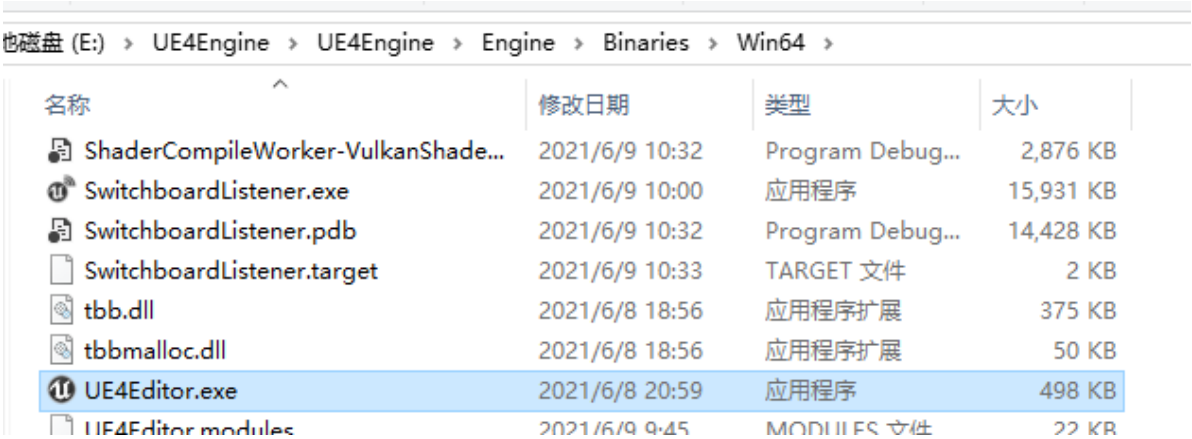
创建项目

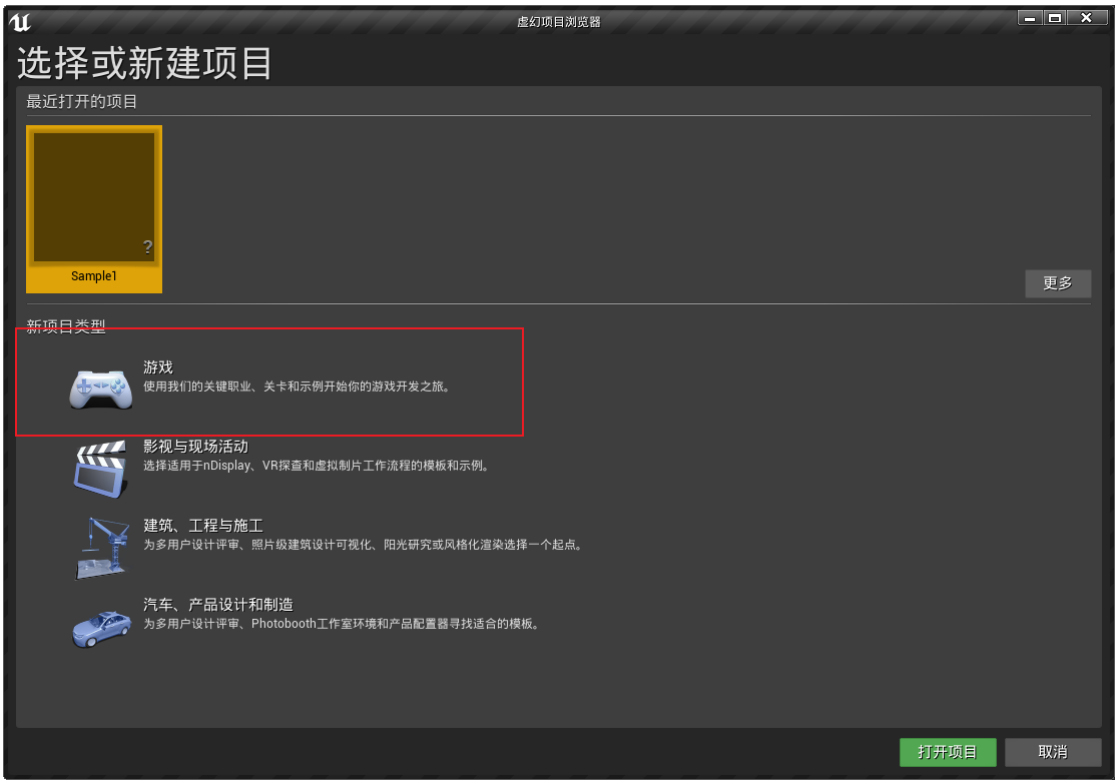
打开UE4

UE4引擎入口在下图所示位置中，为了方便，可以创建快捷链接到根目录下

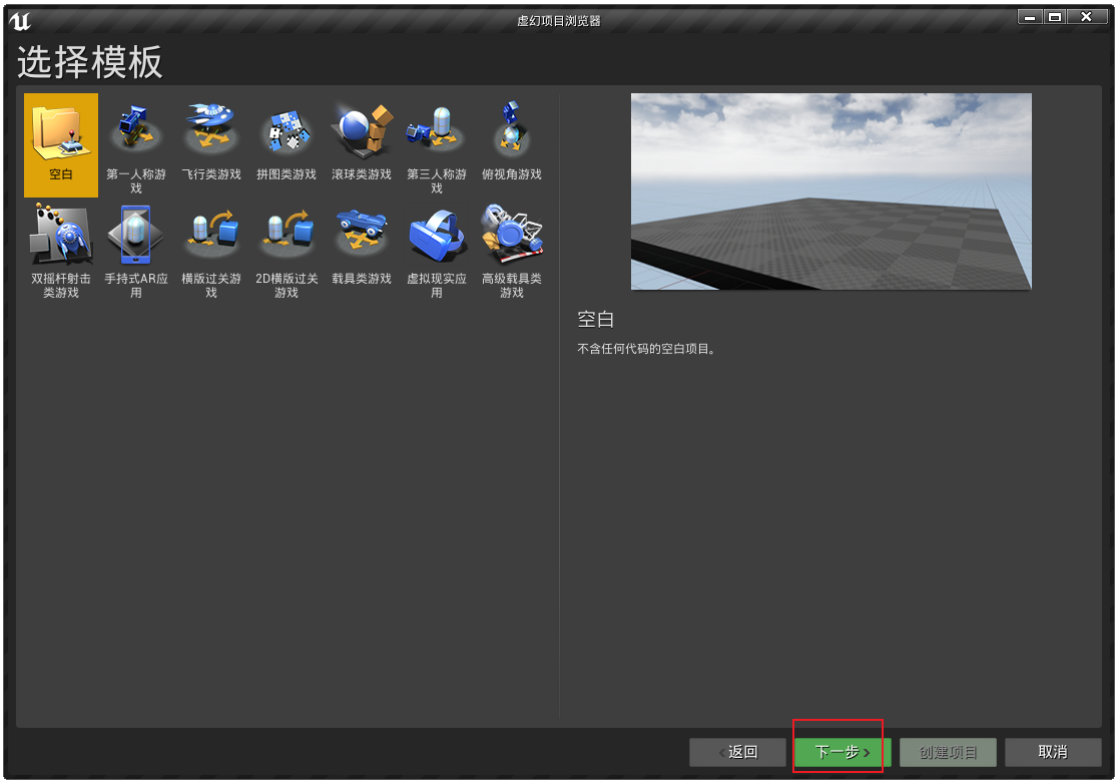


创建空白项目

打开UE4后，会出现一个选择项目的界面，上部分为之前创建过的项目列表，下部分为创建新的项目

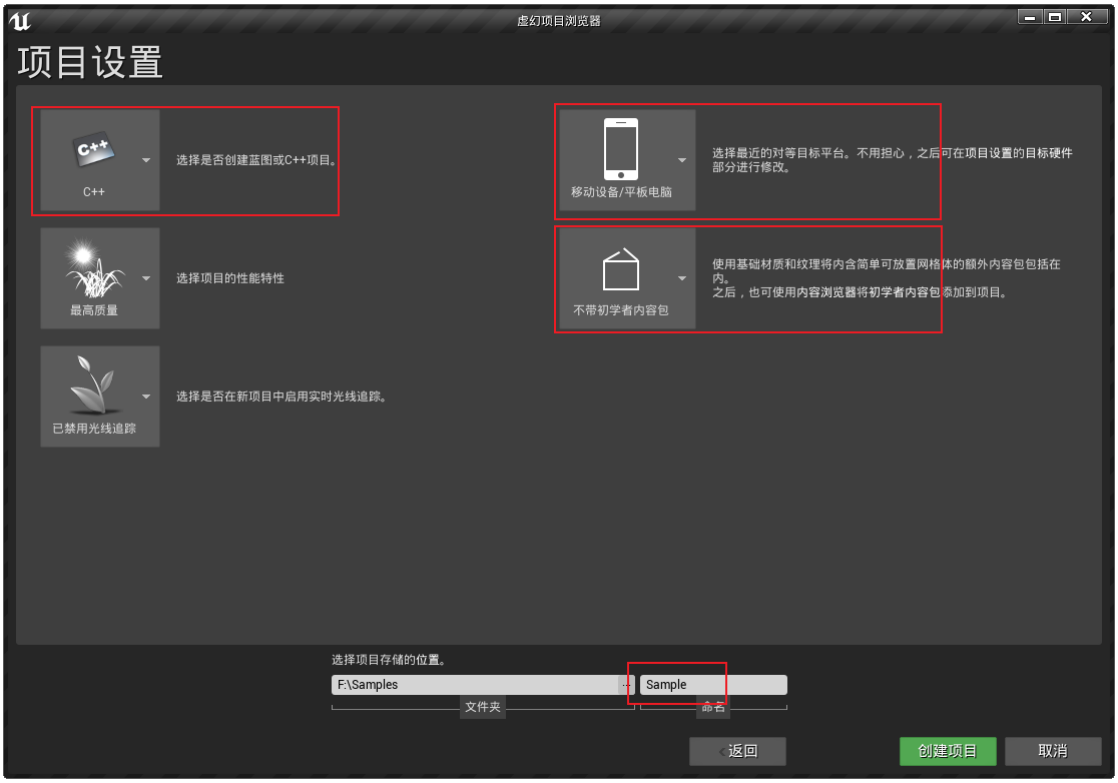


1. 我们此次目标为创建一个名为 Sample 的项目，则选中**游戏**，点**下一步**
2. 选择**空白**，点**下一步**



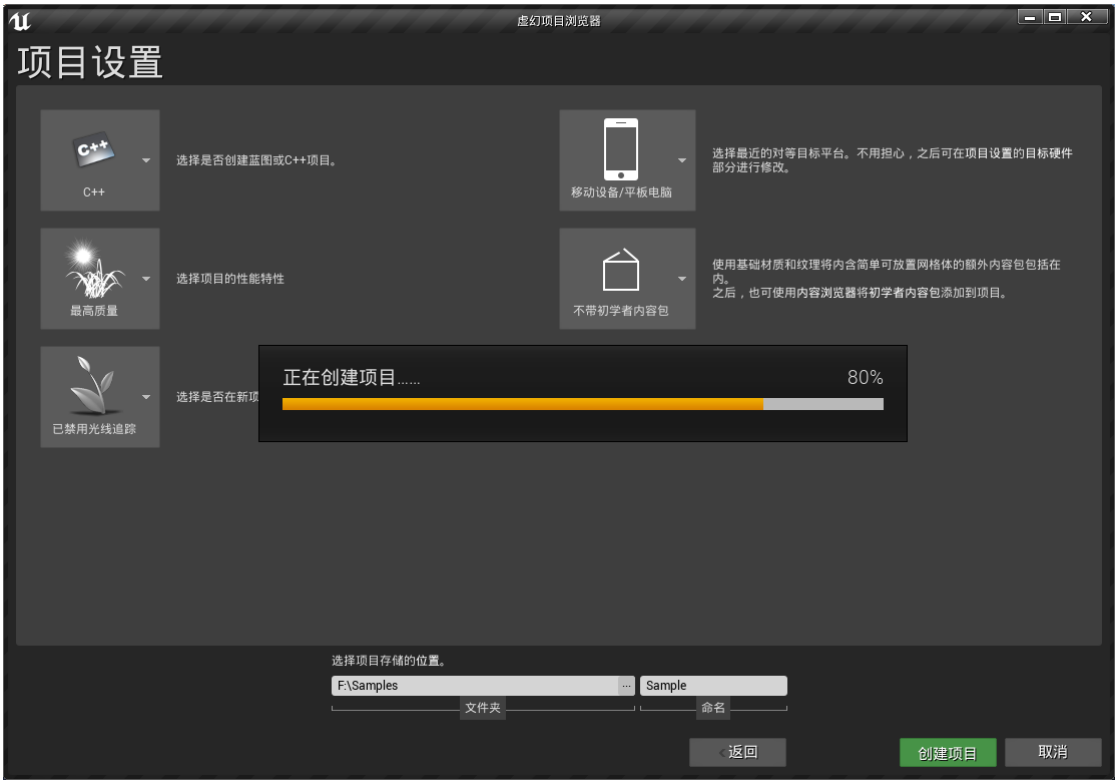
3. 项目设置中，我们修改项目路径，且更换命名为 **Azure**（本文档中使用Sample为名方便演示），设置为

- **C++工程**
- **移动设备/平板电脑**
- 最高质量
- **不带初学者内容包**
- 已禁用光线追踪



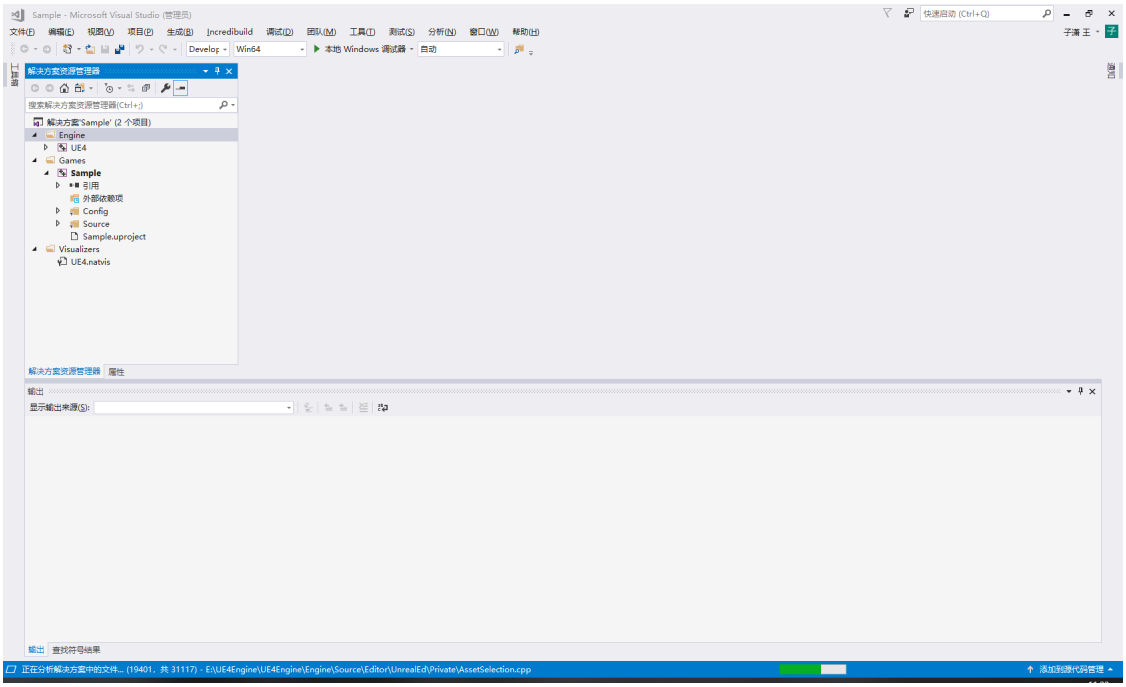
> **将项目名称修改为Azure，因为公司很多配套工具依赖这个名称**

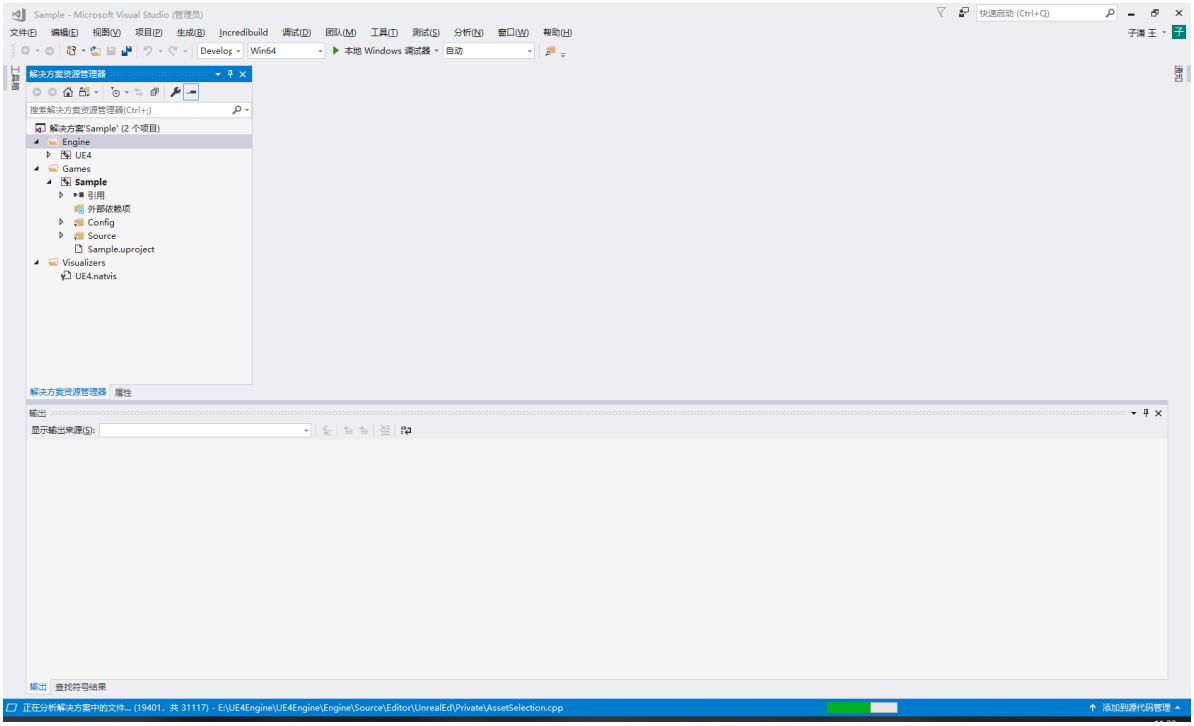
4. 点**下一步**等待创建完项目



5. 创建完项目后，会默认打开 Visual Studio ，展示游戏工程；等待VIsual Studio载入完项目后，点击**本地 Windows 调试器**开启项目

第一次启动游戏会在 45% 处停留一段时间，请耐心等待



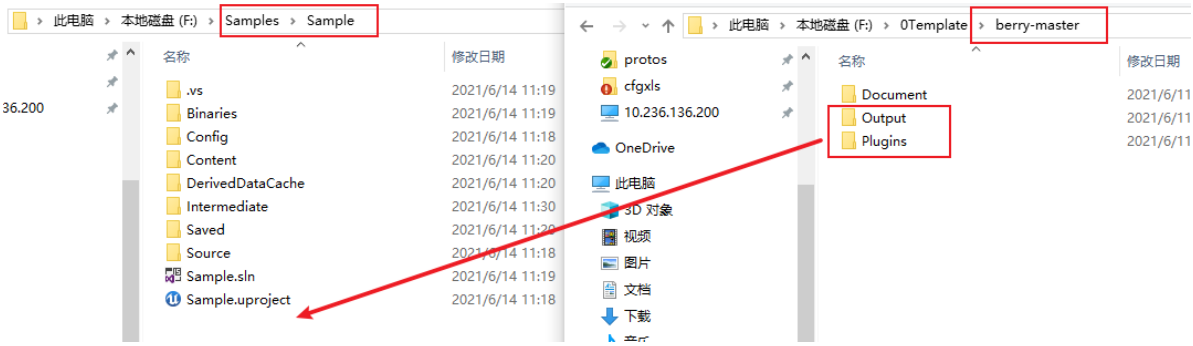


6. 在我们做游戏之前，先得进行下一步骤，导入内部的控件

Output 和 Plugins 导入

拷贝内部代码

找到我们的 berry 库，将 如下两个文件夹 **Output** 和 **Plugins** 拷贝到刚刚创建的 **Sample** 项目中

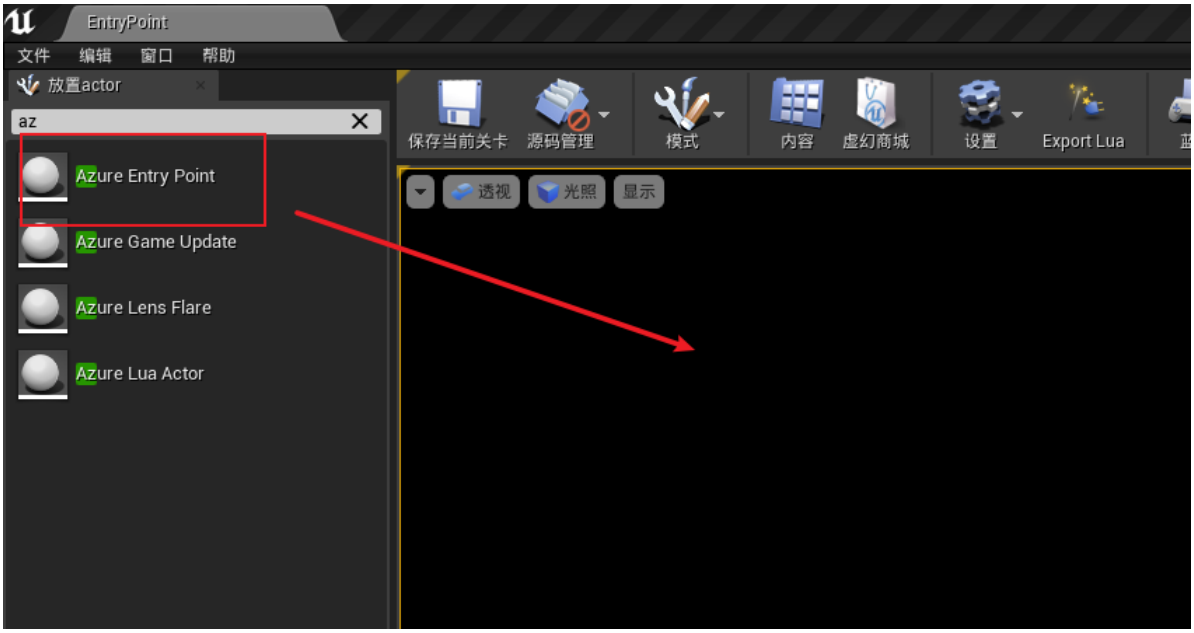


下面是完成后的 **Samepleb**工程目录

场景中 EntryPoint 导入

1. 回到我们的 **Sample** 工程中，点击**本地Windows调试器**开启游戏
2. 打开 内容浏览器，右键新建 Level关卡，命名为 EntryPoint

3. 创建后，在 项目设置中，设置入口为刚刚创建的 EntryPoint
4. 进入EntryPoint，左侧内容搜索 Azure，并将 Azure Entry Point拖拽到场景中，此时已完成基础场景创建，我们点击开启游戏后，会执行到 Lua 的 Main.lua 脚本



开始游戏

LPlus

简介

- LPlus 将Lua开发规范化，加入了类似C++等静态代码关于类的特性，像类(Class)的创建、继承
- LPlus 也添加了针对调用方法参数的类型检查，方法返回值的类型检查
  LPlus 中存在类型检查的开关，帮助开发者调试，提高了开发效率，打包游戏时可关掉类型检查，恢复运行效率

环境及插件安装

> 为方便使用Lua进行开发，首选安装 IntelliJ + EmmyLua + RedefineLua

- IntelliJ安装包：\\10.236.136.200\ShareRead\IdeaIntelliJ\ideaIC-2018.3.3.exe
- 安装完成后，前往 `File/Settings/Plugins`，点击顶部右侧菜单，选择 `Install Plugin from Disk`，分别安装以下插件：
- \\10.236.136.200\ShareRead\IdeaIntelliJ\IDEAPlugins\IntelliJ-EmmyLua-1.2.6-IDEA182.zip
- \\10.236.136.200\ShareRead\IdeaIntelliJ\IDEAPlugins\RedefineLua_1.9.4-sigma.jar

使用

我们在以下举例，展示如何创建、继承类，和变量以及方法等

类的定义

代码为创建一个名为 `MapObject` 的类，类名与文件名相同，类文件的位置在 xxPath下

> Files: xxPath/MapObject.lua

```
local Lplus = require("Lplus")

local MapObject = Lplus.Class("MapObject")     -- 通过 Lplus 的Class方法创建一个类

do
    MapObject.Commit()
end

return MapObject
```

第一行的 require 方法会根据路径将目标类引入，在文件中，如果需要其他类，需要使用require的方式将其引入

> 详情见 Lplus.lua 中的 Lplus.Class 方法

类的继承

我们创建一个继承于 `MapObject` 的类：`MapHero`

> Files: xxPath/MapHero.lua

```
local Lplus        = require("Lplus")
local MapObject    = require("somepath.MapObject")

local MapHero = Lplus.extend(MapObject, "MapHero")

do
    MapHero.Commit()
end

return MapHero
```

Define字段

> 在定义类及其变量与方法前，我们需要使用 Lplus.lua 中定义的 define 字段

```
local def = MapObject.define
```

使用如上代码获取创建类的 define 字段，define字段中包含了我们对这个类的操作，包含变量(field)、方法等的声明

```
local define = {
        field       = InstanceField,
        shared      = StaticField,
        const       = ConstField,
        final       = FinalMethod,
        static      = StaticMethod,
        method      = InstanceMethod,
        virtual     = VirtualMethod,
        override    = OverrideMethod,
        constructor = ConstructorMethod,
    }
```

具体代码请阅读 `Lplus.lua` 中对 `define` 的定义

变量

在这里我们介绍类变量的声明，比如我们声明当前 `MapObject` 类记录着一个名字和位置，首先确保我们拥有这个类的 define变量

> Files: xxPath/MapObject.lua

```
local Lplus = require("Lplus")

local MapObject = Lplus.Class("MapObject")     -- 通过 Lplus 的Class方法创建一个类

do
    local def = MapObject.define

    def.field("string").name = ""    -- 名字

    def.field("number").x = 0        -- 位置 x

    def.field("number").y = 0        -- 位置 y

    MapObject.Commit()
end

return MapObject
```

field支持的变量类型

类型	解释	是否支持初始化为nil
boolean	布尔	不支持
number	数字	不支持
string	字符串	不支持
table	table	支持
userdata	userdata	支持
function	方法	支持
thread	线程	支持
dynamic	动态	支持

方法

方法的定义

> Files: xxPath/MapObject.lua

```
local Lplus = require("Lplus")

local MapObject = Lplus.Class("MapObject")     -- 通过 Lplus 的Class方法创建一个类

do
    local def = MapObject.define

    def.field("string").name = ""    -- 名字

    -- 获取 MapObject 的名称
    def.method("=>", "string").GetObjectName = function(self)
        return self.name
    end

    -- 设定 MapObject 的名称
    def.method("string").SetObjectName = function(self, name)
        self.name = name
    end

    MapObject.Commit()
end

return MapObject
```

以上代码中定义了两个方法：

- `GetObjectName` 用来获取 MapObject 对象中的name变量
- `SetObjectName` 用来设定 MapObject 对象中的name变量

定义方法前，我们通过 `MapObject.define` 获取类的 `define` 字段
`def.method` 用来表示定义一个方法，后面跟的是方法的 `参数` 和 `返回值` 的类型，其中 `"=>"` 用来分隔，前者是参数类型，后者是返回值类型；
Lplus中开启了类型检查，所以返回值数量与类型必须对应

`function`后参数为参数变量名，第一个参数必定是 `self`，用来指向当前变量，方法中需要使用 `self.name` 指向当前对象的`name`变量

类的构造方法

> Files: xxPath/MapObject.lua

```
local Lplus = require("Lplus")

local MapObject = Lplus.Class("MapObject")    -- 通过 Lplus 的Class方法创建一个类

do
    local def = MapObject.define

    def.field("number").x = 0       -- 位置 x

    def.field("number").y = 0       -- 位置 y

    -- 构造方法
    def.constructor().MapObject = function(self)
        self.x = 1000
        self.y = 1000
    end

    MapObject.Commit()
end

return MapObject
```

使用 `define` 的 `constructor` 声明构造方法，方法名为类的名称

类方法重写

> 方法的重写存在于继承的子类中

首先我们假设在 MapObject 中存在方法 `SayHello` 和 `Jump`

```
    ...

    def.virtual().SayHello = function(self)
        Output("阿巴阿巴阿巴")
    end

    def.virtual("number").Jump = function(self, height)
        self.y = height
    end

    ...
```

则在我们的子类 `MapHero` 中使用 `override` 来重写 `SayHello` 方法

> Files: xxPath/MapHero.lua

```
    ...

    -- 重写方法，不调用父类的方法
    def.override().SayHello = function(self)
```

```
        Output("Hello, I'm your friendly neighbor %s", self.name)
    end

    -- 重写方法，调用父类的方法
    def.override("number").Jump = function(self, height)
        MapObject.Jump(self, height)    -- 使用"."来调用父类方法，第一个参数为当前的
self
        Output("吐丝")
    end

    ...
```

重写时，如果想要调用父类方法，则使用 父类名.方法名(self, [参数])

命名规范

1. 所有lua文件通常以驼峰式命名，必要时可增加下划线和数字，不出现中文。如

- Good: Mail.lua UIMail.lua
- Bad: 主界面.lua

2. Lua5.3本身没有package的机制，但可以使用table做到package的组织。推荐绝大部分代码都要用package的概念组织，不污染全局域。
   package使用短且有意义的缩写来命令。比如：

```
cfg     ---- 配置数据的package
UI      ---- UI模块
Net     ---- 网络模块
```

3. 类型名称使用驼峰式命名，如果某文件中只有某个类型，则该文件名要与类型名一致。比如 `UILogin`、`UIServerCell`
4. 类型成员带有公有性质的方法命名, 采用驼峰命名规则。比如：

```
Config:LoadFile()
```

5. **局部变量**和类型成员带有**私有性质**的方法及字段的命名，采用小写开头或加下划线都可以，推荐是的变量名需要尽量反映出变量的含义。
   变量名并不要很长很长，反映出含义即可。比如循环变量，程序员们都用i, j, k，这个就短且含义清晰。但有业务逻辑的代码，就不要i, j,k满天飞了
   局部变量的作用域越短，含义越清晰，所以尽量写精简的函数吧~
   在一般逻辑代码中，尽量用常见单词给局部变量命名，不是大家喜闻乐见的缩写尽量少用
6. 常量（也许lua没有真正的常量，但符合常量概念的都算）的命名：全部以大写字母命名，以_作单词的分隔。如：`OS_IOS`
7. require后局部化的命名，需与require的文件名保持一致(通常情况下，文件名-类型名-局部类型名三者一致，方便后期维护)。如

```
local SceneInst = require("Core.Scene.SceneInst")
```

8. ！！！非常重要的一条：变量与方法的引用（调用）尽量local化，这不仅事关代码的可阅读性，而且还关系到性能!
   在一个代码域中，如果引用别的package的方法或变量，或者引用一个lua对象(表), 特别是多次引用时，应该使用一个local变量引用到方法或变量，然后再使用。如：

- 这是不好的做法：

```
print(gmodel.Login.userid)      -- 打印userId
print(gmodel.Login.userName)    -- 打印userName
```

- 推荐这样：

```
local loginModel = gmodel.Login
local userName = loginModel.userName
local userId = loginiModel.userId
print(userId)                   -- 如果userId只用一次， 则print(loginModel.userId)也是可以授受的
print(userName)
```

简单说下对性能的影响，因为`lua`不是像`c++`这样可以深度编译优化的，即使是`math.abs`这样的调用也涉及到math的查表操作，如果我们abs调用的多，先`local abs = math.abs`,然后再调用，肯定会省掉查表操作

模块

模块简介

模块的基础代码在`Hub.lua`中，架构中使用模块来规范化程序的开发
模块的设计思想为MVC，将数据层(M)逻辑放到了`Model`中，将控制层(C)逻辑放到了`Module`中，View层(V)的逻辑表达为游戏中的UI或Actor

一般我们接收到协议内容后，会将数据缓存到`Model`里，方便管理

创建模块

下面举例创建Chat(聊天)模块

创建模块

1. 在`Lua`根目录`Modules`下创建文件夹`Chat`，文件夹里创建`Chat.lua`文件

> File: Modules/Chat/Chat.lua

```
local Lplus = require("Lplus")

-- Chat 需要继承 Module
local Chat = Lplus.Extend(Module, "Chat")
do
    Chat.Commit()
end

return Chat
```

2. 在`Lua`根目录`Models`下创建文件`ChatModel.lua`

> File: Models/chatModel.lua

```lua
local Lplus = require("Lplus")

-- ChatModel 需要继承 Model
local ChatModel = Lplus.Extend(Model, "ChatModel")
do
    ChatModel.Commit()
end

return ChatModel
```

注册模块

在 `main.lua` 文件中

1. 使用 `gmodel("Chat")` 注册聊天模块的数据层，注册时调用 `Hub.lua` 的 `CreateModel` 方法创建模块
2. 使用 `gmodule("Chat", gmodel.Chat)` 注册聊天模块的控制层，注册时调用 `Hub.lua` 的 `CreateModule` 方法创建模块

可以看到在 Hub.lua 中，`CreateModel` 的获取类路径为：

```lua
require("Models." .. moduleName .. "Model")
```

所以模块放在 Models 目录下即可，但文件名需要以 `Model.lua` 作为后缀

`CreateModule` 的获取类路径为:

```lua
require("Modules." .. moduleName .. "." .. moduleName)()
```

Model与Module的使用

程序中可以全局使用 gmodel 和 gmodule 访问 Model 和 Module，一下例子为在 Chat(Module) 中访问 Model 存储的内容
首先我们声明 `ChatModel`，定义一个 `playerList`，用来存储所有玩家的名称

> File: Models/ChatModel.lua

```lua
local Lplus = require("Lplus")

local ChatModel = Lplus.Extend(Model, "ChatModel")

do
    local def = ChatModel.define

    def.field("table").playerList = nil

    def.constructor().ChatModel = function(self)
        self.playerList = {
            "Player_A",
            "Player_B",
            "Player_C"
        }
    end

    ChatModel.Commit()
```

```
end

return ChatModel
```

然后我们声明 逻辑层 `Chat.lua` ，定义一个方法 `BoradcastToPlayers` 对所有已存储玩家“发送消息”

> File: Modules/Chat/Chat.lua

```
local Lplus = require("Lplus")
local ChatModel = gmodel.Chat                          -- 此处通过 gmodel.Chat 获取 ChatModel 对象

local Chat = Lplus.Extend(Module, "Chat")

do
    local def = Chat.define

    -- 假定方法，向所有玩家对话
    def.method("string").BroadcastToPlayers = function(self, word)
        local playerList = ChatModel.playerList      -- 此处通过获取的 ChatModel拿到对应的 playerList
        for i = 1, #playerList do
            Output("Hello, " .. playerList[i])
        end
    end

    Chat.Commit()
end

return Chat
```

Subject 与 Services

一般是用于项目模块之间防止耦合互相调用，Subjects理解为事件系统，Services可以将模块中的方法暴露出来，供外部使用

Subject

subject 类似于广播，发送端只管发送，不在乎谁接收，一般由别的模块发出，或者收到协议后发出，通常的用法如：

```
MapUIModule:Notify(subjects.MapUI.MonsterSearchSuccess)
```

接收端可以有多个，写一个监听函数就可以如下：

```
self:Listen(subjects.MapUI.MonsterSearchSuccess,self.MonsterSearchSuccess)
```

Subject案例

1. 先我们需要创建文件 ChatSubjects.lua

> File: Modules/Chat/ChatSubjects.lua

```
-- 某个玩家离开了频道
--  playerId: string    玩家ID
subjects:new("SomePlayerLeave")  -- 声明 subject类型： NotifyToSomePlayer
```

没错，该文件只需要这一行即可，当然如果加上注释的话那就更好了，注释一般包括方法名，参数名和类型

2. 然后我们在模块 Chat 中引入subjects

> File: Modules/Chat/Chat.lua

```
...
    def.override().Init = function(self)
        -- 我们在这个初始化方法中引入subjects
        require("Modules.Chat.ChatSubjects")
    end
...
```

2. 其次我们假设在某个 UI面板中，订阅了这个Chat的"玩家离开"事件（UI相关的开发见下文）

> File: Modules/Chat/UI/ChatPlayerListPanel.lua

```
...
    -- 假设在 ChatPlayerListPanel 中，Init为初始化方法
    def.override().Init = function(self)
        -- 我们定义一个方法订阅该事件，SomePlayerLeave 来自 ChatSubjects 的定义
        self:Listen(subjects.Chat.SomePlayerLeave, self.OnSomePlayerLeave)
    end

    -- 定义订阅的方法，参数需与 Notify 的参数相同
    def.method("string").OnSomePlayerLeave = function(self, playerId)
        Output( playerId .. "离开了聊天频道" )
    end
...
```

3. 然后我们再假设在另外一个UI面板中，存在一个按钮，点击后，会通知所有**订阅**了该事件的方法

> File: modules/Chat/UI/PlayerLeavePanel.lua

```
...
    -- 这里，我们假定这是个某个按钮的回调方法，点击按钮后触发
    def.method().OnLeaveBtnClick = function(self)
        -- 在此处使用 gmodule
        gmodule.Chat:Notify(subjects.Chat.SomePlayerLeave,
"playerId_some_player")
    end
...
```

Service

当你想调用别的模块的功能，就在别的模块通过注册服务的功能实则是调用父类Module 的Serve方法如：

```
self:Serve(services.MapUI.ShowContextPanel, self.OnServiceShowPanel)
```

并在对应的模块Proto.lua文件中new 一个服务名，与Module中的保持一致

Service案例

我们此例将上面代码 Chat.lua 中的 BroadcastToPlayers 方法暴露出来

1. 首先创建文件 `ChatServices.lua`

> Files: Modules/Chat/ChatServices.lua

```
-- 设置视野目标
-- param1: words: string
services:new("BroadcastToPlayers")
```

没戳，此文件和上面的 Subjects 一致，都只有一行即可

2. 然后同 Subjects的文件一样，在 Init 方法里注册和引入 Services

> Files: Modules/Chat/Chat.lua

```
...
    def.override().Init = function(self)
        -- 我们在这个初始化方法中引入 services
        require("Modules.Chat.ChatServices")

        -- 在引入后，“绑定”类中的方法 BroadcastToPlayers 与 services 在一起
        self:Serve(services.Chat.BroadcastToPlayers, self.BroadcastToPlayers)
    end
...
```

3. 后在我们想使用的地方通过 `services.Chat` 访问即可

> Files: modules/Chat/UI/PlayerLeavePanel.lua

```
...
    -- 这里，我们假定这是个某个按钮的回调方法，点击按钮后触发
    def.method().OnLeaveBtnClick = function(self)
        -- 此处会调用至 Chat.lua 的 BroadcastToPlayers
        services.Chat.BroadcastToPlayers("Hello Players!")
    end
...
```

Core

lua 核心模块，是其他模块的基础。在游戏初始化的时候，在Core.lua中Core内其他核心模块进行了实例化及部分加入到全局表中。

ResPath

读取Respath.csv 的方法，用于ue4 打包cook 资源的目录，资源路径统一在Respath.csv中索引，资源路径为相对于Ue - Content 目录，

UI

UI 模块，对于ue4Widget层的封装

UI.lua

UI.lua 是管理lua UI 对象的管理类

```
--常见用法： luaUIPath 为相对于lua根目录的相对路径
UI.Show("luaUIPath",...)
UI.Find("LuaUIPath")
UI.Close("luaUIPath")
```

UIBase.lua

继承于Observer.lua,通过注册Msgtable封装了ue4 Widget 的界面控件事件回调,常见的比如，onClick，onPress，onToggle，onTextChange，onScroll等等。

常用UMG控件	常用的支持的事件	适合的使用场景
AzureButton	onClick,onPress,onLongPress,onHovered	
AzureScrollList	onScroll,onScrollEnd	childWidget高度一致，Item较多
AzureFixedList		Item较少且高度一致，不支持滑动
AzureFixedListWithCache		比上一种更省资源
AzureEditableTextBox	onTextChange	输入框
AzureInfiniteList	需要拿到widget对象 手动写SetOnUpdateFunc回调	对于childWidget 高度多变，灵活性高，可操作性高，如排行榜，聊天界面
AzureRadioBox	onToggle	单选框，复选框
AzureCompactScrollList		子列表项嵌套子列表项（常见搭配：AzureCompactScroll 配 AzureFixedList）

常用成员包括 luaPath,及luaUI类对象对应的ue4 UserWidget 对象;

常用成员方法：GetWidget,CreateWidgetWrapper,CreateView，OnUIEvent,FindChild,SetActive,及UI基本的生命周期函数，具体用法见示例demo;

```
self.HtmlTxt_Left = self:FindChild("HTMLTxt_Content_Left")

---@param string UMG 控件名称
---@param string 事件方法名
---@param 回调函数（传参与事件方法类型关联）
self:OnUIEvent("Pnl_Chat_All","onDrag",self.UpdateContent)
self:OnUIEvent("Btn_New","onClick",self.ClickNotRead)
self:OnUIEvent("ETxt_Write","onTextChange",self.EditTextChange)
```

UIPanel.lua

继承于UIBase.lua

比UIBase 更为细化的对于Widget界面的生命周期函数：

Panel常用生命周期函数	
OnInit	Widget创建之前，通常传入一些panel配置如：LuaPath，加载资源模式，UI显示层级
OnCreate	Widget创建之后已经加载到内存中
onShow	Widget已经addToViewPort 之后
OnClose	Widget销毁之前

UIView.lua

继承于UIBase.lua

通常用于UMG蓝图嵌套UMG蓝图,可以使代码分离，更加美观。

```
---@param string 嵌套蓝图对象的名称
---@param string 嵌套蓝图对应lua类的相对路径
self:CreateView("MiniChat",         "Modules.Chat.UI.ChatInterfaceEntryPanel")
```

UITool.lua

工具类，用于界面设置Image 图片。

```
 ---@param uiObj ue.UWidget UIImage/UIAzureImage等Widget
 ---@param textureId any 资源Id，可为string或number(取决于在Respath.csv中的写法)
UITool.SetTexture(uiObj，textureId)
```

UIWidget

文件夹目录下，包括对常见UMG List控件用法的封装。

目前每种类型List 用法都类似：

如：UIFixList,UICompactScrollList,UIScrollList等

```
-- 示例用法：
---@type UIFixedList
def.field(UIFixedList).ChannelStateSign = nil
---@param string 控件名称
---@param Cell 的lua文件相对路径
---@Type UIFixedList
self.ChannelStateSign =
self:CreateListWrapper(UIFixedList,"FL_Channel","Modules.Chat.UI.Cell.ChatEntryC
ell")
```

```
--region List设置数据方法
local dataTable = {
    [1] = 1,
    [2] = 2,
    [3] = 3,
}
--region SetData 方法把数组数据打包处理，将数组Item传递到UICell处理
self.ChannelStateSign:SetData(dataTable)

local cellNum = self.ChannelStateSign:GetCellCount()
for i = 1, cellNum do
    self.ChannelStateSign:GetCellAt(i):UpdateTag(channelId)
end
```

UICell接收数据示例：

```
local ChatEntryCell    = Lplus.Extend(UICell, "ChatEntryCell")

do
    local def = ChatEntryCell.define

    ---@type userdata
    def.field("userdata").WS_Channel = nil

    ---@type string
    def.field("string").channelId = ""

    ---@param self ChatEntryCell
    ---@return void
    def.override().OnCreate = function(self)
        self.WS_Channel = self:FindChild("WS_Channel")
    end

    ---@param self ChatEntryCell
    ---@param channelId string
    ---@return void
    def.method("string").UpdateTag = function(self,channelId)
        if channelId == self.channelId then
            self.WS_Channel:SetActiveWidgetIndex(0)
        else
            self.WS_Channel:SetActiveWidgetIndex(1)
        end
    end

    --region 通过覆写RefreshCell方法，self:GetData()拿到数组Item数据
    ---@param self ChatEntryCell
```

```
    ---@return void
    def.override().RefreshCell = function(self)
        local cellData = self:GetData()
        if not cellData then return end
        self.channelId = cellData
    end
```

UI相关完整demo:

OrdinaryRewardPanel.lua

```
---OrdinaryRewardPanel.lua
--- 通用普通奖励弹窗
---
---@param data table <number,{ItemCid:number,Count:number}>
--- 用法： UI.Show("Core.UI.UICommonPanel.Reward.OrdinaryRewardPanel",data)
-------------------------------------------------------------------------------
-------------

local Lplus = require "Lplus"
local UIPanel = require "Core.UI.UIPanel"

---@class OrdinaryRewardPanel:UIPanel
---@field private _TittleTxt userdata
---@field private _Btn_Txt userdata
---@field public FL_Reward UIFixedList
---@field public CallBackFunc function
---@field public Commit fun():OrdinaryRewardPanel @notnull
---@field public OnInit fun(self:OrdinaryRewardPanel)
---@field public OnCreate fun(self:OrdinaryRewardPanel)
---@field public FillingText fun(self:OrdinaryRewardPanel)
---@field public OnShow fun(self:OrdinaryRewardPanel, data:any,
callBackFunc:function)
---@field public LoadCell fun(self:OrdinaryRewardPanel, dataTable:table)
---@field public ClickReceiveBtn fun(self:OrdinaryRewardPanel)
---@field public ClickClosePanel fun(self:OrdinaryRewardPanel)
local OrdinaryRewardPanel = Lplus.Extend(UIPanel,"OrdinaryRewardPanel")

do
    local def = OrdinaryRewardPanel.define

    ---@type userdata
    def.field("userdata")._TittleTxt = nil    -- 标题奖励

    ---@type userdata
    def.field("userdata")._Btn_Txt = nil      -- 领取按钮文字

    ---@type UIFixedList
    def.field(UIFixedList).FL_Reward = nil

    ---@type function
    def.field("function").CallBackFunc = nil

    ---@param self OrdinaryRewardPanel
    ---@return void
```

```
    def.override().OnInit = function(self)
        self:SetPath(ResPath.General_RewardsO2) --UI/UMG/General_RewardsO2_vO2
        self:SetLayerId(UIPanel.ELayerId.POPUP)
        self:SetLoadMode(UIPanel.ELoadType.ASYNC)
    end

    ---@param self OrdinaryRewardPanel
    ---@return void
    def.override().OnCreate = function(self)

        self._TittleTxt = self:FindChild("Common_BG_Min.Txt_Title")
        self._Btn_Txt = self:FindChild("Btn_Max.Txt_Label")
        self:OnUIEvent("Btn_Max.Button","onClick",self.ClickReceiveBtn)

 self:OnUIEvent("Common_BG_Min.Btn_Close.Button","onClick",self.ClickClosePanel)
        self.FL_Reward =
self:CreateListWrapper(UIFixedList,"FL_Reward","Core.UI.UICommonPanel.Reward.Cel
l.OrdinaryRewardCell")
    end

    ---@param self OrdinaryRewardPanel
    ---@return void
    def.virtual().FillingText = function(self)
        self._TittleTxt:SetText("奖励")
        self._Btn_Txt:SetText("确认")
    end

    ---@param self OrdinaryRewardPanel
    ---@param data any
    ---@param callBackFunc function
    ---@return void
    def.override("varlist").OnShow = function(self,data,callBackFunc)
        if data == nil then return  end
        ---@type table <number,{ItemCid:number,Count:number}>
        local dataTable = data
        self:LoadCell(dataTable)
        self.CallBackFunc = callBackFunc
    end

    ---@param self OrdinaryRewardPanel
    ---@param dataTable table
    ---@return void
    def.virtual("table").LoadCell = function(self,dataTable)

        self.FL_Reward:SetData(dataTable)
    end
    ---@param self OrdinaryRewardPanel
    ---@return void
    def.virtual().ClickReceiveBtn = function(self)
        if self.FL_Reward then
            local cellCount = self.FL_Reward:GetCellCount()
            if cellCount >= 0 then
                for i = 1,cellCount do
                    self.FL_Reward:GetCellAt(i):ReceiveRewardCallback()
                end
            end
        end
        self:Close()
```

```
        self.CallBackFunc()
    end
    ---@param self OrdinaryRewardPanel
    ---@return void
    def.virtual().ClickClosePanel = function(self)
        self:Close()
    end
    OrdinaryRewardPanel.Commit()
end
return OrdinaryRewardPanel
```

OrdinaryRewardCell.lua

```
--- 通用奖励物品Cell
---
local Lplus         = require "Lplus"
local UICell        = require "Core.UI.UIWidget.UICell"

---@class OrdinaryRewardCell:UICell
---@field public ItemName userdata
---@field public ItemNumTxt userdata
---@field public itemBuffNumTxt userdata
---@field public FrontImage ue.UWidget
---@field public itemCid number
---@field public itemCount number
---@field public itemBuffNum number
---@field public Commit fun():OrdinaryRewardCell @notnull
---@field public OnCreate fun(self:OrdinaryRewardCell)
---@field public ClickItem fun(self:OrdinaryRewardCell)
---@field public RefreshCell fun(self:OrdinaryRewardCell)
---@field public SetContent fun(self:OrdinaryRewardCell)
---@field public ReceiveRewardCallback fun(self:OrdinaryRewardCell)
local OrdinaryRewardCell   = Lplus.Extend(UICell, "OrdinaryRewardCell")

do
    local def = OrdinaryRewardCell.define

    ---@type userdata
    def.field("userdata").ItemName = nil

    ---@type userdata
    def.field("userdata").ItemNumTxt = nil

    ---@type userdata
    def.field("userdata").itemBuffNumTxt = nil

    ---@type ue.UWidget
    def.field("userdata").FrontImage = nil

    -----------------data------------------

    ---@type number
    def.field("number").itemCid = 0      -- 奖励配置id

    ---@type number
    def.field("number").itemCount = 0    --物品数量
```

```lua
---@type number
def.field("number").itemBuffNum = 0  -- 单个物品增益

---@param self OrdinaryRewardCell
---@return void
def.override().OnCreate = function(self)
    self.ItemName = self:FindChild("Txt_Name")
    self.ItemNumTxt = self:FindChild("Txt_Num")
    self.itemBuffNumTxt = self:FindChild("Txt_Top_Normal")
    self.FrontImage = self:FindChild("Img_Normal")
    self:OnUIEvent("Button","onClick",self.ClickItem)

end

---@param self OrdinaryRewardCell
---@return void
def.method().ClickItem = function(self)
    -- todo Something  that we will know in the future
    -- todo maybe show anything of Item introduce
end

---@param self OrdinaryRewardCell
---@return void
def.override().RefreshCell = function(self)
    ---@type {ItemCid:number,Count:number}
    local cellData = self:GetData()
    self.itemCid = cellData.id     --todo ItemCid
    self.itemCount = cellData.num  -- todo Count
    self:SetContent()
end

---@param self OrdinaryRewardCell
---@return void
def.method().SetContent = function(self)
    --todo 假设奖励配表 CReward ,等待后续奖励功能完善
    local itemCid = self.itemCid
    local numCount = self.itemCount
    local rewardConf = cfg.CItemBase[itemCid]

    self.ItemName:SetText(rewardConf.name)
    self.ItemNumTxt:SetText(tostring(numCount))

    local itemBuffConf = cfg.CItemResource[itemCid]
    if itemBuffConf then
        self.itemBuffNum = cfg.CItemResource[itemCid].useItemNum
        self.itemBuffNumTxt:SetText(tostring(self.itemBuffNum))
    else
        self.itemBuffNumTxt:SetText("")
    end

    ImageTools.SetItemIcon(self.FrontImage, itemCid)
end

---@param self OrdinaryRewardCell
---@return void
def.method().ReceiveRewardCallback = function(self)
    -- todo 根据不同的物品，及货币类型
```

```
        -- todo 每个物品做一些相应的动画播放效果
        -- todo  对用户的货币，物品等进行加减计算
    end
    OrdinaryRewardCell.Commit()
end
return OrdinaryRewardCell
```

Level

LevelMan.lua

场景管理类，包括加载关卡，卸载关卡，设置场景的显隐等；insts table 存放各个场景对象实例。

加载关卡

```
---@param string levelNameOrId  （关卡的名称）
---@param function (回调函数，形参为 LuaLevel 对象)
---return Boolean
local ret = LevelMan.Load("levelNameOrId", function(luaLevel)
            if not luaLevel then
                errorf("Failed to load level, name")
                return
            end
        end)
```

卸载关卡

```
LevelMan.Unload("levelNameOrId")
```

生成场景对象

```
local LevelMan = require "Core.Level.LevelMan"
local asset = GameUtil.SyncLoad(assetPath, g_LoadType_BlueprintGeneratedClass)
---@param string 场景名称或者ResPath索引id
---@param ue.UClass 加载的资源
self.m_ViewModel = LevelMan.SpawnActor("levelNameOrId", asset)
```

获取场景中的对象

```
---@param string 关卡名称或者Id
---@param string actor名称
---@param bololean 是否包括搜索子关卡
LevelMan.FindActor(nameOrId, actorName, alsoSearchSub)
```

场景的一些常用接口：

```
LevelMan = require "Core.Level.LevelMan"

---@param nameOrId any
---@return LuaLevel
LevelMan.Find(levelNameOrId)

-- 根据名称判断level是否正在加载
---@param nameOrId any
```

```
---@return boolean
LevelMan.Isloading(levelNameOrId)

-- 根据名称判断level是否已加载完成
---@param nameOrId any
---@return boolean
LevelMan.IsLoaded(levelNameOrId)

-- 根据名称判断level是否已处于显示状态
---@param nameOrId any
---@return boolean
LevelMan.IsVisible(levelNameOrId)

-- 根据名称判断level是否存在（尚未加载或已被卸载）
---@param nameOrId any
---@return boolean
LevelMan.IsExists(levelNameOrId)
```

Fx

特效模块

Fx.lua

```
local fxPath = ResPath[fxId]

---@param resName string
---@param pos table (播放特效的位置)
---@param rotation table （特效的转向FRotation）
---@param lifetime number (播放时间)
---@param needHighLod boolean  --todo
---@param prio number          --todo
---@return userdata
local fxObj =  Fx:Play(fxPath, effPos, rot, 0, false, FXPRIO.MIDDLE)
```

Sound

声音模块

SoundManager.lua

常用播放音乐接口：

```
SoundMan:PlayBackgroundMusic("Play_Main")
---@param string 音效名称
---@param Vector3 位置
---@param Vector3 角度
SoundMan:PostEventOnLocation(EventName, location, rotator)
```

NetWork

> 网络模块

- Http.lua 用于下载,
- DirLinkMan.lua 整理网络地址http的路径
- NetWorkClient.lua 待模块初始化后，执行 Connect，传入服务器Ip地址和端口，一般从登录模块拿到对应的Ip和端口号，并在收到C++层协议i数据后，分发事件回调
- GameNetwork.lua C++处的一些回调事件处理，注册PbProcessor ,将协议和类型注册进Pbtype
- PbProcessor.lua 初始化加载 对应proto 协议到PbType ,协议路径为 “data/proto/index.lua”, index.lua 后边会讲怎么生成。PbProcessor 注册lua层注册的事件。

  使用流程：
- 设置最初的http路径，从登录模块中拿到下载好的服务器列表IP,和端口数据传给NetWorkClient 链接

```
Net:Close()
Net:Connect(curServer.address, curServer.port)
Net:Close()
```

  - 定义Pb协议
  - ```
    Protobuf 项目结构：
    \protocol_buffers
        |-- - - pb3gen                                          <- - 内部有C++ 的 vs 解决方案，用于编译Protobuf
        |-- - - protos                                          <- - 存放按照Protobuf 语法自定义的一系列 .proto 文件
                    |-- - - alliance.proto
                    |-- - - common.proto
                    |-- - - etc...
                    |-- - - ...
        |-- - - gen.bat                                         <- - 更改.proto 协议文件后，双击之后， gen.bat 即可在对应的输出目录中生成lua所需要的 .pb 文件 ，和 index.lua文件
        |-- - - pb3gen.exe
        |-- - - protoc.exe
        |-- - - readme.txt
    ```

    自定义 .Proto 数据结构示例：

    ```
    // 例1: 在 xxx.proto 文件中定义 Example1 message
    message Example1 {
      optional string stringVal = 1;
      optional bytes bytesVal = 2;
      message EmbeddedMessage {
        int32 int32Val = 1;
        string stringVal = 2;
      }
      optional EmbeddedMessage embeddedExample1 = 3;
      repeated int32 repeatedInt32Val = 4;
      repeated string repeatedStringVal = 5;
    }
    ```
    ```

```
message xxx {
  // 字段规则：required -> 字段只能也必须出现 1 次
  // 字段规则：optional -> 字段可出现 0 次或1次
  // 字段规则：repeated -> 字段可出现任意多次（包括 0）
  // 类型：int32、int64、sint32、sint64、string、32-bit ....
  // 字段编号：0 ~ 536870911（除去 19000 到 19999 之间的数字）
  字段规则 类型 名称 = 字段编号;
}
```

在上例中，我们定义了：

类型 string，名为 stringVal 的 optional 可选字段，字段编号为 1，此字段可出现 0 或 1 次
类型 bytes，名为 bytesVal 的 optional 可选字段，字段编号为 2，此字段可出现 0 或 1 次
类型 EmbeddedMessage（自定义的内嵌 message 类型），名为 embeddedExample1 的 optional 可选字段，字段编号为 3，此字段可出现 0 或 1 次
类型 int32，名为 repeatedInt32Val 的 repeated 可重复字段，字段编号为 4，此字段可出现任意多次（包括 0）
类型 string，名为 repeatedStringVal 的 repeated 可重复字段，字段编号为 5，此字段可出现 任意多次（包括 0）
注意在Proto文件中备注好各个字段的含义，方便开发人员查看

lua对协议的使用:
◉详细参考Lua代码读写PB协议规范

生成二进制数据：
●编辑完自己写好的 .proto 文件保存后，双击\sigma_client\Tools\protocol_buffers路径下的 gen.bat 生成客户端对应的 .pb 文件。 输出的.pb 文件在

```
        \sigma_client\Azure\Output\data\proto 文件夹下
```

SVN提交规范：
●提交代码时，协议部分 . proto 和 对应的 .pb 文件，只提交自己确认更改过的文件，自己改动的 .proto 文件和相对应的 .pb 文件

都要提交，防止他人svn check out 后编译报错。

```
* 在 模块名+Proto.lua文件中注册handler 回调函数接收协议数据

* 在  模块名.lua 文件中写协议发送函数

  demo:

  ```lua
  --发送 Chat.lua
  local msg = PbType.chat.CSendChat()
  msg.channelId = channelId
  msg.chatInfo.contentType = contentType
  if contentType == ChatConst.ContentType.ContentType_TXT then     -- 图文混排
      local stringSend = StringContentBuild.BuildSendInputText(inputString)
      msg.chatInfo.content = stringSend
      Pb:Send(msg)
  end
```
```

```
  --接收 ChatProto.lua
  local function SSendChat(ptc)
      if ptc.channelId == "" or ptc == nil then return end
      local speakerId =
LuaInt64.ToString(ptc.syncChatInfo.speakerInfo.speakerId)
      local bSpeakerPlayerShield =
services.Friend.IsShieldByplayer(speakerId)
  end
  Pb:Handle(PbType.chat.SSendChat,SSendChat)
```

Type

包括 Vector2,Vector3,Vector4,Rect 等

```
  --- 常见用法
  local a = Vector2.zero
  local b = Vector2.one
  local c = Vector2.right
  local d = Vector2.up
  print(a.x,a.y)
  local targetPos = Vector3.new(5, 4, 0)
  local curPos = Vector3.new(2, 3, 0)
  local sqrDis = Vector3.get_SqrLength(targetPos - curPos)
```

Input

Input.lua

一些输入事件handerEvent，将回调函数注册进相应的目标事件中就可以。

```
  -- 输入类型很多：具体可查看 Event类型的成员,用法类似
  --用法demo:

  --注册回调
  self.handle = Input.OnTapped:Add(function(_x, _y)
              self:Close()
          end)
  -- 解除注册
  Input.OnTapped:Remove(self.handle)
```

Utility

通用工具模块

Int64Patch

```lua
local a = LuaInt64.ONE
local b = a.ToString()            --int64转string

local c = 34242342
local d =LuaInt64.FromString(b)   --string 转int64
local d = LuaInt64.FromDouble(c)  -- number转int64
```

Math.lua

```lua
-- 常见用法
local a = math.round(12.5)        -- 四舍五入
local b = math.clamp(10,5,7)      -- 输出：7
```

StringPatch.lua

```lua
--获取字符串长度
string.chlen(str)
```

Table.lua

```lua
-- table 的常用操作
local dataTable = {}
table.empty(dataTable)
-- 新建一个弱引用表（k）
NewWeakKeyTable()
-- 新建一个弱引用表（v）
NewWeakValueTable()
-- 新建一个弱引用表（kv）
NewWeakKeyValueTable()
```

LoadType.lua

```lua
-- 可加载的ue资源的类型
_G.g_LoadType_Texture2D = "Texture2D"
_G.g_LoadType_Material = "Material"
_G.g_LoadType_MaterialInstanceConstant = "MaterialInstanceConstant"
_G.g_LoadType_BlueprintGeneratedClass = "BlueprintGeneratedClass"
_G.g_LoadType_SkeletalMesh = "SkeletalMesh"
_G.g_LoadType_StaticMesh = "StaticMesh"
_G.g_LoadType_ParticleSystem = "ParticleSystem"
_G.g_LoadType_AnimMontage = "AnimMontage"
_G.g_LoadType_AnimBlueprintGeneratedClass =
"AnimBlueprintGeneratedClass"
_G.g_LoadType_LevelSequence = "LevelSequence"
_G.g_LoadType_AzureAtlas = "AzureAtlas"
_G.g_LoadType_WidgetBlueprintGeneratedClass =
"WidgetBlueprintGeneratedClass"
_G.g_LoadType_UserDefinedStruct = "UserDefinedStruct"
_G.g_LoadType_FileMediaSource = "FileMediaSource"
_G.g_LoadType_MediaTexture = "MediaTexture"
_G.g_LoadType_MediaPlayer = "MediaPlayer"
_G.g_LoadType_MediaPlaylist = "MediaPlaylist"
_G.g_LoadType_LevelSequence = "LevelSequence"
_G.g_LoadType_StringTable = "StringTable"
```

malut.lua

lua tale 数据序列化的相关方法，用法不再介绍，自行查看。

Coroutine.lua

协程相关

StartCoroutine 会一直执行function，直到返回true

```
--demo:
StartCoroutine(function()
        if not GWorldCameraCtrl:CoSetViewLocation(Vector3.new(MonsterPostionX,
MonsterPostionY, 0), false,500) then
            return false
        end
        local objType
        if gmodel.MapUI.CurrentSearchType == 0 then
            objType = PbType.common.OT_Monster
        else
            objType = PbType.common.OT_Mine
        end
        if not MapHelper.WaitForMapObject(objType, ObjectID) then
            return false
        end
        services.MapStage.SelectObject(objType, ObjectID)
        self:Close()
        local SearchPanel = UI.Find("Modules.MapUI.UI.MapSearchPanel")
        if SearchPanel then
            SearchPanel:Close()
        end
        return true
    end)
```

Convex 文档

Convex 介绍

> 文档先介绍了Convex，后在新建一个表程序配置XML处讲解了配置中具体字段说明，在策划配置数据里介绍如何使用Convex相关的策划用配置工具，接着后面介绍了程序使用配置的方法和一个完整的案例

Convex工具简介

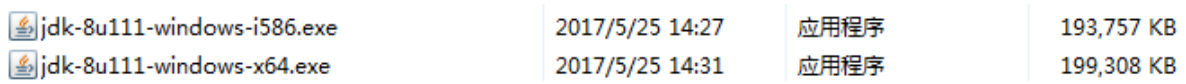
- convex是一个用于处理游戏开发中数据文件的工具集合。其核心功能是通过定义xml文件来生成对应的excel、程序代码，并将策划填写的数据导出为游戏运行时的数据文件。
- 代码类型包括：c++(分为读取xml和lua两种)、java、go、lua
- 运行时数据文件类型包括：xml文件、lua文件、bcfg二进制文件
- 支持导出数据源文件包括：excel文件、tmpl散文件、csv文件

需要安装的软件

- 安装java，版本1.8以上
- 启用excel宏
- 安装svn命令行，svn版本1.8以上，推荐安装1.9

安装JDK8

- 从\10.236.100.206\public!_4.程序分享\JDK 中下载对应版本JDK并安装。
- 使用的JDK版本如下图所示



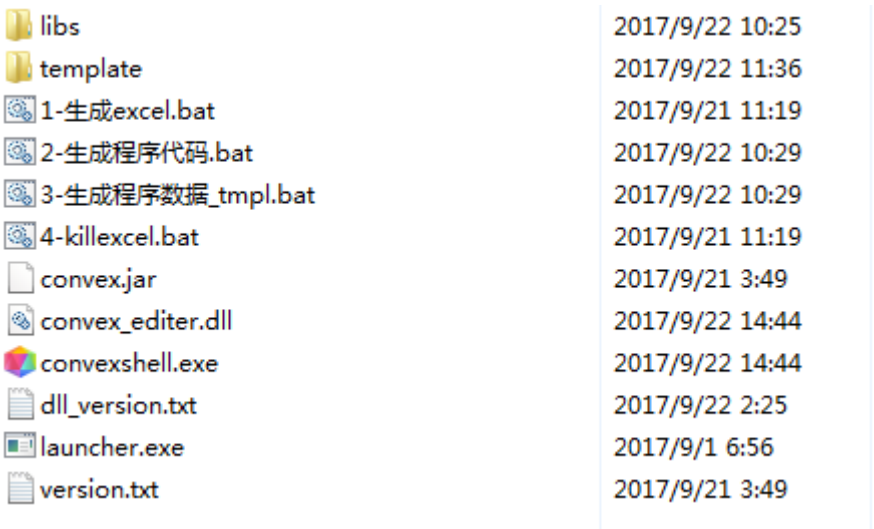
下载 convex 工具

- 从 http://convex-update.zulong.com/ 中下载样板目录.zip解压后得到convex工具，包含内容如下图所示：

目录说明：

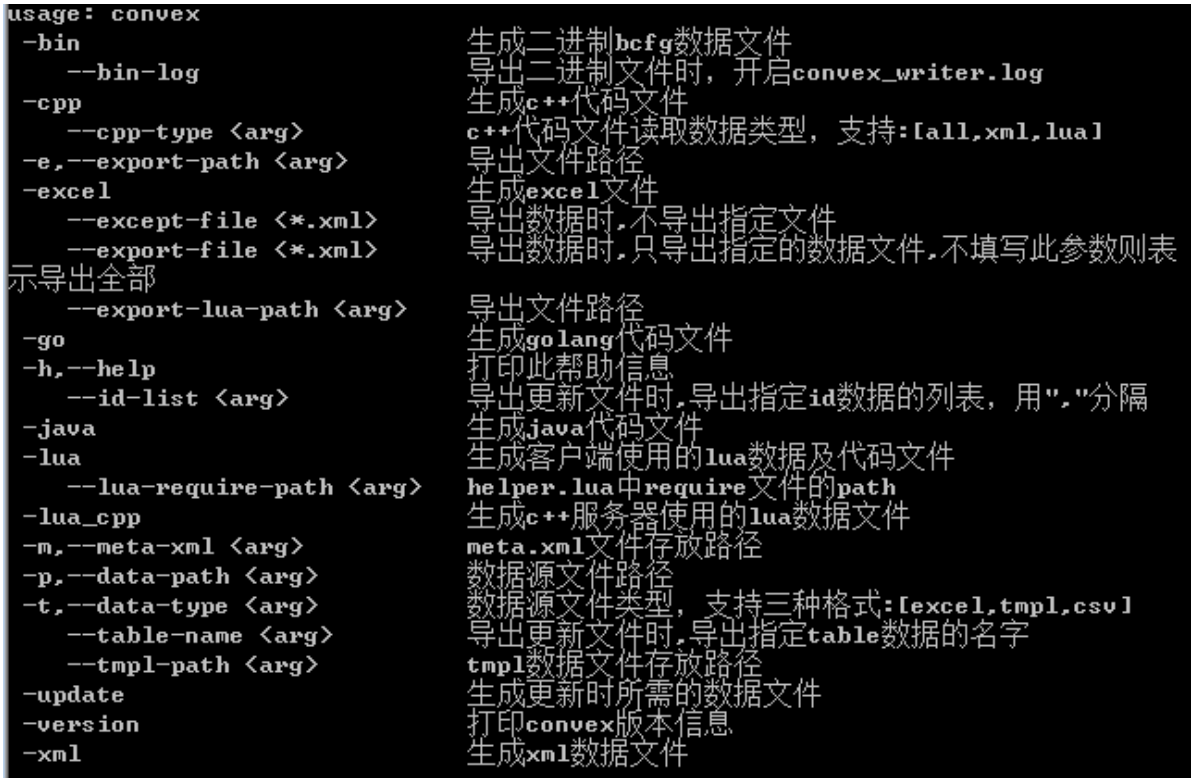
- cfgxls目录中存放excel文件
- client目录中存放客户端相关数据及代码文件
- convex目录中存放各种脚本以及工具所需的lib文件等
- metaxml目录中存放定义的xml文件
- server目录中存放服务器相关数据及代码文件
- tmpl目录中存放策划数据的tmpl散文件

convex目录说明



- **libs** 目录中存放convex工具使用的一些库文件
- **template** 目录中存放生成代码、excel所需的template文件
- **1-生成excel.bat** 用于生成excel文件，生成的excel文件放在cfgxls文件夹中，旧文件中的- 数据会被对应转移到新生成的文件中

- **2-生成程序代码.bat** 用于生成对应代码文件，生成的代码放置在对应的文件夹中，如go、cpp、java等，对应文件夹会自动创建
- **3-生成程序数据_tmpl.bat** 用于将tmpl散文件转换成对应的服务器、客户端配置文件，服务器- 文件在server文件夹中，客户端暂时使用lua，与lua代码放在一起。
- **4-killexcel.bat** 快捷关闭所有excel进制，使用前确保保存excel数据，避免丢失数据。
- **convex.jar** 为工具相关程序。
- **convexshell.exe**、**convex_editer.dll**为可视化数据编辑工具。
- **launcher.exe**、**version.txt** 为更新程序，执行**launcher.exe**会根据本地**version.txt**文件中- 记录的版本号去 http://convex-update.zulong.com/ 中比对并更新最新版本。
- 可以使用**java -jar convex.jar -h**查看帮助内容，如下图所示：



> **注意事项**
>
> - mask类型需要启用excel的宏功能(见附录1)，如果生成execl时出错，可以启动任务管理器->进程，关闭所有的excel进程(注意保存本地excel文件数据)
> - 每个脚本会自动调用launcher.exe检查convex工具是否有更新，如果直接使用convex.jar时，程序本身会自动检查当前是否为最新版本，如果不是最新版本则无法继续运行

新建一个表程序配置XML

简介

- 本文介绍了convex工具二维xml表的定义规则，通过工具生成相应的excel表格、程序代码以及服务器、客户端的配置数据
- 主要定义的xml文件分为两类：
  1. 数据二维xml表，用于定义数据结构
  2. meta.xml特殊二维表，用于汇总使用的xml二维表

数据二维xml表说明

文件功能

- 定义的xml文件(例如card.xml)，是程序根据具体功能定义的配置描述文件，工具根据xml文件生成对应的excel配置、xml配置文件，代码等。
- 生成文件根据用途分为三大类：

用途	生成文件类型
策划数据配置	.xlsm
程序配置代码	.cpp .h .java .go .lua .js
程序数据	.xml .bcfg .lua

- 一个简单的例子 card.xml 如下图所示，具体说明参见文件格式说明

```
<conf name="card" alias="卡牌配置">
  <enum name="CQuality" alias="卡牌品质">
    <var name="White"  default="1" alias="白"/>
    <var name="Blue"   default="2" alias="蓝"/>
    <var name="Purple" default="3" alias="紫"/>
    <var name="Orange" default="4" alias="橙"/>
    <var name="Gold"   default="5" alias="金"/>
  </enum>

  <mask name="CCardType" alias="卡牌类型">
    <var name="Attack"  default="0x00000001" alias="攻击卡"/>
    <var name="Defence" default="0x00000002" alias="防御卡"/>
    <var name="Assist"  default="0x00000004" alias="辅助卡"/>
    <var name="Persion" default="0x00000008" alias="人物卡"/>
    <var name="Monster" default="0x00000010" alias="怪物卡"/>
  </mask>

  <bean name="CardSkillInfo">
    <var name="skillId"     type="uint" alias="技能id" ref="skill.CSkill"/>
    <var name="probability" type="float" alias="触发概率" probSum="1" />
    <var name="week"  type="CWeek" alias="星期"/>
  </bean>

  <const name="CConstCard" alias="卡牌常量表">
    <var name="InitSkillNum"  type="int" alias="卡牌初始化技能数量" default="1"/>
    <var name="TestCardId"    type="int" alias="测试卡牌id"/>
    <var name="InitCardId"    type="int" alias="初始化卡牌id"/>
  </const>

  <table name="CCardBase" alias="卡牌基类表">
    <var name="cardName"  type="string" alias="卡牌名称"/>
    <var name="quality"   type="CQuality" alias="卡牌品质"/>
    <var name="cardType"    type="CCardType" alias="卡牌类型"/>
    <var name="icon"      type="string" alias="icon资源"/>
  </table>

  <table name="CCardBiology" alias="生物卡牌表" base="CCardBase" export="server,client">
    <var name="hp"          type="int"  alias="血"/>
    <var name="attack"      type="float"  alias="攻击力"/>
    <var name="taunt"       type="bool" alias="是否嘲讽"/>
    <var name="oddsSkillProbCount" type="list"  alias="随机技能数概率-技能数" startIndex="0" valueType="float" size="3" comment="出现0、1、2个技能的概率"/>
    <var name="skills"      type="list" alias="技能信息" valueType="CardSkillInfo" size="3"/>
    <var name="skillInfo"   type="skill.SkillInfo"  alias="附加技能"/>
  </table>
</conf>
```

文件格式说明

元素	元素描述	包含属性	属性类型	是否非空	属性描述
conf	xml文件根元素	name	string	可为空	程序代码对应包名，为空时表示代码全部生成在代码根目录下
alias	string	可为空	excel文件所属子目录，为空表示excel全部生成在excel根目录下		
table	配置表定义元素	name	string	非空	程序代码表名
alias	string	非空	excel文件名		
base	string	可为空	父表名称，填写后继承对应表所有字段，无需在本表中重复定义		
export	string	可为空	导出目标集合，用逗号分隔，支持server(服务器)，client(客户端)，为空时等同export="server,client"		
origin	string	可为空	导出程序数据文件时，策划数据源类型，目前支持：excel、tmpl、csv，默认为tmpl		
idspace	string	可为空	id空间，每个空间内的id独立分配，space在meta.xml中定义，默认会放在normal空间		
transpose	bool	可为空	转置表，当transpose=true时，将表中所有id转置为1个id，数据变为一个list，如果定义个var大于1个，自动生成一个名字为tablename_bean的结构		
enum	枚举定义元素	name	string	非空	程序代码枚举名
alias	string	非空	excel列选项		
export	string	可为空	同table元素中的exprot属性		
type	string	可为空	默认都为int16，如果需要定义，可以设置为int16、uint16、uint、int		

元素	元素描述	包含属性	属性类型	是否非空	属性描述
mask	复选定义元素	name	string	非空	程序代码枚举名
alias	string	非空	excel列选项		
export	string	可为空	同table元素中的export属性		
bean	自定义结构元素	name	string	非空	程序代码类型名
alias	string	非空	excel列名中的标记		
export	string	可为空	同table元素中的export属性		
const	常量定义元素	name	string	非空	程序文件名
alias	string	非空	excel文件名		
export	string	可为空	同table元素中的export属性		
stringTable	字符串表	name	string	非空	程序代码表名
alias	string	非空	excel文件名		
export	string	可为空	导出目标集合，用逗号分隔，支持server(服务器)，client(客户端)，为空时等同export="server,client"		
var	字段类型定义元素	name	string	非空	程序代码字段名
alias	string	非空	excel列名中的标记，**不能含有":"字符**		
type	string	非空	字段类型，支持：基本类型int16,uint16,int，uint，int64,uint64,float，string，bool，数组类型：list，自定义类型：enum，bean		
valueType	string	可为空	当type=list时，表示数组中元素类型，支持全部基本类型+bean类型		

元素	元素描述	包含属性	属性类型	是否非空	属性描述
size	int	非空	当type=list时，表示数组长度，可以定义为enum类型		
export	string	可为空	同table元素中的export属性		
default	string	可为空	字段默认值，当数据没有填写(excel中为空cell)时，生成数据使用默认值。当type=enum、mask时，需要设置成对应的数字，不要填enum、mask的英文变量名		
ref	string	非空	关联的表名，填写后限制excel中数据范围等于关联表的id范围		
comment	string	可为空	字段注释，显示在excel的列头当中		
startIndex	int	可为空	当type=list时，表示list列头拼接的字符串的起始值，默认不填写为1		
probSum	int	可为空	字段的归一化处理，在生成数据时会将excel中填写的值做归一化处理，type必须为float		

补充

- int、uint为32位，暂时不支持64位的整数
- 每个table定义元素包含一个隐藏的id属性，用来表示数据id索引，根据定义的idspace来决定table处于哪个空间，每个空间内id不会重复，默认都属于normal空间
- 当type=”list”时，需定义valueType与size字段，其中size表示数组的最大长度，实际生成程序数据时会根据配置的数据来决定数组大小，去掉空数据
- startIndex字段在拼接excel列头时使用，默认为1，只在type="list"时有效，正常如举例的skills字段的列头为“技能信息1”，“技能信息2”，“技能信息3”。oddsSkillProbCount字段的列头则为“随机技能数概率-技能数0”，“随机技能数概率-技能数1”，“随机技能数概率-技能数2”
- enum类型默认为int16，如需要定义更大的值则需要自定义type属性
- mask类型在定义时值为自动生成，默认最多可以复选32个值，如果定义的mask值不是2的整数次幂，而是其他值的复合值，在excel中的复选框不可选，策划需要手动去勾选对应的值
- var的alias中不能含有":"
- 每个table中定义的所有var元素的name和alias属性不能重复

meta.xml二维表说明

- meta.xml文件是所有定义的xml文件的一个汇总，新增xml配置文件后，需要在meta.xml中增加对应的描述，生成工具会根据此文件来生成对应的配置文件
- 一个简单的meta.xml文件如下图所示，具体说明参见文件格式说明

```xml
<?xml version="1.0" encoding="UTF-8"?>

<conf package="heroh5.conf">
  <idspace>
    <space name="card"/>
  </idspace>
  <import file="card.xml"/>
  <import file="skill.xml"/>
  <table id="1" name="CCardBase"/>
  <table id="2" name="CCardBiology"/>
  <table id="3" name="CSkill"/>
</conf>
```

文件格式说明

元素	元素描述	包含属性	属性类型	是否非空	属性描述
conf	meta.xml文件根元素	package	string	非空	程序代码对应根目录
import	包含的xml文件	file	string	非空	包含的xml文件名
idspace	id空间	space	string	非空	id空间的名字
table	生成的配置表， 根据xml文件**自动生成**	id	int	非空	配置表起始id，单位万，如id=1表示起始id是10000
name	string	非空	配置表对应名		

补充

- table中的id属性自动生成，最终会导出到lua代码中，作为cfg.CTABLE_TYPE
- meta.xml中的idspace默认会定义一个normal的space，不需要手动填写，没有指定space的table会默认在normal中

策划配置数据

> Excel 的 Convex 插件安装请看 附录1
>
> 此外需要安装 svn，详情见 附录2

Convex插件使用说明

- 

在当前表中新建指定数量的模板并赋默认值，默认值为当前表中第一行所填写的数据，当前约定单次新建模板数量上限为100个。 新建模板在提交时会自动分配ID, 不要手动改动ID或填写指定ID

模板名为必填项，项目可以设定<类别1>是否为必填项。项目可以在定义表时设定每张表的<创建目录限制>, 限制有且只有几层类别，不可填写类别单元格会标为红色，每个类别代表一级子目录。

创建目录限制：`<table create_dir_level = "0-3"> create_dir_level="0"` 时不能填写类别， 值 "1" 只能填写类别一,值为"2" 必须填写类别一和类别二，值为"3" 必须填写类别一，类别二，类别三；不需要限制时不要配置此属性

- 

更新当前表结构定义文件， 生成新定义的列头。 当前表中列头与表结构定义不匹配时，无法锁定编辑当前表中的模板，需要执行当前操作后进行锁定编辑提交。

当表结构有变化时，建议相关策划执行更新表头操作后，提交当前表，避免其它策划每次新打开编辑此表时需要更新表头

- 

读取并显示当前表中所有模板的索引数据， 不刷新所有模板的数据，不会清空当前表中原有信息

用于查看当前表中拥有的所有模板，用于查看自己所需修改的模板，速度快

- 

读取并显示当前表中所有模板的数据，会清空当前表原有的信息。

如果表中有被锁定的模板，刷新后用<查看锁定>刷新锁定状态

- 

读取并显示当前表中所选模板的数据。可以手动输入当前表中已有ID进行刷新，也可以选中已有索引进行刷新

- 

读取大于当前表中显示的最大模板ID的未显示模板数据刷新在表的尾部，小于最大ID的模板不会被刷新。
可用于刷新其它策划新增的模板，便于多人协作

- 

读取并显示与当前选中类别相同的模板数据

选中<类别1> 单元格时只对<类别1>进行匹配， <类别2>不相同也会刷新

例如：选中<类别1=副本怪物> 会同时刷新<类别2=希腊神庙> <类别2=遗迹入口>的模板
选中<类别2> 单无格时会对<类别 1><类别2>都进行匹配

例如：选中<类别2=希腊神庙> 只会刷新<类别1=副本怪物 > <类别2=希腊神庙>的模板

- 

清空前当表中所有数据。如果有清理数据的需要请使用此功能， 避免因为数据清理不完全导致新刷新数据没有在第一行，出现在一些奇怪的地方

- 

当前选中单元格如果是复选类型，可以查看当前复选类型的定义

- 

将选中模板的进行锁定， 避免多人协作时同时操作同一模板

当表结构定义发生变化时，无法锁定模板，需要执行[更新表头]的操作

- 

恢复当前表中已经刷新显示的，并且被自己锁定的模板的锁定状态

多用于第二次打开此表时继续完成上次未完成的修改操作，或者重新刷新数据后恢复锁定状态

- 

解锁当前选中模板，放弃当前修改，还原为SVN版本数据

- 

提交当前表中所有已显示并且处于新建或锁定状态的模板

当表结构定义发生变化时，无法提交模板，需要执行[更新表头]的操作

新建模板在提交时不要手动填写ID， 提交后<模板名，id，类别>无法修改

- 

提交当前表中所有选中（支持连续或不连续多选）并且处理新建或锁定状态的模板

当表结构定义发生变化时，无法提交模板，需要执行[更新表头]的操作

新建模板在提交时不要手动填写ID， 提交后<模板名，id，类别>无法修改

-

保留表中选中（支持连续或不连续多选）的模板数据，新建和锁定状态的模板数据未被选中也会保留，清除其它数据，便于针对性的处理相应模板数据

- 

导出本地修改差异文件

- 

开ConvexShell工具，ConvexSehll 工具使用看 Convex Shell 使用说明 并打开当前选中模板方便进行编辑

如果当前选中单元格为引用ID类型（需要在XML定义中配置引用表），则同时根据配置和引用ID打开引用模板

- 

快捷打开Convex 工作文件夹

- 

根据选中模板打开当前模板所在文件夹

- 

更新所有模板数据

- 

打开日志所在文件夹

- 

查看当前插件的版本号

- 

在当前EXCEL窗体中新建Sheet页签打开选中的模板表

支持输入表名进行查询，可进行模糊匹配。避免打开多张表时需要去文件夹的多级目录中查找打开EXCEL表

表头每次动态生成，避免表结构定义多次修改后旧表中可能保留冗余数据

- 

  切换不同语言版本（需要在meta.xml中定义locales）本地化的数据

  同一个ID的模板可以分别保存不同语言的本地化数据，通过切换语言版本独立修改本地化数据

  EXCEL中只能修改当前模板中已有语言版本的本地化数据， 不能为模板增加新的语言版本。没有对应语言版本的数据时，显示基础版本的数据；需要增

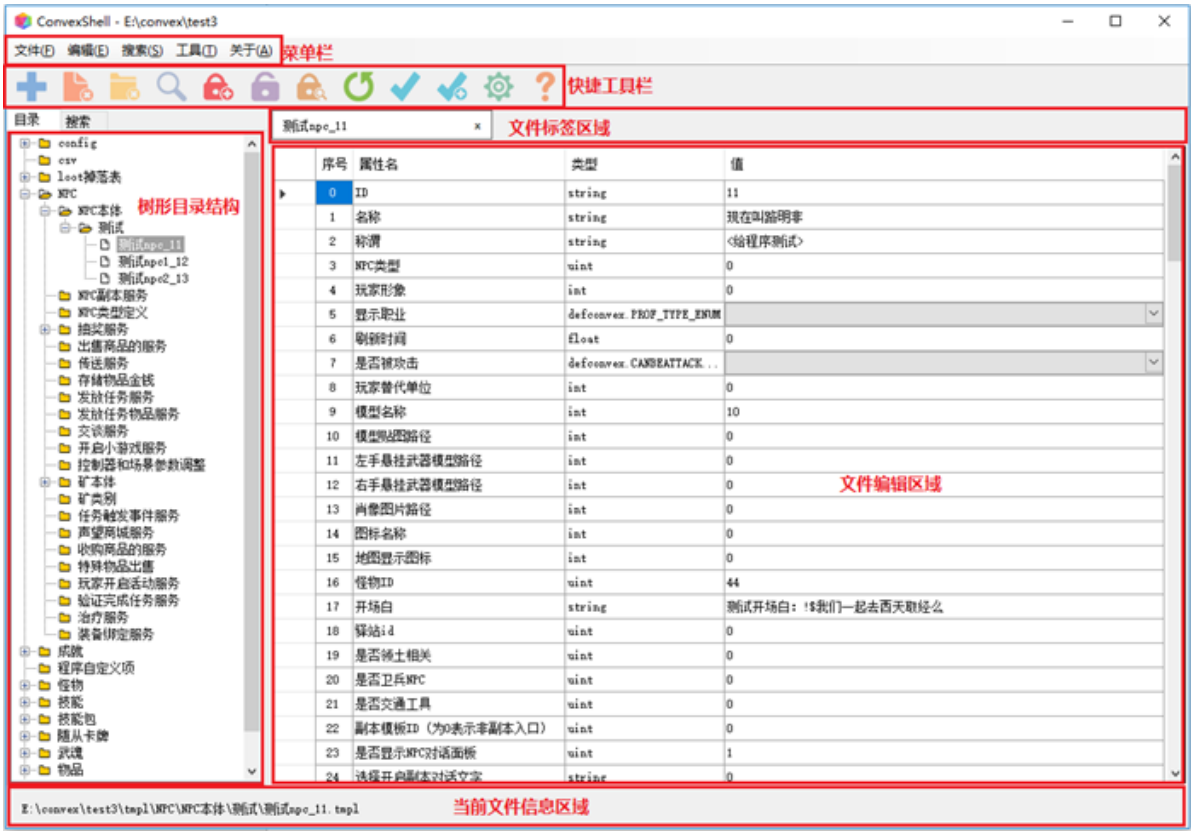
  模板的（在meta.xml中定义的）本地化版本，使用ConvexShell增加

ConvexShell使用说明

ConvexShell简介

- ConvexShell，一个convex编辑器的外壳，一个可视化的数据编辑工具
- 集常用模板功能于一身，特别添加精确条件搜索
- 效率即一切，数据编辑与svn操作无缝衔接
- 所见即所得，可视化数据处理时代已经来临
- 采用轻量级框架，最大程度简化操作流
- 功能配备多入口，不同习惯无障碍使用
- ConvexShell，策划工作的最佳伴侣
- 原来编辑数据，也可以如此妙不可言

界面图示



主要功能

新建文件

- 方式1：先于左侧目录结构中选中目录/文件结点，然后使用菜单栏 文件-新建文件或者快捷工具栏  ，于所选目录位置/所选文件同级位置，新建单个文件。
- 方式2：在左侧目录结构中对应目录结点单击右键，打开菜单，选择 新建文件/批量新建文件选项。

新建目录

在左侧目录结构中对应目录结点单击右键打开菜单，选择新建目录选项，如下图所示

打开文件

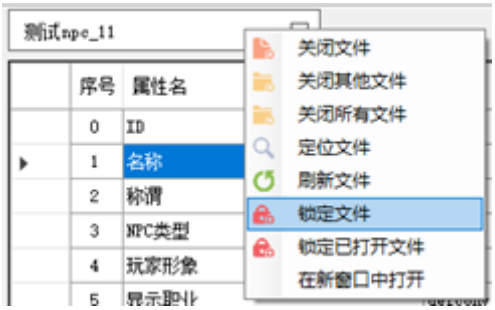
- 方式1：先于左侧目录结构中选中要打开的文件结点，然后使用菜单栏 文件-打开文件，即可在右侧打开单个文件
- 方式2：在左侧目录结构中的文件结点单击右键打开菜单（可使用ctrl键多选），选择 打开文件（打开选中文件） 选项，如下图所示

- 方式3：在左侧目录结构中直接双击文件结点，即可打开单个文件

锁定文件

> 由于ConvexShell是基于svn进行版本管理的，因此只有已锁定的文件才能进行编辑，锁定后文件会有标识出现，如下图所示
>
> 
>

- 方式1：先于左侧目录结构中选中要锁定的文件/目录结点，然后使用菜单栏 编辑-锁定文件，即可锁定单个/多个文件
- 方式2：使用快捷工具栏中的按钮，即可根据所操作的位置，锁定左侧目录结构中的文件/目录（可使用ctrl键多选），或者 锁定右侧当前已打开的文件
- 方式3：在左侧目录结构中的文件/目录结点单击右键打开菜单（可使用ctrl键多选），选择 锁定文件（锁定选中文件） 选项
- 方式4：对于已打开的文件，右键单击文件标签页，在所弹出菜单中选择 锁定文件 选项，如下图所示



修改文件

- 只能修改已锁定文件
- 双击属性名可以快捷进入属性搜索
- mask复选类型需要双击 值列 进行浏览/编辑
- 每一行都有类型校验
- 文件标签页可以拖动，使得文件在新窗口中打开；同理，返回主窗体时也可拖动标签页完成

解除锁定

操作方式同锁定文件

查看锁定文件

点击快捷工具栏中的相应按钮 

，即可查看/解锁已经锁定了的文件

提交文件

- 方式1：先于左侧目录结构中选中要提交的文件/目录结点，然后使用菜单栏 编辑-提交文件，即可提交单个/多个文件
- 方式2：使用快捷工具栏中的  提交，批量提交 按钮，即可根据所操作的位置，提交左侧目录结构中的文件/目录（**可使用ctrl键多选**），或者 提交右侧当前已打开的文件（其中，批量提交按钮可以扫描所有与svn存在差异，即需要提交的文件）
- 方式3：在左侧目录结构中的文件/目录结点单击右键打开菜单（**可使用ctrl键多选**），选择 提交文件（提交选中文件） 选项

> 注意：只能提交已锁定文件，并且提交完毕将会自动将文件解锁

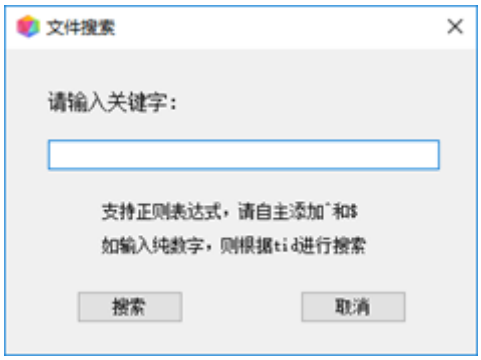
导出文件

- 方式1：先打开文件，然后选择顶部菜单栏 编辑-导出当前文件，之后选择导出位置即可
- 方式2：在左侧目录结构中的目录结点单击右键打开菜单，选择导出目录下已锁定文件选项，如下图所示

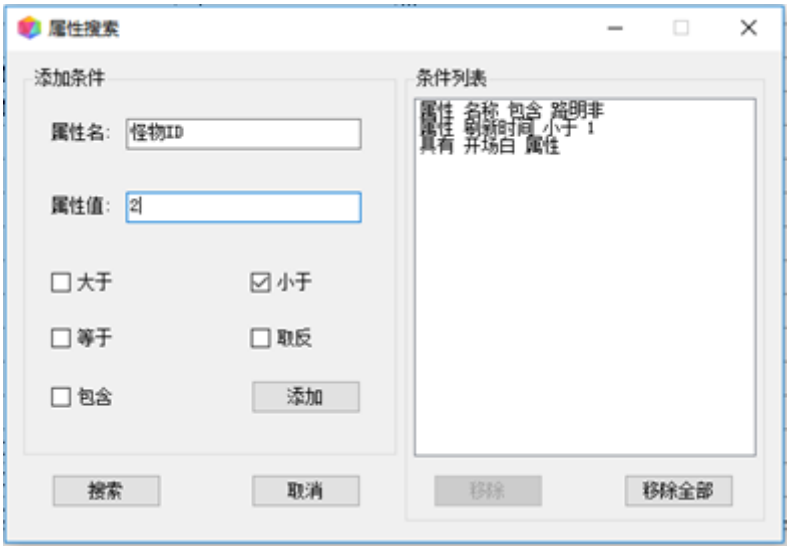
搜索

> 随着项目不断的开发，配置文件会越来越多，为了便于查看，因此一个强大的搜索功能必不可少，ConvexShell工具提供了一下几种搜索方式：

- 文件搜索：搜索文件名中含有匹配字符串的文件，如输入纯数字（如：50）即对表id进行搜索，如下图所示



- 属性搜索：可添加多条件，搜索满足所有条件的文件，如下图所示。（详细用法请参考附录）



- 内容搜索：仅对于当前已打开的文件，在右侧文件编辑区域匹配字符串，重复点击可跳转至下一个匹配项。

其他功能

- 保存文件:将当前所做改动保存至本地,不涉及svn操作
- 刷新文件:放弃当前所做更改,并从本地重新读取文件内容
- 定位文件:对于已打开文件，可以快速的在左侧目录结构中找到对应的文件结点

convex常用命令行参数说明

> convex.jar包命令行使用说明，可通过java -jar convex.jar -h命令查看

指定路径相关参数

```
--meta-xml <arg> meta.xml文件存放路径
--data-path <arg> 数据源文件路径
--data-type <arg> 数据源文件类型，支持三种格式:[excel,tmpl,csv]，通常都为tmpl类型
--data-csv-path <arg> csv数据源文件路径
--data-excel-path <arg> excel数据源文件路径
--data-tmpl-path <arg> tmpl数据源文件路径
--export-path <arg> 导出文件路径
```

生成代码相关参数

```
-go     生成golang代码文件
--go-one-package 导出的golang代码在同一个package中

-cpp  生成c++代码文件
--cpp-type <arg> c++代码文件读取数据类型，支持:[all,xml,lua]
--locker-init <arg> c++代码中locker锁的初始化值

-cpp_use_define_map c++使用自定义的map类型，默认false时使用abase::ash_map，生成代码会
#include <hashmap.h>
--cpp-namespace <arg> c++导出代码时，namespace的名字

-java 生成java代码文件

-cs      生成c#代码文件

-lua    生成客户端使用的lua数据及代码文件
--lua-require-path <arg> helper.lua中require文件的path
--export-lua-path <arg> 导出文件路径
--sub-helper <arg> 导出客户端helper.lua时,只导出export-file中指定的表

-js      生成客户端使用的js数据及代码文件
```

生成数据文件相关参数

```
-bin 生成客户端的二进制bcfg数据文件
--bin-log 导出二进制文件时，开启convex_writer.log
--bcfg-cpp-const 导出客户端bcfg文件时,常量表导出bcfg文件

-xml 生成xml数据文件

-lua_cpp 生成c++服务器使用的lua数据文件

-pb_bin 生成pb二进制数据文件
--pb-desc-file <arg> pb描述文件

-excel 生成excel文件
--tmpl-path <arg> tmpl数据文件存放路径，生成excel时同时创建对应的tmpl文件夹。
```

本地化相关参数

```
-extract 提取数据中的字符串,生成xml文件

-translate 根据xml文件,翻译数据中的字符串
--translate-file-path <arg> 翻译文件路径
--locale <arg> 数据版本
```

生成数据文件相关参数

```
-bin 生成客户端的二进制bcfg数据文件

--bin-log 导出二进制文件时，开启convex_writer.log
--bcfg-cpp-const 导出客户端bcfg文件时,常量表导出bcfg文件
```

```
-xml 生成xml数据文件

-lua_cpp 生成c++服务器使用的lua数据文件

-pb_bin 生成pb二进制数据文件
--pb-desc-file <arg> pb描述文件

-excel 生成excel文件
--tmpl-path <arg> tmpl数据文件存放路径，生成excel时同时创建对应的tmpl文件夹。
```

本地化相关参数

```
-extract 提取数据中的字符串,生成xml文件

-translate 根据xml文件,翻译数据中的字符串
--translate-file-path <arg> 翻译文件路径
--locale <arg> 数据版本
```

指定导出表相关参数

```
--export <arg> 导出类型:可选值 "server"-服务器 "client"-客户端"server,client"-服务器和客户端
--export-file <*.xml> 导出数据时,只导出指定的数据文件，多个xml用","分隔，不填写此参数则表示导出全部
--except-file <*.xml> 导出数据时,不导出指定文件，多个xml用","分隔
--export-table <arg> 导出数据时,只导出指定table的数据文件，多个table用","分隔，不填写此参数则表示导出全部
--except-table <arg> 导出数据时,不导出指定table的数据文件,多个table用','分隔
--export-tmpl-table <arg> 导出数据时,只导出指定tmpl文件对应的table的数据,需传入从tmpl开始的相对路径，多个tmpl文件用','分隔,不填写此参数则表示导出全部
--export-file-tmpl-table <arg> 导出数据时,读取文件中tmpl文件对应的table的数据,不填写此参数则表示导出全部
```

其他参数

```
--list-size-variable <arg> list的长度是否可变，默认为true

-high_verify 增强校验模式，会对enum、int类型等做强制校验，类型不匹配直接报错中断，不会继续生成数据文件。

-h,--help 打印此帮助信息

-version 打印convex版本信息
```

配置与导出

Lua客户端接入convex说明

如何生成程序使用的配置

todo 如图所示

生成文件说明

convex客户端生成的文件分为两个版本: Lua源文件版与二进制版本

- Lua源文件版只生成lua源码, 生个配置对象对应成Lua的表, 其中使用元表与字符串提取减小文件大小. 主要用于开发期调试用, 方便程序员查看配置对象的属性
- 二进制版本会生成:
  - **二进制数据文件**: 存放配置对象与配置对象的元信息
  - **少量lua源文件**: 助手函数与常量定义

> 二进制版本用于发布使用, 针对内存占用与读取速度作了优化与平衡
> 如果项目有自己的需求, 可以通过导出工具与加载参数配置导出文件的存放路径
> 二进制文件的目录bcfg之所以放到cfg的子目录, 是因为cfg一般是引用策划目录的外链, 以子目录的形式工作可以减少外链数

如何接入convex?

- lua版本的接入, 一行代码:

```
require "cfg.helper"
cfg.loadAll("lua/cfg/bcfg")    -- 可以没有, 加上这行是为了与二进版统一
```

- 二进制版本的接入, 在lua脚本处是两行代码:

```
require "cfg.bcfg.helper"
cfg.loadAll("lua/cfg/bcfg")
```

> - loadAll函数的说明:
> - loadAll是定义在helper.lua中的助手函数, 帮助加载所有的配置文件
> - 参数是一个字符串, 填入bcfg目录的路径
> - 如果是lua版本, 也定义了一个空的loadAll()函数, 也就是说, 二进制版本与lua版本的差异可以控制到一行:
>   `requrie "cfg.bcfg.helper"` 与 `require "cfg.helper"`, 推荐使用参数控制两个版本的使用

如何加载卸载配置对象?

- 加载: 如上所说, cfg.loadAll()提供了加载的统一入口, 注意需要一个参数是bcfg目录的路径.
  该函数返回加载配置文件出错的数量, 项目可以根据需要定制出错时的行为
- 卸载: cfg.unloadAll()
  - 该函数会把所有配置对象卸掉, 方便热更新时使用
  - 该函数只针对二进制版本, lua版本是一个空实现
- 加载行为的控制:
  - 使用convex.derferload()函数可以控制是否启用延时加载, 默认行为是不启用
    - convex.derferload(1) – 启用延时加载
    - convex.derferload(0) – 关闭延时加载
  - 延时加载是指配置对象在真正使用时才读取二进制文件. 当然, 如果这时读取出错, 将会返回nil
  - 延时加载可以加快app的启动速度, 因为启动时并没有去读配置文件

- 延时加载可能会对游戏运行时的某一帧造成卡顿.
- 当然, 只是有这个可能, 从游戏时加载美术资源的对比来看, 配置的加载并不会慢于美术资源的加载, 一般情况应该没有卡顿问题.
- 延时加载只针对二进制版本启用, lua版本是空实现

如何取出配置对象来使用?

- 需要取出配置对象, 必须要有对象的id, 对象的id是一个32位的整数且必须大于0
- 知道配置对象的类型时, 这是性能最高的方式, 比如:
  `local crune = cfg.citemrune[570001]`
  这是从符文配置对象里取出一个符文配置
- 只知道对象的id, 不关心类型时, 可以使用统一的函数:
  `local crune = cfg.get(570001)`
  由于lua是动态语言, 拿到的crune实际就是一个符文配置, 有符文配置的所有配置项
- 如果配置有继承关系, 也可以从基类取:

```
local crune = cfg.citembase[570001] -- 这里的citembase是一个物品基类配置表, citemrune继承自citembase
```

由于lua是动态语言, 拿到的crune实际就是一个符文配置, 有符文配置的所有配置项

取出配置对象后, 对配置项的使用有三种情况:

- 如果配置项是基本数据类型(bool, int, uint, int16, uint16, float, string), 使用.操作符可以直接使用, 比如符文的名称是crune.name
- 如果配置项是bean, 这项在lua版本中是个表, 在二进制版本中是个userdata, 对bean的使用, 只用.操作符取成员.
  一定不要针对bean作遍历, 在lua中userdata没有对pair的支持.
  同样的, 由于一个配置对象(即excel中的一行)也是一个userdata, 也不支持使用pair/ipair进行遍历.
  取出的bean对象, 应该声明为local变量, 这种对象不能缓存(即bean变量的生命期一定不能比行级对象长), 否则可能会读取不到值! 这不是bug, 这是一个设计: convex对象以行为单位进行回收.
  以下截图来自于一个实际项目, 缓存了bean对象的代码, 在原来的convex代码可能会崩溃, 新的代码可以避免进程崩溃, 但可能在真正使用bean时会读到nil值:

```lua
function UITop:setpOperater(  )
    local cfightModel = cfg.cfightmodelandowner[65]
    self._curMaxTime = cfightModel.time

    local stepInfo = cfightModel.stepInfo
    local stepData = {[0] = stepInfo[1]}
    local overStepTime = 0
    local clastStep
    local isFirst = true
    for i,v in ipairs(cfightModel.stepInfo) do

        if isFirst then
            isFirst = false
            stepData[overStepTime] = v
        else
            overStepTime = overStepTime + clastStep.time
            stepData[overStepTime] = v
        end
        clastStep = v
    end
    self._setpInfo = stepData

end
```

- 如果配置项是list, 无论是lua版本还是二进版本, 都是一个lua的table, 可以支持遍历, 也可以使用[]下标操作符.

对应于excel中的每行的数据对象, 可以拿到id字段, 代表了配置表唯一id. 如 `print(crune.id)`

如何遍历一个配置表?

为了两个版本的兼容性, 遍历配置表时, 不要直接对cfg.citemrune这样的配置表作遍历, 比如以下代码是错误的:

```lua
for id, cobj in pairs(cfg.citemrune) do

...

end
```

正确的遍历方式是使用配置表的_ids()方法, 返回一个id的table, 拿到id后, 再作遍历, 比如:

```lua
local idTable = cfg.citemrune._ids()
for , id in ipairs( idTable) do

    local crune = cfg.citemrune[id]

    ...

end
```

也可以使用cfg的助手函数, 提供一个回调作遍历:

```
cfg._foreach(cfg.citemrune, function(crune)

    ...

end)
```

> 注意: 返回所有id的 `_ids()` 与遍历助手函数 `_foreach()` 都以_开头, 想表明这是一个副作用比较大的函数

- 只在配置表的条目比较少时使用
- 只在必要时使用, 比如上面符文的例子, 在软软项目中, 所有的符文一共才几十个, 在启动时需要所以符文作整理才使用
- 对于数据大的配置表, 使用遍历会造成频繁的内存分配与回收

二进制中的字符串编码

二进制文件bcfg中字符串, 是以utf-8编码

使用convex读取出来的字符串也是以utf-8作编码

如何查看内存消耗

convex提供了 `convex.memorybytes()` 方法, 返回三个值: 已经载入配置表的数量, 配置数据内存占用, convex模块内部的管理对象内存占用.

其中第三个返回值所示的内存占用, 不包含lua对象在lua vm上所占的内存(由于数据都放在convex内部, lua对象占用的内存是lua对象管理的消耗, 理论上是比较小的).

示例:

```
function info:onMemoryBytes()
     local tableCount, dataSize, objSize = convex.memorybytes()
     local allKB = (dataSize + objSize) / 1024
     local dataKB = dataSize / 1024
     local str = string.format("loaded %d tables, %.2f KB , %.2f KB",
tableCount, dataKB, allKB)
     dbg.infof(str)
end
```

配置与导出案例

其它

convex字段显示名设置

在meta.xml文件中增加 `displayName = "true"`

```
<conf package="com.zulong.conf" forcecategory="false" displayName="true">
```

执行生成EXCEL表的脚本后生成 **TMPL目录** 下 `_display_name` 文件夹， 对应每张表一个文件，用于设置表字段的显示名

附录

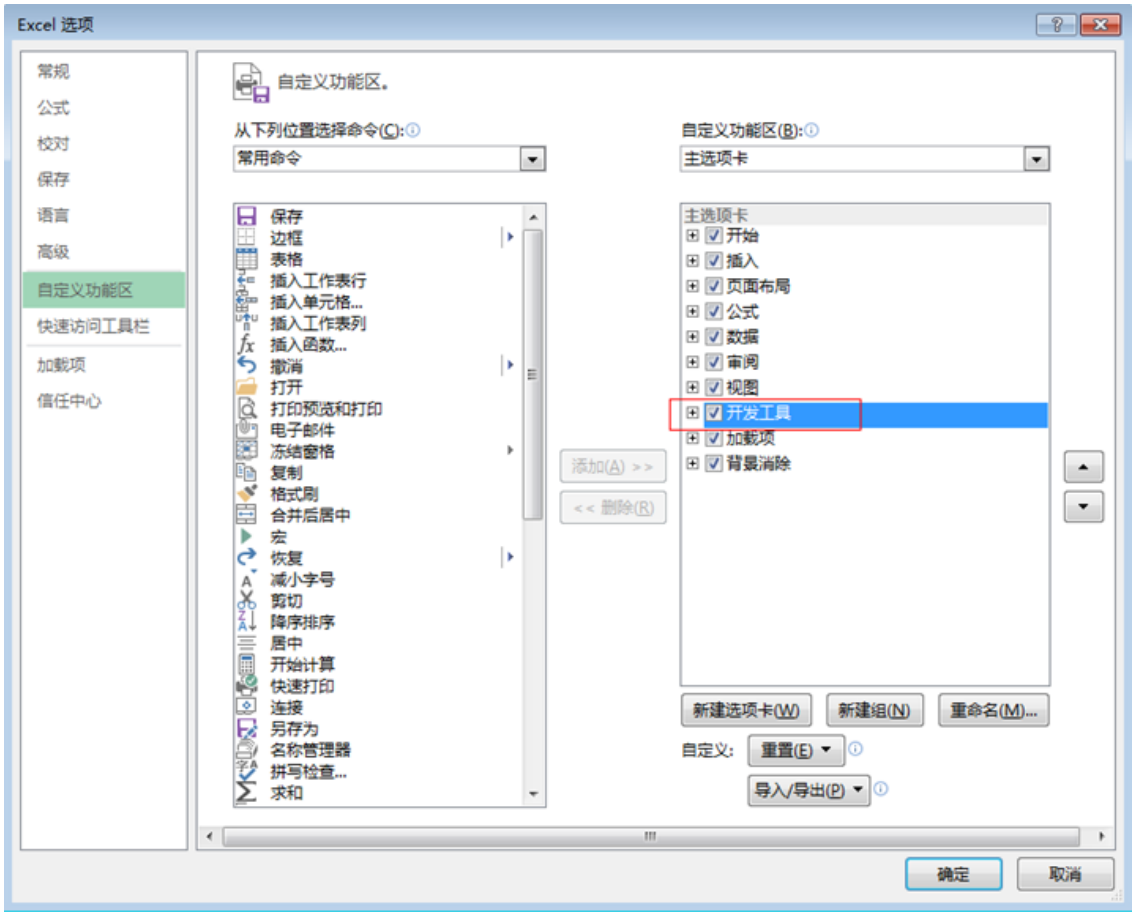
附录1：开启excel宏功能

由于工具中使用了excel的宏功能，因此需要手动配置开启excel宏功能。

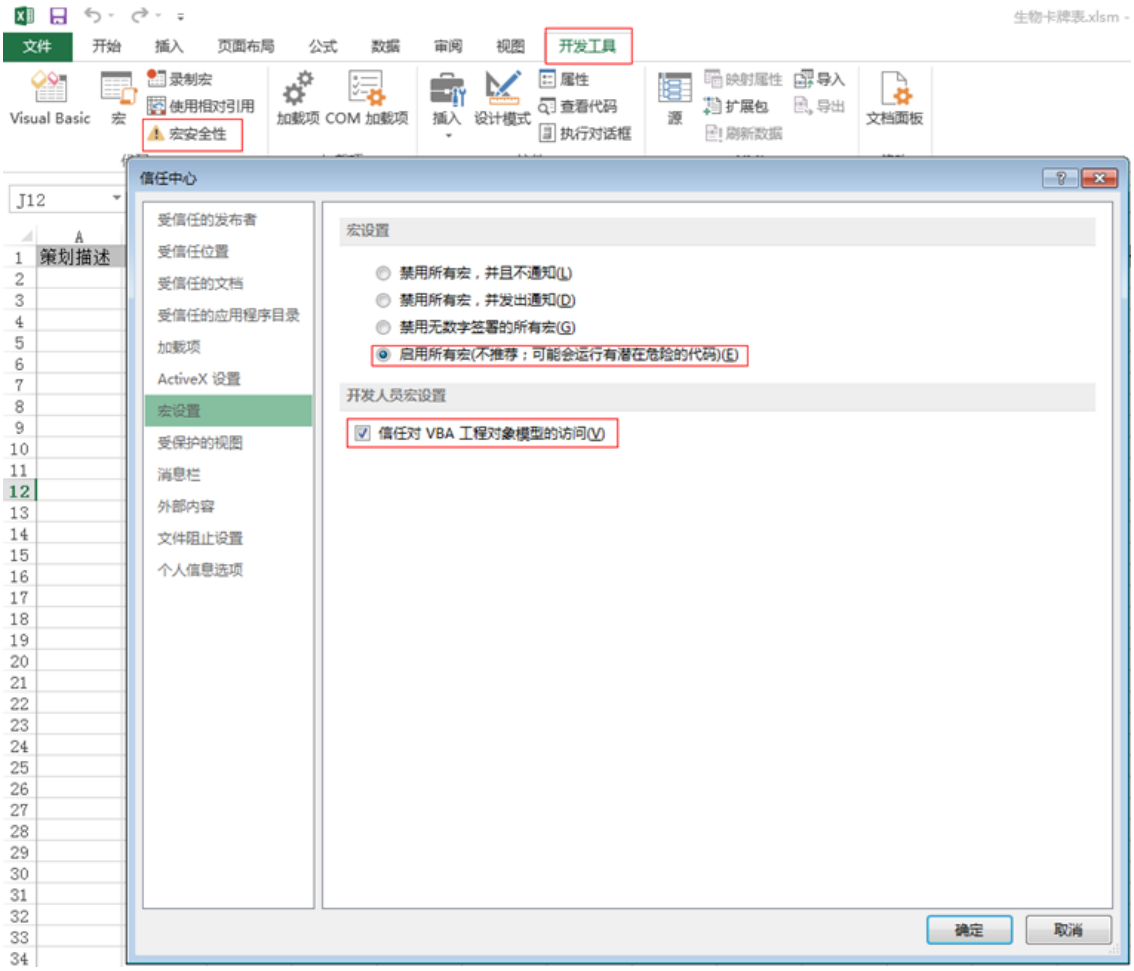
以excel2013为例，开启步骤如下：

1. 开启excel宏功能，勾选文件->选项->自定义功能区->开发功能
2. 点击“文件”按钮
3. 点击“选项”按钮

4. 点击“自定义功能区”，勾选“开发工具”，点击“确定”按钮，开启Excel宏功能

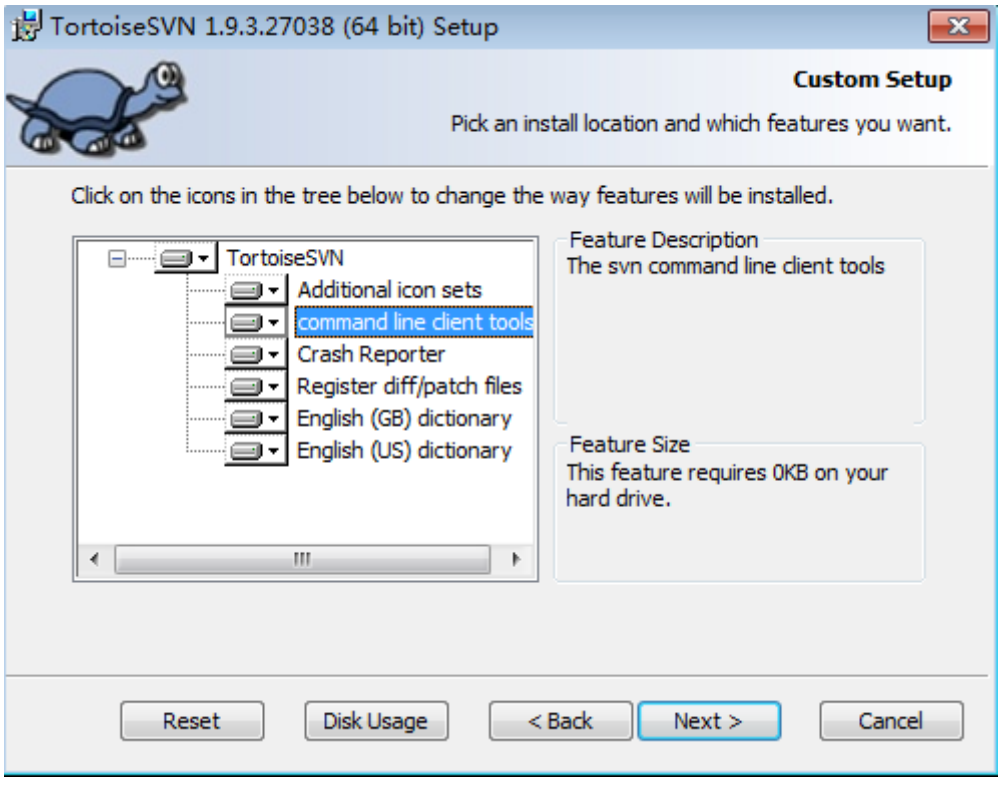


5. 启用excel宏，点击开发工具->宏安全性，选择“启用所有宏”，勾选“信任对VBA工程对象模型的访问(V)”后，点击“确定”按钮



附录2：安装svn命令行工具

- 在使用convex插件操作excel以及shell编辑工具时，需要配合使用svn命令行工具，因此在安装svn时，需要同时安装svn命令行，svn版本需要1.8以上，最好安装1.9
- 安装svn时勾选command line client tools选项，如下图所示，即可安装svn命令行工具



附录3：ConvexShell属性搜索的用法

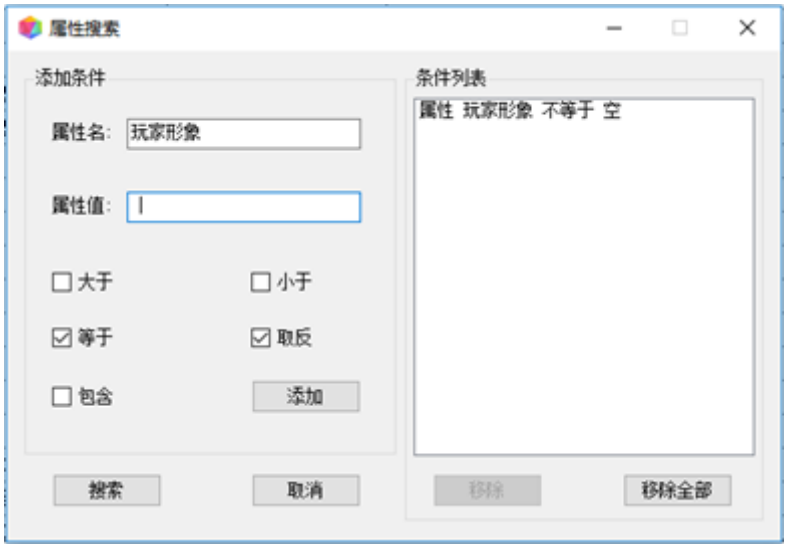
以下面的文件举例：

	序号	属性名	类型	值
	0	ID	string	11
	1	名称	string	现在叫路明非
	2	称谓	string	<绘程序测试>

- xxx 属性 等于 a：该属性的值必须等于a，如果xxx属性为int型，那么是值相等，如果为string，则为完全匹配
- xxx 属性 大于/小于 a：该属性如果为int/float类型，那就不说了，如果是string，那么为大于小于就对应字典序
- xxx 属性 包含 a：比如上图中，以下条件为真：名称 属性 包含 路明非； 模型名称 属性 包含 1

以下列出一些其他常用条件：

- xxx 属性 不等于空：（属性值填了一个空格）



- 具有 xxx属性：（属性值啥也没填）