**探索-前端项目开发规范**

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 修改时间 | 修改人 | 简要 |
| 1 | 2016-11-09 | 向金保 | 创建文档 |
| 2 | 2016-11-29 | 向金保 | 修改文档 |
| 3 |  |  |  |

目录

[目的 2](#_Toc471217501)

[美术资源 2](#_Toc471217502)

[资源分类 2](#_Toc471217503)

[资源制作规格 3](#_Toc471217504)

[资源命名 4](#_Toc471217505)

[编码模块 4](#_Toc471217506)

[模块层次 4](#_Toc471217507)

[功能整合 6](#_Toc471217508)

[命名空间 7](#_Toc471217509)

[注意事项 7](#_Toc471217510)

## 目的

主要目的快速熟悉项目开发流程和规范，快速融入项目开发团队，同时也为规范项目结构，保证项目资源和代码相对规范。

## 美术资源

美术资源是游戏开发中起着非常重要作用，本质无非使用技术驱动资源按照主观的表现。然而在游戏制作过程中，资源的类型繁多、规格不一、格式多样等问题都是项目开发中必须面对的问题。如何确保美术资源明确分类、规格统一显得尤为重要。这里就美术资源分类、规格、命名阐述美术资源开发规范。

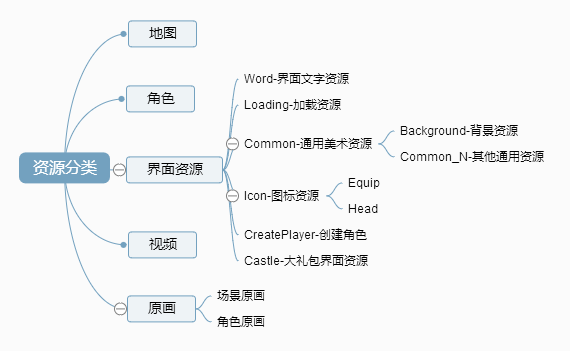
### 资源分类

游戏开发中资源分类主要分为：

1. 场景资源（也称地图或关卡资源）：无论是2D/3D游戏都有场景的概念，场景的资源也是游戏资源重要资源，资源量相对也是比较大。
2. 角色资源：这里角色资源是一个通用的概念，可以指主角、怪物、宠物等角色美术资源。
3. 界面资源（也称UI资源）：界面资源量也是比较大，而且种类繁多，如何来组织分类界面资源显得非常重要，合理的分类不仅可以起到规范资源，便于查看作用，更重要的是会大大的节省美术资源。

* 通用界面：界面背景、界面的按钮、提示框等资源可能都是通用资源，这时我们不可能所有界面都独立引用一份，这时便需要一个通用的模块，让所有界面统一引用。
* 图标资源：图标资源是一个相对比较特殊资源，主要有道具图标、角色图标、技能图标等等。
* 功能界面：除了上面资源，其他主要是一些功能模块资源，一般游戏会有背包、战斗、活动等等。

三国志项目资源分类为例，整理思路是按照上面资源分类规范来分类，这里不再一一说明具体分类原因，目前界面资源和角色资源已经按照规范整合进来，其他美术资源正在收集和整合中。



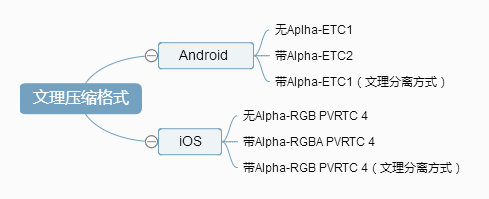
### 资源制作规格

游戏前端，不同的游戏引擎，对不同格式资源处理上都略有不同，庆幸的是存在一些通用的业界标准的资源处理标准，这些标准对于项目开发非常有帮助，也为资源的通用性，留下非常的接口。

1. 模型资源：模型资源对于3D游戏来说是一种很常见的资源，三国项目模型主要分为角色模型和场景模型，角色模型采用Spine动画实现，想具体了解Spine动画的可以参考这个论坛：<http://zh.esotericsoftware.com/spine-in-depth>，场景资源主要采用模型片的方式实现。
2. 动画资源：动画采用是新动画系统和旧动画系统混合的方式实现，不同的情况下，采用不同的动画系统，UI动画采用旧动画系统、角色模型和一些复杂的状态动画采用新的状态机的动画系统实现。可参考项目的界面动画和模型动画部分。
3. 文理资源：对于三国项目来说，处理文理资源变得非常重要，项目本身的定位是偏2D的项目，游戏资源主要依赖是2D文理，这里主要说一些界面、模型、特效文理，作为理解文理图片的现状和处在问题。

* 界面：如上面的所作，界面是按照模块进行划分，不同模块切图最终会被打入独立的图集中为开发使用，界面图集可能会存在重复切图资源以及某些资源划分不合理，优化期将重点进行整理。
* 模型：模型文理，采用Spine工具方式制作的图集，图集原理和NGUI图集是一样的，目前主要问题某些角色图集资源分配不合理，占用资源量过大，性能较低。
* 特效：特效资源文理是游戏中最多文理图集，游戏中大概有100个以上角色，每一个角色技能3-5个，所有角色技能特效资源之间相互独立，重复资源多，这也是后期优化的一个关键，合并特效文理资源。

文理资源压缩格式也是非常重要的，这里重点强调一下，资源压缩格式应采用如下压缩格式：



### 资源命名

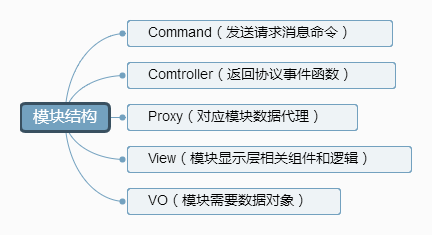
资源命名按照规范文档确定，了解项目资源命名规范的可以阅读资源命名规范相关文档。

## 编码模块

编码模块是前段程序接触最多的模块，也是最容易出现混乱模块，这里项目

### 模块层次

前端是按照功能模块来划分，开发期间，不同的人负责不同的功能模块，各模块之间没有逻辑上的关联性，模块主要分为：Command、Controller、View、VO等，可参考下图。



1. **Command层**

命令层作用发送消息，项目中如果需要向到服务器发送请求，就必须写一个命令，然后调用网络层的接口NetworkManager.Send(msg)将消息发送至服务器。例如发送S2C\_XXX消息到服务器，便可定义如下接口。

public class XXXCommand : SimpleCommand {

public const string NAME = " XXXCommand";

public override void Execute(INotification notification){

C2S\_XXX c2s = new C2S\_XXX();

NetworkManager.Send(c2s);

}

}

一个功能模块可能会有多个操作需要请求服务器，同理可以定义多个Command命令来发送不同消息。

1. **Controller层**

控制层作用是处理请求响应的逻辑，将请求返回的数据保至Proxy层，同时通知View显示层更新渲染（界面，技能等）逻辑。如何来写一个模块的控制层呢？只需在控制层里定义响应回调函数即可。例如要接受S2C\_XXX响应消息，便可定义如下响应回调函数。

public class XXXController {

public static void OnXXXRefresh(S2C\_XXX s2c) {

// 更新数据代理层数据

// ProxyPool.XXXProxy.Update(s2c);

// 通知渲染层更新渲染

// GameFacade.SendNotification(NAME, null);

}

}

和消息发送时一样，一个模块可能会有很多响应操作，这时只需在XXXController里面定义多个消息回调函数即可。

1. **Proxy层**

数据代理层，作用是提供数据的存储和操作，所有网络层返回的数据将被控制层存到数据代理层。数据代理会提供基本的增、删、改、查的接口。例如XXX功能模块的数据代理可以这样定义。

public class XXXProxy : Proxy {

public static string NAME = "XXXProxy";

private *Dictionary*<int, XXXData> xxxMap = null;

public XXXProxy() : base(NAME) {

xxxMap = new *Dictionary*<int, XXXData>();

}

public void Add(XXXData data){}

public void Del(int id) {}

public void Modify(XXXData data) {}

public void Get(int id) {}

}

一般情况一个模块会存在对应的数据代理层，当然不是所有的功能模块都需要对应数据层，如果功能模块不需要存储数据，而是直接使用其他模块数据，这时便没有必要增加额外的数据代理。

1. **View层**  
   显示层，主要作用是存放前端渲染相关的显示操作逻辑，显示层主要分为两部分，中间层和渲染层。

* 中间层（Mediator）

所有界面都需要一个中间层来接受事件，中间层在项目都以XXXPanel命名，中间层会持有渲染层的显示对象，可以洞察渲染对象任何状态。

* 渲染层（Renderer）

中间层操作便是渲染层的渲染对象，这里渲染对象可能是一个UI界面，也可能是一个特效，或者是一个角色模型，甚至是场景模型。

1. **VO**

数据对象，功能模块内部可能会出现数据传递，这时可能需要定义一些数据对象，这些数据对象统一放在VO目录下面即可。

### 功能整合

上面介绍一个功能模块可能包含的层次，如何才能保证一个模块功能正常运转呢？因为上面只是编写模块各个层次逻辑代码，各模块之间没有任何关系，相当于，这些只是汽车零部件，还没有将汽车组装起来，下面就将这些部件组装起来，让汽车能够跑起来。

1. 注册请求消息命令，在ProtocolReceiverInitCommand类中注册需要请求的所有命令。例如：

GameFacade.RegisterCommand(XXXCommand.NAME, typeof(XXXCommand));

1. 注册响应消息回调，在ProtocolReceiverInitCommand类中注册响应服务器返回数据函数回调。例如：

NetworkManager.Register<S2C\_XXX>(XXXController.OnXXXHandle);

1. 注册数据代理类，在ProxyInitCommand类中注册所有的数据代理类，直接new一个代理对象即可。例如：

GameFacade.RegisterProxy(new XXXProxy());

所有的注册相关命令类，都在目录下面Logic/Init目录下面。经过上面注册，功能模块就可以运行起来，当然你不需要担心你使用的其他模块的数据Proxy对象是否被注册，所有数据代理对象都已被注册。

### 命名空间

功能代码命名空间命名都统一在Xsjm命名空间下，大模块功能可以在此命名空间单独加入新的命名空间如：Xsjm.Combat、Xsjm.Guide。

框架代码命名空间主要分为：Xsjm.Base和Xsjm.Core两个模块下，命名方式也是按照大模块可以在此命名空间基础下加入新的命名空间如：Xsjm.Core.Network、Xsjm.Core.Protocol等。

编辑器代码命名空间统一在：Xsjm.Editor命名空间下。

### 注意事项

1. 数据代理必须显示指定构造函数，且继承基类的带String参数构造函数。例如：

public XXXProxy() : base(NAME) {

xxxMap = new *Dictionary*<int, xxxData>();

}

1. Proxy对象注册完毕后，建议在ProxyPool中添加对象静态索引，当然不添加也没问题，也可以用户GameFacade.RetrieveProxy(NAME);来获取已经注册好的代理对象，只是相对比较麻烦一点。
2. 数据代理可以理解为一个数据管理类，管理的数据对象目前放在Tpl模板下面ValueObjects下面，由于一些历史原因，改动相对比较大，建议将这些数据对象统一放到这个目录下面。
3. 应该避免渲染层出现修改、删除、更新数据层的操作，显示做应该是当收到更新指令后，获取数据层数据，用最新的数据更新渲染逻辑。数据的变更操作，应该由控制层来处理，以此来避免数据一致性和数据混乱。
4. 应该避免直接将协议对象作为Notification参数发送至渲染，逻辑上没有问题，为了代码层次的清晰，应该避免这种操作。
5. 应该避免在发送消息地方注册响应回调函数，无论是请求命令还是响应回调函数，或者是数据代理对象，都必须在对应的初始化命令中统一进行注册。