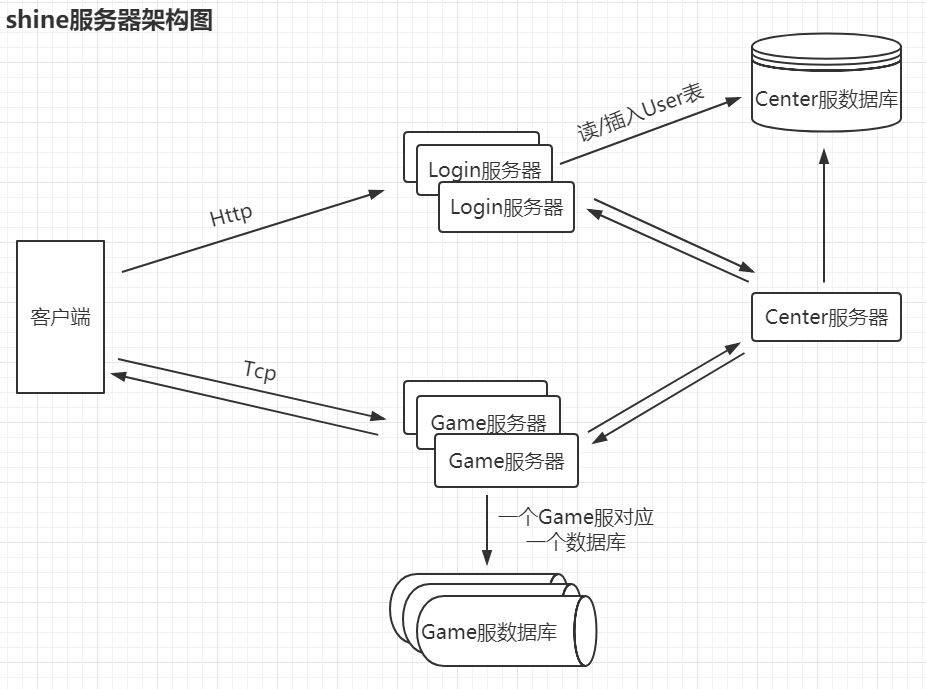
****

**框架结构**

一.shine工程 **(home\core\server\project)**

多个工程（base、shine、commonBase、commonCenter、commonGame、commonLogin、commonManager） 会以svn外链的形式 添加到需要使用到的工程文件夹中

1.commonCenter 登录服、游戏服、管理服启动后根据develop\server\bin\serverConfig\server.xml与中心服建立连接

储存玩家登录数据（账号id、角色id、登录服id、游戏服id、登录时间等）

提供匹配插件、排行榜插件、社交功能插件

二.数据工程 **(develop\common\data)**

用来存放生成协议部分、数据库表、以及用到的数据存储类相关的模板

三.机器人工程 **(develop\server\clientProject)**

可以做各种测试

四.GM工程 **(develop\server\gmClientProject)**

提供一些常见gm线上操作

五.工具工程 **(home\core\server\toolProject) （只Git上有）**

数据工程相关、数据库相关、开发中用到的一些便捷工具 最后会打成jar 供python调用

**MMO部分**

**一.战斗**

1.相关配置表说明

Bullet、BulletLevel 描述了子弹类型技能的类型、最大有效距离、子弹类型技能等级差异等信息

SkillProb 技能触发几率

SkillVar 技能变量表

Attribute 描述了各种属性信息

cd、cdGroup 描述了技能cd的冷却时间、cd组等信息

SkillStep、SkillStepLevel 描述了关于技能释放后多久显示伤害、等级差异等信息

Skill、SkillGroup、SkillLevel 描述了关于技能的释放方式、类型、消耗、等级差异和技能组等信息

Buff、BuffGroup、BuffLevel 描述了关于buff的影响类型、叠加方式、等级差异和buff组等信息

Attack、AttackGroup、AttackLevel 描述了关于攻击的是否带特殊效果、影响类型、等级差异和攻击组等信息

2.服务器驱动技能和客户端驱动技能的区别

AI逻辑执行的为服务器驱动技能

客户端发送请求为客户端驱动技能（引导、单人副本等可以选择用客户端驱动技能的模式）

3.战斗单位攻击流程 （UnitFightLogic）

根据技能id和技能等级查出配置表数据 用技能类型做差异处理

如果是服务器驱动的技能检测几率触发技能（BuffDataLogic里存储的） 成功触发后按SkillActionType来执行逻辑

执行技能步逻辑 设置技能步总时间和技能步时间比率 tick顺序执行技能步

检测到触发伤害或触发子弹类技能

直到技能步结束

4.buff相关说明 （BuffDataLogic）

每秒10次的tick 来检测间隔类buff触发的增加属性、造成的伤害、buff持续时间

提供多个对外接口 供攻击中几率触发的buff、攻击动作、物品触发的buff、活动特殊事件调用

5.怪物AI （UnitAILogic）

UnitAICommandType：无状态、守护原地、移动到、特殊移动到（没添加逻辑）、攻击到（没添加逻辑）、技能攻击到、攻击移动到

UnitAIFightStateType：无状态、待机、移动中、追集中、攻击中、回位

每秒10次的tick 驱动AI逻辑和AI命令逻辑

每秒1次的tick 驱动怪物的徘徊状态

6.AOI算法（有处理方案 还没添加）

**二.场景**

1.逻辑线程的执行器 初始化scenePool 数量可通过scenePoolSize配置 被初始化的scene会添加逻辑体（进出场景逻辑体、AOI逻辑体、位置逻辑体、战斗逻辑体、玩法逻辑体、触发器逻辑体）

进出场景逻辑体：在场景初始化后 通过scenePlaceConfig表来初始化显示单位 （载具未实现）

AOI逻辑体：分为全视野和九宫灯塔（未实现）

位置逻辑体：提供坐标、朝向计算判断的接口 （寻路未实现）

战斗逻辑体：提供计算伤害逻辑、处理攻击时刻逻辑、执行攻击动作、取范围内单位组等接口

玩法逻辑体：可根据需求自定义玩法

触发器逻辑体：可以做场景内特殊事件触发

2.通过sceneId查找配置来初始化场景驱动类型（区分了客户端和服务器）、特殊摆放物件等属性

**通信相关**

**一.通信部分说明**

通信底层是netty，序列化部分为自定义

支持proto数据解析 proto解析请求 -> 转shine请求逻辑 -> shine回复逻辑 -> 转proto回复

**二.自定义的序列化的特点**

1.变长写入 跟proto一样的zigzag编码类型

2.支持多种数据集合，原生数组、List、Set、Map

3.基础类型的集合 未使用java原生集合 而是基于koloboke改良版本的自定义集合（具备无迭代器高效遍历，无checkCast的类型转化开销，并且可支持遍历中修改的特性）

4.支持继承 每个结构 会生成唯一的标识id 用来反序列化（类似proto的any）

5,支持空传（类似proto的optional）

6.支持增量（类似proto的tag，不过使用的是向后添加）

7.支持增量读写 可以在生成的类中 添加你想要的属性和方法

8.工具支持版本检查 以防开发期的版本错误 如旧版客户端连服务器、服务器访问过期数据库、双端加载过期配置表（此项需要客户端用shine才行）

9.支持存库数据增量更新检查 标记线上模式后 不得作出不可逆修改（即非向后添加，新增类之外的修改，如修改原有字段的名字和类型）

工具会直接报错提示 以防开发人员误操作 造成数据库毁

10.支持刷库操作 如特别想删除已废弃字段 可做差异化生成 洗数据库

11.序列化区分完整读写和简版读写

简版读写无头长验证包体更小 适用于严格版本匹配的客户端（如页游和支持热更的手游）

**登录流程**

1.获取客户端版本信息 客户端进行app更新 资源更新

2.http连接登录服校验数据版本 未注册的User会创建User表

3.成功后将用户登录到中心服 分配区服逻辑

4.将用户数据发送到游戏服 游戏服生成token回复中心服 （预备登录）

5.中心服将token回复登录服 登录服转发客户端

6.连接游戏服 用token进行验证 成功后返回角色列表

无角色 客户端会进行创建角色。

有角色 登录过程会异步从db中取出角色数据 反序列化好并且组好发到中心服的数据包

7.发送角色登录协议 成功后将角色登录消息发到中心服

8.中心服缓存数据 将center的数据包返回游戏服 游戏服会用Player对象存储

选择线程切换线程 将用户首包数据发送给客户端

9.游戏服会继续执行进入场景的逻辑（如场景如要跨服，此时会执行跨服） 然后发送首次进场景的消息给客户端

**数据库**

1.数据库表的生成 支持自定义表结构（通过在数据工程中 填写TO模板类）

2.一个玩家对应数据库中的一条数据 功能数据用二进制存储到字段中

3.数据库入库为异步逻辑 不会占用主线程

4.每一次入库操作都是一个事务操作 能保证数据的一致性

5.添加好友、交易等行为（包括跨服操作） 是通过订单的形式保证数据的一致性

6.自定义表同样支持批量的增删改查 其中batchSelect不支持自定义的语句

**线程控制**

**一.线程分类**

观测守护线程、主线程、杂项线程、日志线程、DB写线程、IO线程组、池线程组、DB池线程组

**二.线程负责事务说明**

1.netty连接成功后在IO线程组中取出一个IO线程进行绑定 当接收到一个请求会从netty线程转到绑定好的IO线程 解析数据后添加到和玩家绑定的逻辑线程中等待处理（concurrentLinkedQueue）根据设置好的线程每轮方法上限（ShineSetting.cThreadRoundFuncMax） 来消费队列中的事务

2.序列化/反序列化都在IO线程

3.同场景的玩家会在同一个线程处理事务

4.杂项线程：检查子工程代码版本 如果版本号不对就重新加载配置（热更） 热更时所有逻辑线程（主+池）暂停

5.DB写线程：负责间隔入库操作、写入离线角色数据 成功写入后 会切换回主线程执行写DB后的逻辑

6.DB池线程组：做基础正删改查的

7.池线程组：根据ShineSetting.poolThreadNum来定义池线程数 用来绑定逻辑线执行器

8.观测守护线程：在其他线程启动后会添加到守护线程里的SMap（自定义的Map 默认为HashMap）中 在tick中循环检测SMap中的线程是否陷入死循环 其他线程也都守护着守护线程，如果守护线程挂掉了，重启一个守护线程。

9.日志线程：负责输出各种日志、输出到控制台

10.主线程：负责读角色表的操作和全局事务的处理（例如工会、组队、找角色、上下线等）

**框架内用到的枚举 （develop\common\config\game\enum）**

定义以xlsx表格的形式存在 方便策划查看枚举类型

**配置表**

跟业务逻辑关联 可兼容自定义类型

**Proto结构体定义**

1.protoDefine.xml说明

isonce创建协议号和初次包体 只生成一次 需要手写赋值的情况

proto中可以增加自定义结构体 msgProto再增加自定义枚举

msgProto里边的Request和Response协议号 根据protoDefine来生成

2.使用流程（需要完善）

protoDefine.xml

改完proto文件

LOST\develop\server\tools\dataExportAll.py

LOST\develop\server\toolsGame\protoExport.py

LOST\develop\common\tools\main2.py