分布式自动化版本发布系统

目　　录

第1章　分布式自动化版本发布系统

1.1. 概述

在页游或手游行业，游戏的版本发布是运维日常工作主要部分，在短暂的停机维护时间内要将游戏的新版本发布到几十台甚至数百台服务器中。这种工作强度下，运维必须采用一些自动化运维的手段，在实现自动化运维时，不同的团队会采用不同的技术，大致如下：

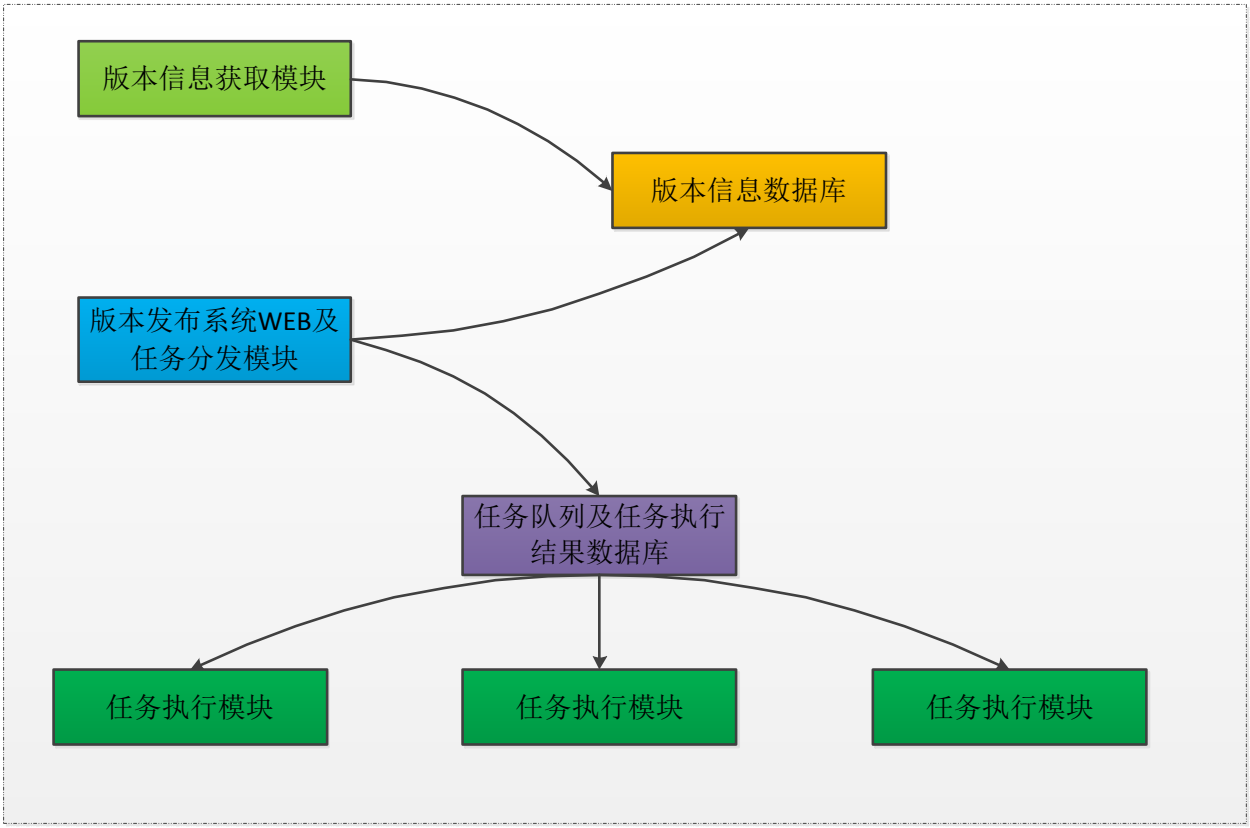
1. 采用 puppet、chef、func、salt 等开源工具
2. 写一些 shell 或 python 脚本
3. 开发自己的自动化运维系统

笔者采用的是第 3 种方式，理由是：

1. 第 3 方开源工具的安装、配置、维护比较麻烦，有一定的学习成本，且功能不太灵活；
2. 第 2 种方式需要登陆一台运维服务器，根据需求输入一些命令，对其他 SA、运营、策划的同学有一定的学习成本，且容易发生误操作；
3. 自主开发自动化运维系统，对团队的开发水平有一定的要求，但是灵活性高，可以根据业务情况开发出非常适合自己公司业务需要的工具，提供友好的 WEB 操作界面的话，可以招一些水平较差的 SA 来培养，降低人力成本。

1.2. 架构说明

以下为笔者开发的自动化运维工具的架构图：

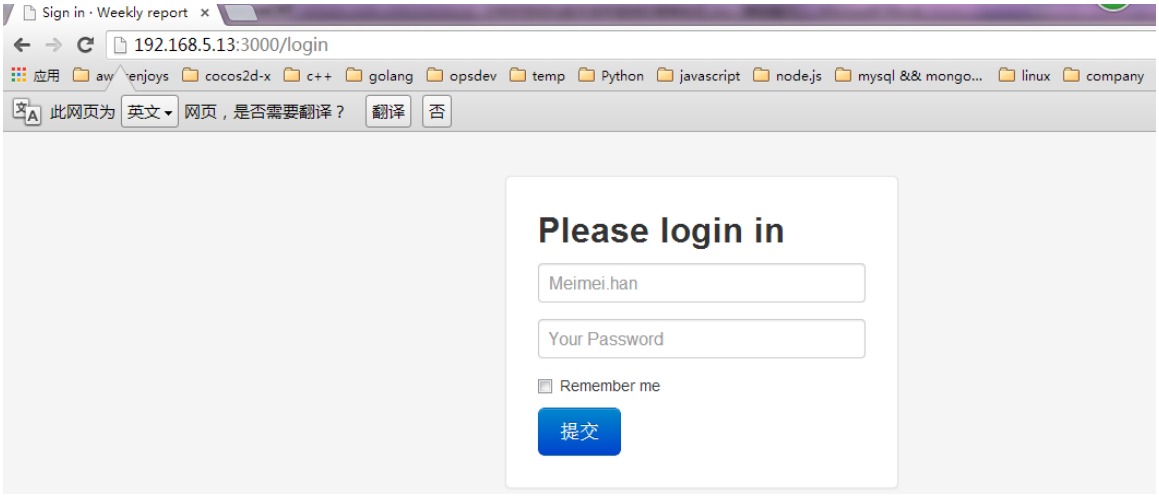


架构说明：

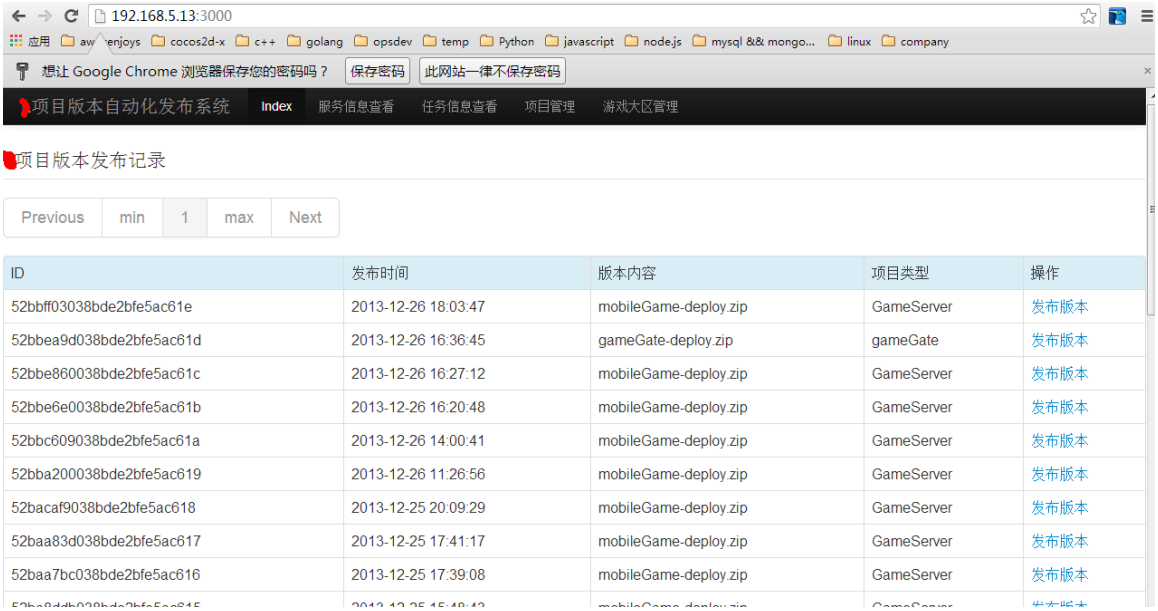
1. 版本信息获取模块的功能是从 SVN 或 maven 中每隔 1 分钟获取一次版本信号，然后将版本信息存入数据库；
2. 版本发布系统提供一个 WEB 界面，用户可以进行版本发布、服务器开、关、重启、项目录入、游戏大区录入等操作，操作指令将作为任务存入任务队列；
3. 任务队列数据库将保存每个任务信息及执行结果
4. 任务执行模块会实时从任务队列中拉取任务信息并执行，可以部署 1 个或多个，加快任务执行速度。

1.3. 系统 UI

在使用前需要通过域账户登陆：

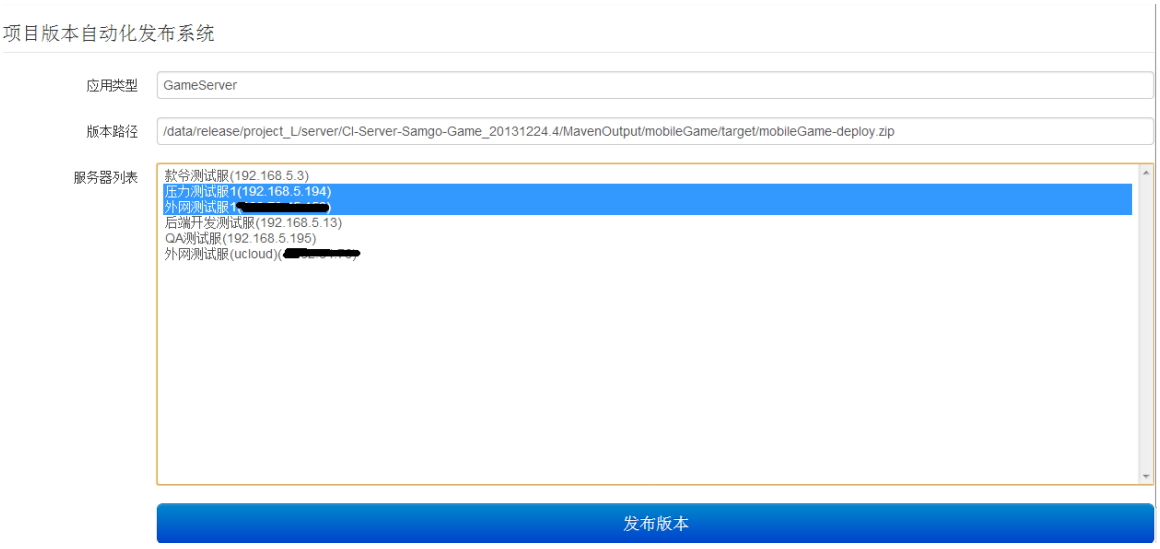


登陆后如下所示（这个是一个游戏后端的项目，代码提交后 maven 会自动构建成 zip 文件）：

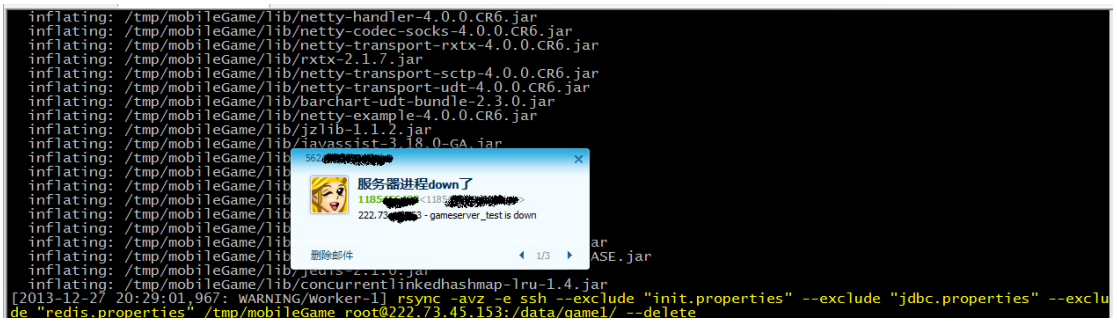


在版本发布前需要录入项目信息及游戏大区信息：

项目和游戏大区录入后就可以使用本系统进行版本发布了，如下图所示：



版本发布时，可以在任务执行服务器中看到详细的发布过程，如下图所示：



版本发布完成后，会自动完成重启，正好在此过程中被监控系统发现，然后邮件报警了。重启任务发出后跳转到服务管理页面，如下图所示：

项目服务器运行状态

p	大区名	进程名	进程组	运行状态	进程信息	操作					
192.1[redacted]	gameGate	gameGate	RUNNING	pid 23955, uptime 8 days, 9:47:13	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
	gameLogin	gameLogin	RUNNING	pid 23977, uptime 8 days, 9:46:49	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
	gameserver_test	gameserver_test	RUNNING	pid 32275, uptime 0:15:40	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
192[redacted]	gameGate	gameGate	RUNNING	pid 25200, uptime 10:03:18	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
	gameLogin	gameLogin	RUNNING	pid 25238, uptime 10:01:49	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
	gameserver_test	gameserver_test	RUNNING	pid 27674, uptime 0:15:21	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
42.[redacted]	gameGate	gameGate	RUNNING	pid 3665, uptime 8:08:19	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
	gameLogin	gameLogin	RUNNING	pid 3695, uptime 8:07:58	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
	gameserver_test	gameserver_test	RUNNING	pid 3724, uptime 8:07:52	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
192.16[redacted]	gameGate	gameGate	RUNNING	pid 2578, uptime 1 day, 6:15:03	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
	gameLogin	gameLogin	RUNNING	pid 2391, uptime 1 day, 6:17:41	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
	gameserver_test	gameserver_test	RUNNING	pid 3521, uptime 0:00:30	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	
[redacted]	gameGate	gameGate	RUNNING	pid 28395, uptime 1 day, 4:25:04	启动	停止	重启	ProcessLog	StdoutLog	StderrLog	

第2章　实现原理

2.1. 版本信息获取模块

2.1.1.SVN 版本库

SVN 版本库的检查可以使用 pysvn 模块，代码片段如下：

```
#检查 SVN 的更新情况
def project_svn(path, team):
        #pathA = "/data/release/tech"
        project = path['name']
        ret = svnchanges(path['path'])

        old_rev = ret['old_rev']
        new_rev = ret['new_rev']
        del ret['old_rev']
        del ret['new_rev']

        svnupstr =  '「%s」updated from %s to %s.' % (project, old_rev, new_rev)
        print svnupstr
        values    = {}
```

```
        if int(new_rev) > int (old_rev):
                entry = svninfo(path['path'])
                print entry
                for i in entry:
                        print i

                messagea = u"SVN 版本号：%s\n" % entry.commit_revision.number
                messagea += u"提交者：%s\n" % entry.commit_author
                messagea += u"更新日期：%s\n\n" %
str(datetime.datetime.fromtimestamp(entry.commit_time))[:-7]
                messagea += u"更新内容：\n"

                values['project']         = path['name']
                values['revision']        = entry.commit_revision.number
                values['author'] = entry.commit_author
                values['date']            =
str(datetime.datetime.fromtimestamp(entry.commit_time))[:-7]
                values['content']         = ""
```

以下为一个使用该功能的另一个系统的截图：

项目列表

「运维」
「project_[redacted]」
「project_[redacted]」
「project_[redacted]」

更新记录

Previous | min | 3 | max | Next

项目信息		提交内容
项目名	[redacted]	前端资源/美术资源/技能动画/SK011/SK011/atk [added] 前端资源/美术资源/技能动画/SK011/SK011/atk/0003.png [added] 前端资源/美术资源/技能动画/SK011/SK011/atked [delete] 前端资源/美术资源/技能动画/SK011/SK011/atk/0001.png [added] 前端资源/美术资源/技能动画/SK011/SK011/atk/0009.png [added] 前端资源/美术资源/技能动画/SK011/SK011/atk/0008.png [added] 前端资源/美术资源/技能动画/SK011/SK011/atk/0007.png [added] 前端资源/美术资源/技能动画/SK011/SK011/atk/0006.png [added] 前端资源/美术资源/技能动画/SK011/SK011/atk/0005.png [added] 前端资源/美术资源/技能动画/SK011/SK011/atk/0004.png [added] 前端资源/美术资源/技能动画/SK011/SK011/atk/0010.png [added] 前端资源/美术资源/技能动画/SK011/SK011/atk/0002.png [added]
版本号	1170	
提交者	[redacted]	
提交日期	2013-11-23 16:31:58	

2.1.2.Maven

我们的项目代码是提交到 TFS 中，然后通过 maven 构建的，构建出来会保存到 windows 中的一个目录中，然后通过 python 脚本挂载 Windows 的目录，并用 rsync 把新构建的版本信息存入数据库，代码片段如下 ：

```
#coding:utf8

import os
import datetime
import pymongo
import time
#-------------------------------------------------------------------------
--
#同步 maven 编译过的版本到本地服务器
def synsfile():
    #卸载挂载过的磁盘
    umount = "umount -a"
    os.system(umount)

    #挂载 TFS 中的服务器端发布目录
    mount = "mount.cifs -o username=xxxx.zhao,domain=xxxx.com,password=mypas
word //111.111.111.111/output/server /mnt/projectl_server/"
    os.system(mount)

    local = "/mnt/projectl_server/"
    dest = "/data/release/project_L/server"
    rsync = 'rsync -avz --include "*/" --include "*.zip" --exclude "*/*" --ex
clude "*" %s %s' % (local, dest)
    #print rsync
    os.system(rsync)

    #分到到另一台版本发布服中
    local = "/data/release/project_L/server/"
    rsync_cmd = 'rsync -avz -e ssh  %s root@%s:%s --delete' % (local, '192.16
8.5.14', dest)
```

```
    print rsync_cmd
    os.system(rsync_cmd)



#-------------------------------------------------------------------------
---
#列出版本内容
def listversion(path, verpath):
    versionlist = os.listdir(path)
    version = []
    for ver in versionlist:
        version.append(ver + verpath)

    #print version
    return version
#-------------------------------------------------------------------------
---
#更新记录入库
def InsertDB(values):
    conn = pymongo.Connection('111.111.111.111', 27017)
    version_db = conn.version
    release_record = version_db.release_record
    release_record.update({'path' : values['path']}, values, True)




if __name__ == '__main__':
    while True:
        path = "/data/release/project_L/server/"
        verpath = ["/MavenOutput/mobileGame/target/mobileGame-deploy.zip","/
MavenOutput/gameGate/target/gameGate-deploy.zip","/MavenOutput/gameLogin/t
arget/gameLogin-deploy.zip"]
        synsfile()
        for item in verpath:
            version = listversion(path, item)
```

```
            versionlist = []
            for ver in version:
                verpath = path + ver
                try:
                    create_time = os.stat(verpath). st_ctime
                except Exception, e:
                    #print verpath
                    pass
                else:
                    #CI-Server-Samgo-Center_20131213.7/MavenOutput/gameLogin/t
arget/gameLogin-deploy.zip
                    if "mobileGame" in ver:
                        version_type = "GameServer"
                    elif "gameGate" in ver:
                        version_type = "gameGate"
                    elif "gameLogin" in ver:
                        version_type = "gameLogin"
                    else:
                        version_type = "Unknown"

                    verinfo = dict(
                        path = verpath,
                        version_type = version_type,
                        create_time = datetime.datetime.fromtimestamp(create_ti
me)
                        )
                    versionlist.append(verinfo)
                    InsertDB(verinfo)
                #print verinfo

          #versionlist.reverse()
          #print versionlist
        time.sleep(60)
```

2.2. 版本信息数据库

版本信息是存入 mongodb 中的，就不用说了，上面有入库的代码。

2.3. 版本发布系统及任务分发模块

版本发布系统是用 Tornado + mongodb 做的，全部代码有 400 行左右，就不全贴了，只贴一下版本发布功能的代码片段：

```
#-------------------------------------------------------------------------------
   ----
```

```
#版本发布
class ReleaseHandler(tornado.web.RequestHandler):
    def get(self):
        conn = pymongo.Connection('111.111.111.111', 27017)
        version_db = conn.version
        areas = version_db.areas
        username = self.get_cookie("username", "")
        print username
        if not username:
            self.redirect("/login")
        else:
            filename = self.get_argument("v", "")
            v_type = self.get_argument("v_type", "")

            areas_info = areas.find({"app_type" : v_type})
            print filename
            if filename:
               self.render("release.html", filename=filename, v_type=v_type,
areas_info=areas_info)
            else:
                self.redirect("/")

    def post(self):
        username = self.get_cookie("username", "")
        print username
        if not username:
```

```
            self.redirect("/login")
        else:
            filename = self.get_argument("v", "")
            version_type = self.get_argument("v_type", "")
            ids = self.get_arguments("_id")
            print ids

            conn = pymongo.Connection('111.111.111.111', 27017)
            version_db = conn.version
            areas = version_db.areas
            projects = version_db.projects
            print version_type
            projects_info = projects.find({'project_code' : version_type})
            for project_info in projects_info:
                local = project_info['project_path']
                print project_info, local

            for _id in ids:
                area = areas.find({'_id' : bson.ObjectId(_id)})
                for item in area:
                    print item,type(item)
                    server_ip = item['server_ip']
                    process_path = item['process_path']
                    process_name = item['process_name']
                    print filename, server_ip, process_path

                if filename:
                    result = release_version.delay(server_ip, filename, local,
 process_path, process_name)
                    print result.get(timeout=300)
                    #重启服务
                    result = services_manager.delay(server_ip, "restart", proc
ess_name)
                    print result.get(timeout=120)
```

```
            self.redirect("/serverinfo/")
```

大家也许注意到了，版本发布与重启时只有以下几行代码：

```
result = release_version.delay(server_ip, filename, local, process_path, pr
ocess_name)
                    print result.get(timeout=300)
                    #重启服务
                    result = services_manager.delay(server_ip, "restart", proc
ess_name)
                    print result.get(timeout=120)
```

这些代码是调用任务执行模块的函数，接下来会简单说一下任务执行模块的实现。

2.4. 任务执行模块

任务执行模块使用的采用的是 celery，以下是开源中国对 celery 的简单说明：

Celery (芹菜)是一个异步任务队列/基于分布式消息传递的作业队列。它侧重于实时操作，但对调度支持也很好。celery 用于生产系统每天处理数以百万计的任务。celery 是用 Python 编写的，但该协议可以在任何语言实现。它也可以与其他语言通过 webhooks 实现。

Celery 可以执行实时任务，可以指定在特定时间内执行任务，也可以指定再过多少时间执行任务，以下为 pycon 2013 中叶剑烨大牛演讲中 celery 使用场景的一个截图，对于运维来说，可以把维护的工作，实时或计划性的维护加入队列中自动完成，运维只要关注下任务是否成功执行即可。

如果一个 worker 发布版本太慢时，可以在外网的游戏服务器中启动几个 worker，进行分布式版本发布等操作，提高效率。

服务进程管理使用了 supervisor，celery 的 broker 使用了 redis。

下面贴一下发布版本及进程管理的代码片段：

```
# -*- coding:utf8 -*-
from celery import Celery
import os
import xmlrpclib
import tornado.httpclient
from celery.task.http import URL

app = Celery('task', backend='redis://111.111.111.111:6379/0',
broker='redis://111.111.111.111:6379/0')

#-------------------------------------------------------------------------
-----
#版本发布任务
@app.task
def release_version(server_ip, v_filename, local_path, dest_path,
process_name):
        #解压版本 zip 包到/tmp 目录中
        unzip_cmd = "unzip -o %s -d /tmp/" % v_filename
        os.system(unzip_cmd)

        #发布版本到指定的服务器
        rsync_cmd = 'rsync -avz -e ssh --exclude "init.properties" --exclude
"jdbc.properties" --exclude "redis.properties" %s root@%s:%s --delete' %
(local_path, server_ip, dest_path)
        print rsync_cmd
        os.system(rsync_cmd)

#-------------------------------------------------------------------------
-----
#服务进程管理任务
@app.task
```

```
def services_manager(server_ip, active, process_name):
        rpcaddr = 'http://%s:9001/RPC2' % server_ip
        server = xmlrpclib.Server(rpcaddr)

        if active == "start":
                server.supervisor.startProcess(process_name)

        if active == "stop":
                server.supervisor.stopProcess(process_name)

        if active == "restart":
                server.supervisor.stopProcess(process_name)
                server.supervisor.startProcess(process_name)
```

打完收工，行文仓促，如有问题可与我交流。

邮件：netxfly#hotmail.com