

赢三张服务器文档

部署

- Game Server
 - 游戏服务器，玩家直接连接至游戏服务器，处理玩家逻辑。
- Robot Server
 - 机器人,从数据库中加载机器人配置，模拟玩家操作，同样连接至游戏服务器。
- CDKey
 - 生成奖励CDKey。
- iapppay-server
 - 爱贝充值，处理爱贝充值渠道回调，回调处理成功则发送http请求至游戏服务器进行充值。
 - 本服务只处理爱贝平台的充值回调，其它充值回调直接由游戏服务负责。
- 数据库使用MongoDB。
 - 对应的数据库索引可通过script/user_index.js来创建。

目录结构

```
├── cdkey
│   └── main.go(生成CDKey)
├── client
│   └── client.go
├── config(配置)
│   ├── config.go
│   ├── config_db.go
│   └── match_config.go
├── game(游戏服务器)
│   ├── domain(游戏业务逻辑)
│   │   ├── cdkey
│   │   │   ├── cd_key.go
│   │   │   └── cd_key_gain_record.go(CDKey领取记录)
│   │   ├── forbidWords(关键词过滤)
│   │   │   └── forbid_words.go
│   │   ├── game(游戏逻辑)
│   │   │   ├── game_desk_manager.go
│   │   │   ├── game_item.go(游戏主逻辑)
│   │   │   ├── game_log.go
│   │   │   ├── game_logic.go(赢三张规则)
│   │   │   └── game_manager.go
│   └── offlineMsg(离线消息,服务器端向玩家消息队列发送消息，如玩家离线，此模块负
```

责保存至数据库)

```

└── offline_msg.go
└── pay(充值日志)
    ├── hm_pay_log.go
    ├── iapp_pay_log.go
    ├── lenovo_pay_log.go
    ├── pay_log.go
    └── qf_pay_log.go
└── prize(奖励相关)
    ├── exchange_goods.go(兑换物品)
    ├── exchange_goods_log.go(兑换物品日志)
    ├── online_prize.go(在线奖励)
    ├── online_prize_gain_record.go(在线奖励领取记录)
    ├── online_prize_gain_records.go
    ├── prize_mail.go(邮件奖励)
    ├── prize_mails.go
    ├── sign_in_record.go(签到记录)
    ├── task_prize.go(任务奖励)
    ├── user_task.go
    ├── user_tasks.go
    └── vip_prize.go(VIP奖励)
└── randBugle(随机大喇叭，服务器端随机广播的系统消息)
    └── rand_bugle.go
└── rankingList(排行榜)
    ├── ranking_item.go
    └── ranking_list.go
└── report(处理玩家举报)
    └── report_log.go
└── slots(老虎机)
    ├── lucky_wheel_config.go(大转盘配置)
    └── slot_machine.go(老虎机)
└── stats(统计)
    ├── ai_fortune_log.go(AI财富统计)
    ├── match_log.go(比赛日志)
    └── online_log.go(在线日志)
└── user(玩家相关)
└── background_user_manager.go(游戏前端切到后台时会通知游戏服务器，此时游戏

```

逻辑相关消息将不会再发往前端)

```
| | | | ——— fake_ranking_list.go(伪造的充值排行榜->)
```

```
| | | | ——— game_player.go(玩家对象)
```

```
| | | | ——— match_record.go(比赛记录)
```

```
| | | | ——— player_manager.go(在线玩家管理器,管理当前在线玩家, 向玩家发消息, 踢出玩家
```

等)

- | | |—— ranking_list_updater.go(每隔一分钟刷新一次排行榜)
- | | |—— shop_log.go(玩家购买商品日志)
- | | |—— user.go(玩家信息)
- | | |—— user_cache.go(缓存当前在线玩家信息，可获取任意玩家信息)
- | | |—— user_fortune.go(玩家财富信息)
- | | |—— user_fortune_log.go(玩家财富更新日志)
- | | |—— user_fortune_manager.go(玩家财富信息管理器)
- | | |—— user_log.go(玩家日志，主要是登录记录)
- | | |—— vip_config.go(VIP配置)
- | |—— handlers(消息处理器)
 - | |—— admin(后台相关接口)
 - | |—— get_online_count_handler.go(获取当前在线人数)
 - | |—— lock_user_handler.go(冻结/解冻玩家)
 - | |—— query_user_by_id_handler.go(根据玩家ID查询玩家信息)
 - | |—— query_user_by_name_handler.go(根据玩家名字查询玩家信息)
 - | |—— send_prize_mail_handler.go(向玩家发送邮件奖励)
 - | |—— send_system_msg_handler.go(发送系统消息)
 - | |—— set_ai_win_rate_handler.go(设置AI胜率)
 - | |—— set_cur_version_handler.go(设置当前版本号，用于客户端版本更新时发放奖

励)

- | | |—— set_user_fortune_handler.go(修改玩家财富信息)
- | |—— cdkey
 - | |—— exchange_cd_key_handler.go(兑换CDKey奖励)
- | |—— chat
 - | |—— chat_msg_handler.go(聊天消息)
- | |—— game
 - | |—— app_enter_background_handler.go(客户端进入后台消息)
 - | |—— app_enter_foreground_handler.go(客户端切到前台消息)
 - | |—— enter_game_handler.go(进入游戏)
 - | |—— join_wait_queue_handler.go(万人场上桌)
 - | |—— leave_game_handler.go(离开游戏)
 - | |—— leave_wait_queue_handler.go(万人场下桌)
 - | |—— lookup_bet_gold_handler.go(万人场旁观下注)
 - | |—— match_result_handler.go(比赛结果,游戏结束后 (game_item.go处理)，会将

结果发至玩家消息队列，然后由此处理器处理,主要是完成相关游戏任务)

- | | |—— op_card_handler.go(牌桌内操作跟，弃，加注等)
 - | |—— reward_in_game_handler.go(打赏消息)
- | |—— msg_registry.go(消息路由表)
- | |—— pay(充值相关)
 - | |—— hm_pay_handler.go(海马充值渠道回调)
 - | |—— iapp_pay_handler.go(爱贝充值渠道回调)

```

| | | |—— lenovo_pay_handler.go(联想充值渠道回调)
| | | |—— qf_pay_handler.go(起凡充值渠道回调)
| | | |—— prize(奖励)
| | | |—— bind_prize_address_handler.go(绑定地址)
| | | |—— buy_daily_gift_bag_handler.go
| | | |—— exchange_goods_by_score_handler.go(积分兑换奖励)
| | | |—— gain_mail_prize_handler.go(领取邮件奖励)
| | | |—— gain_online_prize_handler.go(领取在线奖励)
| | | |—— gain_task_prize_handler.go(领取任务奖励)
| | | |—— gain_vip_prize_handler.go(领取VIP奖励)
| | | |—— get_prize_mails_handler.go(前端获取奖励邮件列表)
| | | |—— server_prize_mail_handler.go(后台接口发送奖励邮件)
admin/send_prize_mail_handler.go, 消息会路由到此处理器处理)
| | | |—— sign_in_handler.go(签到)
| | | |—— sign_in_record_handler.go(前端获取玩家签到记录)
| | | |—— subsidy_prize_handler.go(东山再起奖励)
| | | |—— rankingList(排行榜)
| | | |—— get_ranking_list_handler.go(前端获取排行榜列表)
| | | |—— register_handlers.go(注册消息处理器)
| | | |—— report
| | | |—— report_user_handler.go(玩家举报)
| | | |—— slots
| | | |—— gain_slot_machine_prize_handler.go(领取老虎机奖励)
| | | |—— play_lucky_wheel_handler.go(大转盘)
| | | |—— play_slot_machine_handler.go
| | | |—— play_slot_machine_handler_test.go
| | | |—— replace_slot_machine_card_handler.go(大转盘换牌)
| | | |—— stats
| | | |—— get_online_status_handler.go
| | | |—— user
| | | |—— exchange_game_goods_handler.go(兑换物品)
| | | |—— exchange_gold_handler.go(兑换金币)
| | | |—— get_match_record_handler.go(前端获取比赛记录)
| | | |—— get_recharge_info_handler.go(前端获取充值信息)
| | | |—— get_shipping_address_handler.go(前端获取玩家绑定地址信息)
| | | |—— get_shop_logs_handler.go
| | | |—— get_user_info_handler.go(前端获取玩家信息)
| | | |—— lock_user_handler.go(admin/lock_user_handler.go冻结玩家, 由此处理器处理)
| | | |—— login_handler.go(玩家登录)
| | | |—— robot_set_gold_handler.go(机器人修改财富处理器)
| | | |—— update_gold_handler.go(更新金币消息)
| | | |—— update_recharge_diamond_handler.go

```

- | | |—— update_user_info_handler.go
- | | |—— use_magic_item_handler.go(使用道具)
- | |—— main.go
- | |—— server
- | |—— server.go(此模块处理客户端连接, 接收客户端消息, 发至相关session的消息队列)
- | |—— session.go(客户端会话)
- |—— game_logic.go
- |—— pb(protobuf协议编译后生成)
 - |—— client_msg.pb.go
 - |—— config.pb.go
 - |—— game.pb.go
 - |—— msgid.pb.go
 - |—— rpc.pb.go
 - |—— server_msg.pb.go
 - |—— server_msgid.pb.go
- |—— qf.go
- |—— readme.md
- |—— robot(机器人)
 - |—— game_logic.go
 - |—— main.go
 - |—— robot
 - |—— robot.go
- |—— script
 - |—— user_index.js(数据库索引)
- |—— test.go
- |—— test2.go
- |—— util(工具类)
- |—— bugle.go(大喇叭)
- |—— common_error.go
- |—— func.go
- |—— game_type.go
- |—— hash_util.go(计算对象的hash值, 通常用于判断数据是否发生变化, 是否需要入库)
- |—— lru.go
- |—— mongo.go(mongodb数据库相关)
- |—— msgid_name.go
- |—— stack.go
- |—— task_type.go
- |—— time_util.go
- |—— util.go

协议

通信协议使用protobuf,可执行proto目录下的make.go生成后的程序来编译协议。 前后端数据包格

式为proto/pb/client_msg.proto,

```
message ClientMsg {  
    required int32 msgId = 1;  
    optional bytes msgBody = 2;  
}
```

消息至服务器后，统一转换成如下格式:

```
message ServerMsg {  
    optional bool client = 1;(用于区分消息来源， 是否是从客户端发来的消息)  
    optional string srcId = 2;  
    repeated string dstId = 3;  
    optional int32 msgId = 4;  
    optional bytes msgBody = 5;  
}
```

服务器接收完客户端消息后，会将其转换成ServerMsg,然后投递到对应玩家的消息队列中，然后将由msg_registry.go来分发至对应消息处理器。服务器逻辑可通过player_manager.go的SendServerMsg系列接口向特定玩家的消息队列发送消息。

SendServerMsg(srcId string, dstIds []string, msgId int32, body proto.Message)

参数:

srcId: 发送者用户ID,系统发送可设为空字符串""

dstIds: 消息接收者用户ID列表

msgId: 消息Id

body: 消息体

如果玩家在线，则发送成功，不在线，关键消息可通过offline_msg.go的PutOfflineMsg保存至数据库，待玩家上线时会重新向其发送。

关键代码说明

- 通信部分

前后端使用websocket通信，每个客户端会启动2个goroutine来处理逻辑，一个用于从客户端接收消息，一个从消息队列读取消息，处理业务逻辑。分别由server.go及session.go来处理。

server.go::handleClient(第一个goroutine)

```

for {
    var data []byte
    // 客户端超时时间，当前设置10分钟，超过10分钟没有数据到达服务器，则将客户端踢出
    conn.SetReadDeadline(time.Now().Add(time.Minute * 10))
    err := websocket.Message.Receive(conn, &data)
    if err != nil {
        glog.Info("error receiving msg:", err)
        break
    }

    conn.SetReadDeadline(time.Time{})

    clientMsg := &pb.ClientMsg{}
    err = proto.Unmarshal(data, clientMsg)
    if err != nil {
        glog.V(1).Info("unmarshal client msg failed!")
        break
    }

    msg := &pb.ServerMsg{}
    msg.Client = proto.Bool(true)
    msg.MsgId = clientMsg.MsgId
    msg.MsgBody = clientMsg.MsgBody

    // 将接收的客户端消息发送到会话消息队列
    sess.mq <- msg
}

```

session::run(第二个goroutine)

```

for {
    select {
        // 服务器停止时关闭此channel,然后当前在线玩家离线,保存相关数据,待所有玩家都退出以后,服务停止。
        case <-GetServerInstance().stopChan:
            return
        case msg, ok := <-s.mq:
            if !ok {
                return
            }

            // 将消息队列中的消息取出,交由相关逻辑处理器处理,无消息时,阻塞。
            res := dispatcher.DispatchMsg(msg, s)
            if res != nil {
                s.SendToClient(res)
            }
        case <-s.exitChan: // 踢出玩家时可关闭此channel,调用session::Kickout即可。
            glog.Info("==>Kickout sess:", s)
            return
    }
}

```

- 游戏逻辑部分

游戏逻辑主要由domain/game/game_logic.go及domain/game/game_item.go处理。

game_logic.go处理赢三张规则部分, game_item.go处理牌桌内逻辑。

洗牌规则:game_logic.go::ShuffleCards


```

func (logic *GameLogic) ShuffleCards(pos []int) {
    // 随机打乱
    sort.Sort(ByteSlice(logic.cardList))

    logic.pos = pos
    logic.sortedPos = []int{}
    logic.sorted = false

    // 首先按后台设置的各个牌型的概率抽牌,然后按pos的位置进行大小排序,pos为按玩家幸运值排序后的
    的玩家位置列表。
    for i := 0; i < len(pos); i++ {
        t := config.GetCardConfigManager().GetRandCardType(logic.gameType)

        if t == CARD_TYPE_SINGLE {
            logic.changeSingle(i)
        } else if t == CARD_TYPE_DOUBLE {
            logic.changeDouble(i)
        } else if t == CARD_TYPE_SHUN_ZI {
            logic.changeShunZi(i)
        } else if t == CARD_TYPE_JIN_HUA {
            logic.changeJinHua(i)
        } else if t == CARD_TYPE_SHUN_JIN {
            logic.changeShunJin(i)
        } else if t == CARD_TYPE_BAO_ZI {
            logic.changeBaoZhi(i)
        }
        logic.sortedPos = append(logic.sortedPos, i)
    }

    cardTypeComp := NewCardTypeComp(logic)
    cardTypeComp.pos = logic.sortedPos
    sort.Sort(cardTypeComp)
    logic.sortedPos = cardTypeComp.pos
    logic.sorted = true
}

```

编译

- 服务器端采用go语言编写, 首先安装go语言编译器,建议使用Go1.4.2版本。
- 将golib目录下的库拷贝至server2/src(只需一次即可)
- 设置GOPATH环境变量export GOPATH=server2的目录
- 进入server2/src/game执行go build编译game
- 进入server2/src/robot执行go build编译robot
- 启动服务可通过script/service.sh执行service.sh start|stop, 或自行编写启动脚本。

- 爱贝充值渠道回调服务器iapppay-server可直接使用其目录下编译好的iapppay-server.jar。如需自行编译可安装<http://leiningen.org/>,然后至iapppay-server目录,执行lein ring uberjar即可在target目录下生成对应jar包。本服务默认绑定3000端口,可通过环境变量PORT来更改。