**任务系统设计文档**

**概况**：本任务系统中，一个任务包含多个步骤，一个步骤有可能包含一个完成步骤需要的条件表达式。玩家接取任务后，根据提示完成任务所需要的条件，系统就会完成该步骤，如果所有步骤完成，就会让玩家领奖，并将任务设置成已经完成。任务之间只有父子任务一种关系，父任务完成后，才可以接取子任务。

**配表部分:**

task配表字段（和后台编码有关的）：

id： 任务的id  
parentList: 父任务的id  
repeate: 任务一天可以完成的次数  
unaccpet:任务是否需要接取  
setpCnt：任务一共有几步  
bonus: 任务完成获取的奖励，值中的每一个数字对应于Prize表的一个Id,用”,”隔开.

<complete></complete> 元素： 当任务的某步需要完成的条件表达式时，就会有此元素。其中：

Id：条件的id

Type： 条件的类型（具体值的含义参见配表目录conf/task下边的readme.txt中的介绍  
operate: 条件判定的操作符， 0大于，1小于，2两数之间，3等于

Params:条件判断的参数

<combine></combine>元素：条件表达式， 在express属性中配

在服务器启动时，会将配表的内容存储到内存中，内存中有以下几个结构体或类和配表相关（都在task\_info.h文件中）

class task\_conf\_mgr\_t : 这个类用来管理task的配表信息。这个类只有一个成员变量，就是一个存放task\_conf\_t的map. 类中的成员函数都是对这个map的操作

struct task\_conf\_t: 此结构存储从配表中读取的信息。

struct step\_conf\_t: 用来存某一步骤完成条件的信息

condition\_conf\_t: 用来存完成某步需要的条件

**在这几层结构中，task\_conf\_t 是 task\_conf\_mgr\_t的map成员变量task\_infos\_中的value的类型. 而struct task\_conf\_t包含一个std::vector<struct step\_conf\_t>类型的成员变量，struct step\_conf\_t 又包含std::vector<condition\_conf\_t>类型的成员变量.这是一个map套vector，然后vector又套vector的结构，使用要格外小心，很容易core掉**

**数据库部分：**

数据库中的字段玩家userid和task\_id共同为key，仅保存玩家已经接取的任务的状态。

status存玩家任务到达了哪一步，每一步由一个bit位表示. done\_times存该任务完成了几次。

玩家在登录时，会将数据库中的任务信息加载到内存中。内存中和任务数据库相关的有如下几个结构体或者类：

Class TaskInfo: 内存中的任务信息管理类。成员变量为task\_infos\_。 类型是std::map<uint32\_t, task\_t>.其他方法均为对此map的操作

task\_t：任务从数据库存入内存中的数据结构

step\_t：任务的步骤在内存中的结构

此部分详情见task\_info.h中的代码和注释

**协议部分：**

AcceptTaskCmdProcessor ::proc\_pkg\_from\_client：接任务。首先解析需要接取的任务的id，根据id读取配表中的任务信息，初始化一个新任务加载到内存中，并将任务写入数据库，然后通知客户端。

**注意**：此处判断了一下是否已经接过任务，已经接过的任务会有一个任务重温，这是adam从精灵派继承来的代码，由于任务形式的不同，本任务系统可能不需要任务重温。

AbandonTaskCmdProcessor::proc\_pkg\_from\_client：放弃任务。首选根据任务id判断该任务是否存在，如果存在，判断是否接取过任务，如果接取过该任务，则分别从内存中和db中删除该任务，并通知客户端。

TaskCompleteCmdProcessor::proc\_pkg\_from\_client：步骤完成（本任务系统步骤完成和任务完成都是这一个协议，每次发此协议时，完成一步，当判断所有步骤都完成时，完成整个任务）。首先判断任务id是否存在和是否已经接了任务。然后从配表和内存中读取任务信息，判断是否可以领奖（判断任务的stepCnt的那一位是否为1，例如任务一共有三步，就判断标志位右起第三位是否为1）。如果可以领奖，就读奖励列表，将任务奖励状态设置为已领取，然后发奖。

然后从客户端读一个step号，不论之前是否可以领奖，都将该step设置为完成，任务状态分别向内存，数据库，客户端同步，再回包给客户端。

**注意**：大部分任务完成判定前端可以监听，但是有些前端无法判定。例如type11（type信息可以看配表目录conf/task/readme.txt）副本完成，这时需要后台监听。本案中，在task\_utils。Cpp中TaskUtils::listen\_duplicate\_complete中完成。副本完成时，调用此函数，此函数读取所有任务信息，判断是否有某个任务需要完成该副本，如果有，则将该任务的当前步骤置为已经完成。

**坑：本任务案代码从精灵派搬来，因策划需求不同，经adam做修改，又后经过colin修改，所以代码结构上，就显得有些杂乱。由于测试量的有限，代码在健壮性上可能存在问题，尤其是map套vector,vector又套vector这种多层嵌套结构，要格外小心。**

**随着任务案的完善，会加更多的步骤完成条件type在任务中，type信息见conf/task/readme.txt， 实现添加在task\_utils.cpp中， 函数名在xml\_configs.cpp中用一个宏生成，例如 #define CONDITION\_FUN(i) g\_condition\_fun[i] = TaskUtils::condition\_fun\_##i;**

**CONDITION\_FUN(11);**

**如果写了CONDITION\_FUN(11);，则要在task\_utils中实现一个函数名为TaskUtils::condition\_fun\_11的函数**

**数据库中的done\_times还没有使用到，因为暂时没有日常任务. 加入每天固定可以完成特定次数的日常任务后，就需要用到此字段来记录玩家当日已完成某任务的次数。**