player\_dict：玩家进程字典；

跨服服务端API：

%% @doc 发消息给游戏服  
%% 返回：undefined-失败，ok-成功  
send\_to\_game(Bin) -> %%Bin为发送数据  
 GameList = mod\_cross\_server:get\_game\_list(),  
 Fun = **fun**(#ets\_cross\_server{send\_pid = SendPid}) ->  
 cross\_server:send\_to\_sid(SendPid, Bin)  
 **end**,  
 lists:foreach(Fun,GameList)**.**send\_to\_game(ServerId, Bin) -> %%ServerId为发送进程  
 **case** mod\_cross\_server:get\_send\_pid(ServerId) **of** undefined ->  
 ?ERROR\_MSG("Not send\_pid:~p", [{ServerId,Bin}]),  
 undefined;  
 SendPid ->  
 cross\_server:send\_to\_sid(SendPid, Bin),  
 ok  
 **end.**

%% 玩家record定义

-record(r\_player\_info, {  
 player\_id = 0 % 玩家id  
 ,user\_id = 0 % 帐号id  
 ,user\_name = "" % 帐号名  
 ,name = "" % 角色名  
 ,union\_id = 0 % 军团ID  
 ,sex = 0 % 性别  
 ,exp = 0 % 经验值  
 ,level = 1 % 等级  
 ,avatar\_id = 0 % 头像  
 ,fight = 0 % 战斗力  
 ,create\_time = 0 % 创建时间  
 ,reset\_time = 0 % 重置时间  
 ,offline\_time = 0 % 下线时间  
 ,last\_login\_time = 0 % 最后一次登录时间  
 ,ip = "" % 用户ip  
 ,heat\_error = 0 % 心跳错误次数(跨天重置为0)  
 ,state = 0 % 状态  
 ,ban\_chat\_date=0 % 禁言时间  
 ,login\_day = 0 % 登录天数  
 ,vip\_lev = 0 % vip等级  
 ,channel\_id = "" % 渠道id  
 ,is\_merge\_first\_login = 1 % 是否是合服第一次登陆 -- 用来做一些合服处理 1 是 0 否  
 ,platform = 0 % 0=其他 1=android 2=ios  
})**.**

**Union\_id 军团Id；**

**增加军团的经验值函数：**

add\_union\_exp(AddExp) ->  
 #r\_player\_info{union\_id = UnionId} = player\_dict:get\_player\_info(),  
 ?IF(UnionId == 0, skip, mod\_union:cast({add\_union\_exp, misc:get\_player\_sid(?PLAYER\_ID), UnionId, AddExp})),  
 ok**.**

**mod\_cache.erl**

%% 保存标识  
-define(SAVE\_FLAG\_KEY(PlayerId), {?MODULE, save\_flag, PlayerId})**.**get\_save\_flag(PlayerId) -> erlang:get(?SAVE\_FLAG\_KEY(PlayerId))**.**set\_save\_flag(PlayerId, Flag) -> erlang:put(?SAVE\_FLAG\_KEY(PlayerId), Flag)**.**del\_save\_flag(PlayerId) -> erlang:erase(?SAVE\_FLAG\_KEY(PlayerId))**.**

do\_cast({clear\_offline\_player, IsForce}, #state{ets\_name = EtsName} = State) ->  
 erlang:statistics(wall\_clock),  
  
 NowSec = misc\_timer:now\_seconds(),  
 ClearPlayers = ets:foldl(  
 **fun**(EtsPlayer, Acc) ->  
 **case** EtsPlayer#ets\_player**.**offline\_time **of** 0 -> Acc;% 玩家在线  
 OfflineTime ->

%% 未到清除时间并且没有进行强制清除  
 **case** (NowSec - OfflineTime) > ?CACHE\_CLEAR\_TIME **of** false **when** IsForce =/= 1 ->  
 % 还没到清除时间  
 Acc;  
 \_ ->  
 PlayerId = EtsPlayer#ets\_player**.**player\_id,  
 % 清除时存档  
 **case** get\_save\_flag(PlayerId) **of** true -> save\_cache\_player(EtsName, EtsPlayer);  
 \_ ->  
 skip  
 **end**,  
  
 ets:delete(EtsName, PlayerId),  
 [PlayerId | Acc]  
 **end  
 end  
 end**, [], EtsName),

**模块说明：**

**玩家背包仓库接口模块：(api\_package)**

**获取道具数量：**

%% @doc 获取道具数量  
%% 参数：  
%% PackageType = integer()  
%% Kvs = {#item.id, Id, Bind, Dealine} |  
%% {#item.pos, Pos, Bind, Dealine} |  
%% {#item.tid, Tid, Bind, Dealine}  
%% 返回值：integer()  
num(PackageType, Kv) ->  
 lists:foldl(**fun**(#item{num = Num}, Acc) ->  
 Num + Acc  
 **end**, 0, lib\_player\_package:get(PackageType, Kv))**.**

**根据某背包类型指定模板id的计算道具数量：**

num\_by\_tid(PackageType, Tid) ->  
 lib\_player\_package:num(PackageType, {#item**.**tid, Tid, all, deadline})**.**

**从背包删除指定消耗id和指定数量的道具，并附上道具使用源头**

?SOURCE\_ARTIFACT\_ACTIVE)

api\_package:del\_by\_tid(?PACKAGE\_TYPE\_BAG,NeedItemTid,NeedNum,?SOURCE\_ARTIFACT\_ACTIVE),

**事件派发：**

%% @doc 派发某个类型的事件  
%% 玩家进程调用，事件安全  
dispatch(Event) ->  
 dispatch(Event, [])**.**dispatch(Event, Data) ->  
 do\_dispatch(Event, Data)**.**

**当使用某些道具和装备属性发生改变时重新计算玩家属性：**

%% @doc 属性变化时计算玩家属性  
calc\_attribute() ->  
 HeroList = hero\_dict:get\_battle\_list(),  
 [calc\_attribute(Hero,false)**||**Hero<-HeroList],  
 AllPower = get\_all\_power(),  
 player\_event:dispatch(?EVENT\_FIGHT, AllPower)**.**

**获取出站武将列表：**

get\_battle\_list() ->  
 player\_dict:get(?PD\_BATTLE\_HERO,[])**.**

**计算玩家属性的算法：**

calc\_attribute(Hero,IsSend) ->  
 **case** player\_dict:get(?PD\_LOGIN\_DONE) **of** true ->  
 {NewFight,NewAttrList} = calc\_attribute\_impl(Hero),  
 OldAttrList = Hero#r\_hero**.**attr\_list,  
 OldFight = Hero#r\_hero**.**fight,  
 List1 = diff\_fight(OldFight,NewFight,NewAttrList),  
 List2 = diff\_attr\_list(OldAttrList, NewAttrList),  
 List = List1 ++ List2,  
 **case** List =/= [] **of** true ->  
 {ok, Bin} = ?PT\_HERO\_ENCODE(?SC\_HERO\_UPDATEATTR, #sc\_hero\_updateattr{id = Hero#r\_hero**.**id,attr\_list = List1}),  
 lib\_send:send\_to\_me(Bin);  
 false ->  
 ok  
 **end**,  
 **case** IsSend **of** true ->  
 AllPower = get\_all\_power(),  
 player\_event:dispatch(?EVENT\_FIGHT, AllPower);  
 false ->  
 ok  
 **end**,  
 %player\_event:dispatch(?EVENT\_ATTR\_FINISH, [0]),  
 ok;  
 \_ ->  
 ok  
 **end.**

%% @doc 计算各个模块属性  
calc\_module\_attribute(Hero) ->  
 Fun = **fun**(Module, {Acc0, Acc1}) ->  
 **case** Module **of** {Mod, 1} ->  
 %%不计算战斗力  
 {Acc0, attr\_tool:merge\_attr(Mod:calc\_attr(Hero), Acc1)};  
 Mod ->  
 {attr\_tool:merge\_attr(Mod:calc\_attr(Hero), Acc0), Acc1}  
 **end  
 end**,  
 lists:foldl(Fun, {[], []}, player\_module:hero\_attr\_modules())**.**

**具体实现：**

calc\_attribute\_impl(Hero) ->  
 {Attribute0, Attribute1} = calc\_module\_attribute(Hero), %%?LDEBUG(Attribute0),  
 AttrList1 = attr\_tool:sum\_attr(Attribute0, Attribute1), %%?LDEBUG(AttrList1),  
 %attr\_tool:wrire\_log("属性合并",[{Type,Value}||#r\_attrib{type = Type,value = Value}<-AttrList1] ),  
 AttrList =

属性合并：

sum\_attr(First, Second) ->  
 AttrList = merge\_attr(First,Second),  
 lists:foldl(**fun**(#r\_attrib{type = Type,value = Value},Acc) ->  
 **case** lists:keyfind(Type,1,?ATTR\_PER\_MAP) **of** false -> Acc;  
 {\_,FinnalType} ->  
 Acc2 = lists:keydelete(Type,#r\_attrib**.**type,Acc),  
 **case** lists:keyfind(FinnalType,#r\_attrib**.**type,Acc) **of** false ->  
 Acc2;  
 #r\_attrib{value = OldValue} ->  
 NewValue = util:ceil(OldValue\*(10000+Value)/10000),  
 lists:keyreplace(FinnalType,#r\_attrib**.**type,Acc2,#r\_attrib{type = FinnalType,value = NewValue})  
 **end  
 end  
 end**,AttrList,AttrList  
 )**.**

属性过滤：

filter\_attr(Hero,AttrList1),%%?LDEBUG(AttrList),  
 %attr\_tool:wrire\_log("属性过滤",[{Type,Value}||#r\_attrib{type = Type,value = Value}<-AttrList] ),  
 Fight = attr\_tool:calc\_sum\_fight(AttrList),  
 %attr\_tool:wrire\_log("总战力 物理攻击 \* 1.2 + 法术攻击 \* 1.2 + 物理防御 \* 1 + 法术防御 \* 1 + 生命 \* 0.05 ",Fight ),  
 %attr\_tool:wrire\_log(" ","~n~n" ),  
 NewHero = Hero#r\_hero{fight = Fight,attr\_list = AttrList},  
 hero\_dict:set\_battle\_hero(NewHero),  
 {Fight,AttrList}**.**

**玩家缓存模块：**

**mod\_cache 玩家缓存操作函数模块；**

**api\_cache 玩家缓存接口模块，通过接口对数据库进行相应的持久化操作（使用数据库代理db\_agent实现）；**

**数据库模块：**

**db\_mysqlutil的函数用于编译相应的sql语句，以供db\_mysql函数调用，有了db\_mysql的数据库持久化操作，再使用具体模块的代理生成具体的持久化操作db\_agent\_xxx**

**项目包说明：**

**db\_agent:游戏数据库代理模块；**

**mod\_player/lib ,lib\_xxx用于实现具体的算法和也无逻辑；**

**数据库启动：**

-module(db\_agent)**.**

**----🡪**

-module(db\_agent\_mysql)**.**

**----🡪(mysql连接客户端)**

-module(mysql)**.**

**数据库操作模块：(以玩家player模块为例)**

-module(db\_agent\_player\_mysql)**.**

**-------🡪**

-module(db\_mysql)**.**

**-------🡪**

-module(db\_mysqlutil)**.**

**-------🡪(mysql连接客户端)**

-module(mysql)**.**