当有必要为object（entity/missile/dropitem）增加一个变量时，针对不同的情况，我们提供了很多实现方法。游戏专用的变量必须由游戏自己定义，不能写在引擎中。即便是引擎通用变量，也应对优先考虑带自动同步、自动维护的更简便方法。详情参看本文介绍。

### 可用方法列表

（按考虑顺序排序，先考虑上面的方法，不适用再用下面的）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 方式 | 可用范围 | 游戏定义 | 可变 | 任意数据 | C/S同步 | 自动维护 | 自动存盘 |
| 配置（config） | Lua & C++ | √ | × | √ | √ | √ | √ |
| 属性（prop） | Lua & C++ | √ | √ | × | √ | √ | √ |
| Action对象值 | Lua | √ | √ | √ | × | × | √ |
| value封装 | Lua & C++ | × | √ | √ | √ | × | × |
| lua对象值 | Lua | × | √ | √ | × | × | × |
| c++对象值 | C++ | × | √ | √ | × | × | × |

列解释：

任意类型： 支持大多数数据类型，支持各种数据操作（计算、修改等）

C/S同步： Client、Server之间会保持同步数值，包括第三者客户端

自动维护： 无需关心变量具体的设置、改变逻辑

自动存盘： 如需玩家下线保存，无需手动处理具体逻辑

### 各个方法的详解

【配置（config）】

概述：针对一类object的配置，由json配置

（其实item、block也有相似的配置）

缺点：运行时不可变更

定义：

一般： entity/missile的json配置文件中

c++： EntityConfig.h、MissileConfig.h文件中

使用：object:cfg().xxx

Actions.GetObjectVar

【属性（prop）】

概述：由配置和buff共同维护的一套动态属性值

缺点：数据类型受限、仅entity支持

定义：

游戏专用： game/sample/setting.json的customProp字段

引擎lua： lua/common/entity.lua的EntityProp（C/S还有扩充定义）

引擎c++： EntityProp.h中

使用：object:prop(“xxx”)

Actions.GetPropVar

【Action对象值】

概述：专门为特定游戏的Action提供的object对象值

缺点：不会同步客户端，引擎不会依赖此类值

定义：entity配置的saveVars（不存盘无需定义）

使用：Actions.GetObjectVar

【value封装】

概述：基于lua/c++对象值做了一层封装，方便同步等

缺点：不提供游戏专用，仅entity可用，更多维护需自行实现

定义：lua/common/entity.lua的ValueDef、ValueFunc

使用：entity:getValue(“xxx”)

【lua对象值】

概述：针对object对象的一切lua数据

缺点：自行管理、自行定义

使用：object:data(“groupKey”).xxx

【c++对象值】

概述：既c++的object对象的一切成员变量

缺点：自行管理、自行定义、自行使用

### buff机制的扩展

暂时我们希望prop的初值由配置（config）唯一决定，而prop的动态改变仅由buff提供。这样一来，buff的自动维护、自动同步等机制就可以使得prop天然拥有了自动维护和同步等功能。未来也许会有其它途径修改prop值，但是不应破坏这一特性。

【buff如何改变prop】

当entity获得一个buff时，会将此buff所携带的prop“加”到entity自身prop上。

当entity失去一个buff时，会将此buff所携带的prop从entity自身prop中“减”掉。

对于一般数值型prop，“加”、“减”意味着正常的数学加减法。对于table、string、function型的prop，又有其它的行为定义，详情可参考代码Entity:calcBuff。

【如何获得buff】

目前常用的加buff方式：

道具装备buff、道具手持buff、区域buff、踩方块buff、技能buff、……

所有这些方式都可以在填写配置后自动加删buff，而buff又会自动加减prop，所以可以不再关心麻烦的同步、维护等问题。