密级：公开

编号：14250599

中山大学

**工程 硕士专业学位论文**

基于Unity3D的自动化构建发布系统

Automation Build System Based on Unity3D

学位申请人： 钱锋

导师姓名及职称： 李中华 副教授

专业、领域名称： 工程硕士（计算机技术）

二〇一七年十一月

**中山大学硕士学位论文**

**基于Unity3D的自动化构建发布系统**

**Automation Build System Based on Unity3D**

**专 业：计算机技术**

**学位申请人：钱锋**

**指导教师：**

论文答辩委员会主席：

成员：

二0一七 年十一 月

**原创性及使用授权声明**

论文原创性声明内容

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：2017 年11月 日

**学位论文使用授权声明**

本人完全了解中山大学有关保留、使用学位论文的规定，即：学校有权保留学位论文并向国家主管部门或其指定机构送交论文的电子版和纸质版；有权将学位论文用于非赢利目的的少量复制并允许论文进入学校图书馆、院系资料室被查阅；有权将学位论文的内容编入有关数据库进行检索；可以采用复印、缩印或其他方法保存学位论文；可以为存在馆际合作关系的兄弟高校用户提供文献传递服务和交换服务。

保密论文保密期满后，适用本声明。

学位论文作者签名：         导师签名：

日期：2017年11月 日  日期：2017年11月 日

基于Unity3D的自动化构建发布系统

**专 业：计算机技术**

**硕士生：钱锋**

**指导教师：李中华**

# 摘 要

Unity3D是一个三维开发引擎，由于其所见即所得的编辑开发环境及跨平台、多语言脚本支持等特性而得到很多开发者的喜爱，目前广泛应用在游戏、动画、建模交互的创作中。Unity3D的编辑器集成度高，并且涵盖整个开发流程，让项目制作的各个人员都可以参与其中，大部分工作也都会在编辑器上进行。不过其自带的发布功能比较简单，在资源组织和平台兼容上隐含了不少细节，故在实际针对各个平台发布工程时往往需要繁琐的步骤。另外，其打包处理资源耗时、不同平台需求环境不一等等问题也让整个发布过程变得漫长和容易出错。

本文通过分析基于Unity3D开发的游戏项目所经历的构建发布流程，提出自动化构建发布需求，对Unity3D、网页开发、版本管理等相关技术进行研究。随后，进行详细设计，描绘整体框架，结合使用Python、C#、PHP、JavaScript语言实现了采用网页交互，在独立的计算机上进行构建发布的系统。针对程序异常处理，图片智能合并，着色器多平台编译优化等难点进行详细探讨与细致说明。

此系统支持自定义配置和平台差异化处理，对项目的需求处理、资源检测、资源打包、脚本编译、集成发布一系列任务进行全自动化操作，为项目持续集成、迭代开发提供良好支持。在实际应用中获得良好反馈并极大地促进了产品测试、发布效率。

关键词：Unity3D，网页交互，版本管理，构建，发布

Automation Build System Based on Unity3D

**Major：**Computer Technology

**Name：**Qian Feng

**Supervisor：Li Zhong Hua**

# **ABSTRACT**

Unity3D is a three-dimensional development engine, because of it’s WYSIWYG editing environment and cross-platform, multi-language scripting support features, it is widely used in the creation of games, animation, modeling interaction. The Unity3D Editor is highly integrated and covers the entire development process so that everyone in the project can work on the editor. But it’s own build and release function is relatively simple, and it’s resource organization and platform compatibility implies a lot of details, so in the actual release for the various platforms often need complex steps. In addition, the resource packing process is time consuming, different platforms require different environments, because of these issues, the entire process is prone to error.

This paper analyzes the Unity3D-based game project build release process, put forward the need of automation build system, elaborates the related technologies such as Unity3D, web development and version management. Use Python, C#, PHP and JavaScript languages to design and implement a system that is built on a separate computer. This paper discusses the issues of program exception handling, image merging, and shader multi-platform compiler processing.

This system supports configuration and platform differentiation processing, and fully automates the project demand processing, resource detection, resource packing, script compilation and integrated release, it provides good support for project integration and iterative development. It has greatly promoted the product testing and release efficiency in practice.

**Key Words**：Unity3D, web interaction, version management, build, release

目录

[摘 要 I](#_Toc2584)

[ABSTRACT II](#_Toc28729)

[第一章 绪论 1](#_Toc30935)

[1.1 Unity3D引擎 1](#_Toc16432)

[1.2 应用平台介绍 5](#_Toc20474)

[1.3 迭代式开发需求 7](#_Toc4814)

[1.4 Unity3D自动化构建发布系统 8](#_Toc18608)

[1.5 论文目录与组织 9](#_Toc27933)

[1.6 小结 10](#_Toc10760)

[第二章 自动化构建发布系统需求 11](#_Toc16222)

[2.1 Unity3D资源与构建发布相关API 11](#_Toc31496)

[2.2 网页服务器 12](#_Toc30011)

[2.3 Python语言 13](#_Toc24190)

[2.4 Unity3D构建发布相关知识 14](#_Toc15566)

[2.5 数据配置存储 17](#_Toc11905)

[2.6 小结 17](#_Toc25267)

[第三章 框架设计 18](#_Toc23728)

[3.1 网页服务模块 18](#_Toc21425)

[3.2 任务服务模块 22](#_Toc1458)

[3.3 小结 29](#_Toc6350)

[第四章 网页服务模块实现 30](#_Toc22928)

[4.1 网页服务环境配置 30](#_Toc2236)

[4.2 文件与代码组织 32](#_Toc18381)

[4.3 交换网页实现 33](#_Toc522)

[4.4 网页更新支持 38](#_Toc5494)

[4.5 小结 39](#_Toc23418)

[第五章 任务服务模块实现 40](#_Toc29823)

[5.1 文件与代码组织 40](#_Toc25709)

[5.2 任务驱动逻辑 45](#_Toc12065)

[5.3 关键任务逻辑 52](#_Toc23873)

[5.4 难点及注意事项 58](#_Toc20248)

[5.5 小结 59](#_Toc20505)

[第六章 项目应用 60](#_Toc10233)

[6.1 项目应用的其它配置 60](#_Toc13413)

[6.2 项目应用情况介绍 60](#_Toc9452)

[6.3 小结 64](#_Toc23585)

[第七章 总结与展望 65](#_Toc32150)

[参考文献 66](#_Toc24491)

[致谢 68](#_Toc21241)

# 绪论

## 1.1 Unity3D引擎

Unity3D是一个三维开发创作引擎，支持发布可运行在各个主流平台的应用（如Windows、Linux、macOS等桌面平台，Android、iOS等手机平台）。Unity3D最出色的功能特性就是其集开发与运行于一体的编辑器，开发人员可以快速的在此编辑器上布置资源，编写脚本，运行测试效果。

### 1.1.1 Unity3D引擎编辑器常用窗口

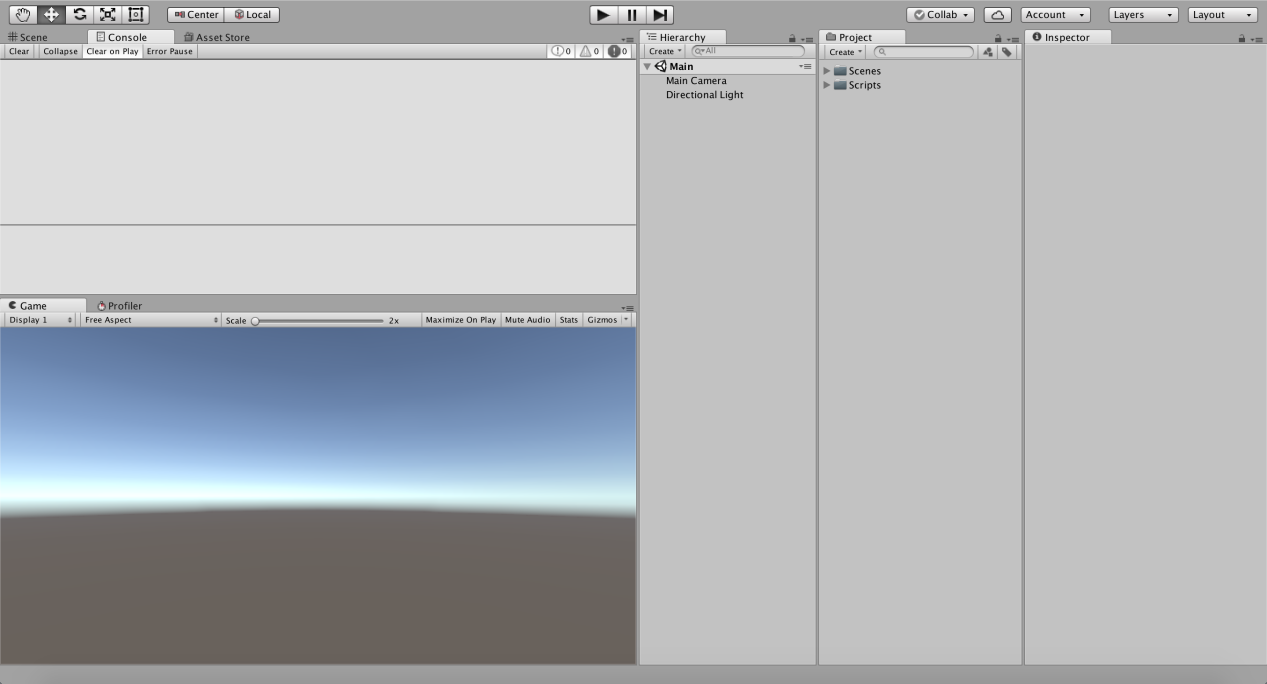


图1-1 Unity3D编辑器

Project（项目资源窗口）类似资源管理器，项目所有文件的组织在此处展现。管理的常见资源包括贴图、着色器、材质、三维模型、模型动作数据、预设、脚本等等，窗口上方有搜索栏可以根据输入的名称进行资源检索，方便查找相关资源。通过此窗口可以对资源进行诸如导入添加、改名调整以及删除操作，其中的导入是非常重要的一环，Unity3D会在资源导入时进行各类初始化操作，并对应生成同文件名但以meta结尾的文件(元文件，记录资源唯一标识以及相关属性设置)，同时会在工程目录下Library文件夹内生成相关资源缓存（Library文件夹不存在时会进行创建，资源缓存即是对资源格式参照当前设置的平台转换调整为Unity3D更易操作的文件）。

Hierarchy（层级窗口）展示当前场景及场景内存在的各类对象，其层次结构对应场景中各类对象的从属及前后顺序关系。调整对象的层级结构均会在场景中相应的反映出来，此外它也具备搜索栏搜索其下对象的功能。

Scene（场景窗口）将层级窗口中的所有对象进行实际绘制表现出来，是整个编辑器体现所见即所得的主要窗口。在场景窗口中，当前场景就同真实世界一样呈现给使用者，同时也会具象表现出各种可编辑属性，可方便调整查看视角以及改变各类对象的位置、大小。层级窗口中进行对象搜索时，场景窗口中也会对应将满足搜索条件的对象进行高亮表现，与层级窗口配合发挥功能。

Inspector（组件参数编辑窗口）显示具体对象挂载的组件和组件的各类属性数据，当在上述几个窗口里选中对象时，其就会激活显示对应对象的信息。其常用的操作是激活或关闭对象以此进行对比、查看。

Game(预览窗口)可以预览到项目最终效果，配合工具栏上的播放、暂停、逐帧播放三个按钮完成响应操作。同时由于Unity3D面向多平台，其分辨率设置便需要进行各种情况下的适配，预览窗口可以方便调整当前分辨率实时查看当前项目是否表现正常，而状态按钮会显示运行时的各个主要参数信息，帮助开发人员获知状态，查找定位问题。

Console（控制台）是Unity3D引擎中用于观察及调试脚本运行状态，输出各类日志信息的窗口。在实际开发中，经常会自定义输出信息来处理问题，同时支持对显示日志字体大小、颜色控制，方便进行定制。

Unity3D常用窗口大致介绍如上，此外还有像光照、动作编辑、帧调试等功能窗口，都是根据自身特点进行相应布局，力求直观的可视化操作[[[1]](#endnote-0)]。

### 1.1.2 Unity3D脚本开发集成环境

Unity3D引擎本身基于C++构建，为了便于上层开发，引入Mono虚拟机并基于Mono虚拟机进行脚本驱动，支持C#/JavaScript等语言开发[[[2]](#endnote-1)]。目前还可同时混合使用上述语言，为开发者提供多种选择和便利，也大大扩展了受众面，让各类开发人员可以选择自己熟悉或易学的语言进行开发。所以往往可以看到不少美术相关从业人员直接进行Unity3D的脚本编写。

Mono是由Xamarin公司为创建一系列匹配ECMA标准的.Net工具而主持的一个自由开源项目，可运行在各个平台之上[[[3]](#endnote-2)]。MonoDevelop脚本编辑器也是一个开源项目，可以在主流平台上运行编写Mono软件[[[4]](#endnote-3)]。Unity3D引入MonoDevelop作为默认脚本编辑器，同时在新的版本中支持微软的Visual Studio开发工具集，为开发者提供多种代码编辑器选择。

Unity3D目前大部分版本使用的Mono都是比较早期的版本（2系列版本），其自身编译时对部分C#语法编写的代码会产生内存分配请求，所以在编写时需要注意部分语法（如迭代foreach语句处理非数组的容器对象时，老的Mono版本会申请大量的临时变量），从测试反馈表明，使用新的Mono或Visual Studio的编译工具进行编译可得到优化的版本，其内部相关逻辑不产生或者产生较少内存分配请求。开发人员可以从编写或编译这两个方面对代码进行优化处理。

### 1.1.3 Unity3D历史版本及遗留问题

Unity3D第一个版本发布于2005年[[[5]](#endnote-4)], 随着版本逐步升级，支持平台越来越多，特性越来越丰富，但也留下不少问题。早期Unity3D主要固定在Resources文件夹下使用所有项目资源，统一的Resources操作接口简单易用，但是在发布时整个Resources文件夹会被其内部处理并针对各种平台进行适配，造成开发者丢失对资源的控制权，不能更改、调整资源。同时，随着不少应用项目需求增大，资源量也会越来越多，这种整体Resources管理模式就会在资源打包和整合时耗费大量时间，另外由于引擎启动时会创建Resources索引表，过多的资源也会造成索引表变大，占用更多内存，影响程序启动时间，增加耗时。后面Unity3D引入AssetBundle模块来处理这些问题[[[6]](#endnote-5)]，不过整个模块是全新的一套加载使用机制，并且隐含资源依赖、卸载等注意规则，同时将内部针对各种平台的适配变为外部自行调用打包，形成一个学习和使用的陡坡式曲线，也因此造成开发和实际发布对资源使用的不统一。

Resources模式开发期的简单易用与发布期的不能更新及效率低下，对比AssetBundle模式开发期复杂耗时与发布期的可更新及良好效率，往往让开发人员为了发布的可维护性而舍弃Resources模式，这种做法也是目前Unity3D官方建议。不过这样也就损失了Unity3D开发中的一大便利特性，特别对于非技术人员在处理美术资源时增加不少繁琐步骤，往往在增改资源时需要进行资源打包的操作以及相关配置生成后才能看到实际展示效果，不能专注于资源调整本身。

此外，Unity3D不少API接口及各平台处理方式也在版本迭代中调整变换，如为应对iOS强制要求64位编译需求，引入IL2CPP机制，内部将C#转换为CPP文件，而Android平台继续沿用C#脚本模式，造成了平台相关处理存在较大的不同。这些都需要开发者在发布时针对不同平台进行适配调整，项目在Android平台发布运行顺利，在iOS平台下却不能正常运行的例子比比皆是。

### 1.1.4 Unity3D自身发布功能

Unity3D自身发的布功能非常简洁，切换到需要发布平台后，选择构建需要的场景，并对是否需要发布支持开发模式等选项进行勾选后，点击Build按钮即可进行构建发布。

不过实际开发中，我们会发现切换平台极其耗费时间。因为Unity3D会在切换平台时对当前所有资源进行针对对应平台设定格式并进行转换（项目资源窗口介绍提及的工程下的Library文件夹，即是用来缓存这些具体格式资源），会涉及到图片纹理格式、音频、视频格式转换。这当中如果开发者又存在对部分图片设置使用了高质量压缩的设定，则转换效率开销更会雪上加霜。以笔者参与的几个游戏项目来看，平均下来使用的原始资源每100M需耗费转换时间10分钟左右，而这几个项目使用的原始资源都是在700M以上，所以在切换Android、iOS等手机平台时均是需要等待一个多小时的时间。

切换平台后，开发者还需针对不同平台进行资源适配性调整，开启对应平台资源打包的任务，又会耗费不少时间，其主要就是分析依赖、对资源进行压缩封装消耗巨大。处理完打包任务后还需部署打包后的资源，再进行最终Unity3D的打包构建环节。

Unity3D开发商也意识到这个问题，推出了Cache Server[[[7]](#endnote-6)]（缓存服务）功能。此缓存服务可以部署在本地或者服务器之上，内部机制大致是在第一次处理指定平台资源时，将生成的内部格式资源保存起来，完整地切换某个平台并运行资源转换后，其对应平台的资源就会被保存起来，同时更新或新增资源时会对相关资源重新进行缓存，这样后面再进行此平台切换时就可以快速地从保存的资源中获取，传输给发出请求的Unity3D编辑器，达到节省转换生成时间的目的。由此也可得知Unity3D的缓存服务功能需要占用大量存储空间，并且运行服务的计算机也需要较高的运算处理性能，官方文档也建议部署在高配置、大存储空间、具备高速网络的远程计算机上。

不过如此处理，目前也只是解决了切换平台资源重新缓存的问题，适配调整以及打包等步骤仍然未得到好的解决。

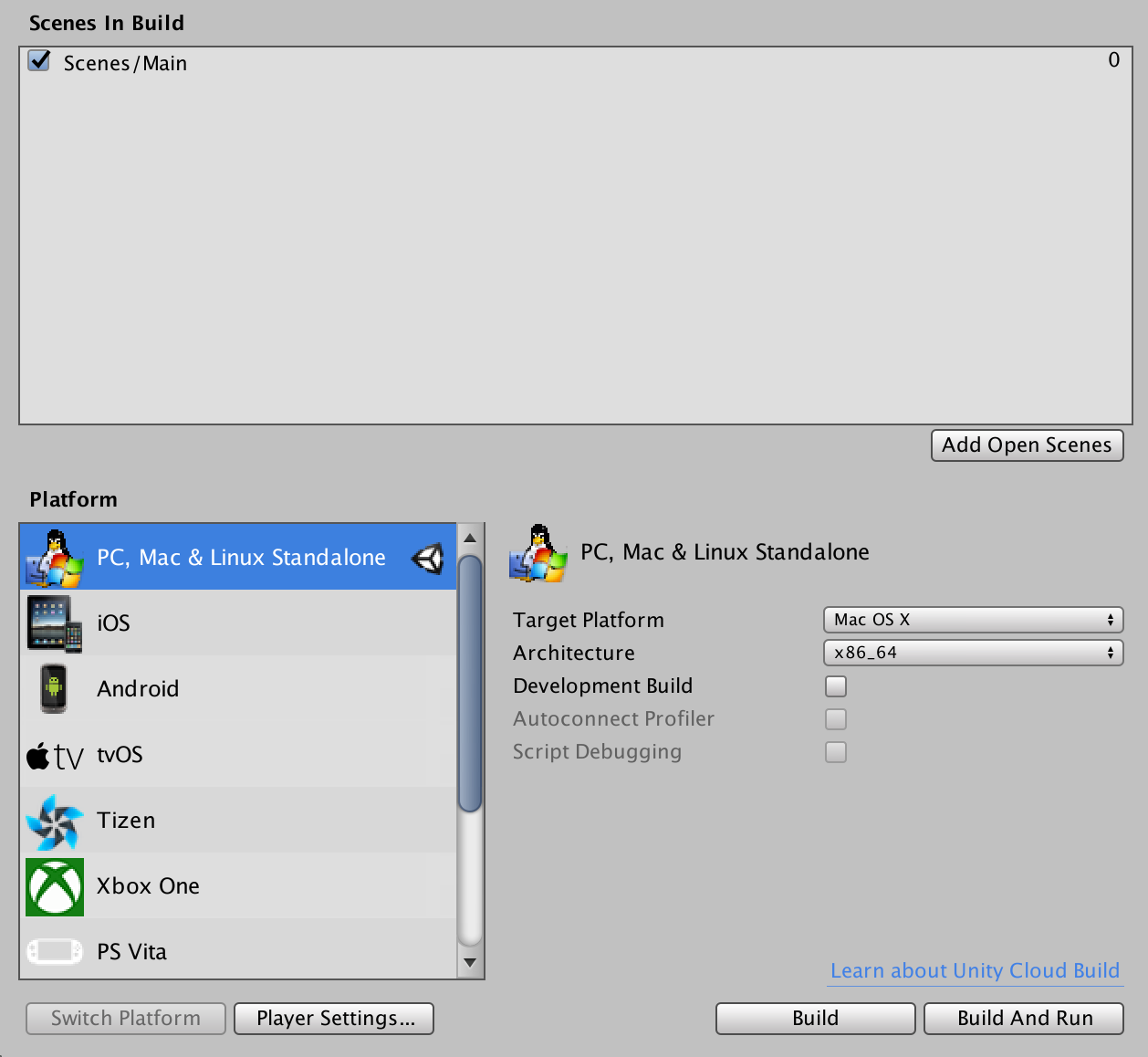


图1-2 Unity3D 发布窗口

## 1.2 应用平台介绍

Unity3D支持发布十几个平台的应用，这里主要针对应用广泛的桌面平台以及手机平台做相关描述，了解Unity3D的运行环境及涉及到的平台调整、适应问题。

### 1.2.1 桌面平台

桌面平台主要是微软的Windows平台与苹果的macOS平台，也是Unity3D编辑器主要运行的平台。Windows平台下Unity3D即可方便构建其它平台运行所需的资源以及除苹果的macOS及iOS平台之外的发行版本。同时借助微软自身的Visual Studio开发工具集和低门槛发布运行程序需求，Windows平台具有Unity3D开发制作的不少便利优势。苹果的macOS平台可以发布自身macOS与iOS平台版本，其它方面与Windows平台相近，不过会存在更严格的安全证书需求。随着三方辅助工具的逐渐增多、增强，苹果的macOS平台也与Windows平台在开发制作的便利性上不断缩小差距。

桌面平台的开发主要以Unity3D脚本为主，与系统扩展交互相关的通过C++也可比较轻松地扩展实现（由于桌面系统应用形式比较统一，大部分基于Unity3D开发的项目通过Unity3D自身提供的API与脚本支持即可，扩展需求比较少）。

这两大桌面平台基于的硬件环境相近，Unity3D应用发布流程也基本类似，使用Unity3D提供的统一系统路径变量即可平滑处理其在文件存储系统上的差异。

### 1.2.2 手机平台

手机平台当前主要是Android与iOS平台，其中Android由Google为主成立的OHA（Open Handset Alliance,开放手机联盟）开发、维护，凭借其开源特性，获得大量使用、迅速普及，如今已经成为市场占有率第一的手机开发平台[[[8]](#endnote-7)][[[9]](#endnote-8)]。iOS运行在苹果公司的系列手机及平板之上，是一个封闭的系统，但依靠大量的终端销量也成为目前仅次于Android的手机开发平台[[[10]](#endnote-9)]。

手机平台特点主要由硬件体现，手机是由电池供电的便携式设备，大部分手机均是具备一块支持触控的屏幕，同时根据具体硬件设备的不同，往往还具备旋转、定位等等特性。所以相应应用开发时需要考虑输入、性能、三方特性支持等问题。

Android平台进行扩展主要通过Java语言开发实现，iOS平台进行扩展则主要通过Objective-C语言与Swift语言开发实现，Unity3D分别进行了相关适配支持。

上述这些特点决定当前Unity3D主流开发模式是基于桌面平台开发，部分模拟手机平台特性，最终需要在手机平台上运行验证。

另外，在手机平台的应用发布、安装方面，Android平台由于开放的原因，其自身版本及三方自定义调整版本众多，提供应用下载的应用市场更是数量庞大。项目开发完毕进行发布，需要面临繁多的版本与设备兼容性测试以及各个应用市场的接入。对比之下，iOS平台由于只运行在苹果的硬件设备之下，其版本及设备类型少、兼容性高，并且应用统一由苹果的应用商店发布，项目开发发布所面临的测试与接入任务相对要轻松不少。

## 1.3 迭代式开发需求

使用Unity3D开发的项目，基本涉及图片、三维模型、模型动作数据、音视频的资源管理、使用。尽管可以直接在编辑器中运行查看工程功能、效果，但是平台的差异化让项目相关人员仍然需要在各个平台上发布运行版本进行体验、复核。这个在涉及到需要发布手机平台的项目中会更为重要，手机平台的处理速度与显示屏幕及操作方式与桌面平台存在极大的不同，所以可以在项目开发过程中就能方便及时的在手机上运行是项目各个开发、管理人员的迫切需求，这样也是更好地满足迭代式开发。

迭代式开发将整个项目的开发工作进行划分组织，形成很多个微小项目，每一个小项目均涉及到需求的分析设计，系统实现及测试一系列完整过程，完成这个过程即为一次迭代，整个项目开发工作即变为一系列迭代的过程，可以使项目具有更高的成功率和生产率[[[11]](#endnote-10)[[[12]](#endnote-11)]。

整个过程同时也让项目人员更贴近用户视角，可进一步推进项目需求完善和开发进展。

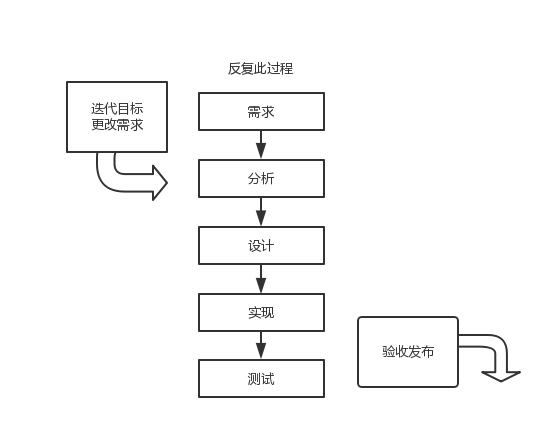


图1-3 整合需求与系统测试的迭代示意图

支持所见即所得，运行时即可调整查看应用功能的Unity3D编辑器为开发带来很大便利，也很好契合了迭代式开发的需求，不过在发布这块因上述种种原因形成短板，成为开发中一个痛点。

## 1.4 Unity3D自动化构建发布系统

### 1.4.1 自动化构建需求

笔者有幸较早接触Unity3D并应用在实际项目开发中，期间经历了直接使用Unity3D自带构建发布，打包资源和手动整理构建发布，辅助配合脚本与批处理整理构建发布等等阶段，也参与了部分公司、学校Unity3D相关培训、应用之中，和自身经历类似，发布构建是开发人员接触了解较为薄弱的一环，大部分均是手动到半自动处理构建发布，同时需求专门相关人员进行处理。一个常见的场景是：指定一个开发人员专门负责打包发布相关操作，而这其中部分相关开发人员会借助Unity3D的脚本对部分处理进行封装处理，简化打包发布步骤，不过仍时常碰到由于新资源、新规则加入而需改动流程的问题，此外项目成员各种不定时的打包测试需求也挤占大量工作时间。

使用Unity3D开发的项目涉及人员比较多，文案策划、美术设计、程序实现、核查测试、上线运营等等流程复杂。相关人员知识结构往往存在比较大的偏差，但是对项目、产品都存在实时查看，了解进度的诉求。同时由于上面所述Unity3D自身的一些要求与注意事项，对当前项目方便快捷，依赖要求低的构建发布功能是广大Unity3D项目人员的心声。

### 1.4.2 解决方案

如何快速、自动化的完成资源打包和项目工程发布，本论文即针对此提供思路和方法，对自动化发布系统进行探讨和实现。

此系统采用单独的计算机，通过提供网页访问服务，接受打包、制作安装文件等指令进行对应任务处理，并输出结果信息提供反馈，做到轻量部署，快捷使用，将繁琐、重复的任务完全自动化解决处理。

采用网页的方式主要目的就是降低对使用人员本地环境需求，简单的通过浏览器访问和相关点击操作即可提交任务需求，对非技术开发人员提供便利。单独的计算机可以统一配置软硬件环境，提供针对性的优化处理（如使用固态硬盘，关闭文件检索以及避免Unity3D多版本的干扰），部属连接网络提供远程访问、控制。整个系统操作使用力求简单，快速完成发布。

## 1.5 论文目录与组织

论文首先通过第一章对Unity3D引擎及其编辑器做了大概介绍，并且指出在实际开发构建发布时所面临的问题，提出自动化构建发布系统的需求。第二章针对此构建发布系统进行分析及对相关知识点进行阐述。第三章进行系统详细的设计，描绘整体框架。第四章、第五章依托前面章节所述知识点与整体框架设计，对整个系统进行了实现，特别针对关键点进行细致说明，确保读者可完整掌握自动化构建发布系统的实现流程。论文第六章展示实际的项目应用及其对项目开发带来的帮助与收益，随后第七章进行总结和后续展望，和读者共同探讨可改进增强的方案。

## 1.6 小结

本章介绍了Unity3D开发引擎，涉及常用开发界面及其所支持的开发功能。其中特别针对Unity3D的资源管理与编译发布进行了说明，结合实际使用开发经验，陈述使用Unity3D进行构建发布中的痛点，对使用Unity3D开发的项目的构建发布需求进行分析，引出自动化构建发布系统。

# 自动化构建发布系统需求

## 2.1 Unity3D资源与构建发布相关API

Unity3D对资源打包与项目构建经过多个版本调整，使用方式多样、多变，这里主要介绍5系列版本中相关API，Unity3D 在5系列版本中，打包构建已经简化为BuildPipeline类的两个静态方法处理，两个静态方法均存在不同参数的重载，下面列出主要使用的重载接口。

表2-1 UnityEditor.BuildPipeline.BuildAssetBundles API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | public static AssetBundleManifest BuildAssetBundles(string outputPath, AssetBundleBuild[] builds, BuildAssetBundleOptions assetBundleOptions, BuildTarget targetPlatform) | | |
| 功能简述 | 根据指定参数表打包构建所有AssetBundle资源 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | outputPath | String 字符串 | AssetBundles输出路径 |
| AssetBundleBuild | UnityEditor.AssetBundleBuild 结构数组 | AssetBundle打包信息（需要打包的资源名与打包生成的资源名） |
| assetBundleOptions | UnityEditor.BuildAssetBundleOptions 枚举量 | 指定打包操作类型（如不进行压缩，对打包资源进行哈希确定等等） |
| targetPlatform | UnityEditor.BuildTarget枚举量 | Unity3D设置的平台枚举类型 |
| 输出项 | 无 | AssetBundleManifest | 打包生成资源的所有信息集合 |
| 备注 | 内部自动处理依赖关系，需分析处理返回信息 | | |

资源依赖一般发生在集合多个组件的预设上面，如果多个预设均使用了同样的贴图等基础资源，在打包成AssetBundle时不处理好资源依赖，则会造成这些使用了同样资源的预设在打包生成AssetBundle时均包含一份冗余副本，进而影响占用存储空间和运行内存。一般有两种方式避免:一种为通过资源依赖处理将贴图这类基础资源单独打包并设定相关预设是依赖此单独打包资源；另一种则是直接指定多个预设中某个预设为基础资源而其它预设依赖此基础资源预设进行打包。依赖处理也是目前进行Unity3D开发时需要重点防范的步骤。

在Unity3D之前的版本（例如4系列版本）需要手动处理资源依赖关系，通过其给定类似堆栈进出的接口调用来设定资源之间的依赖。而在新版本（5系列及之后版本）中，使用上面所列静态方法已经可以在内部进行自动分析依赖关系智能打包，不过要注意的是如果在未对贴图、三维模型这些基础资源设定打包输出路径的情况下，仍会造成资源打包冗余问题。

表2-2 UnityEditor.BuildPipeline.BuildPlayer API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | public static string BuildPlayer(string[] levels, string locationPathName, BuildTarget target, BuildOptions options) | | |
| 功能简述 | 构建整个项目工程 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | levels | String 字符串数组 | 需求包含至构建工程的场景名称列表 |
| locationPathName | String 字符串 | 构建应用输出路径 |
| target | UnityEditor.BuildTarget枚举量 | Unity3D设置的平台枚举类型 |
| options | UnityEditor.BuildOptions枚举量 | 构建类型（如不做特殊处理，编译支持调试的版本） |
| 输出项 | 无 | String 字符串 | 构建时如果产生错误则指明错误信息 |
| 备注 | 无 | | |

构建发布应用的接口中的构建类型主要用来控制构建的应用中是否包含调试信息，以此支持调试，这个在开发期是一个常用的设置，通过设置此类型构建出来的应用可以方便的连接Unity3D的调试功能窗口，反馈当前应用的性能热点和输出更多调试信息。

## 2.2 网页服务器

网页交互方式的选择为使用带来方便，也为系统引入了网页服务需求，即需要在系统上部署网页服务器。

这里的网络服务器是指提供从网络接受HTTP请求，然后提供HTTP回复给请求者的服务处理。目前Apache、IIS、Nginx是目前互联网比较常用的几种网页服务器[[[13]](#endnote-12)]。

其中Apache全称为Apache HTTP Server，它和它的各种分支版本长期活跃在互联网市场上，其代码开源，支持多平台，并且大部分功能可以进行定制。其特性大部分通过模块扩展实现[[[14]](#endnote-13)]，Apache自身定义了一套模块注册、装载、卸载的机制，像权限验证、URL重写、PHP（超文本预处理器）语言均是使用这种方式与Apache结合使用。其中PHP语言是目前最常与Apache配合实现网页控制操作[[[15]](#endnote-14)]的模块。

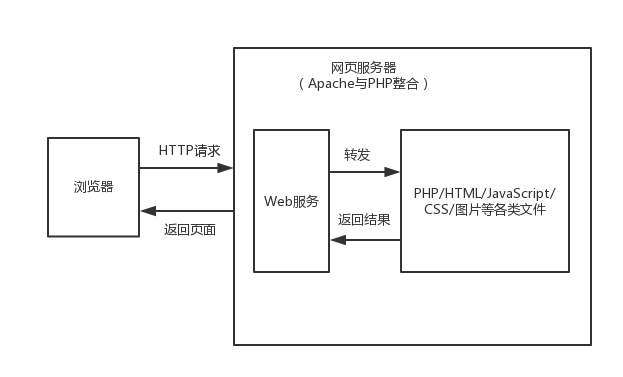


图2-1 Apache与PHP构建网页服务器

本自动化构建系统选用Apache与PHP的组合来提供网页服务正是考虑到这些特点，基于当前成熟的安装配置方案和广泛使用基础，可以快速部署和实现所需网页交互的功能。

## 2.3 Python语言

完成自动化构建各种任务的具体执行涉及到版本管理，文件文本操作，自定义配置解析，图片处理，Unity3D调用与执行，C#编译，以及其它相关工具配合等等，所以需要一个强大的’胶水语言’将这一系列流程串联起来，同时还能支持后续各类扩展需求。本文选择易用性强，配合三方库可提供丰富功能的Python语言。

Python语言由Guido van Rossum在二十世纪八十年代末设计推出，具有解释性、交互性、面向对象的特点。Python语言开发时的使用形式上往往倾向于直观易懂的简明表达。在Python的解释器中输入 “import this” 会打印输出“Python格言”， 内容正是体现这种思想精神[[[16]](#endnote-15)]。整个系统的实现过程中也在尽力学习和贯彻这种思想，以此处理自动化构建发布系统中各种繁复过程。

Python同大多数的脚本语言一样，内存管理交由其解释器负责[[[17]](#endnote-16)]，这样开发人员可以从内存事物中解放出来而专注于最直接的目标，提升效率。

Python具有异常处理机制，标准的Python库中的每个模块都使用了异常，可以说异常在Python中无处不在（如使用不存在的字典关键字时会引发KeyError异常、调用不存在的方法时会引发AttributeError异常）[[[18]](#endnote-17)]。异常如果不进行捕获处理，整个Python程序就会因此崩溃退出，但这也正是异常的设计的目的之一，让问题更早出现，正确的捕获处理异常可以实时反馈错误信息，也让程序运行的更稳健。作为长时间提供服务的应用程序则更需要注意对异常的处理。

同Untiy3D有些类似，Python经过长时间发展，目前有两个主流版本（2系列版本与3系列版本），两个版本在语法上有些许区别，其中自2.6开始的版本（目前主要是2.6，2.7）会在一定程度上支持和兼容Python 3系列版本的语法[[[19]](#endnote-18)]。本文主要采用Python 3系列版本进行开发。

此外上面所提到Python解释器自身虽然具备内存管理，但是仍得留意编写过程中是否存在一直申请内存资源并保存未释放的情况，因为这样会造成Python解释器不能确定此块内存数据已经不再使用而可以进行垃圾回收，进而使内存占用越来越高引发崩溃。处理这类问题可以借助三方库：Python Object Graphs(包名为objgraph)[[[20]](#endnote-19)]，它可以显示各个对象引用关系以及统计程序运行时增加的对象。像每次执行任务完毕后调用其中的show\_growth方法便可方便的知道是否产生了新的对象未释放，辅助处理内存泄露问题。

## 2.4 Unity3D构建发布相关知识

网页服务器以及Python语言工具的选择确定了自动化构建发布系统的主体开发所使用的技术，不过内部流程涉及的任务广泛、多样，分析和处理时需要用到计算机领域的各类相关知识，这里将其中重要的几个知识点进行整理，描述。

### 2.4.1 字符编码

字符编码指各个字符对应的编码方式，其种类繁多，不同系统、软件、文本存储使用的字符编码都有可能不同。字符编码的处理也是实际开发中容易忽略的操作，往往会引起让开发者困惑的问题。

从兼容性及通用性考虑，推荐在编写处理字符相关逻辑时统一采用UTF-8(8-bit Unicode Transformation Format)编码格式，涉及到的任何其它编码都在获取和输出的地方做对应转换工作，保证内部一直为UTF-8编码处理状态。这样只需在各个交互处理的地方留意编码转换问题，而不会在具体逻辑使用和处理中造成混乱。

### 2.4.2 JavaScript

JavaScript也是一种脚本语言，由网景(Netscape)公司创建，大部分使用在Web开发上。目前所有的现代Web浏览器软件都包含JavaScript解释器，通过JavaScript可以操作页面文档（document），控制网页的表现形式和数据内容，使得静态的HTML文档变为交互式的Web应用。此外，作为HTML文档视觉表现标准的层叠样式表（Cascading Style Sheet, CSS）也可由JavaScript进行脚本化设置[[[21]](#endnote-20)]。

jQuery是开源的JavaScript库，封装了操作页面文档的各个步骤，可以方便简单的以对象的方式管理操作页面文档。jQuery在内部还针对各类浏览器及其不同版本对网页解析的差别进行适配，上层使用者可以不用担心网页显示兼容性问题。开源、跨浏览器、轻量、插件扩展这些特性让jQuery被广泛应用[[[22]](#endnote-21)]。

### 2.4.3 版本管理系统SVN

版本管理系统SVN实际名称为Apache Subversion，是Apache Software Foundation下的一个项目[[[23]](#endnote-22)]。它采用仓库与工作副本的形式管理控制文件版本。大概流程为：一开始创建空的仓库，并由此仓库地址检出内容至工作副本，在工作副本中进行增加、删除、改动等文件操作，每次处理完毕后进行提交，即对应一个版本，仓库会记录版本之间的所有区别，并支持指定版本完整检出对应文件。一个仓库可以对应多个工作副本，多个工作副本通过更新、融合处理、提交来协同合作。

由于上述的特点，SVN经常被用来管理项目开发。对于自动化构建发布系统，主要使用它完成资源同步、提交、还原。

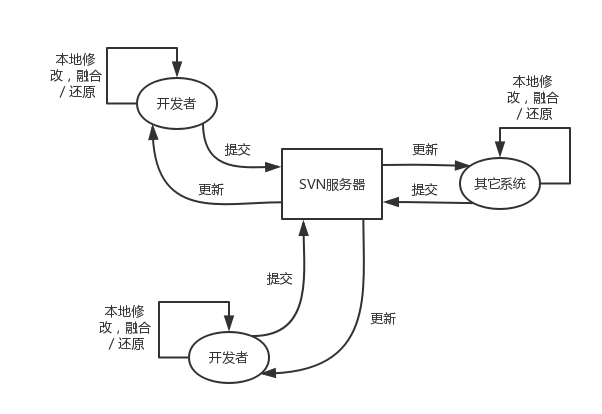


图2-2 SVN协作流程示意图

### 2.4.4 纹理压缩

纹理压缩是图像压缩的一种，不过与通常的图像压缩不同，纹理压缩针对渲染应用做了对随机获取纹理像素数据的优化，同时对解压速度提出更高要求，但在质量与压缩方面则偏向于使用有损处理以获得更高的压缩率，另外，由于大部分情况都是使用压缩后的纹理数据，对纹理压缩往往只在制作时进行，故对压缩编码的速度要求不高[[[24]](#endnote-23)]。目前常用的有S3TC(S3 texture compression)、ETC（Ericsson texture compression）、PVRTC（PowerVR texture compression）纹理压缩格式，这些格式各自包含具体的细分格式（如ETC中就有ETC1和ETC2格式）和对应的压缩工具，相应程序开源可进行调整。

使用Unity3D进行项目开发少不了要处理纹理贴图，而其中纹理压缩便是非常重要的一环，特别针对移动平台，选择合适的纹理压缩格式可以让内存显存占用与渲染性能效率处于良好的平衡点。Unity3D自身集成多种纹理压缩格式，支持自动判断选用格式的处理，不过实际开发中仍然需要对特定的纹理贴图做格式调整。另外，使用特定的压缩处理方法可以获取更好的质量表现。

在移动平台，一个常见的情景是部分手机只支持ETC1格式而不支持ETC2格式，而ETC1格式只支持非透明贴图，所以为了使用纹理压缩保证效率的一种处理方式是将透明贴图转换为不透明的贴图和存储透明通道的贴图（将透明信息存放在任意一个颜色通道中），然后使用ETC1格式压缩，在实际使用中配合指定着色器使这两张纹理贴图还原出完整含透明通道的贴图数据用于渲染。

## 2.5 数据配置存储

XML作为可扩展标记语言可以方便的由各类程序处理[[[25]](#endnote-24)]。与HTML不同，XML作为元标记语言可以自行定义与需求相关语义的标记，而不是固定的。这样大大增大其扩展性和灵活性。

自动化构建系统正是存在大量特定任务和配置处理需求，使用XML可以进行良好处理。同时由于XML的一个设计目标即是要相关处理代码容易编写[[[26]](#endnote-25)]，所以基本大部分的程序语言都由对XML读写支持的相关处理库，而不需求额外构建程序逻辑。

配合本地文件存储信息，基本可为自动化构建发布系统提供简单、方便的数据支持。

## 2.6 小结

本章对自动化构建发布系统所涉及的相关技术进行了大致的阐述，结合以往人工、半人工发布的工作经验及实际项目开发、构建发布涉及的具体问题，进行了相关技术选型，对大部分重要的知识点进行了了解和学习，以便顺利推进系统的设计与实现。

# 框架设计

## 3.1 网页服务模块

网页服务模块主要完成与用户交互获取任务及将任务递交、通知处理的功能，网页需要呈现接受任务的表单和任务的状态信息。任务划分为三种状态：等待执行，正在执行，执行完毕。

网页服务天然具备并发特性，面对可能同时产生的用户任务请求，对应也有两种处理方式：并发处理，队列处理。并发处理一般是通过线程或进程的方式同时处理多个任务[[[27]](#endnote-26)]，可充分利用多核处理器等硬件性能，不过也需要处理其中的公有量，做好锁[[[28]](#endnote-27)]处理。像在大部分Unity3D项目的开发中，就有不少并发机制的应用：资源加载服务采取同时进行几个加载任务；网络收发通信使用多线程处理；多个对象的动画计算分工作线程处理等等。队列处理则是通过将待办任务放置在一个队列中（通常可按照创建时间排序），然后依次取出任务处理。

由于自动化构建发布涉及的任务基本都涉及到对资源文件的处理，资源文件会成为多数任务的公有量，处理数量多、耗时长，如果使用并行处理，这些地方就都需要加锁处理，相关并行任务会在这些加锁的地方进行阻塞等待，这样基本和队列处理效率相差不多，其并行运行提升有限，而且因此还会带来多创建的线程、进程的开销。

另外，Unity3D针对一个项目会使用一个编辑器进程进行独占处理，本身也不能对一个项目工程同时启动多个编译构建任务。在实际使用中，一种可行的处理方式是复制多份项目工程处理，不过这样占用大量存储空间，并且也会引入新的多副本项目同步的问题。

综上所述，自动化构建发布系统的任务处理采用队列方式，把多用户多任务的请求加入队列然后依次处理。同时出于对可查询、存档、容灾的考虑，采取任务对应文件，状态对应文件夹的方式，文件名包含时间、任务名相关描述，文件后缀表明状态，同时文件内容记录更详细信息。这样通过写入存储系统来保障相关任务及状态记录。在面向所有项目成员使用和存在多人同时操作、取消等情况下，用户下达的任务变为文件依次在等待执行任务文件夹、正在执行任务文件夹、已完成任务文件夹中传递，文件处理通过队列机制处理，在各个阶段改变文件命名，文件内容，最终将文件移动存储在已完成任务文件夹中，如此循环推动流程。即使在各种突发状态下（程序崩溃、断电等等）也可以借助操作系统层面对文件处理的稳定机制来保证当前执行状态，快速恢复运行。文件与文件夹这个处理对比数据库在事务[[[29]](#endnote-28)]的保证下进行信息记录的优点是简单，并可直观查看，也可以更轻松配合各个脚本进行逻辑处理（各类脚本语言基本都会集成文件相关的操作库）。

明确主要工作流程及采取的基本方案特点，结合实际项目开发中的需要，下面分别从交互、管理、辅助三个方面进行网页服务模块的功能划分和设计，完整涵盖自动化构建发布系统通过网页服务模块与项目用户交互的功能需求。

### 3.1.1 主交互网页服务

完整的网页服务模块流程图如下所示：

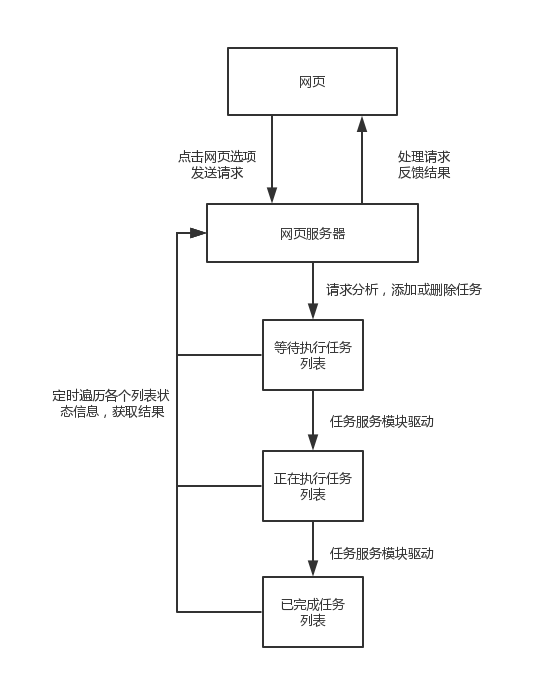


图3-1 主交互网页服务

用户访问指定网页时，网页服务器通过PHP解释器获取各个状态任务列表文件夹下面的文件，然后交由JavaScript脚本在用户浏览器上运行呈现出可操作的任务指令列表，以及任务所有状态信息。PHP与JavaScript分别处理后端与前端数据，在具体实现中可以灵活运用它们的特点进行相互配合实现功能。PHP与JavaScript的数据交互方式有许多种，如通过Cookie或通过HTML中转，PHP更可直接输出JavaScript语句，声明变量给JavaScript使用，最后提到的这种处理方式简单易操作，可依据实际情况做相关使用。

用户点击相关指令列表，网页服务器对应生成文件存储在待执行列表中，在网页被重新请求时，PHP更新各任务状态列表文件夹下面的文件状态等信息，对应网页获取新的数据进行同步更新显示，提示当前存在的待执行任务，任务创建时间，同时提供可取消选项。当用户点击取消时，对应删除待执行任务列表文件夹中对应的任务文件，如果此任务文件不存在（已经移至其它状态列表文件夹中，说明当前状态为正在执行中或已经执行完毕），则丢弃这次取消任务操作。如此设计比较简单明了，因为打包构建这些任务均是起到对项目迭代推进的作用，已经在执行或执行完毕的任务没有非常必要取消回退的需求，此外即使需要回退，也可通过版本控制管理系统取出历史版本，在任务本身执行时进行中断的意义不大，也会增加额外复杂度。

完成接收用户下达的任务指令并对应显示后，流程由后面即将介绍的任务服务模块进行推动，任务被依次放置在正在执行任务与已完成任务列表文件夹中，网页服务器在这个对应时间段中可通过PHP遍历查看这些列表文件夹中的文件名及文件内容来获取信息，再通过JavaScript脚本的处理呈现在网页之上。

正在执行任务因为会按照队列依次顺序执行，所以只会显示一项信息，具体内容为任务描述、创建时间和开始执行时间，以此提示用户。已完成列表则是显示当前仍存储在已完成任务列表文件夹中的所有任务文件，具体内容为任务描述、创建时间、开始执行时间、任务耗时、任务完成状况（根据具体状态划分为正常、警告、错误）。点击任务完成状况则弹出新的日志页面，显示记录在任务文件里的具体任务执行反馈信息。

任务完成状况的设计参考程序编译的状态表现，即正常编译，编译时发现有不规范的地方而进行警告，编译错误。对应将部分贴图格式问题，脚本编译警告，非关键资源缺失等不影响项目构建发布运行的问题归至警告类，Unity3D内部异常，脚本编译错误，关键资源缺失或损坏，自动化构建发布任务本身执行出错等影响项目构建及正常运行的问题归至错误类。确定好任务完成状况后，配合在网页上相应的颜色显示（正常对应白色底色，警告对应黄色，错误对应红色），项目开发人员可以非常自然、快速的了解到此任务执行结果，并可点击打开详细记录进行查看确定问题所在。

### 3.1.2 管理网页服务

在实际项目使用中，自动化构建发布系统还会存在如下两处需求：

1. 项目在不同阶段其构建发布需求会存在差异。例如在开发阶段往往是需要频繁的打包和构建安装包，此时只需要发布存放在局域网以及单台外网服务器中即可。而在正式发布的阶段，由于需要面对市场用户，往往要将打包文件及安装包文件推送至CDN(Content Delivery Network，内容分发网络[[[30]](#endnote-29)])，以方便软件用户可以更快地获取到数据内容。发布后从市场角度出发对于用户在一定周期内属于更新阶段，不需要重新推送完整安装包，这时主要使用的是打包及推送CDN任务指令，而不进行新的安装包构建，故构建安装包任务最好能够关闭以免项目人员误操作发布了新的安装包。

2. 部分平台需要发布多个安装包，安装包之间往往只是部分参数设定细微的差别，这时任务指令需要根据实际需求增加。

应对上面的需求，网页服务模块增加管理网页来对主交互网页进行配置，这样便可实现在浏览器上进行调整、维护，提升自动化构建发布系统的快速响应能力。

同时部分常用功能也可附加在主交互网页之上，在主交互网页产生约定的操作时进行网页配置调整，例如指定快捷键操作选中任务指令即可开关其功能。



图3-2 管理网页服务

### 3.1.3 其它辅助服务

借助网页服务器对外呈现数据的能力，可以将自动化构建发布系统涉及的文档内容、注意事项放置在网页上方便项目人员阅读、获取最新信息。

类似项目安装包下载地址，以及面向移动平台增加下载地址的二维码图片，均可设置网页、链接以供使用。

## 3.2 任务服务模块

任务服务模块属于核心模块，其特点是任务的描述简单，即主要是进行各个平台的构建和发布，但是每个任务涉及实际事项繁杂，需要根据需求进行步骤划分，变为一系列小的特定任务，再依次处理。

针对网页服务模块设定的任务管理方式，任务服务模块主流程需要定时查看等待执行任务文件夹，从中获取最早提交的任务文件进行处理。获取需要处理的任务文件后，将此文件调整命名与内容并移至正在进行执行任务文件夹中，然后根据任务文件在移动前获取到的具体描述信息进行实际任务的操作，实际任务操作完毕后，主流程继续推进，将任务文件名调整为含有创建时间、开始执行时间、耗时、任务命令、状态信息命名，内容则保存完整执行日志，并存储在已完成任务列表文件夹中。同时可对其中已完成任务文件进行整理，按完成时间将旧的任务文件进行打包备份后删除，维护整体存储大小。

完整的任务服务模块流程图如下所示：

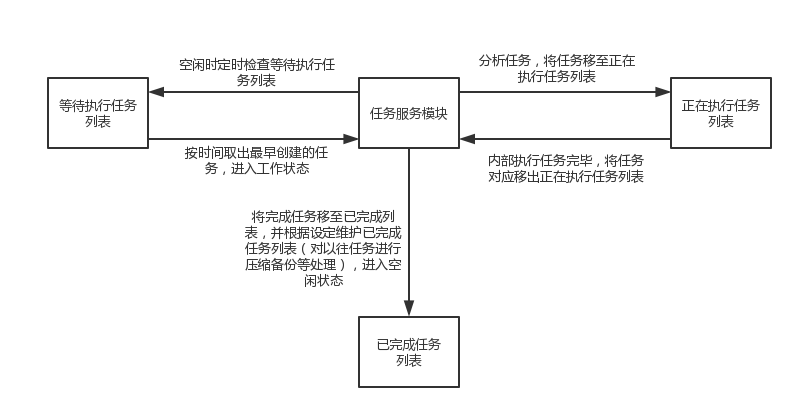


图3-3 任务服务模块

文件与文件夹的设计保证了状态记录工作的稳定运行，任务处理则需要保证可以从中断时的状态进行恢复处理，也就是意味着关键的任务处理应该可以重置状态，重新执行，这里借助版本管理系统，对开始执行任务时项目工程中的文件进行还原重置，来保证任务开始的输入状态一致。另外，在实际处理中还需要对项目文件进行更新，保证项目工程中的所有文件为当前最新状态。本论文描述此还原、更新处理操作时，统称为项目更新（即将项目还原修改，更新为当前最新状态，与版本管理系统的仓库中的各个文件一致）。

任务服务模块主要的任务为将各类资源打包为各平台下的AssetBundle及生成或处理项目需求的数据文件，将项目构建发布为各平台下的安装包，以及上传、分发。

### 3.2.1 资源打包任务

资源打包任务主要依次完成项目更新，项目打包前修改，其它三方内容处理，调用Unity3D脚本打包，还原打包前端的修改，版本管理系统SVN提交等子任务，整个流程依次串联，顺次执行。

针对此任务可参考模板方法模式进行设计处理，模板方法模式主要定义的是一个操作中的算法的基本结构，类似骨架的形式，在此基本结构上将具体执行步骤的实现放置在子类当中，子类可以在不改变基础算法及结构的情况下可以进行重新定义调整该算法中的某些行为步骤[[[31]](#endnote-30)]。资源打包主流程可以顺次调用子类的统一接口完成任务，子类具体操作则在它处各自进行定义实现，这样整个主框架清晰明朗，子类逻辑也可独立、有针对性的编写处理。此模式即在本系统的各个任务处理中得到应用。

模板方法模式在支持类、类继承的语言上可以轻松实现，设计好基类，基类属性和基本方法，即可通过继承实现各个子类，在子类中复写基类方法。结合资源打包任务特点，本系统对模板方法模式进行一定的扩展，子任务之间继续按照类型划分，在基类上继承实现的部分子类亦作为同类型、特点子任务的基类。按照不同平台进行资源打包任务便是如上设计，分别存在对应的Windows，Android，iOS子类作为后续任务的基类。

### 资源打包任务

图3-4 资源打包任务

如前所述，资源打包任务首先进行保证项目状态的项目更新操作，然后在开始执行Unity3D的脚本进行打包操作前，还需要进行各类修改。这里面包括开发人员定义在配置文件中的修改内容和业务固定的修改内容，和项目本身设定关系密切，例如在应用此系统的一个战斗类游戏项目中就需要去除指定路径下所有预设的材质球以防止在Unity3D 4系列版存在的重复打包问题或者在Unity3D 5系列版本存在的不必要的依赖问题；特定预设的动作控制器进行移除；场景开发时所需的碰撞体在正式运行时由于程序处理的数据保存信息而不再需要，也可进行移除避免占用空间及运行时的创建、维护而造成的性能开销；对UI等各类使用到的贴图进行对应的格式设定，保证取得压缩大小与显示质量的良好平衡。此外，还有其它三方内容处理，如在Android平台上将Java程序部分编译为jar包以便后面进行打包处理。

打包前的这些修改需要对实际操作进行归类，模块化处理，可以同样使用模板方法模式设计，再由配置文件针对不同资源进行不同操作。

对于Unity3D项目工程来说，需要进行统一处理的资源往往是某一系列同类型的资源，亦或者是同处于一个文件夹下面的资源。这些资源在路径、命名上会有一些公共特性，故配置文件及解析配置文件在驱动具体任务操作时需要支持尽量丰富的批处理，本系统采取简略的正则表达式（这里往往不需要太复杂完整的正则支持，通过语言相关处理库和自定义扩展进行处理）描述指定资源进行指定任务处理，对符合规则的系列资源再依次进行处理。

调用Unity3D脚本打包主要包括对各类脚本进行编译、打包，根据配置文件设定的资源包名进行相应打包调用。

脚本编译打包主要针对Unity3D使用的脚本语言以及项目引入的其它脚本语言进行处理。此处设计是将脚本语言编写的代码文件作为资源对待，编译后同样进行打包发布，这样即可定制编译过程，也可支持对应代码的在线更新（Unity3D自身脚本语言C#在部分平台上受限制不能进行更新替换），提升项目灵活性。

在Unity3D 4系列版本中依赖于配置文件设定的依赖关系进行对应打包接口调用生成，同时生成依赖文件信息便于项目工程实际运行时加载使用。在Unity3D 5系列版本中，由于其自身在打包过程中会自动处理依赖关系，所以从配置文件生成对应的资源打包信息传入Unity3D的打包接口直接调用即可，依赖文件文件信息也会自动生成。项目可根据需求对Unity3D自动生成的依赖文件进行处理生成适配自身使用的依赖数据文件。这些文件本身也再次进行打包处理，得到最终打包资源。

主要工作完成后，由于在实际打包中对部分资源进行了调整、去除部分不必要的组件，所以这里再进行还原处理，然后通过版本管理系统SVN进行打包资源提交，结束整个流程（项目也可在SVN提交后根据需要进行打包资源的相关上传、发布）。

### 3.2.2 构建发布任务

构建发布任务主要完成项目更新，项目构建前修改，调用Unity3D命令行进行发布，还原构建前修改，版本管理系统SVN提交。和资源打包任务流程较为相近，但内部处理方式则是完全不同，前面主要是资源的调整和打包，而这里是对资源的组织和移动以及生成整个项目工程的安装包。

项目构建前的修改流程主要是配合前面所说为同时支持开发与构建发布两种项目状态，依据Unity3D发布和更新特点，将打包处理后的资源复制整理到StreamingAssets文件夹内，同时将Resources文件夹下的已经打包的各类资源进行移除。此外还需当前构建平台进行相关包名，版本号设置，以及针对调试版本的相关设置。

相关处理完毕后调用Unity3D执行指定平台的构建的命令即可进行安装包构建。由于iOS平台的特殊性，构建发布iOS平台下的ipa包时需要相应平台库与编译工具，Unity3D采取的处理方式是导出Xcode项目，Xcode项目设置、资源组织等同步于Unity3D项目，然后在苹果设备上进行编译发布。Xcode项目的编译已经与当前流程完全不同，为统一构建发布任务，这里参照Unity3D的设定，将iOS平台的构建发布任务调整为发布Xcode项目（借助Unity3D发布的XcodeAPI，项目在这个步骤生成Xcode项目后，执行脚本对Xcode项目进行定制化修改）。

构建发布后，同样需要要进行项目还原，然后通过SVN将构建发布获取的结果（安装包或Xcode工程）进行提交上传。

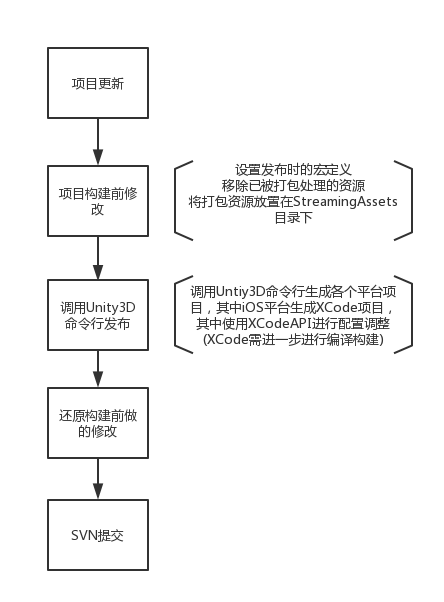


图3-5 构建发布任务

### 3.2.3 Xcode构建发布任务

此任务主要针对iOS平台的Xcode项目进行编译、构建发布，即上面的构建发布任务的补充。Xcode构建发布任务主要依次完成Xcode发布通知，版本管理系统SVN更新，调用Xcode命令行编译构建，版本管理系统SVN提交。

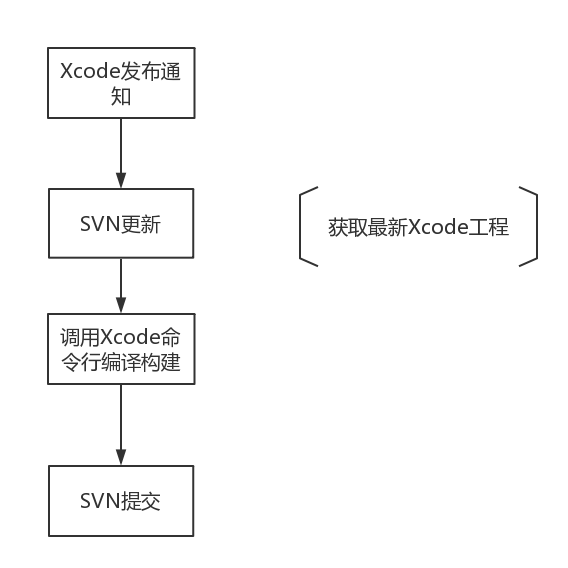
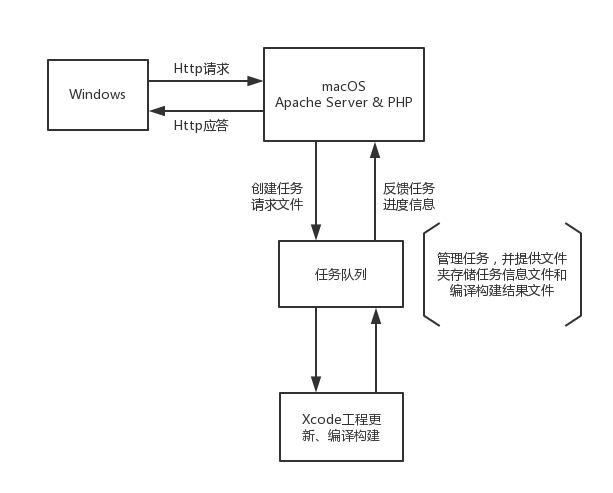


图3-6 Xcode构建发布任务

流程前面的通知和SVN更新获取最新Xcode工程是承接构建发布iOS任务所进行的流程。调用命令行编译构建ipa安装包会涉及到符号表、签名相关的处理，也是此任务的主要环节。相关处理完毕后将设定内容提交至版本管理系统，自动化构建发布系统则轮询获取此流程构建生成的结果及安装包。

### 3.2.4 整体系统配置

由于目前Xcode工程发布iOS版本需要在macOS系统下进行，故要配备一台Mac机器。当前整个系统可以完整搭建在Mac机器上，不过考虑相关硬件、维护开销，最终选择Windows兼容机与Mac机器配合的方式，网页服务模块与任务服务主模块运行在Windows兼容机之上，大部分构建发布任务也运作在Windows兼容机上，Mac机器仅处理Xcode编译发布任务，Windows兼容机在硬件选购配置上可以有更好的性价比。采取此种方案，Mac机器上通过搭建网页服务器提供请求Xcode编译服务和编译后文件下载服务，Windows兼容机通过脚本访问网页命令提交请求、查询编译是否完成、下载结果文件。

图3-7 iOS平台发布特殊流程

此流程设定依然遵循快速、稳定的思想，由设置在Windows兼容机器上的主服务模块通过网页请求进行任务申请、任务查询、安装包获取等事项处理，然后再同其它平台构建发布任务一样，进行SVN提交及后续处理。Mac机器则参照主服务模块的交互网页服务方式，通过网页服务器提供Xcode项目更新，编译构建的请求处理及相关任务文件信息获取和安装包下载服务。结合上节所述Xcode构建发布任务，封装为一个构建发布任务，整体流程设计仍然保持统一。

## 3.3 小结

本章为自动化构建发布系统框架设计，通过流程图的展示和具体流程的详细说明，确定整个系统的实现方案、逻辑。

# 网页服务模块实现

## 4.1 网页服务环境配置

本系统选用了 WampServer[[[32]](#endnote-31)] 三方辅助工具配置环境, WampServer 提供Apache2， PHP 以及 MySQL数据库来构建网页应用，可以快速简单的将自动化构建发布系统需要的 Apache 和 PHP服务安装布置起来。

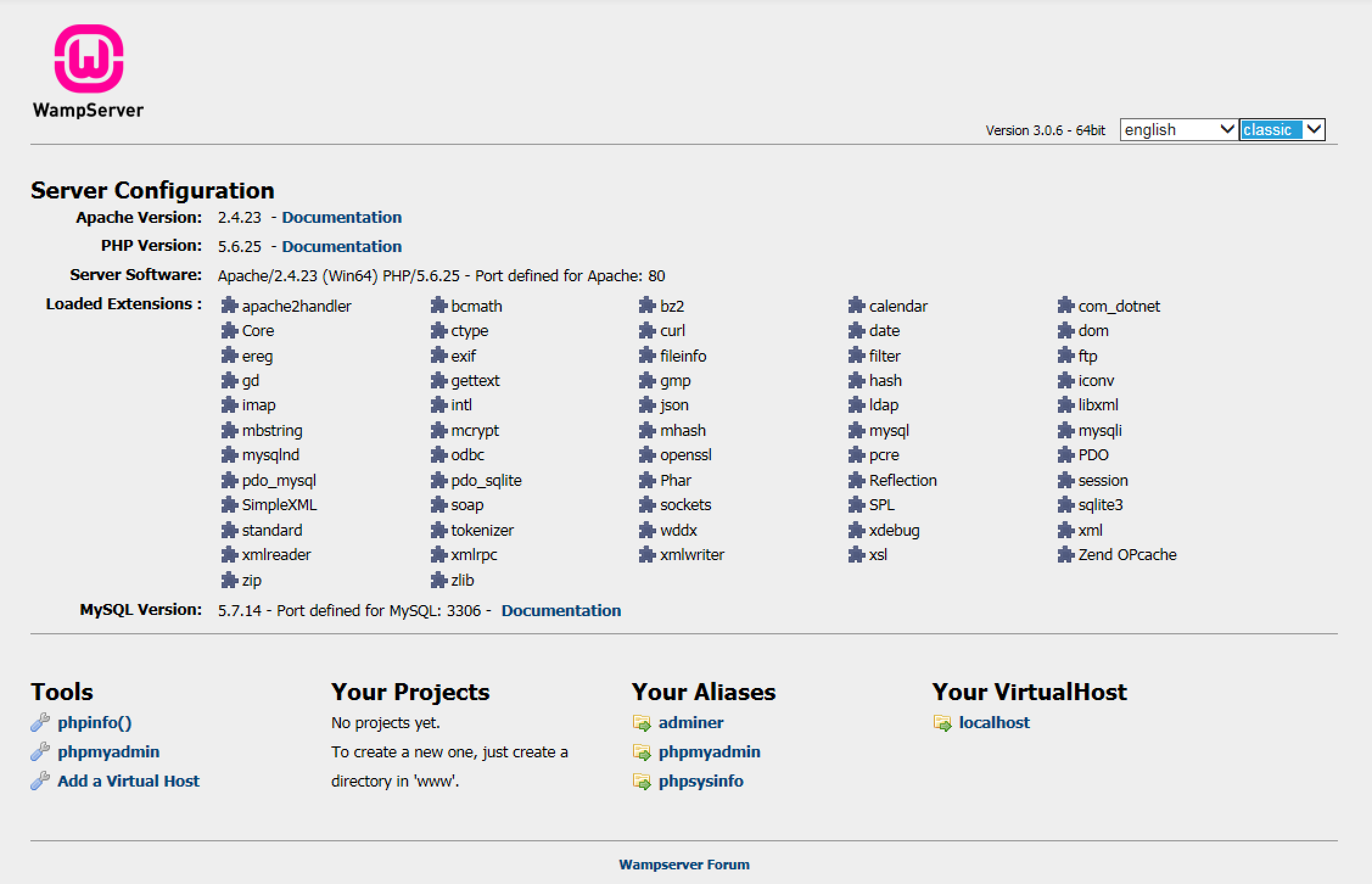


图4-1 WampServer安装后默认网页展示

安装后，可以运行其后台管理程序(Wampserver64)进行相关服务启动等处理，由于目前只使用其Apache及PHP服务，并需要设置开机运行以便可以自动持续地提供服务，故在系统设置中设置对应的Apache服务为自动启动类型即可。

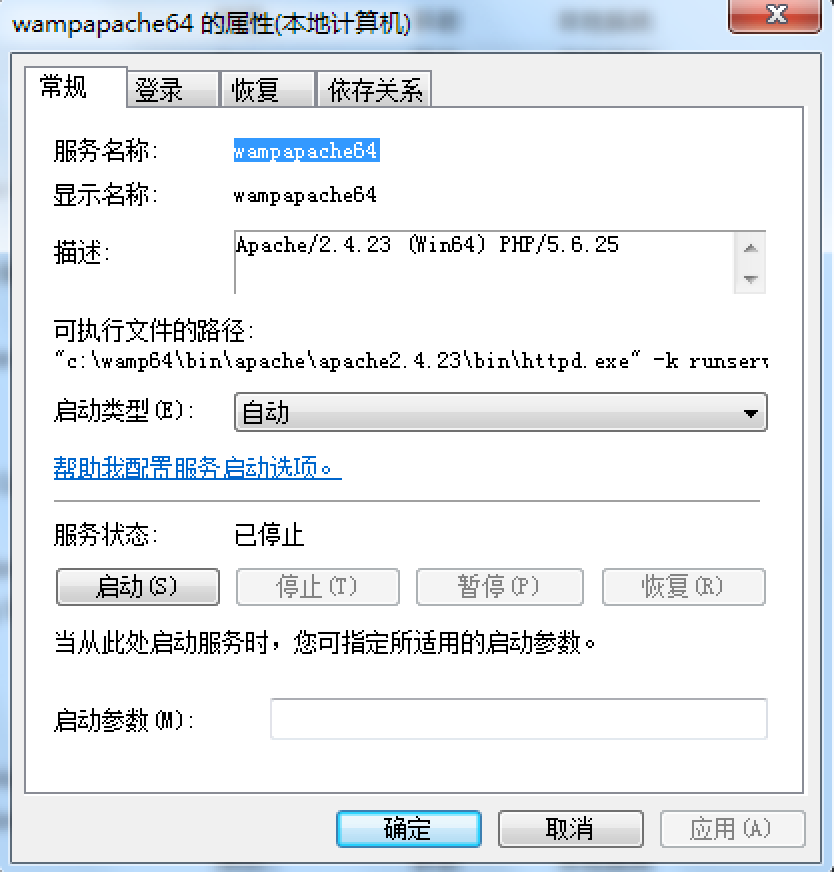


图4-2 Wampserver服务启动设置

打开其 httpd.conf 配置文件， 确认及调整网页服务监听端口，工作目录“www”路径，调整“Require local”为“Require all granted”让外部可以访问网页服务。简单配置后，便可在www所指定路径下创建服务网页进行后续任务开发，同时也可享受到Apache强大、稳定的服务性能。

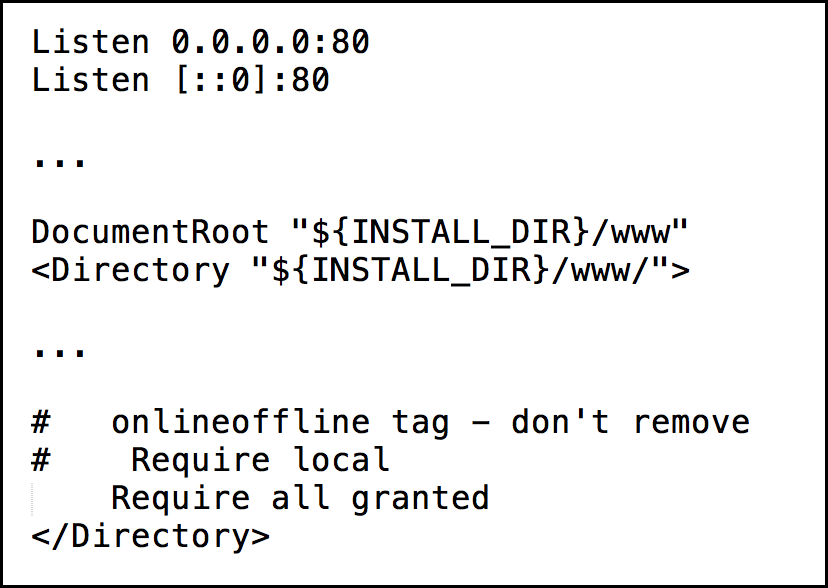


图4-3 httpd部分配置

## 4.2 文件与代码组织

网页服务模块大致实现的文件与代码组织如下，主要使用PHP与JavaScript进行相关实现，后面小节依次针对关键实现进行详细介绍。

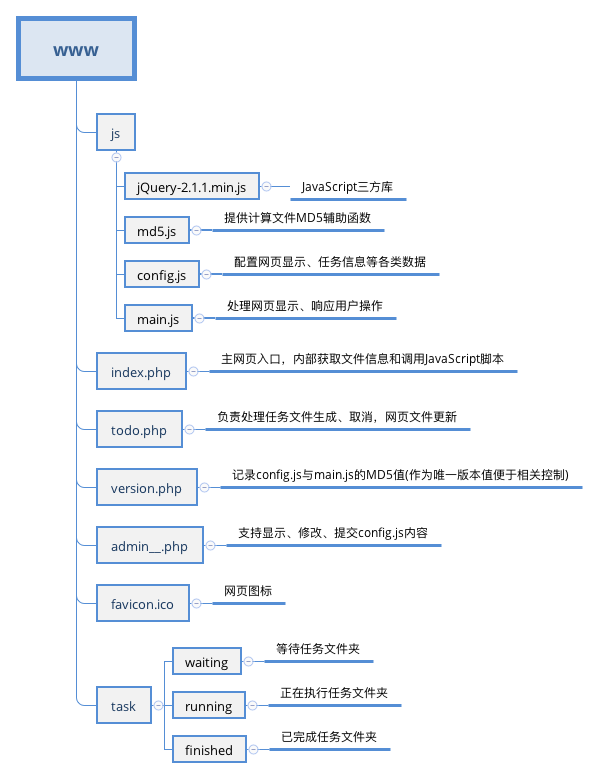


图4-4 网页服务模块文件、代码组织

## 4.3 交换网页实现

交互网页完成接受自动化构建发布任务，反馈任务完成信息两项最核心功能。参照面向项目人员的特点，主要保证功能实现清晰简单，交互网页主要集中在一个网页上完成。表现上划分为三个区域，均以列表的方式展现：页面上部罗列出各个平台下资源打包、安装包构建发布功能；中部显示当前执行任务及等待任务列表；下部显示已经完成的任务信息；详细任务信息通过点击具体已经完成任务的条目在新弹出的网页中展示出来。



图4-5 交互网页展示图

主页面(index.php) 创建表明网页基本元素，主要完成对三个任务状态文件遍历获取文件列表，包含主脚本(main.js) 及其它辅助脚本。主脚本完成功能显示，任务点击响应，任务状态显示功能。当点击完成列表日志栏下的各个任务状态时，会调用显示日志页面(showlog.php)。

### 4.2.1 获取文件列表

PHP脚本函数实现，由指定路径获取下面所有文件（系统部署时创建相关任务文件夹）。遍历逻辑比较简单，主要是使用opendir获取目录句柄，然后由readdir函数从目录句柄中读取条目。失败时会返回FALSE，以此作为循环中断判断。

表4-1 getFileList API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | function getFileList($path, $isRsort=FALSE) | | |
| 功能简述 | 获取指定路径下的所有文件 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | path | String字符串 | 遍历获取文件的文件夹路径 |
| isRsort | Boolean布尔类型 | 是否对获取的文件数组进行逆向排序 |
| 输出项 | 无 | Array数组 | 遍历获取的文件数组 |
| 备注 | 正常获取的数组为安装系统排序（基本是根据文件名排序），如果isRsort输入参数为TRUE，则将其逆向排序后输出 | | |

### 4.2.2 任务指令列表显示

此功能使用JavaScript脚本函数实现，考虑方便之后调整，将具体指令参数配置写在config.js中，由config变量统一保存。



图4-6 config脚本部分配置

函数数据来源即是config中的tasks键值数组，进行遍历处理添加至jQuery构建的table表单对象中，最后将其添加到HTML的body标签中以便显示。

表4-2 showOperations API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | function showOperations() | | |
| 功能简述 | 显示任务指令 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | 无 | | |
| 输出项 | 无 | | |
| 备注 | 读取config.js中的config.tasks进行处理，对应任务指令会加入点击响应 | | |

### 4.2.3 任务指令点击相应

此功能的JavaScript脚本函数onClickTask由上节所述任务指令列表中各个任务指令调用，提示用户当前操作并根据传递值在config.tasks中获取对应任务信息，对todo.php网页提交增加任务的POST请求（action:addTask），再由todo.php进行响应，在waiting文件夹中生成任务文件。

表4-3 onClickTask API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | function onClickTask(row, column) | | |
| 功能简述 | 响应任务指令点击事件 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | row | 数字 | 遍历获取文件的文件夹路径 |
| column | 数字 | 是否对获取的文件数组进行逆向排序 |
| 输出项 | 无 | | |
| 备注 | 读取config.js中的config变量数据，由行列确定是点击到具体task，进行处理。 | | |

在实际实现中，onClickTask完成处理了下述功能：

1. 根据任务配置获取任务提示内容，判断当前是否打开，进行提示。
2. 任务配置的标签(‘tag’)中设定了‘password’关键词的表明为需要验证密码的关键任务指令，此类任务指令在点击执行时网页会弹出一个输入框，用户需输入对应密码来继续执行操作。而不含有‘password’关键词设定的任务指令也均会生成一串随机数字显示给用户并提示需要输入此数字进行验证，以此避免误操作。
3. 快捷键点击开关任务指令，当指定的键盘按键按下时（本项目选用F8功能键），点击列表中的任务指令时会查找HTML对应的表的行列对象，并对其class属性设置成相对立的值(‘open’与‘disabled’)，同时进行记录保存在modifyOpenDict字典变量中。在按键松开时根据modifyOpenDict字典变量记录值生成信息访问todo.php提交POST请求（action:open）,todo.php对应响应对config.js内容进行调整并更新（内部调用网页更新在下面章节会进行详细描述）。

todo.php获取全局变量\_POST的action键值数据，判断当前进行何种操作（为配合后面章节提到的网页更新支持，还会对版本文件进行判断，如果不一致则不执行操作）。



图4-7 处理增加任务与开关任务指令代码片段

### 4.2.4 任务状态显示

任务状态显示主要包括等待执行任务，正在执行任务，已完成任务三个列表显示，对应JavaScript中三个函数(showWaitingList，showRunningList，showFinishedList)处理，其中等待执行任务列表显示中增加取消选项，点击时向todo.php发送GET请求(action:cancel)，todo.php则对应将等待任务文件删除。这里使用GET方式是由于请求数据简单，对应设置action值和file值即可，嵌入JavaScript脚本使用可以直接加入超链接访问指定链接地址完成调用（‘todo.php?action=cancel&file=’+item[‘fileName’])，编写会比较容易，已完成任务列表显示内的日志状态也是类似处理，加入超链接向showlog.php发送GET请求，showlog.php读取对应日志内容进行显示(‘showlog.php?V=task/finished/’+item[‘fileName’])。

表4-4 showWaitingList API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | function showWaitingList() | | |
| 功能简述 | 显示等待执行任务列表 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | 无 | | |
| 输出项 | 无 | | |
| 备注 | 对等待执行任务增加取消操作 | | |

表4-5 showRunningList API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | function showRunningList() | | |
| 功能简述 | 显示正在执行任务列表 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | 无 | | |
| 输出项 | 无 | | |
| 备注 | 无 | | |

表4-6 showFinishedList API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | function showFinishedList () | | |
| 功能简述 | 显示已完成任务列表 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | 无 | | |
| 输出项 | 无 | | |
| 备注 | 对日志状态增加点击获取详细内容操作 | | |

## 4.4 网页更新支持

网页更新支持主要实现支持调用SVN更新网页以及在网页中直接编辑配置文件并提交。同时增加“更新网页”与“提交网页配置”两项任务指令。其中网页配置(config.js)面临调整，修改的需求最多，所以独立出来并增加网页操作处理（SVN更新的设置专门避开网页配置的调整，避免冲突）。这两个任务在config.js中对应进行相关配置。

增加的网页操作编辑config.js的功能由admin\_\_.php网页处理，内部实现文件读取和提交功能。



图4-8 编辑网页配置代码片段

提交操作是向todo.php发送POST请求(action:admin)，内部处理同上面章节的快捷操作开关任务指令，将新的config.js内容字符串提交给handleAdmin函数对config.js进行覆写。

表4-7 handleAdmin API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | function handleAdmin($content) | | |
| 功能简述 | 更新覆写网页配置文件 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | content | String字符串 | 新的配置内容 |
| 输出项 | 无 | | |
| 备注 | 覆写config.js后，还会更新version.php网页中的config.js与main.js的新的MD5值 | | |

## 4.5 小结

本章对网页服务模块进行代码实现，列出PHP，JavaScript语言脚本下实现的主要函数描述和部分代码截取片段，分析了其中的逻辑要点，并对在实现时根据实际使用情况做的防止误操作等功能增加进行了强调。为有类似需求的网页交互开发提供借鉴。

Mac系统中相关网页服务类似，并且由于macOS自身已经集成了Apache与PHP环境，部署和处理也更便捷，具体实现略去介绍及说明。

# 任务服务模块实现

## 5.1 文件与代码组织

任务服务模块主要由Python脚本驱动，涉及自身逻辑流程运作和对Unity3D相关命令及使用C#扩展的自定义方法的调用，此外还有对各类辅助工具的命令行调用。

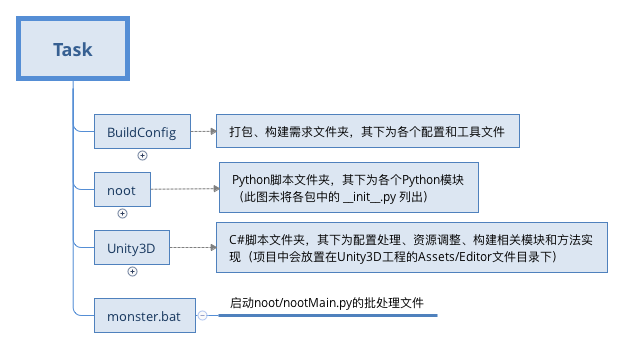


图5-1 任务服务模块文件、代码组织

具体实现中Python脚本采取模块层级包的组织方式，通过包名确认使用的类进行调用执行，而对应的Unity3D项目即是其服务对象也是其部分步骤调用方法的来源。整个模块由批处理(monster.bat)调用Python脚本(noot/nootMain.py)启动运行,批处理调用的设计在于为了实现整个模块代码更新时，可通过Python脚本返回值确认是否重新运行Python脚本，以便满足热更需求。

整个任务服务模块涉及较多文件和代码，本章节对部分组织、代码选取关键和具有代表性的部分进行介绍，其中的资源打包任务由于为配合项目需求多变和资源组织调整，实现采用配置文件进行设定，构建发布版本任务也需要将版本号的设置信息及项目配置文件拷贝至发布文件夹中，故其下先介绍配置、工具文件的组织，再介绍Python与C#脚本组织。

### 5.1.1 配置、工具文件组织

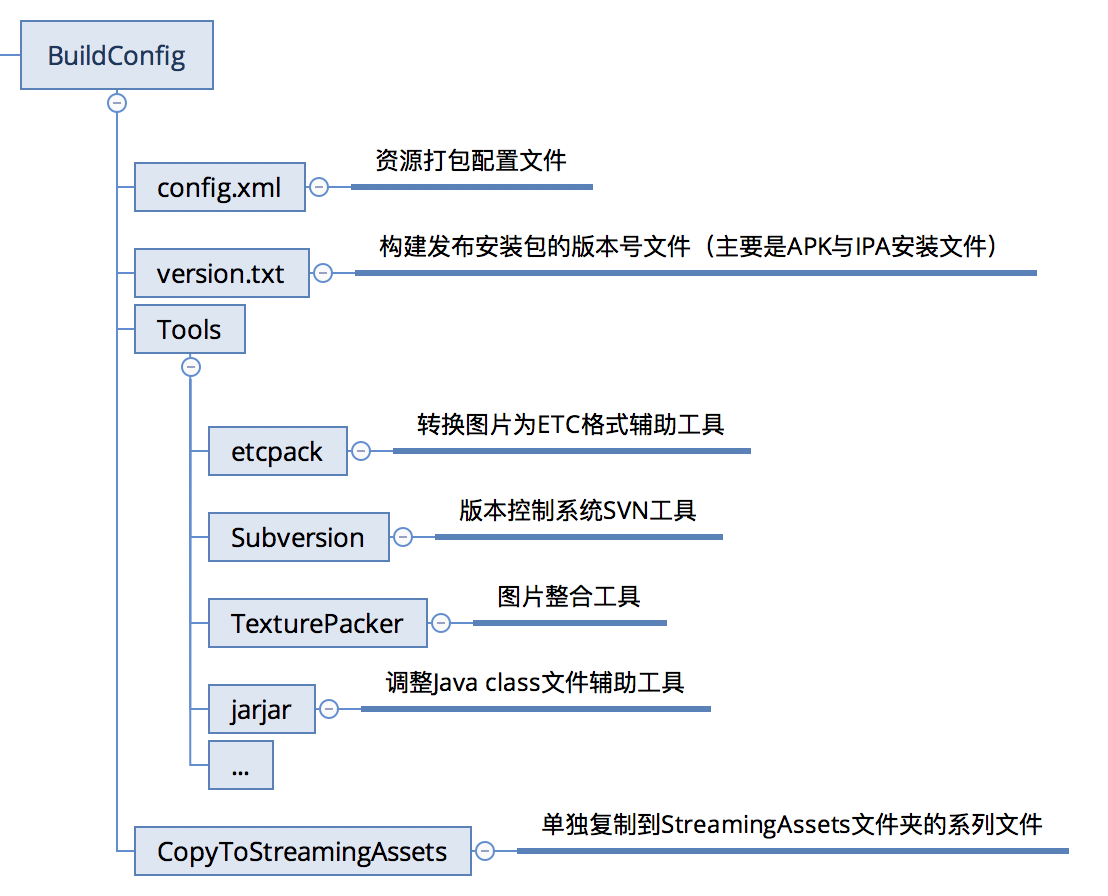


图5-2 配置、工具文件组织

打包配置文件config.xml设置具体打包方式，工具文件夹下则是实际任务会使用到的各类三方工具，包括SVN，图片压缩、整合工具等，放置在此文件夹可保证自动化构建发布系统部署方便，减少依赖以及版本需求（部分功能需要三方工具特定版本实现）。

CopyToStreamingAssets文件夹中放置无需Untiy3D打包而项目在发布时需要的文件（如实现中使用的服务地址列表）。

配置文件主要由Unity3D中的自定义C#脚本读取使用，其它由Python脚本在具体任务类中使用。

### 5.1.2 Python脚本组织

Python脚本采用包组织方式进行管理，对应文件系统即为文件夹表明一个包，内部Python脚本为模块。Python对封装成为包的设定是需要在每个文件夹目录下包含\_\_init\_\_.py[[[33]](#endnote-32)]文件(含有此文件即可使用执行import语句导入相关模块)。\_\_init\_\_.py文件大部分时候留空即可而不需要填写任何内容，但可根据需要定制写入代码来控制自动加载子模块等功能。这里展示Python脚本组织时，已经略去\_\_init\_\_.py文件，只列明关键包、模块。

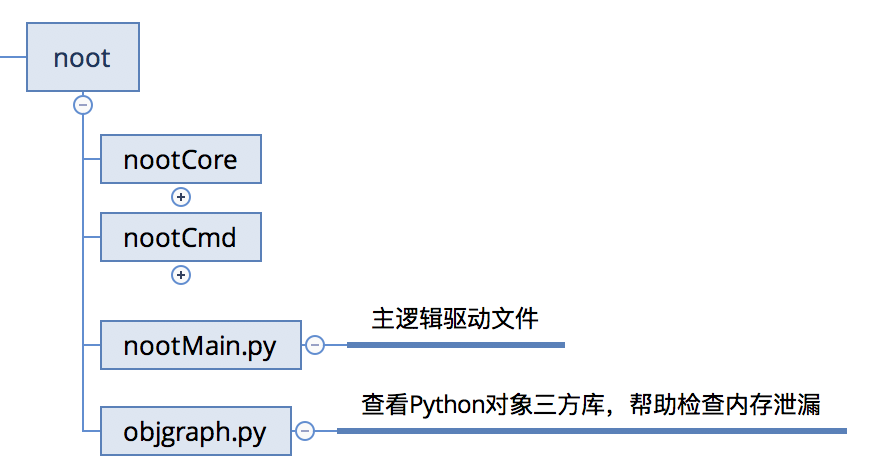


图5-3 Python脚本整体组织

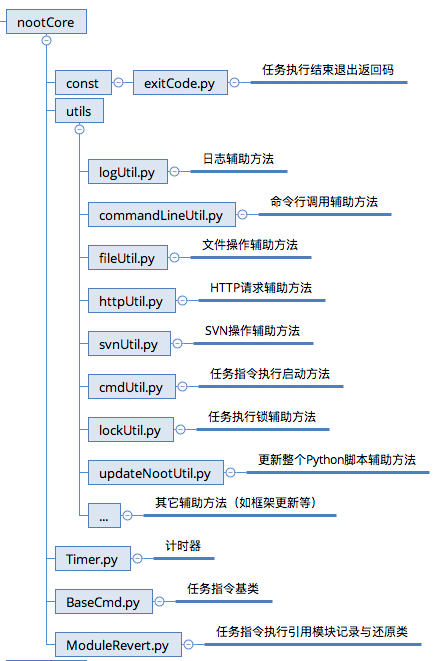


图5-4 Python脚本核心模块组织

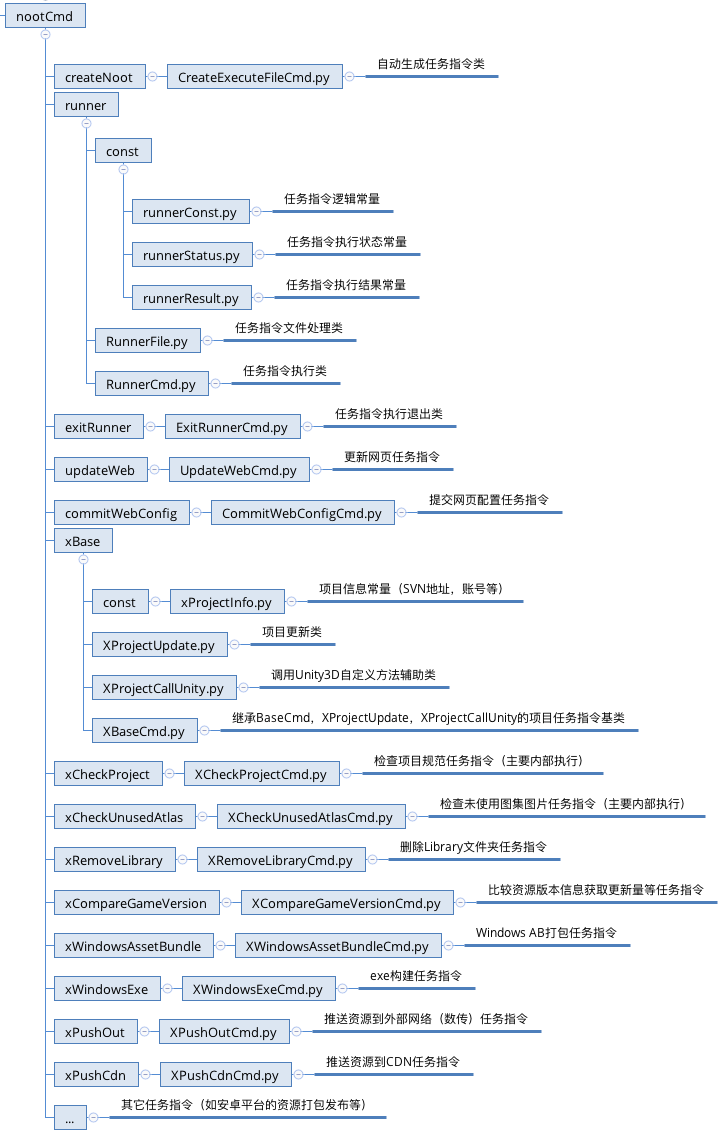


图5-5 Python脚本指令组织

### 5.1.3 C#脚本组织

C#脚本均是放置在Unity3D工程的编辑工具文件夹(“Assets/Editor”)中，由Unity3D内置的Mono虚拟机驱动执行，具备完整C#语言能力并可访问调用Untiy3D内部各类接口，这样便可实现对Unity3D资源进行修改调整相关逻辑。同时此处部分C#脚本实现功能也会在平时开发中使用到，故组织上会对部分工具类单独划分出来（如图集更新辅助类）。

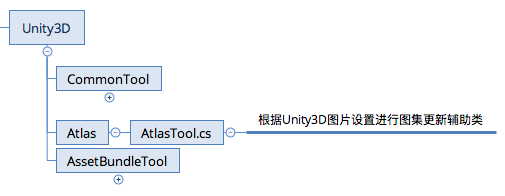


图5-6 C#脚本整体组织

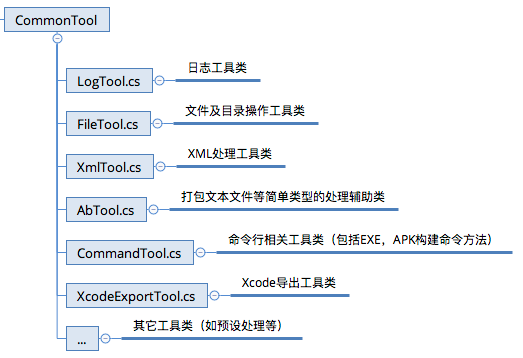


图5-7 C#脚本工具类组织

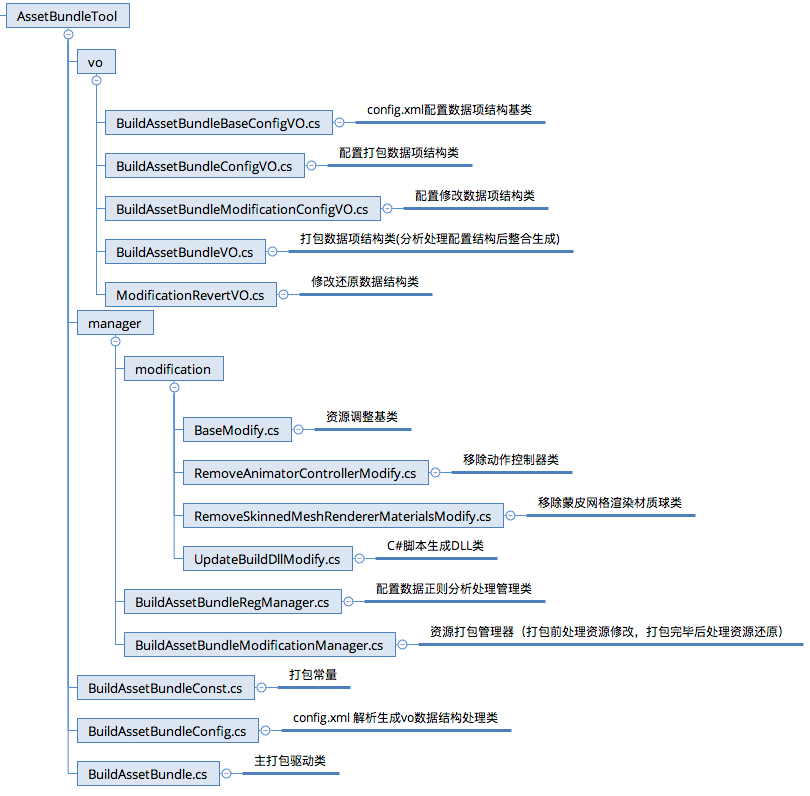


图5-8 C#脚本资源打包相关类组织

## 5.2 任务驱动逻辑

由批处理调用执行的Python脚本驱动整个任务服务模块，Python脚本采用间隔轮询的方式获取任务请求并调用对应任务模块执行。整个核心为定义各级任务指令，流程逻辑调用任务指令执行启动方法对其启动。下面分别阐述几个关键模块的实现。

### 5.2.1 基础指令模块

驱动逻辑的一个基础设计是将整个流程涉及的逻辑执行均作为一个个指令执行驱动，其自动化构建发布系统的实际任务也是由此继承扩展。基础指令模块nootCore.BaseCmd主要实现通用辅助方法（\_\_parseArgv）、基础构造函数(\_\_init\_\_)以及定义统一的API接口(execute, finish)。构造函数对参数解析初始化的设计让继承类可以方便的对实例进行初始化设置，如果定义了需要赋值的成员变量，只需在构造实例化时传入对应以成员变量和实际值为键值对的字典参数即可。

表5-1 nootCore.BaseCmd.\_\_parseArgv API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | def \_\_parseArgv(self, argv) | | |
| 功能简述 | 解析参数赋值成员变量 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | self | Class类 | 模块本身 |
| argv | Dictionary字典 | 上层处理构造传递过来的参数字典 |
| 输出项 | 无 | | |
| 备注 | 由构造函数\_\_init\_\_调用，遍历传入的字典的键值，如果本身也含有此同名的成员变量，则按成员变量当前类型对字典中对应的值进行转换赋值（转换失败的忽略，无对应成员变量的也直接忽略，对应均进行输出信息提示） | | |

表5-2 nootCore.BaseCmd.\_\_init\_\_ API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | def \_\_init\_\_(self, argv) | | |
| 功能简述 | 模块类构造函数 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | self | Class类 | 模块本身 |
| argv | Dictionary字典 | 上层处理构造传递过来的参数字典 |
| 输出项 | 无 | | |
| 备注 | 内部即调用\_\_parseArgv进行具体处理 | | |

表5-3 nootCore.BaseCmd.execute API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | def execute(self) | | |
| 功能简述 | 基础指令驱动执行逻辑 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | self | Class类 | 模块本身 |
| 输出项 | 无 | | |
| 备注 | 指明指令类基本驱动执行接口方法，继承类进行扩展实现 | | |

表5-4 nootCore.BaseCmd.finish API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | def finish(self) | | |
| 功能简述 | 基础指令完成退出逻辑 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | self | Class类 | 模块本身 |
| 输出项 | 无 | | |
| 备注 | 指明指令类基本完成退出接口方法，继承类进行扩展实现 | | |

后续指令类通过覆写上表两个接口进行各个具体逻辑实现。配合nootCore.utils.cmdUtil.executeCmd即可方便的通过传入包名和字典参数进行相应模块指令调用执行。

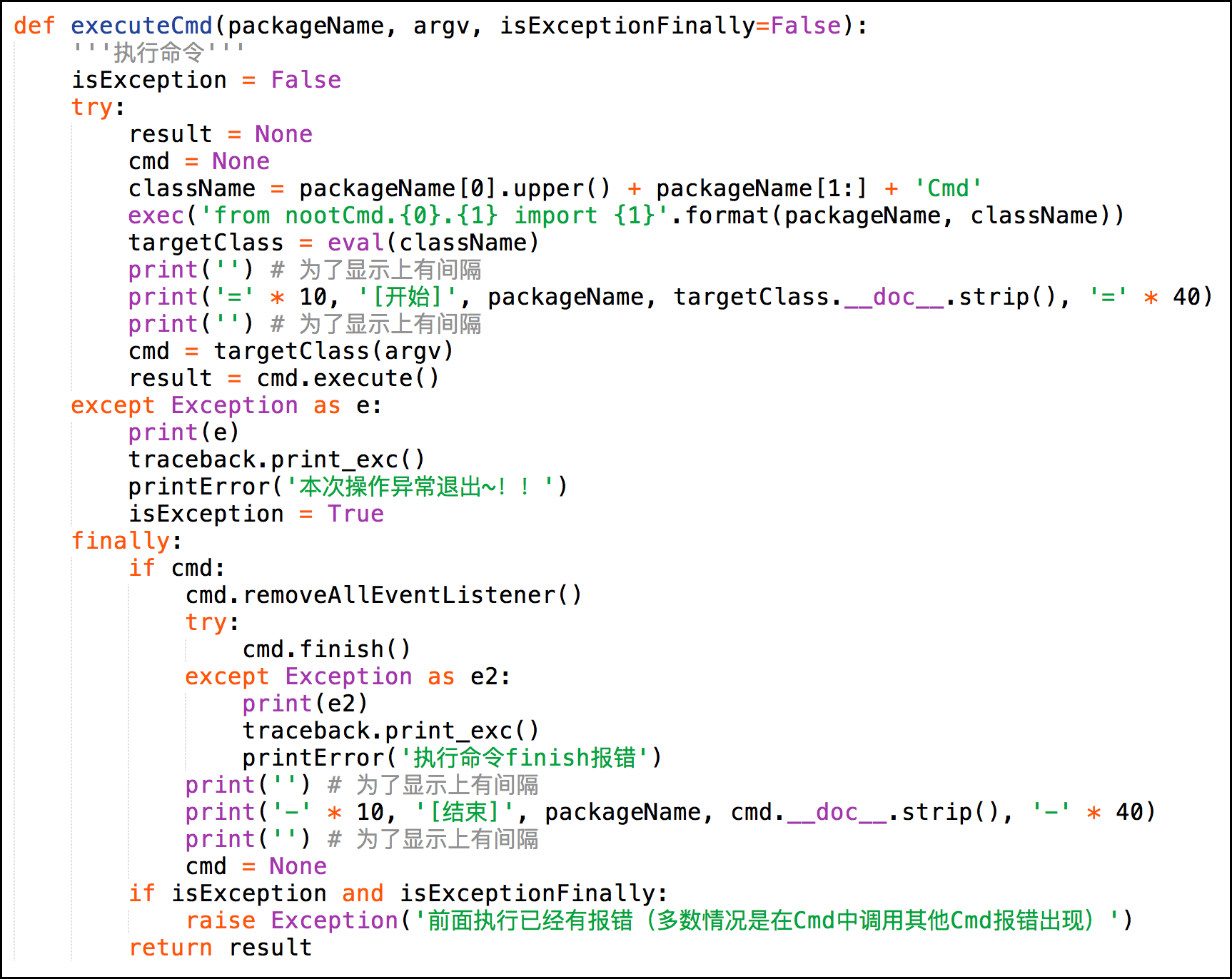


图5-9 nootCore.utils.cmdUtil.executeCmd辅助执行指令类代码片段

nootCore.utils.cmdUtil.executedCmd内部逻辑是在nootCmd包中寻找指定模块名的包，然后对指定模块名首字母大写并补上后缀‘Cmd’字串即为对应模块（例如传入‘task’，则对应查找到nootCmd.task.TaskCmd模块类，传入参数字典进行实例化，然后执行实例的execute方法）。具体实现增加相关信息输出以及异常捕获，实际继承开发具体指令类时可获知内部运行情况，以便跟踪、调试。

### 5.2.2 循环服务模块

批处理(monster.bat)启动主逻辑驱动文件nootMain.py，文件对应nootMain模块执行execute函数启动主循环逻辑，内部依次进行相关初始化处理与调用\_\_initArgv函数获取启动传入参数(即上节所述指令构造实例化所需的参数字典)，然后根据参数指明的主循环服务模块，调用辅助指令执行类方法(nootCore.utils.cmdUtil.excuteCmd)启动执行。

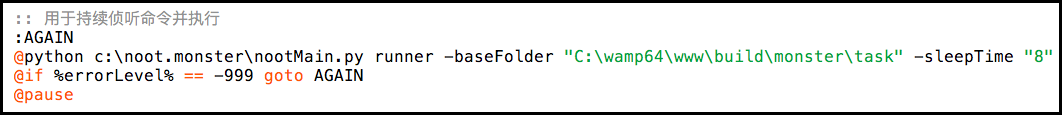


图5-10 monster.bat处理内容

monster.bat与noot.monster为具体项目调整命名，上面的文件代码组织中，对noot.monster等统写为noot目录包名。此批处理判断返回值为指定的值（nootCore.const.exitCode.EXIT\_CODE\_AGAIN,内部实现为自定义的-999）则会重新运行nootMain.py脚本。

表5-5 nootMain.execute API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | def execute() | | |
| 功能简述 | 主驱动逻辑执行 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | 无 | | |
| 输出项 | 无 | Number数字 | 退出返回码 |
| 备注 | 由if \_\_name\_\_ == ‘\_\_main\_\_’: execute()逻辑触发，这样在调用nootMain.py时即会运行此方法，执行结束时会调用exit并指定退出返回码 | | |

表5-6 nootMain.\_\_initArgv API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | def \_\_initArgv() | | |
| 功能简述 | 分析处理执行参数 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | 无 | | |
| 输出项 | argv | Dictionary字典 | 输入参数字典 |
| 备注 | 内部调用函数，对sys.argv进行处理并返回结果 | | |

nootMain.execute执行函数进行日志类，时间类、输入参数的初始化和相关处理，并且检查是否存在标识正在执行任务的文件，如果存在则输出信息返回错误码值（nootCore.const.exitCode），否则创建标识文件，以此保证同一个项目只会有一个任务服务模块运行，避免竞争出错。由传入参数获知要执行的模块任务名及参数字典，即可调用辅助指令执行类方法(内部会最终实例化nootCmd.runner.RunnerCmd类，再执行其excute 方法)，此runner模块是针对处理网页交互服务模块产生的任务文件进行间隔轮询后执行的项目任务驱动模块。

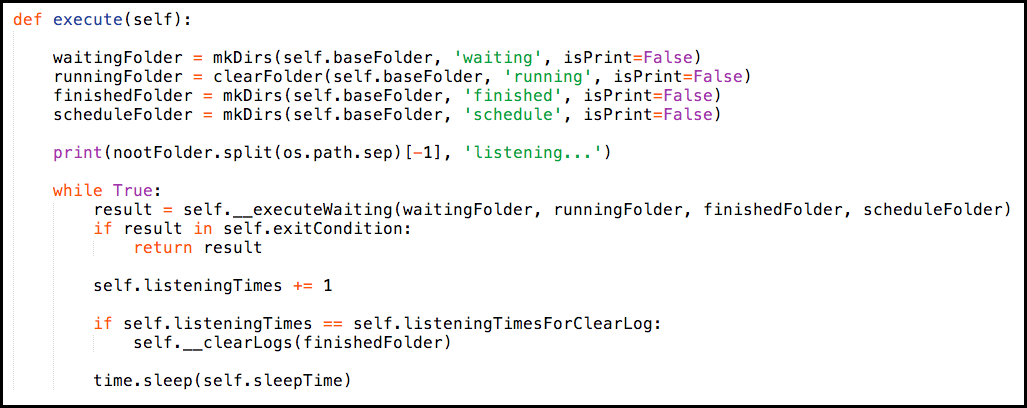


图5-11 nootCmd.runner.RunnerCmd.excute代码片段



图5-12 nootCmd.runner.RunnerCmd.\_\_executeWaiting代码片段

nootCmd.runner.RunnerCmd完成整个任务服务模块代码更新，等待任务列表读取初始化（使用任务文件对象进行解析）并依次进行调用，其中还涉及Python模块状态记录与清理，内存清理及快照输出，便于查看每个任务执行过程是否存在内存泄漏等问题。

### 5.2.3 基础工程指令模块

针对Unity3D打包构建特点，实现对项目工程调用SVN更新与调用执行Unity3D命令的辅助模块(nootCmd.xBase.XProjectUpdate与 nootCmd.xBase.XProjectCallUnity)，以及基于基础指令模块和上述两个辅助模块的基础工程指令模块（nootCmd.xBase.XBaseCmd）。

工程更新模块内部实现的主要流程是对工程执行更新，并且删除掉Unity3D对工程资源生成的Library缓存目录，然后从备份文件夹中获取对应平台的Library目录，移动至当前工程对应位置。备份处理Library目录的方式可以极大加速Unity3D在不同平台切换时所花费的时间（后续任务结束时，会再次将此Library目录移动到对应的备份目录下）。

表5-7 nootCmd.xBase.XProjectUpdate.\_\_updateTargetXFolder API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | def \_\_updateTargetXFolder(self, libName) | | |
| 功能简述 | 更新和准备工程目录 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | self | Class类 | 模块本身 |
| libName | String字符串 | 平台包名 |
| 输出项 | 无 | | |
| 备注 | 无 | | |

执行Unity3D命令模块主要实现callUnity方法，获取工程目录和需要执行的方法名及其它可能携带的参数后，进行命令行生成并调用执行，同时将执行所输出的日志文件进行打印和返回并移除此中介日志文件。

表5-8 nootCmd.xBase.XProjectCallUnity.callUnity API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | def callUnity(self, projectPath, executeMethod, \*\*pars) | | |
| 功能简述 | 执行Unity3D命令行 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | self | Class类 | 模块本身 |
| projectPath | String字符串 | 工程目录 |
| executeMethod | String字符串 | Unity3D工程对应的方法名 |
| \*\*pars | 关键字参数 | 可变关键字，使用时会变为字典 |
| 输出项 | 无 | String字符串 | 执行日志信息 |
| 备注 | 无 | | |

基础工程指令模块使用了Python脚本支持多继承的特性，借助辅助类的继承，封装实现了设置Unity3D工程平台以及宏定义、提交文件至版本控制仓库等大部分任务常用操作方法，并继承实现执行方法(execute)完成Unity3D工程更新、版本信息相关获取与信息输出等基本流程逻辑。各类打包、构建任务均是继承于此基础工程指令模块进行进一步编码实现。



图5-13 基础工程质量模块类代码片段

## 5.3 关键任务逻辑

自动化构建发布系统实际任务涉及各个平台和具体业务细分需求，数量繁多，其中最主要的两个大任务为资源打包处理与构建发版处理，下面选取Windows平台以及Unity3D的5系列版本进行具体实现阐述。

### 5.3.1 资源打包任务



图5-14 nootCmd.xWindows.AssetBundleCmd.execute代码片段

在Windows平台下的资源打包任务逻辑包含执行工程的版本更新，清除Unity3D生成的部分缓存资源以避免额外影响，设置平台和相关宏定义后执行Un打包命令模块(AssetBundleTool.BuildAssetBundle.callByCmd)让Unity3D进行具体的打包任务，打包任务完毕后对新生成的打包资源进行提交及后续资源检查处理。

Unity3D打包命令模块由项目工程中的C#脚本实现，完成打包前资源控制调整，对资源打包配置文件（config.xml）的读取分析，对应生成打包信息结构。其后根据此信息结构依次进行资源修改处理，解析打包数据、整理需要打包的文件及依赖列表，最后执行打包、生成游戏版本文件，还原数据完成任务。

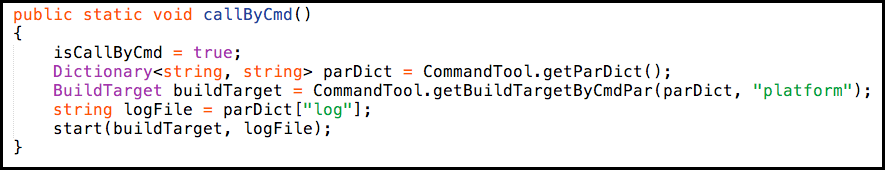


图5-15 AssetBundleTool.BuildAssetBundle.callByCmd代码片段

#### 5.3.1.1 图集更新处理

Unity3D内部会对标志了合并打包标签(Packing Tag)的界面图片资源进行自动合并处理，即将零散的界面资源拼接在一张大的图片上面，可以提升运行效率。图集更新处理设计为按照文件夹分类管理的界面资源进行检查设置图片属性和合并打包标签，并且根据需求定制Unity3D自动合并处理过程（不进行透明和非透明图片区分，统一合并等等），另外部分图集可能存在工程逻辑使用的情况，则需要对应将图集下的图片引用保存在一个列表然后放置到预设中，这样便可通过获取此预设取得相关图集里的图片资源。

表5-9 AtlasTool.updateAtlas API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | public static void updateAtlas() | | |
| 功能简述 | 更新图集相关资源 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | 无 | | |
| 输出项 | 无 | | |
| 备注 | 生成/更新的图集列表预设可附带材质球等其它信息 | | |

#### 5.3.1.2 资源修改处理

使用C#系统库中的XML库加载资源打包配置文件，获取XML操作对象实例(System.Xml.XmlDocument)，并针对配置文件中部分特殊标签进行扩展处理（如自定义的循环迭代子目录、循环迭代子文件、自动大小写变换、子字符串截取、相等等逻辑判断、模板复制），修改扩展XML操作对象实例。

获取到最终的XML操作对象实例后，调用执行资源打包管理器（AssetBundleTool.BuildAssetBundleModificationManager）进行遍历处理需要移除部分组件、资源，设置格式等操作，为后续打包减少重复资源（资源打包管理器还会根据XML操作对象实例对修改的资源进行还原处理）。

表5-10 AssetBundleTool.BuildAssetBundleModificationManager.execute API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | public static void execute(XmlDocument configXml, BuildTarget buildTarget) | | |
| 功能简述 | 根据配置文件修改资源 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | configXml | System.Xml.XmlDocument 对象 | 加载配置文件获取的实例 |
| buildTarget | UnityEditor.BuildTarget枚举量 | Unity3D设置的平台枚举类型 |
| 输出项 | 无 | | |
| 备注 | 无 | | |

#### 5.3.1.3 打包数据与依赖列表相关处理

依据XML操作对象实例中各个属性设置，解析生成配置打包数据项结构类列表（BuildAssetBundleConfigVO,继承自BuildAssetBundleBaseConfigVO）,然后整理、分析依赖文件生成打包数据项结构类列表（BuildAssetBundleVO）。

表5-11 AssetBundleTool.BuildAssetBundleConfig.getBuildConfig API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | public static List<BuildAssetBundleConfigVO> getBuildConfig (XmlDocument configXml) | | |
| 功能简述 | 获取打包数据项结构类列表 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | configXml | System.Xml.XmlDocument 对象 | 处理扩展的配置实例 |
| 输出项 | 无 | AssetBundleTool.BuildAssetBundleConfigVO的List列表 | 根据配置信息生成的打包数据信息列表 |
| 备注 | 无 | | |

表5-12 AssetBundleTool.BuildAssetBundle.getBuildAssetBundleList API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | private static List<BuildAssetBundleVO> getBuildAssetBundleList(List<BuildAssetBundleConfigVO> config, string outputFolder) | | |
| 功能简述 | 获取需求打包的文件及依赖列表 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | config | AssetBundleTool.BuildAssetBundleConfigVO的List列表 | 根据配置信息生成的打包数据信息列表 |
| outputFolder | String 字符串 | 输出打包文件的文件目录 |
| 输出项 | 无 | AssetBundleTool.BuildAssetBundleVO的List列表 | 根据打包数据信息列表生成的需要打包文件及依赖的数据信息列表 |
| 备注 | 无 | | |

#### 5.3.1.4 打包与版本文件生成处理

获取到易用的打包及依赖数据信息列表后，结合平台信息和输出目录信息，依据Unity3D的打包传输格式需求生成对应设置参数，传递给Unity3D的打包方法执行打包（UnityEditor.BuildPipeline.BuildAssetBundles）获取打包文件。

目前Unity3D设置打包生成的文件均为小写，可根据项目需求对应调整打包文件名。

根据SVN中的信息，将新生成或更新的打包文件记录并分配当前SVN版本号，结合其MD5值，生成版本文件，更新提交相关文件后完成资源打包任务。

表5-13 AssetBundleTool.BuildAssetBundle.buildAssetBundle API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | private static void buildAssetBundle (List<BuildAssetBundleVO> buildFiles, BuildTarget buildTarget, string outputFolder) | | |
| 功能简述 | 获取需求打包的文件及依赖列表 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | buildFiles | AssetBundleTool.BuildAssetBundleVO的List列表 | 打包文件及依赖的数据信息列表 |
| buildTarget | UnityEditor.BuildTarget枚举量 | Unity3D设置的平台枚举类型 |
| outputFolder | String 字符串 | 输出打包文件的文件目录 |
| 输出项 | 无 | | |
| 备注 | Unity3D的打包方法会将指定的输出文件名调整为小写，项目需注意此特性 | | |

表5-14 AssetBundleTool.BuildAssetBundle.exportVersionFile API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | private static void exportVersionFile (BuildTarget buildTarget, string outputFolder) | | |
| 功能简述 | 获取需求打包的文件及依赖列表 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | buildTarget | UnityEditor.BuildTarget枚举量 | Unity3D设置的平台枚举类型 |
| outputFolder | String 字符串 | 输出打包文件的文件目录 |
| 输出项 | 无 | | |
| 备注 | 无 | | |

### 5.3.2 构建发版任务



图5-16 nootCmd.xWindowsExe.XWindowsExeCmd.execute代码片段

此任务实现较资源打包任务要简单一些，主要是对工程进行发版相关的项目配置设置，移除已进行打包的普通资源，将打包后的文件放置在Unity3D指定的StreamingAssets目录，调用Unity3D对应的设置启动场景及构建方法(CommonTool.CommandTool类下的SetBuildScenesAndDriver与buildExe)完成构建生成可执行文件，然后执行提交发版。

表5-15 CommandTool.CommandTool.setBuildScenesAndDriver API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | public static void setBuildScenesAndDriver() | | |
| 功能简述 | 设置编译场景以及驱动模块参数 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | 无 | | |
| 输出项 | 无 | | |
| 备注 | 无 | | |

表5-16 CommandTool.CommandTool.buildExe API描述

|  |  |  |  |
| --- | --- | --- | --- |
| API原型 | public static void buildExe() | | |
| 功能简述 | 编译构建可执行文件 | | |
|  | 参数名称 | 参数类型 | 描述 |
| 输入项 | 无 | | |
| 输出项 | 无 | | |
| 备注 | 无 | | |

## 5.4 难点及注意事项

在任务服务模块的具体实现中，面临了不少的问题和挑战，其中常见注意点如下：

1.Unity3D将资源序列化设置为明文格式存储，便于查看对比和直接的文本调整，也利于版本控制。

2.提供给用户操作的任务指令要尽量避免参数输入及选择，可以减少了错误的发生，在本系统中基本做到任务指令均不需要输入参数。

3.自动化流程要有完整的异常保护机制，杜绝各类死循环，中断发生，对异常都有相应处理保证主流程稳定工作。

4.要有详尽的日志记录与时间记录，便于查看问题和效率开销，同时配合三方工具检查内存占用情况，确保程序未有内存泄漏，可长时间持续运行。

5.配置文件统一存放并做到简单易懂，这样可以轻松应对频繁的修改配置需求。

6.网页交互页面使用颜色区分，做到直观简洁、一目了然。

7.发布系统的计算机要注意磁盘空间管理，定时备份以及清理，避免频繁的打包构建任务操作挤占大量空间。

此外，针对Unity3D特定项目中的图片智能合并与着色器多平台编译优化往往会对存储和运行效率造成很大影响。

图片智能合并建议增加自定义的处理算法进行辅助分析，可参考目前各类装箱算法处理[[[34]](#endnote-33)]。其中根据合并贴图的特点，对排列信息进行记录，记录参与合并中宽高在2的各幂次大小附近的图片以及造成合并图大小扩充的图片。可自动对其进行Unity3D的最大尺寸调整设置以及显示反馈给项目人员进行图片大小优化。整合图片大小的合理控制可以极大的减少存储空间和运行空间的占用，降低内存、各处理器传输量，提升性能。

由于面向多平台开发，Unity3D对项目中使用到的着色器会进行预编译处理并在各个平台的应用实际运行时再进行针对当前硬件平台编译处理，这个过程的耗时会随着着色器自身逻辑复杂度，控制关键字以及着色器数量增加而增加[[[35]](#endnote-34)]，部分情况下会达到几分钟之久（本文在初期准备实验中，将所有着色器不做优化调整均放在一开始进行加载编译，在部分低端手机上甚至会耗费十几分钟）。整合优化着色器降低数量，优化关键字，多块打包着色器是处理此类问题的关键解决方法。通过控制关键字和流程变量参数整合多个着色器为单一着色器，并根据项目实际使用的材质球属性分析构建自定义需要的着色器变体（留意multi\_compile多重编译的使用，这个会针对关键字使用排列组合的方式编译生成多个变体），结合提取应用开始运行时所需的着色器成一个小包来降低单个和整体着色器的编译时间。

## 5.5 小结

本章对任务服务模块进行了代码实现，使用Python语言进行整体流程的串联和处理，逻辑内部定义基础执行类，对大量的子任务进行继承实现。此外利用Python胶水语言的特性，与Unity3D编辑器及其他工具进行配合调用，完成各类相关任务。着重对关键功能实现时所碰到的问题进行阐述分析并记录要点。

# 项目应用

## 6.1 项目应用的其它配置

此自动化构建发布系统在实际使用中，还接入了笔者所在公司的其它业务系统，如面向Android平台最终发布打包接入的蜂鸟SDK，发布外网及CDN的数传系统。与其它系统配合也充分体现出系统自动化流程处理的扩展性。

## 6.2 项目应用情况介绍

此套系统已经运用在笔者参与的三款正式游戏项目中，其中一款基于Unity3D 的4.7.2版本开发，另两款则分别是基于Unity3D的5.5.3及5.6.3版本开发。其中两款已经上线正式运营，整个开发、发布运营的过程中此套系统发挥了重要的作用。

笔者所在游戏项目的部门人员主要划分为策划组、技术组、美术组、测试组，同时还与公司其它部门如平台部门、运营部门进行配合。目前基于此套自动化构建发布系统，可以良好的完成各个部门人员对产品的体验、开发、测试、更新、发布需求，大大缓解开发人员的更新、发布的任务压力。

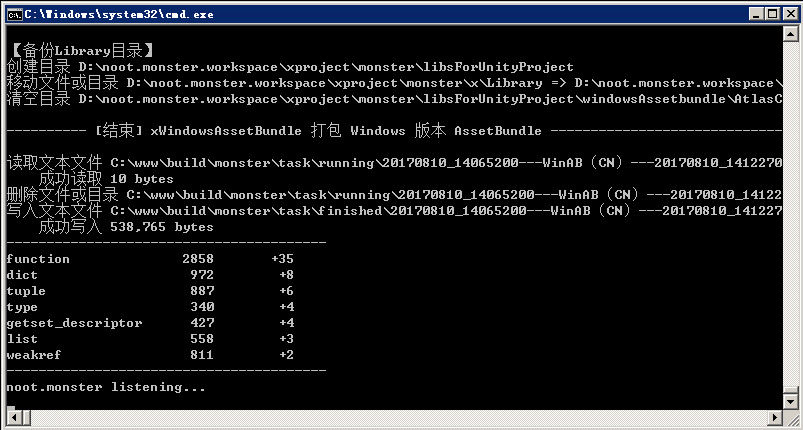


图6-1 系统Windows服务端运行图

日常的开发场景是：技术人员专心开发相关功能，测试人员根据反馈自行打包发布测试，策划在完成某个功能数据配置并提交后直接进行打包更新在手机上查看效果，美术则对某张图片或模型进行调整后自行发布更新，然后便可在手机上对比PC的表现的异同。有正式发布，更新需求时，只要相关程序，资源准备完毕并通过测试后，即可在固定的时间由相关运营同事点击发布或者由系统自动定时发布出去。

早期系统还提供定时任务，设定下午及晚上自动发布资源更新包。不过在后面的实际使用中，由于发布的客户端需要与服务端严格匹配对应，而服务端调整也较为频繁，发布更新还需要部分人工参与，不能与设定的自动发布时间良好配合，所以取消此功能，本论文框架与实现介绍中也未触及。

下列是系统实际运行的截图，图6-2蓝色表明当前开启可执行指令，灰色表明当前关闭不能执行指令。



图6-2 系统指令列表截图



图6-3 系统执行与等待任务列表截图



图6-4 系统完成任务列表截图

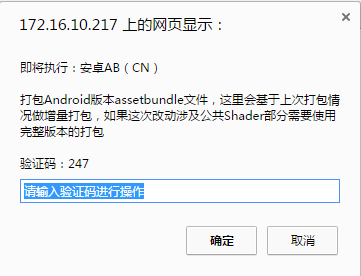


图6-5 防止误操作的验证框截图



图6-6 运行日志网页截图之一

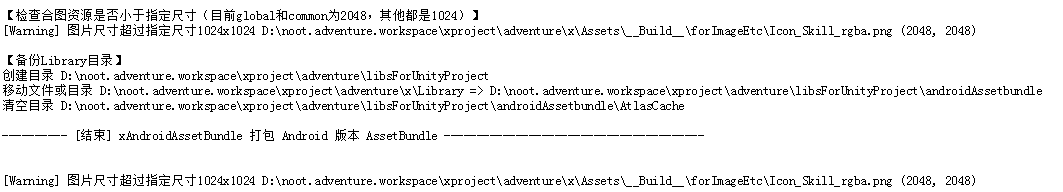


图6-7 运行日志网页截图之二

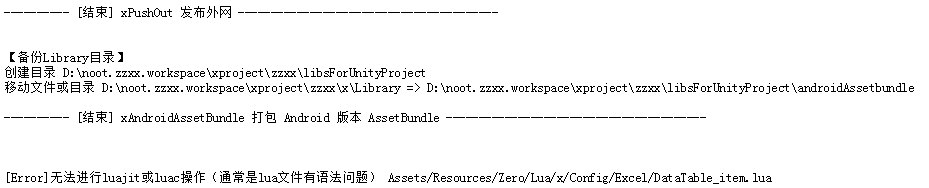


图6-8 运行日志网页截图之三



图6-9 编辑配置网页截图

## 6.3 小结

本章描述了自动化构建系统相关应用情况，补充介绍项目涉及的其它功能模块配置和接入。通过实际案列与系统运作系列截图展示其对项目的推进作用。

# 总结与展望

整个系统在独立的计算机上运行，采用网页交互的方式获取用户指令进行构建发布，对项目的需求处理、资源检测、资源打包、脚本编译、集成发布一系列任务进行全自动化操作，为项目持续集成、迭代开发提供良好支持。

在逐步总结、搭建、使用此套系统的过程中经历了Unity3D版本的替换更新支持，移动平台升级与外部应用市场的各类应用开发规则调整，应用资源及项目特性对应的支持和错误检测等调整。同时，持续收集项目成员使用、反馈意见对其改进，这些在本文中都有相关阐述。

目前系统也遗留了一些优化改进的地方，总结如下：

1. 网页的自动刷新提示。利用网页脚本询问任务完成情况，自动更新当前状态，省去手动刷新获知状态的步骤。此外还可与通讯软件（如腾讯的办公通讯软件RTX）的结合，更实时的通知用户任务完成情况。
2. 定时预备打包。如上所述，Unity3D在处理资源调整更新时往往比较耗时，通过设置定时启动Unity3D对资源进行处理，可以节省此处开销，加快后续打包时间。
3. 图片压缩的多进程、分布式加速。图片格式的压缩处理也是目前观察到的一个消耗资源、时间的‘大户’，但图片也具备数量多可分批处理的特点，利用此特性，用多进程甚至多台机器并行处理各个图片，可以极大缩短压缩处理时间。
4. 增加指定SVN版本构建发布功能。
5. 将目前网页服务器替换成纯Python脚本服务。这样可简化自动化构建发布系统的开发运行环境，只通过Python脚本语言进行相关驱动。不过此前提是构建在目前系统整体对网页服务要求不高的情况之下。
6. 适应性调整，例如针对不需要服务端的项目重新配置定时处理功能。

另外要推进在其它项目组应用，让更多使用Unity3D的开发人员受益。同时对多个项目构建发布需求充分发挥并行特性，充分发挥自动化构建系统所在服务器性能，研究探讨多计算机备份、扩展的支持。

# 参考文献

1. 张帆.Unity3D游戏开发基础.浙江工商大学出版社,2013.31-62
2. 陈嘉栋.Unity3D脚本编程--使用C#语言开发跨平台游戏,2016.22-57
3. Wikipedia.Mono.https://zh.wikipedia.org/zh-cn/Mono,2017
4. Wikipedia.MonoDevelop.https://zh.wikipedia.org/wiki/MonoDevelop,2017
5. Unity Technologies.Unity 5.X从入门到精通.中国铁道出版社,2016.1-4
6. 商宇浩,李一帆,张吉祥.Unity 5.x完全自学手册.电子工业出版社,2016.1-16
7. Unity.CacheServer.https://docs.unity3d.com/Manual/CacheServer.html,2017
8. Wikipedia.Android.https://zh.wikipedia.org/wiki/Android,2017
9. 韩超,梁泉.Android系统原理及开发要点详解.北京:电子工业出版社,2010.1
10. Wikipedia.iOS.https://zh.wikipedia.org/zh-cn/IOS,2017
11. Wikipedia.迭代式开发.https://zh.wikipedia.org/wiki/迭代式开发,2013
12. John W.Statzinger,Robert B.Jackson.系统分析与设计:敏捷迭代方法(原书第6版).北京:机械工业出版社,2017.21-23
13. Wikipedia.Web server. https://en.wikipedia.org/wiki/Web\_server,2017
14. Ken Coar,Ricb Bowen.Apache Cookbook(2nd Edition).O’Reilly Media Inc,

2007.27-41

1. julie C. Meloni.PHP、MySQL和Apache入门经典(第5版).北京:人民邮电出版社,2013.42-52
2. Python Software Foundation.The Zen of Python.

<https://www.python.org/dev/>peps/pep -0020

1. Wesley J. Chun.Python核心编程（第二版）.北京:人民邮电出版社,2008.7
2. Mark Pilgrim.Dive Into Python.APress Media LLC,2005.95-115
3. James Payne.Beginning Python:Using Python 2.6 and Python 3.1.

Wiley Publishing Inc,2010.7-13

1. Marius Gedminas.Python Object Graphs.http://mg.pov.lt/objgraph,2016
2. David Flanagan.JavaScript权威指南(原书第6版).北京:机械工业出版社,2012
3. jQuery Community Experts.jQuery Cookbook.O’Reilly Media Inc,2009.1-4
4. Apache Software Foundation.Apache Subversion.https://subversion.apache.org
5. Andrew C. Beers,Maneesh Agrawala,Navin Chaddha.

Rendering from Compressed Textures.

Computer Graphics,Proc.SIGGRAPH,1996:373-378

1. 伍逸.XML实用技术自学经典.北京:清华大学出版社,2016.2
2. 孙更新,李玉玲.XML编程与应用数据.第三版.北京:清华大学出版社,2017.1-7
3. 张效祥.计算机科学技术百科全书.第二版.北京:清华大学出版社,2005.985
4. 米切尔.程序语言设计概念.北京:清华大学出版社,2005.389
5. 西尔伯沙茨.数据库系统概念（原书第六版）.第一版.北京:机械工业出版社，2012.355-371
6. Wikipedia.CDN.https://en.wikipedia.org/wiki/CDN,2017
7. Eric Freeman,Elisabeth Freeman,Kathy Sierra,Bert Bates.Head First设计模式.中国电力出版社,2007.277-312
8. Romain Bourdon.wampserver.http://www.wampserver.com/en,2017
9. David Beazley,Brian K.Jones.Python Cookbook(3rd Edition).O’Reilly Media Inc,2013.398
10. Janusz Januszewski.Packing rectangles into a large square.

Periodica Mathematica Hungarica,2016,72(1):90-101

1. Alan Zucconi,Kenneth Lammers.Unity 5.x Shaders and Effects Cookbook.Packt Publishing Ltd.,2016.199-216

# 致谢

从开始使用Unity3D开发应用到整理构建发布需求，系统地提出解决方案并实现和改进，这个过程花费了两年多的时间。期间在学校学习过程中获得不少可供类比的知识点和可参考的宝贵资料，这些离不开导师，各位授课老师及教务员的辛勤工作和孜孜教诲。特别是在准备和撰写论文的这段时间，导师帮忙分析选题，指导框架改进，为当时在运行测试的系统提供帮助，并持续提升了系统的可扩展、可维护性。

感谢同学们一直以来的关心和鼓励，帮忙收集资料，和我一起携手进步。感谢公司，在这里从入门学习Unity3D到开发三款正式上线的游戏，都依赖公司的支持。感谢一起工作奋斗的同事，帮助测试和反馈系统的应用情况，并对系统涉及的繁多任务提供技术支持，加速系统的完善。那些为游戏测试、上线发布的日日夜夜我会一直铭记于心。

感谢亲爱的家人一直以来对我的照顾和理解。工作和学习占据了我大部分的时间，家庭全依靠你们的照料，衷心感恩。

1. [] 张帆.Unity3D游戏开发基础.浙江工商大学出版社,2013.31-62 [↑](#endnote-ref-0)
2. [] 陈嘉栋.Unity3D脚本编程--使用C#语言开发跨平台游戏,2016.22-57 [↑](#endnote-ref-1)
3. [] Wikipedia.Mono.https://zh.wikipedia.org/zh-cn/Mono,2017 [↑](#endnote-ref-2)
4. [] Wikipedia.MonoDevelop.https://zh.wikipedia.org/wiki/MonoDevelop,2017 [↑](#endnote-ref-3)
5. [] Unity Technologies.Unity 5.X从入门到精通.中国铁道出版社,2016.1-4 [↑](#endnote-ref-4)
6. [] 商宇浩,李一帆,张吉祥.Unity 5.x完全自学手册.电子工业出版社,2016.1-16 [↑](#endnote-ref-5)
7. [] Unity.CacheServer.https://docs.unity3d.com/Manual/CacheServer.html,2017 [↑](#endnote-ref-6)
8. [] Wikipedia.Android.https://zh.wikipedia.org/wiki/Android,2017 [↑](#endnote-ref-7)
9. [] 韩超，梁泉.Android系统原理及开发要点详解.北京:电子工业出版社,2010.1 [↑](#endnote-ref-8)
10. [] Wikipedia.iOS.https://zh.wikipedia.org/zh-cn/IOS,2017 [↑](#endnote-ref-9)
11. [] Wikipedia.迭代式开发.https://zh.wikipedia.org/wiki/迭代式开发,2013 [↑](#endnote-ref-10)
12. [] John W.Statzinger,Robert B.Jackson.系统分析与设计:敏捷迭代方法(原书第6版).北京:机械工业出版社,2017.21-23 [↑](#endnote-ref-11)
13. [] Wikipedia.Web server. https://en.wikipedia.org/wiki/Web\_server,2017 [↑](#endnote-ref-12)
14. [] Ken Coar,Ricb Bowen.Apache Cookbook(2nd Edition).O’Reilly Media Inc,2007.27-41 [↑](#endnote-ref-13)
15. [] julie C. Meloni.PHP、MySQL和Apache入门经典(第5版).北京:人民邮电出版社,2013.42-52 [↑](#endnote-ref-14)
16. [] Python Software Foundation.The Zen of Python. https://www.python.org/dev/peps/pep-0020 [↑](#endnote-ref-15)
17. [] Wesley J. Chun.Python核心编程（第二版）.北京:人民邮电出版社,2008.7 [↑](#endnote-ref-16)
18. [] Mark Pilgrim.Dive Into Python.APress Media LLC,2005.95-115 [↑](#endnote-ref-17)
19. [] James Payne.Beginning Python:Using Python 2.6 and Python 3.1.Wiley Publishing Inc,2010.7-13 [↑](#endnote-ref-18)
20. [] Marius Gedminas.Python Object Graphs.http://mg.pov.lt/objgraph,2016 [↑](#endnote-ref-19)
21. [] David Flanagan.JavaScript权威指南(原书第6版).北京:机械工业出版社,2012 [↑](#endnote-ref-20)
22. [] jQuery Community Experts.jQuery Cookbook.O’Reilly Media Inc,2009.1-4 [↑](#endnote-ref-21)
23. [] Apache Software Foundation.Apache Subversion.https://subversion.apache.org [↑](#endnote-ref-22)
24. [] Andrew C. Beers,Maneesh Agrawala,Navin Chaddha.Rendering from Compressed Textures.Computer Graphics,Proc.SIGGRAPH,1996:373-378 [↑](#endnote-ref-23)
25. [] 伍逸.XML实用技术自学经典.北京:清华大学出版社,2016.2 [↑](#endnote-ref-24)
26. [] 孙更新，李玉玲.XML编程与应用数据.第三版.北京:清华大学出版社,2017.1-7 [↑](#endnote-ref-25)
27. [] 张效祥.计算机科学技术百科全书.第二版.北京:清华大学出版社,2005.985 [↑](#endnote-ref-26)
28. [] 米切尔.程序语言设计概念.北京:清华大学出版社,2005.389 [↑](#endnote-ref-27)
29. [] 西尔伯沙茨.数据库系统概念（原书第六版）.第一版.北京:机械工业出版社，2012.355-371 [↑](#endnote-ref-28)
30. [] Wikipedia.CDN.https://en.wikipedia.org/wiki/CDN,2017 [↑](#endnote-ref-29)
31. [] Eric Freeman,Elisabeth Freeman,Kathy Sierra,Bert Bates.Head First设计模式.中国电力出版社,2007.277-312 [↑](#endnote-ref-30)
32. [] Romain Bourdon.wampserver.http://www.wampserver.com/en,2017 [↑](#endnote-ref-31)
33. [] David Beazley,Brian K.Jones.Python Cookbook(3rd Edition).O’Reilly Media Inc,2013.398 [↑](#endnote-ref-32)
34. [] Janusz Januszewski.Packing rectangles into a large square.Periodica Mathematica Hungarica,2016,72(1):90-101 [↑](#endnote-ref-33)
35. [] Alan Zucconi,Kenneth Lammers.Unity 5.x Shaders and Effects Cookbook.Packt Publishing Ltd.,2016.199-216 [↑](#endnote-ref-34)