新玩家初始化模块信息：

?NEW\_PLAYER ->  
 PlayerInfo1 = PlayerInfo#r\_player\_info{  
 state = ?OLD\_PLAYER, reset\_time = NowSec, ip = Ip  
 },  
 player\_dict:set\_player\_info(PlayerInfo1),  
 init\_module()

老玩家加载模块信息：

?OLD\_PLAYER ->  
 % 将阵营设置为0  
 player\_dict:set\_player\_info(PlayerInfo#r\_player\_info{ip = Ip}),  
 load\_module();

model\_player player\_login

do\_init

gen\_server:init

do\_init

do\_cast

gen\_server:cast({load\_data},States)

load\_data

player\_dict：玩家进程字典；

data配置表erl文件模块：

命名规则：data\_xx\_xx

装备400级满级；

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}**.**

**背包模块：**

**在往背包里面添加道具时判断背包是否已满：**

HeroBagFull = lib\_player\_hero:check\_hero\_bag\_full(length(HeroItemList))

**将背包数据存储到武将背包进程中：**

set\_hero\_bag(Bag) **when** is\_record(Bag,r\_hero\_bag) ->  
 api\_cache:dirty\_data(#ets\_player**.**hero\_data),  
 player\_dict:set(?PD\_HERO\_BAG,Bag)**.**

**获取武将背包：**

get\_hero\_bag() ->  
 player\_dict:get(?PD\_HERO\_BAG)**.**

**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）；**

%% 从数据库加载玩家数据  
load\_player\_data\_from\_db(PlayerId, Pos) ->  
 **case** Pos **of** #ets\_player**.**info\_data ->  
 db\_agent\_player:load\_player\_info(PlayerId);  
 #ets\_player**.**task\_data ->  
 db\_agent\_task:load\_player\_task(PlayerId);  
   
 #ets\_player**.**package\_data ->  
 db\_agent\_package:load\_player\_package(PlayerId);  
   
 #ets\_player**.**mail\_data ->  
 db\_agent\_mail:load\_player\_mail(PlayerId);  
 #ets\_player**.**hero\_data ->  
 db\_agent\_hero:load\_player\_hero(PlayerId);  
 #ets\_player**.**shop\_data ->  
 db\_agent\_shop:load\_shop\_data(PlayerId);  
 #ets\_player**.**dup\_data ->  
 db\_agent\_dup:load\_player\_dup(PlayerId);  
 #ets\_player**.**assets\_data ->  
 db\_agent\_assets:load\_player\_assets(PlayerId);  
 #ets\_player**.**tactical\_data ->  
 db\_agent\_tactical:load\_player\_tactical(PlayerId);  
 #ets\_player**.**recruit\_data ->  
 db\_agent\_recruit:load\_player\_recruit(PlayerId);  
 #ets\_player**.**other\_data ->  
 db\_agent\_player\_other:load\_player\_other(PlayerId);  
 #ets\_player**.**mask\_data ->  
 db\_agent\_mask:load\_player\_other(PlayerId);  
 #ets\_player**.**west\_beast\_data ->  
 db\_agent\_beast:load\_player\_beast(PlayerId);  
 Pos ->  
 RecList = record\_info(fields, ets\_player),  
 ?ERROR\_MSG("Error player\_id=~p pos=~p", [PlayerId, lists:nth(Pos - 1, RecList)])  
 **end.**

**数据操作规范：**

save\_player\_hero(PlayerId, #r\_player\_hero{battle\_heros = BattleHeros, hero\_bag = #r\_hero\_bag{next\_id = NextId,hero\_list = HeroList},  
 hero\_equips = Equips,open\_pos = OpenPos,open\_embattle\_pos = EmbattlePos,hero\_lifestar = LifeStar, hero\_talisman = Talisman, deity\_beast = DeityBeast}) ->  
 #r\_deity\_beast{battle\_id = BattldId, deity\_beast\_list = BeastList} = DeityBeast,  
 NewData = #r\_player\_hero{  
 battle\_heros = util:pack\_data([**begin** ?RECORD\_TO\_KVLIST(Rec, r\_hero) **end ||** Rec <- BattleHeros]),  
 hero\_bag = util:pack\_data(?RECORD\_TO\_KVLIST(#r\_hero\_bag{next\_id = NextId,hero\_list = util:pack\_data([**begin** ?RECORD\_TO\_KVLIST(Rec, r\_hero) **end ||** Rec <- HeroList])},r\_hero\_bag)),  
 hero\_equips = util:pack\_data([**begin** ?RECORD\_TO\_KVLIST(Rec, item) **end ||** Rec <- Equips]),  
 open\_pos = OpenPos,  
 open\_embattle\_pos = EmbattlePos,  
 hero\_lifestar = util:pack\_data([**begin** ?RECORD\_TO\_KVLIST(Rec, item) **end ||** Rec <- LifeStar]),  
 hero\_talisman = util:pack\_data([**begin** ?RECORD\_TO\_KVLIST(Rec, item) **end ||** Rec <- Talisman]),  
 deity\_beast = util:pack\_data(?RECORD\_TO\_KVLIST(#r\_deity\_beast{battle\_id = BattldId,deity\_beast\_list = util:pack\_data([**begin** ?RECORD\_TO\_KVLIST(Rec, r\_player\_deity\_beast) **end ||** Rec <- BeastList])},r\_deity\_beast))  
 },

**数据库相关数据的操作类型包括三类：**

**基本数据类型：不作任何额外的转换操作**

**列表数据类型：遍历列表中的日志，并将日志转换成键值列表（**RECORD\_TO\_KVLIST**）,再进行组包或解包；**

**元组数据类型：将日志转换成键值列表（**RECORD\_TO\_KVLIST**），在进行组包或解包；**

-define(LOAD\_MYSQL\_POOL\_EX(PlayerId, Table, Record, Fun),  
 erlang:hd(  
 [  
 ?IF(\_\_Data\_\_ =:= undefined, undefined, Fun(\_\_Data\_\_))  
 **||** \_\_Data\_\_ <- [ ?LOAD\_MYSQL\_POOL(PlayerId, Table, Record) ]  
 ]  
 ))**.**

-define(LOAD\_MYSQL\_POOL(PlayerId, Table, Record),  
 erlang:hd(  
 [  
 ?IF(\_\_KVList\_\_ =:= [], undefined, ?KVLIST\_TO\_RECORD(\_\_KVList\_\_, Record) )  
 **||** \_\_KVList\_\_ <- [  
 db\_mongoutil:document\_to\_kvlist(  
 **begin** R = db\_mysql:select\_one(?GAME\_DB\_POOL,Table,["data"],[{"player\_id",PlayerId}]),  
 **case** R **of** null -> {};  
 \_ -> util:bitstring\_to\_term(R)  
 **end  
 end** )  
 ]  
 ])  
)**.**

**条件查询：**

db\_mysql:select\_one(?GAME\_DB\_POOL,Table,["data"],[{"player\_id",PlayerId}])

**查询字段：data**

**条件语句：player\_id=PlayerId**

**玩家武将数据：（hero\_dict）**

**（从武将背包里拿取武将）：**hero\_dict:get\_hero(Id)

**首发（首发最大数量，首发最大位置）：**

#r\_system\_param\_cfg{parameter = FirstMaxNum} = data\_system\_param:get(16),  
#r\_system\_param\_cfg{parameter = FirstMaxPos} = data\_system\_param:get(4),

**援军：**

#r\_system\_param\_cfg{parameter = SencondMaxPos} = data\_system\_param:get(5),

**获取出战武将列表:**

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

**进入替补名单：**

\_ ->  
 lists:min( lists:seq(FirstMaxPos + 1,FirstMaxPos + SencondMaxPos) -- SecondHeroList)

**获取出战武将：**

get\_battle\_hero(Id) ->  
 get\_battle\_hero(Id, [])**.**get\_battle\_hero(Id, Default) ->  
 HeroList = get\_battle\_list(),  
 **case** lists:keyfind(Id, #r\_hero**.**id, HeroList) **of** false -> Default;  
 Hero -> Hero  
 **end.**

**从配置中获取出战武将的最大的最大上限：**

#r\_system\_param\_cfg{parameter = MaxBattleNum} = data\_system\_param:get(17),

**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(api\_cache)**.(玩家数据缓存模块)**

**-------🡪**

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

**-------🡪**

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

**-------🡪**

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

**-------🡪**

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

**-------🡪(mysql连接客户端)**

-module(mysql)**.**

**保存数据到MySQL数据库池里面**

将数据替换保存（replace）到数据库池面：

?SAVE\_MYSQL\_POOL(PlayerId, NewData, ?T\_PLAYER\_HERO, r\_player\_hero)**.**

**——》**

-define(SAVE\_MYSQL\_POOL(PlayerId, Data, Table, Record),  
 [\_|ValueListTmp] = tuple\_to\_list(Data),  
 FieldList = record\_info(fields, Record),  
 db\_mysql:replace(?GAME\_DB\_POOL, Table, [player\_id,data],  
 [PlayerId,util:term\_to\_bitstring(  
 db\_mongoutil:make\_insert(FieldList, ValueListTmp)  
 )])  
)**.**

从数据库池里面加载获取数据：

-define(LOAD\_MYSQL\_POOL\_EX(PlayerId, Table, Record, Fun),  
 erlang:hd(  
 [  
 ?IF(\_\_Data\_\_ =:= undefined, undefined, Fun(\_\_Data\_\_))  
 **||** \_\_Data\_\_ <- [ ?LOAD\_MYSQL\_POOL(PlayerId, Table, Record) ]  
 ]  
 ))**.**

-define(LOAD\_MYSQL\_POOL(PlayerId, Table, Record),  
 erlang:hd(  
 [  
 ?IF(\_\_KVList\_\_ =:= [], undefined, ?KVLIST\_TO\_RECORD(\_\_KVList\_\_, Record) )  
 **||** \_\_KVList\_\_ <- [  
 db\_mongoutil:document\_to\_kvlist(  
 **begin** R = db\_mysql:select\_one(?GAME\_DB\_POOL,Table,["data"],[{"player\_id",PlayerId}]),  
 **case** R **of** null -> {};  
 \_ -> util:bitstring\_to\_term(R)  
 **end  
 end** )  
 ]  
 ])  
)**.**

select\_one(DB\_Pool, Table\_name, Fields\_sql, Where\_List) ->  
 ?STAT\_DB\_ACCESS(Table\_name, select),  
 Sql = db\_mysqlutil:make\_select\_sql(Table\_name, Fields\_sql, Where\_List),  
 get\_one(DB\_Pool, Sql)**.**

%% @doc Make select sql sentence.  
make\_select\_sql(Table\_name, Fields\_sql, Where\_List)

->  
 make\_select\_sql(Table\_name, Fields\_sql, Where\_List, [], [])**.**

%% @doc Make select sql sentence.  
make\_select\_sql(Table\_name, Fields\_sql, Where\_List, Order\_List, Limit\_num) ->  
 {Wsql, Count1} = get\_where\_sql(Where\_List),  
 WhereSql =  
 **if** Count1 > 1 -> lists:concat(["where ", lists:flatten(Wsql)]);  
 true -> ""  
 **end**,  
 {Osql, Count2} = get\_order\_sql(Order\_List),  
 OrderSql =  
 **if** Count2 > 1 -> lists:concat([" order by ", lists:flatten(Osql)]);  
 true -> ""  
 **end**,  
 LimitSql = **case** Limit\_num **of** [] -> "";  
 [Num] -> lists:concat([" limit ", Num])  
 **end**,  
 lists:concat(["select ", Fields\_sql, " from `", Table\_name, "` ", WhereSql, OrderSql, LimitSql])**.**

document\_to\_kvlist(Document) ->  
 **case** Document **of** {} -> [];  
 {Doc} -> document\_to\_kvlist\_1(Doc);  
 Doc **when** is\_list(Doc) -> [document\_to\_kvlist\_1(In) **||** In <- Doc];  
 Doc -> document\_to\_kvlist\_1(Doc)  
 **end.**

db\_mysql:select\_one(?GAME\_DB\_POOL,Table,["data"],[{"player\_id",PlayerId}]),

lists:concat(["select ", Fields\_sql, " from `", Table\_name, "` ", WhereSql, OrderSql, LimitSql])**.**

**-----------------🡪**

WhereList=[{"player\_id",PlayerId}]

{Wsql, Count1} = get\_where\_sql(Where\_List)

WhereSql =  
 **if** Count1 > 1 -> lists:concat(["where ", lists:flatten(Wsql)]);  
 true -> ""  
 **end**,

**数据库连接池：**?GAME\_DB\_POOL

Fields\_sql=[”data”];

**玩家英雄表：**Table\_name=?T\_PLAYER\_HERO

-define(T\_PLAYER\_HERO, t\_player\_hero)**.** % 注意添加合服处理：合

OrderSql =[]

LimitSql=[]

save\_player\_package(PlayerId, Data) ->  
 NewData = Data#r\_player\_package{  
 bag = util:pack\_data(package\_to\_kvlist(Data#r\_player\_package**.**bag))  
 , equip = util:pack\_data(package\_to\_kvlist(Data#r\_player\_package**.**equip))  
 , fragment = util:pack\_data(package\_to\_kvlist(Data#r\_player\_package**.**fragment))  
 , lifestar = util:pack\_data(package\_to\_kvlist(Data#r\_player\_package**.**lifestar))  
 , talisman = util:pack\_data(package\_to\_kvlist(Data#r\_player\_package**.**talisman))  
 , dragon = util:pack\_data(package\_to\_kvlist(Data#r\_player\_package**.**dragon))  
 , diagrams = util:pack\_data(package\_to\_kvlist(Data#r\_player\_package**.**diagrams))  
 },  
 ?SAVE\_MYSQL\_POOL(PlayerId, NewData, ?T\_PLAYER\_PACKAGE, r\_player\_package)**.**

% 从数据库加载玩家数据  
load\_player\_data\_from\_db(PlayerId, Pos) ->  
 **case** Pos **of** #ets\_player**.**info\_data ->  
 db\_agent\_player:load\_player\_info(PlayerId);  
 #ets\_player**.**task\_data ->  
 db\_agent\_task:load\_player\_task(PlayerId);  
 %#ets\_player.module\_data ->  
 % db\_agent\_task:load\_player\_module(PlayerId);  
 #ets\_player**.**package\_data ->  
 db\_agent\_package:load\_player\_package(PlayerId);  
 %#ets\_player.skill\_data ->  
 % db\_agent\_player:load\_player\_skill(PlayerId);  
 %#ets\_player.title\_data ->  
 % db\_agent\_player:load\_player\_title(PlayerId);  
 %#ets\_player.friend\_data ->  
 % db\_agent\_friend:load\_player\_friend(PlayerId);  
 #ets\_player**.**mail\_data ->  
 db\_agent\_mail:load\_player\_mail(PlayerId);  
 #ets\_player**.**hero\_data ->  
 db\_agent\_hero:load\_player\_hero(PlayerId);  
 #ets\_player**.**shop\_data ->  
 db\_agent\_shop:load\_shop\_data(PlayerId);  
 #ets\_player**.**dup\_data ->  
 db\_agent\_dup:load\_player\_dup(PlayerId);  
 #ets\_player**.**assets\_data ->  
 db\_agent\_assets:load\_player\_assets(PlayerId);  
 #ets\_player**.**tactical\_data ->  
 db\_agent\_tactical:load\_player\_tactical(PlayerId);  
 #ets\_player**.**recruit\_data ->  
 db\_agent\_recruit:load\_player\_recruit(PlayerId);  
 #ets\_player**.**other\_data ->  
 db\_agent\_player\_other:load\_player\_other(PlayerId);  
 #ets\_player**.**mask\_data ->  
 db\_agent\_mask:load\_player\_other(PlayerId);  
 #ets\_player**.**west\_beast\_data ->  
 db\_agent\_beast:load\_player\_beast(PlayerId);  
 Pos ->  
 RecList = record\_info(fields, ets\_player),  
 ?ERROR\_MSG("Error player\_id=~p pos=~p", [PlayerId, lists:nth(Pos - 1, RecList)])  
 **end.**

**邮件模块：**

-record(r\_mail,{  
 player\_id = 0,  
 id = 0,  
 common\_mail\_id = 0, %% 共享邮件id  
 title = "",  
 content = "",  
 type = 0, % 0系统 1公会 2后台  
 send\_time = 0,  
 read\_flag = 0, %% 0.未读 1.已读 2.提取  
 attach\_list = [],  
 source =0, %% 邮件来源  
 sender\_name = "" %% 发送者  
})**.**

**type=（0 系统,1公会,2后台）**

**将上述邮件分为两大类：**

MailType = **case** Type **of** 2 -> 1;  
 \_ -> 2  
**end**,

**2后台为类型1**

**其余归为类型2；**

check\_equip\_pos(Pos) ->  
 ?IF(Pos > 0 , ok, ?THROW\_ECODE(?LANG\_SYS\_ERR)),  
%%武将上阵位置

MainPos = Pos **div** 100,

%%武将上阵开放的最大位置   
 MaxPos = hero\_dict:get\_hero\_open\_pos(),

%%判断武将上阵位置是否大于开放的武将上阵的最大位置  
 ?IF(MainPos =< MaxPos,ok,?THROW\_ECODE(?LANG\_HERO\_POS\_ERR)),  
 SubPos = Pos **rem** 100,  
 ?IF(SubPos =< ?EQUIP\_MAX\_POS,ok,?THROW\_ECODE(?LANG\_HERO\_POS\_ERR))**.**

?IF(SubPos =< ?EQUIP\_MAX\_POS,ok,?THROW\_ECODE(?LANG\_HERO\_POS\_ERR))**.**

EQUIP\_MAX\_POS在下图标注位置：

****

%% 交换位置（武将更换装备）

%%将原先穿戴的老装备重新放入背包  
api\_package:add([OldEquip],true,?SOURCE\_MOUNT\_HERO\_EQUIP),

%%从武将进程字典里删除已穿戴的老装备  
hero\_dict:del\_equip(OldId),

%%将新装备重新加入到武将进程字典  
hero\_dict:add\_equip(NewEquip),

%%% @doc 卸下装备  
demount(Id,\_Pos) ->  
 %% 道具是否存在（是否穿戴在武将身上）  
 Equip = hero\_dict:get\_equip(Id),  
 ?IF(Equip =/= [],ok,?THROW\_ECODE(?LANG\_ITEM\_NO\_ENOUGH)),  
 api\_package:add([Equip],false,?SOURCE\_DEMOUNT\_HERO\_EQUIP),  
 hero\_dict:del\_equip(Id),  
 hero\_send:send\_hero\_equip\_del(Id),

通过相应的位置获取武将  
 HeroInfo = hero\_dict:get\_hero\_by\_pos(Equip#item**.**pos **div** 100),

%%在卸下装备后重新计算武将的属性  
 lib\_hero\_attr:calc\_attribute(HeroInfo)**.**

%%装备强化 Type 1：升1级 2：升10级 10级已经废弃  
strength(EquipId,Type) ->

%%检测装备强化是否开启  
 lib\_player\_module:check\_open(?MOD\_EQUIP\_UP),

%%从武将进程字典获取装备信息  
 EquipInfo = hero\_dict:get\_equip(EquipId),

%% 判断强化类型（强化1级和10级）  
 Result = **case** Type **of** 1 -> strength\_one(EquipId,1);  
 2 -> **catch** strength\_one(EquipId,10)  
 **end**,  
 NewEquipInfo = hero\_dict:get\_equip(EquipId),  
 **case** EquipInfo =/= NewEquipInfo **of** true ->  
 %%发送装备信息更新 hero\_send:send\_hero\_equip\_update(NewEquipInfo),  
 HeroInfo = hero\_dict:get\_hero\_by\_pos(NewEquipInfo#item**.**pos **div** 100),

%%装备强化之后重新计算武将的属性信息  
 lib\_hero\_attr:calc\_attribute(HeroInfo),  
 ?THROW\_SCODE(?LANG\_OK);  
 false ->  
 **case** Result **of** ok -> ?THROW\_SCODE(?LANG\_OK);  
 {\_,Code} -> ?THROW\_ECODE(Code)  
 **end  
 end.**

**装备强化后的级数不能超过主公等级的2倍：**

%% 主公等级PlayerLev  
StrengthLev = EquipInfo#item**.**strength\_lev,  
?IF(StrengthLev >= PlayerLev \* 2,?THROW\_ECODE(?LANG\_PLAYER\_LEVEL\_LIMIT),ok),

**获取能够强化的等级配置是否达到最大等级：**

%% 配置最大等级

%%强化到下一级对应的配置  
Cfg = data\_equip\_strength:get(ItemCfg#r\_item\_data\_cfg**.**quality,StrengthLev + 1),

%%配置是否存在  
?IF(Cfg =:= [], ?THROW\_ECODE(?LANG\_EQUIP\_STRENTH\_LEV\_LIMIT),ok)%%通过配置获取消耗的元宝数量,

[\_,NeedCoin] = get\_equip\_strength\_cost(ItemCfg#r\_item\_data\_cfg**.**quality,StrengthLev),  
**case** lib\_player\_assets:check(coin,NeedCoin) **of** ok -> skip;

%%元宝不足，抛出异常  
 \_ -> ?THROW\_ECODE(?LANG\_COIN\_NO\_ENOUGH)  
**end**,

%%从玩家资产中扣除元宝  
lib\_player\_assets:dec(coin,NeedCoin,?SOURCE\_EQUIP\_STRENGTH),

%%提升强化等级  
NewStrengthLv = StrengthLev + 1,  
NewEquipInfo = EquipInfo#item{strength\_lev = NewStrengthLv},

%%重新设置新的装备信息  
hero\_dict:set\_equip(NewEquipInfo),

%%派发玩家强化的事件  
player\_event:dispatch(?EVENT\_EQUIP\_STRENGTH, NewStrengthLv),  
strength\_one(EquipId,Times-1);

set\_equip(#item{id = Id} = Equip) ->  
 List = get\_equip\_list(),  
 NewList = lists:keyreplace(Id, #item**.**id, List, Equip),

%%设置新的装备列表  
 set\_equip\_list(NewList)**.**

set\_equip\_list(Bag) ->  
 api\_cache:dirty\_data(#ets\_player**.**hero\_data),  
 player\_dict:set(?PD\_HERO\_EQUIP,Bag)**.**

%% @doc 设置脏数据  
%% 玩家进程数据较缓存ETS有改变  
dirty\_data(Pos) ->  
 PlayerId = player\_dict:get\_player\_id(),  
 dirty\_data(PlayerId, Pos)**.**dirty\_data(PlayerId, Pos) ->  
 Vsn = **case** get\_dict\_vsn(PlayerId, Pos) **of** undefined -> 0;  
 TmpVsn -> TmpVsn  
 **end**,  
 set\_dict\_vsn(PlayerId, Pos, Vsn + 1)**.**

**根据变更装备信息的所在位置获取武将信息，获取后 以便对武将的属性进行重新计算：**

HeroInfo = hero\_dict:get\_hero\_by\_pos(NewEquipInfo#item**.**pos **div** 100),

**玩家资产模块：**

**lib\_player\_assets:**

**检测资产是否充足：**

do\_check(KeyPos, Num) **when** Num > 0 ->  
 **if** KeyPos == #r\_player\_assets**.**guild\_contribute ->  
 #ets\_union\_member{contribution = Val} = lib\_union:get\_member\_info(player\_dict:get\_player\_id());  
 true ->  
 Data = get\_data(),  
 Val = erlang:element(KeyPos, Data)  
 **end**,  
 **case** Val >= Num **of** true -> ok;  
 false -> {error, {not\_enough, Val}}  
 **end.**

**客户端和服务端socket消息发送：**

send\_to\_me(Bin) ->  
 send\_to\_sid(player\_dict:get(?PD\_SEND\_PID), Bin)**.**

**将2进制数据发送给指定发送进程id的发送进程**

send\_to\_sid(SendPid, Bin) **when** is\_integer(SendPid) ->  
 lib\_cross\_server:remote\_cast(SendPid, {send\_to\_sid, Bin});

**道具物品格式转换（**#item{}**🡪**{Tid,Num}**）：**

%%转换#item{}为{Tid,Num}  
translate\_item(ItemList) ->  
 [{Tid, Num}**||**#item{tid = Tid, num = Num} <- ItemList]**.**translate\_item2(ItemList) ->  
 Fun = **fun**(Item) ->  
 **case** Item **of** {Tid, Num} -> {Tid, Num};  
 #item{tid = Tid, num = Num} -> {Tid, Num}  
 **end  
 end**,  
 lists:map(Fun,ItemList)**.**

**客户端和服务端消息传输：**

**服务端将数据组码发送给客户端**

{ok,Bin} = ?PT\_FORGE\_ENCODE(?SC\_FORGE\_PREVIEW\_ACK,#sc\_forge\_preview\_ack{type = 2,target\_list = lists:map(Fun,HeroIdList)}),