# role\_data设计综述

* 简介

role\_data模块负责单个agent对应角色的本地信息的维护，所有“单机”玩法的数据都会统一由role\_data管理，一般来说一个功能对应的info在role\_data中，会提供他的set和get方法作为操作接口，set接口可以指定当次set是否为脏数据，最后role\_perisent模块会根据role\_data中的数据信息在agent离线的时候将数据进行落盘

* 数据存储

所有的数据都存储在role\_data 模块的本地变量data\_set中，data\_set不直接对外开放，所有对数据的操作必须调用对应的role\_data接口

同时data\_set作为数据存储的元表，为他指定了\_\_index和\_\_newindex方法

\_\_index的实现确保所有的索引操作必须是在ROLE\_DATA\_LIST定义中的一员，同时确保了惰性访问的功能，在访问却没有初始化的时候，会调用ROLE\_DATA\_LIST中对应项指定的module.init函数来初始化一份当前角色对应的数据，需要注意的就是init函数本身只是数据生成器，不要用他来做data\_set操作，\_\_index函数会根据返回值做set

\_\_newindex 实现只是为了防止对data\_set的错误操作，可以确保不会引入未在ROLE\_DATA\_LIST中定义的字段

* 数据结构描述

data\_set的内容可以描述成

{“keystr” = {val = info, is\_dirty = false}}

这种的描述

其中keystr就是ROLE\_DATA\_LIST中的定义，val为具体的数据存放的地方，而is\_dirty标记的是当前数据是否已经被染色，如果为true则role\_perisent会负责将他落盘并将标记清除

以$module.init函数的返回值为例，一般来说init函数只需要从数据库get出来数据，然后将数据返回即可，但是如果初始化的时候对数据进行了改变，且这个改变不能重入，则必须再加一个返回值true表明这个值已经被染色

* 数据操作

一般我们会对一个数据段包装一个get和set函数，需要注意的是lua在传递table的时候，传递的是table本身，也就是如果传递出去的值被修改了，会立刻体现在data\_set中，因此基础的 get\_info函数本身会对数据进行一次copy才将数据返回

而对于set操作set函数暴露的接口为set(info, is\_dirty), is\_dirty是可选字段，这里如果没填的缺省值是true，而不是false，也就是一般逻辑调用的时候只需要直接调用set(info)即可，一般来说，只有在初始化的时候才会用到is\_dirty=false的情况

* 扩展

绝大多数情况下，简单的get和set操作就可以满足平时的需求，如果info的结构比较复杂，但是又不可拆分，允许在role\_data中加入专门的get,set函数。例如ger\_info中经常操作的是ger\_list字段，那就可以包装一个role\_data.set\_ger\_list和role\_data.get\_gerlist，注意实现的使用还是用role\_data.get\_gerinfo 和role\_data.set\_gerinfo来实现

ROLE\_DATA\_LIST的实现，ROLE\_DATA\_LIST的最终结果并不是我们定义的内容，最终数据会变成

{{key =”keystr”, module = module},

….

}

这样的列表方便使用，同时会遍历key，对key做是否多次使用的检查，因为Lua没有宏定义，只能做成这样，同时还会确认每个key都对应有data\_base指定，防止忘写对应集合名而无法落盘，如果确实不需要落盘，可以指定数据库到”test”,然后将set操作改写成永远只会标记is\_dirty是false，init函数不要填充Is\_dirty字段或者显式的声明为false，确保不会调用落盘。