServerPacket>, String>) 注册Packet

* initializeSession()
  + IoSessionInitializer.excute()
    - IoSession.IoFilterChain.add("logger", codec", "fontColor", "appVersion", authentication", "parser") AbstractProtocolHandler.processor.create\*Filter()

责任链IoFilterChain [双向链表]

HeadFilter => LoggingFilter => ProtocolCodecFilter => FontColorFormatFilter => AppVersionFilter => AuthenticationFilter => ProtocolParseFilter => MsgCallbackConfirmFilter => IoHandlerFilter(tailFilter)

Socket正在连接/连接成功/连接失败：

Connection.socket.onConnecting/onConnected/onConnectFailed

* sesson. fireConnect
  + filterChain.fireConnect [责任链模式]
    - callNextConnect
      * filter.onConnect (all filter) 按以上链顺序传递

。。。

* + - * + AbStractProtocolHandler. listeners. onConnect

XmlProtocolProcessor $ MessageListenerImpl

JavaBehindSocketMessageListener

SocketConnectionManager $ MessageListenerImpl

连接成功发出协议

Connection.get().send(Client\*Packet)

* session.write(message) message = packet
  + filterChain. fireFilterWrite
    - callPreviousFilterWrite 逆向传递

。。。

filter. filterWrite(object message)

* + - * HeadFilter.filterWrite(String message)
        + SocketSession.doWrite(String message)

SocketConnection.wirteData(String message)

收到协议

Connection.socket.onData(byte[] data)

* fireMessageReceived(new String(data))
  + session. fireMessageReceived
    - filterChain. fireMessageReceived
      * callNextMessageReceived 正向传递

。。。

* + - * + ProtocolCodecFilter. onMessageReceived

XmlParser.parse

callNextMessageReceived

。。。

ProtocolParseFilter. onMessageReceived

XmlProtocolParser.parseMessageAsync

packet.parse(element)

parseElement(element);

doAfterParse();

fire();

处理协议

packet.parse(element)

* parseElement(element)

特别地，对于PreprocessedServerPacket：将element解析为Map<String, String>；

* + 特别地，有：
    - \*NoticeParser. parse、\*NoticeMapParser. parse (return \*NoticePacketData)
    - \*MessageMapParser. parse (return \*MessageData)
    - \*MapParser. parse (return \*ResponseData)

对于其他ServerPacket：通过\*XmlParser.parse将element解析为\*ResponseData

* doAfterParse(element)

特别地，对于与Message有关的ServerPacket：进行db io操作；

对于其他ServerPacket：方法为空

* fire()
  + onEvent
    - 遍历MessageListenerManager. Listeners中所有MessageListener，并触发onEvent中的listener.on\*()方法
      * 特别地，对于PreprocessedServerPacket：触发listener.on\*ReceivedFromServer
        + Socket\*PacketProcessor. process(Map<String, String>)

特别地，对于ServerMobileNoticePacket：\* NoticePacketMapProcessor().process(Map<String, String>)

parse(Map<String, String>) into data

store(data)

fireEvent(data)

build(data) into Notice

fireEvent(Notice)

* + - * + 特别地，对于ServerMobileNoticePacket及ServerMobileTalkMSGPacket，还会将Notice或Message数据存储到本地数据库中：

storeUnrecognizedNoticeData(Map<String, String>);

storeNotice/MessageCursorIntoLastFourCursorsCache(Map<String, String>);

# Http:

Http请求参数处理

HttpProtocolPacketProcessWrapper

sendHttpRequestWithoutToken()

= super.sendHttpRequest( [version])

sendHttpRequestWithoutToken(String)

= super.sendHttpRequest( [ “data”, version])

sendHttpRequestWithoutToken(String, List<BinaryFileData>)

= super.sendHttpRequest( [“data”, version, uuid, “attachmentData附件”])

sendHttpRequestWithoutTokenForGetProxyServerInfo()

= super.sendHttpRequest()

sendHttpRequestWithoutTokenForRefreshToken(String)

= super.sendHttpRequest( [“data”, version, user\_id])

sendHttpRequestForGetPrivateKey(String)

= super.sendHttpRequest( [uuid, “public\_key”, version])

sendHttpRequestForGetChannelStat(String)

= super.sendHttpRequest( [ “data”])

sendHttpRequestForStaticUrl()

= super. sendHttpRequest()

sendHttpRequestWithToken(String, String)

= super.sendHttpRequest( [ “data”, version, “token”, user\_id])

sendHttpRequestWithToken(String, String, List<BinaryFileData>)

= super.sendHttpRequest( [ “data”, version, “token”, user\_id, “attachmentData附件”])

sendHttpRequest()

= super.sendHttpRequest([version, token, user\_id])

sendHttpRequest(String)

= super.sendHttpRequest( [“data”, version, token, user\_id])

sendHttpRequest(String, List<BinaryFileData>)

= super.sendHttpRequest( [“data”, version, token, user\_id, “attachmentData附件”])

sendHttpRequestForImageWithoutToken()

= super. sendHttpRequestForImage ( [version, uuid])

String token = MobileClientSettings.getToken()

String version = MobileClientSettings.*shanliaoVersion*

String user\_id = String.valueOf(MobileMyself.get().getUserId())

String uuid = MobileClientSettings.getUUID()

Http连接

HttpProtocolPacketProcessor

sendHttpRequest(RequestParameter... params)

* MobileCreators.getUrlAccessor().getData(getHttpURLRoot() + getUrl(), new RequestObserver<ResponseEntity>(){}, params)
  + UIThread.run(Runnable)
    - new AsyncRequestForString(observer, url)
      * urlConnectionProxy = new HttpUrlConnectionProxy(url)

。。。

* + - * + **doInBackground**

excuteRequest(params)

urlConnectionProxy.sendRequest(params)

(HttpURLConnection) url.openConnection()

writeDataToConnection(httpURL, true, params)

createResponseEntity

RequestObserver<ResponseEntity>. onData

onTextResponseThenFire

处理响应数据

onTextResponseThenFire

* parseTextResponse
  + createResponseData
  + 。。。
  + parseJSONString

JSON解析器ErrorProofJsonParser，核心解析器CoreJsonParser

特别地，\*NoticeContentProcessor、\*MessageContentProcessor Extends ErrorProofJsonParser

所有的\*MessageContentProcessor. buildJson及parseJson方法 都统一封装在MessageContentProcessor. buildJson及parseJson方法中（工厂模式），根据MessageContent.Type选择不同的\*MessageContentProcessor的buildJson及parseJson方法。

NoticeCacheDataBuilder：\*NoticeContentProcessor. buildJson 重载NoticeCacheData build方法

NoticeBuilder：\*NoticeContentProcessor. parseJson 统一封装在Notice build(NoticeCacheData data)中，根据NoticeType选择不同的\*NoticeContentProcessor. parseJson方法

\*MessageParser. parse (return \*MessageData)

\*MessageContentParser. parseJson (return \*MessageContent)，返回值传至message. setConent中

\*NoticeContentParser. parseJson (return \*NoticeContent)，返回值传至notice.setContent()中。

* + storeOrParseCacheData
    - 。。。
      * storeCacheData
  + storeInfoIntoCache
* fireListenerAndCallback
  + fire
    - onEvent
      * listeners.on\*方法
  + callback.execute

请求缓存数据

|  |  |  |  |
| --- | --- | --- | --- |
| new CachedAndNetRequest.sendRequest | | | |
| RequestSource | unExpired | Expired | None |
| isFromCache | Cache | Server | Server |
| **!** isFromCache | Cache | Server | Server |
| new CachedFirstOrNetRequest.sendRequest | | | |
| RequestSource | unExpired | Expired | None |
| isFromCache | Cache | Cache | Error |
| **!** isFromCache | Cache | Server | Server |
| new CachedOrNetRequest.sendRequest | | | |
| RequestSource | unExpired | Expired | None |
| isFromCache | Cache | Cache | Error |
| **!** isFromCache | Server | Server | Server |

若向本地缓存请求数据，则触发

onCachedResponseThenFire

* parseTextResponse
  + createResponseData
  + 。。。
  + parseJSONString
  + storeInfoIntoCache
* fireListenerAndCallback
  + fire
    - onEvent
      * listeners.on\*方法
  + callback.execute

AbstractNioSocket

—PayloadNioSocket

—EncryptedPayloadNioSocket

—SnappyEncryptedPayloadNioSocket

—SnappyEncryptedPayloadSocks5NioSocket

SocketConnection :: get

* (SocketConnection) AndroidDeferObjectCreator :: getConnection
  + (SmartSockect) SocketConnetion :: createSocket
    - new SnappyEncryptedPayloadNioSocket(new DirectSocketListener)
  + SocketSession :: get
    - (IoFilterChain) JavaDeferObjectCreator :: createIoFilterChain
    - IoFilterChain :: setIoHandler(

(ProtocolHandlerImpl) AndroidDeferObjectCreator :: createProtocolHandler

* + - * + (JavaBehindXmlProtocolProcessor) AndroidDeferObjectCreator :: createXmlProtocolProcessor

)

* SocketConnection :: initializeSession
  + IoSessionInitializer :: execute
    - IoFilterChain :: add

JavaBehindMessageListener :: onGetServerInfo

* SocketServerInfoContainer :: fillServerList
* AndroidSocketManager :: connect
  + AndroidSocketManager :: setUpServerList
    - SocketConnectManager :: setConnectionList
      * DefaultServerProvider :: resetDefaultFrom
        + DefaultServerProvider :: restart
  + SocketConnectionManager :: connect
    - Check Network Connection
    - SocketConnection :: get
    - Check SocketConnection Connectable(SmartSocket -> state == NOT\_STARTED)
    - Check Token
    - Check ServerAddress
      * DefaultServerProvider
    - SocketConnection :: connect
      * SmartSocket :: connect
        + SmartSocket -> state = DIRECT\_CONNECTING
        + SnappyEncryptedPayloadNioSocket :: connect

AbstractNioSocket :: handleConnecting

AbstractNioSocket -> state = CONNECTING

DirectSocketListener :: onConnecting

SmartSocket -> listener :: onConnecting

SocketSession :: fireConnecting

* IoFilterChain :: fireConnecting
  + IoHandlerFilter :: onConnecting
    - ProtocolHandlerImpl / AbstractProtocolHandler :: onConnecting
      * JavaBehindSocketMessageListener :: onConnecting
        + MobileFacade :: getServerInfo

…

…

…

SocketConnectionManager :: connect

Check SocketConnection return

* AbstractNioSocket :: handleSocketEvents
  + Selector :: open 创建Selector
  + SocketChannel :: open 创建并打开通道
  + SocketChannel :: connect 连接通道
  + SocketChannel :: register(Selector, SelectionKey.OP\_CONNECT) 向Selector注册通道：监听Channel时对连接事件感兴趣，即连接就绪，准备好接收新进入的连接。若interest参数为SelectionKey.OP\_READ，则读就绪；OP\_WRITE，则写就绪。
  + Selector :: select(TIMEOUT) 阻塞到Selector上注册的通道有否（连接或读或写）就绪或阻塞时间超过TIMEOUT
  + AbstractNioSocket :: processSelectedKeys 遍历已选择的键集合SelectionKeys来访问就绪的通道。
    - AbstractNioSocket :: processConnect
      * AbstractNioSocket :: removeInterestOp 移除连接或读或写键
      * AbstractNioSocket :: addInterestOp 添加键
        + SnappyEncryptedPayloadNioSocket :: handleConnected

new SnappyFramedEncoder()

new SnappyFramedDecoder()

AbstractNioSocket -> state = CONNECTED

SmartSocket -> state = DIRECT\_CONNECTED

SocketConnection :: fireConnected

SocketSession :: fireConnected

* IoFilterChain :: fireConnected
  + AbstractProtocolHandler :: onConnected
    - MessageListenerImpl :: onConnected
      * JavaBehindXmlProtocolProcessor :: sendInitPacket
        + Connection :: send(new ClientMobileAuthPacket())

WebSocket:

|  |  |  |
| --- | --- | --- |
| Events | Handler | When to Fire |
| open(simple event) | onopen | 1. A new WebSocket is created and then WebSocket connection is established(readyState: OPEN) |
| MessageEvent | onmessage | 1. readyState is OPEN and WebSocket message has been received |
| CloseEvent | onclose | 1. Successfully invoke close method 2. Required to close ‘cause errors occur |
| Error(simple event) | Onerror | 1. fail the WebSocket connection 2. WebSocket connection is closed with prejudice(单方面断开连接，未成功完成关闭握手) |
|  |  |  |
|  |  |  |
|  |  |  |