# 游戏服Socket通讯宿主基类

GameSocketHost

支持http和socket连接

GameSocketHost()

初始化socket和http的监听，等候建立连接

开启消息处理线程和线程回调函数ProcessQueue

添加数据到来, 连接建立, 断开连接三个回调接口

socketLintener\_DataReceived

数据到来, 保存数据到消息队列,

singal.Set()发送信号通知消息处理线程处理消息

socketLintener\_OnConnectCompleted

连接建立,创建全局的GameSession

socketLintener\_Disconnected

断开连接，回收GameSession

ProcessQueue

收到信号singal.WaitOne(), 进行消息处理,

Interlocked.Increment(ref runningNum);计数加一

threadPool.QueueWorkItem(ProcessPackage, package);使用SmartThreadPool threadPool对象调用线程进行消息处理, 调用逻辑层的接口OnRequested

# Redis

缓存数据库，快速的读取数据

ZyGames.Framework.Redis

使用接口

public static bool CheckConnect();

public static void Initialize();

public static RedisConnection Pop();

public static void Put(RedisConnection connection);

CheckConnect

分配一个RedisClien尝试建立连接，如果建立失败，服务器启动失败, 这个RedisClient不保存。

redis初始化调用顺序

GameEnvironments :: initialize()

CacheFactory :: initialize()

RedisConnectionPool.Initialize() 根据配置参数决定初始连接个数并保存到

private static ConcurrentQueue<RedisConnection> \_connectionPools

使用方式:

保存数据:

获取连接对象 var client = RedisConnectionPool.Pop();

保持数据

byte[] data = ProtoBufUtils.Serialize(\_globalSession);

填写key和value

client.Set(sessionRedisKey, data);

归还连接对象

RedisConnectionPool.Put(client);

读取数据

获取连接对象 var client = RedisConnectionPool.Pop();

读取数据

传入key获取value

byte[] data = client.Get<byte[]>(sessionRedisKey) ?? new byte[0];

if (data.Length == 0) { return; }

var temp = ProtoBufUtils.Deserialize<ConcurrentDictionary<Guid, GameSession>>(data);

if(temp!=null) { \_globalSession = temp; }

归还连接对象

RedisConnectionPool.Put(client);

# ActionFactory

进行消息解析，调用到对应的脚本或者类进行处理.

静态对象

消息流程

GameSocketHost消息处理线程得到信号有消息到来

OnRequested(httpGet, response);

ActionFactory.Request(httpGet, response, GetUser); 获取合适的BaseStruct baseStruct；然后调用子类实现的baseStruct.DoAction(),baseStruct.DoInit();