Ducter任务与操作脚本

编写及使用指南

**(V1.0)**

*变更记录*

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **状态** | **描述** | **作者** |
| 2014-12-10 | V0.1 | 初稿 | cwinux |

目录

[1 简述 4](#_Toc408836198)

[2 任务脚本 4](#_Toc408836199)

[2.1 任务脚本说明 4](#_Toc408836200)

[2.2 任务脚本的运行环境 5](#_Toc408836201)

[2.3 任务脚本的执行结果文件 6](#_Toc408836202)

[2.4 任务最佳实践 7](#_Toc408836203)

[2.4.1 服务软件包存储 7](#_Toc408836204)

[2.4.2 服务的安装位置 8](#_Toc408836205)

[2.4.3 实现实例 9](#_Toc408836206)

[3 操作脚本 21](#_Toc408836207)

[3.1 操作脚本说明 21](#_Toc408836208)

[3.2 操作脚本例子： 21](#_Toc408836209)

[3.2.1 获取服务器运行进程信息脚本：ps 21](#_Toc408836210)

[3.2.2 获取服务器/proc下信息的脚本：proc\_info 22](#_Toc408836211)

## 简述

在产品与服务器的控制与操作上，Ducter只是提供了命令执行的通道、策略及执行的环境，而具体的命令动作完全是由用户自己编写的任务脚本或操作脚本实现的。

本手册详细介绍ducter为任务、操作脚本提供的执行环境及编写要求，以便使用者可以正确、快速的编写自己项目所需要的脚本。

## 任务脚本

任务的本质是为用户提供在一个服务器集群上安全、可靠、可视、自动执行命令的平台。

执行的命令由用户采用各种脚本语言开发，可以用作服务重启、上线等各种目的。而Ducter任务只是按照用户的要求，调度任务脚本在不同服务器上有次序自动执行，并为脚本执行提供所需要的运行环境。

下面从任务脚本的运行环境、任务脚本输出、任务最佳实践三个方面描述任务脚本的编写。

### 任务脚本说明

* 任务指令名称：

任务指令名称为[a-z,A-Z,0-9,\_]范围内的字符。

* 任务指令script文件：

dcmd\_task\_【任务指令名称】.script，其保存在dcmd\_center所在的服务器上，**存放位置由dcmd\_center配置文件的【common：task\_script\_path】的参数项指定**

* 任务指令的参数：

参照【**任务脚本的运行环境**】

* 任务指令的输出：

参照【**任务脚本的执行结果文件**】

### 任务脚本的运行环境

Ducter通agent完成任务脚本在服务器上的执行，而且是按照【服务】所指定的OS用户执行。

Agent在服务器上执行任务脚本的时候，会清空所有的OS环境变量信息，仅仅提供如下的环境变量供任务脚本使用

* DCMD\_APP\_NAME

产品名字，可由【字母】、【数字】、【\_】组成的任何字符。

* DCMD\_SVR\_ENV\_V

服务池子的环境版本，可由【字母】、【数字】、【\_】组成的任何字符。

其在服务的**服务池**中设定。

* DCMD\_SVR\_IP：

执行任务的注册服务器的IP地址

* DCMD\_SVR\_NAME：

服务的名字，可由【字母】、【数字】、【\_】组成的任何字符

* DCMD\_SVR\_NODE\_MUTIL\_POOL：

服务是否允许在一个设备上有多个服务池。1：允许；0：不允许

其在**服务**中设定

* DCMD\_SVR\_PATH：

服务在设备上的安装路径，此为OS的绝对路径。

其在**服务**中设定

* DCMD\_SVR\_POOL：

服务池的名字，可由【字母】、【数字】、【\_】组成的任何字符

此为设备所属的服务池。

* DCMD\_SVR\_PROCESS：

是否实时输出执行进度，在**创建任务**时指定

* DCMD\_SVR\_REPO：

服务的版本库地址，在**服务池**中设定。

此可以是svn、http、ftp、git等等的地址

* DCMD\_SVR\_RESULT\_FILE：

任务执行结果文件名，由ducter agent指定

* DCMD\_SVR\_UPDATE\_ENV

是否无条件更新服务的环境配置信息，在**创建任务**时指定

* DCMD\_SVR\_UPDATE\_TAG：

是否无条件更新服务软件包，在**创建任务**时指定

* DCMD\_SVR\_USER：

任务执行的OS账号，在**服务**中设定

* DCMD\_SVR\_VERSION

当前任务对应的服务的版本号，可由【字母】、【数字】、【\_】组成的任何字符

在**创建任务**时指定

* DCMD\_SYS\_AGENT\_PID：

当前ducter agent的进程ID

* DCMD\_SYS\_AGENT\_PPID：

当前ducter agent的父进程ID

* DCMD\_TASK\_xxxx：

用户脚本定义的参数xxxx，值在**创建任务**时指定，任务模板可以设置缺省值

值可由【字母】、【数字】、【\_】组成的任何字符

* DCMD\_TASK\_POOL\_yyyy：

服务池的属性yyyy，值在**服务池**属性中指定

值可由【字母】、【数字】、【\_】组成的任何字符

### 任务脚本的执行结果文件

任务脚本是否执行成功，是通过任务脚本执行输出结果文件指定的。

任务脚本的执行结果文件，是由任务脚本的环境变量$DCMD\_SVR\_RESULT\_FILE指定。

$DCMD\_SVR\_RESULT\_FILE环境变量指定文件的内容如下：

* + - * process=进度

此为任意的字符串，必须在一行。可以多次输出，最后一次输出的值为真正的进度值。

* + - * state=success/failure

输出state表示执行状态；success：成功；failure：失败。必须在一行。

* + - * err=错误信息

若任务执行失败，则记录错误信息。错误信息可以多行。

### 任务最佳实践

#### 服务软件包存储

要通过ducter的任务实现【服务】的发布，必须有地方存储待发布的服务软件包，这就牵扯上线服务软件包的存储问题。

由于ducter的任务脚本可由用户自由编写，因此，ducter对软件包的存储及获取方式没有任何限制，可以通过svn、ftp、http、git、甚至共享存储的方式获取。同样对服务软件包的命名也没有限制。

但，基于ducter的设计，无论采用哪种方式获取软件包，ducter建议采用如下的方式进行服务软件包的存储及软件包的命名，以便根据已知信息获取软件包与配置的位置：

--软件包存储根目录

|---package ##软件包的存储

| |---产品名字A

| | |--服务名字A-1

| | | |--【服务名】\_【OS版本】\_【服务版本号1】.tar.gz

| | | |--【服务名】\_【OS版本】\_【服务版本号2】.tar.gz

| | |--服务名字A-2

| |---产品名字M

| | |--服务名字M-1

| | | |--【服务名】\_【OS版本1】\_【服务版本号1】.tar.gz

| | | |--【服务名】\_【OS版本2】\_【服务版本号1】.tar.gz

| | |--服务名字A-2

|---conf

|---产品名字A

|---服务名字A-1

|---具体服务池子1

| |---env1 //env1的配置目录，内放各配置文件

| |--env2 //env2的配置目录，内放各配置文件

|---具体服务池子2

通过以上的存储目录结构，任务脚本根据传入的Repository、产品名、服务名、软件包版本号、一级服务器的OS类型，就可以自动计算出服务软件包的存储位置。

同样，也可以计算出服务配置文件的目录。

#### 服务的安装位置

Ducter自身对于服务的服务器安装位置没有任何特殊要求。但为了实现所有的服务都采用一个任务脚本实现发布，建议服务在线上服务器的安装位置及目录结构如下：

---Base Path

|---产品名 ##产品目录

| |--服务名【service\_pool】 ##服务目录，若服务允许同一个设备有多

| | | ##个服务池，则需要加上服务池名的目录

| | |--bin ##服务的bin目录

| | | |--start.sh ##服务的启动文件

| | | |--stop.sh ##服务的停止文件

| | | |--install.sh ##服务的安装文件

| | | |--其他文件 ##服务的其他bin目录下的文件

| | |--服务的其他数据及日志目录

| | |--.install ## 安装相关的目录

| | | |--version：##当前运行的版本文件，内部记录版本号

| | | |--package：##上线时下载的软件包文件，最新N个

| | | | |--【服务名】\_【OS版本】\_【服务版本号1】.tar.gz

| | | | |--【服务名】\_【OS版本】\_【服务版本号2】.tar.gz

| | | |--env\_conf：##当前的conf文件目录

| | | |--tmp: 安装历史目录

|---产品名2 ##产品目录

|--服务名【service\_pool】 ##服务目录，若服务允许同一个设备有多

| ##个服务池，则需要加上服务池名的目录

|--bin ##服务的bin目录

|--.install ## 安装相关的目录

通过这样组织的目录结构，可以通过一个脚本实现所有符合此要求的服务的上线。当前，各个使用者可以根据自己的规范，定义自己服务的线上结构，只要编写自己的任务脚本脚本就可以了。

#### 实现实例

下面的代码是根据上面的规范，实现的agent的上线

##### Agent的start.sh脚本实现

* + 脚本参数：

无

* + 此脚本内容：

#!/bin/sh

killall -9 dcmd\_agent

cd `dirname $0`

./dcmd\_agent -f dcmd\_agent.conf>/dev/null

##### Agent的stop.sh脚本实现

* + 脚本参数：

无

* + 此脚本内容：

#!/bin/sh

#cd `dirname $0`

#./dcmd\_agent -stop

killall -9 dcmd\_agent

##### install.sh

* + 脚本参数：

第一个：安装的目录

第二个：下载解压后安装包的路径

第三个（可选）：下载的配置文件路径

* + 此脚本内容：

内容如下：

#!/bin/sh

if [ $# -lt 2 ]; then

echo "Too less arguement."

exit 1

fi

agent\_path=$1

package\_path=$2

if [ 3 -eq $# ];then

conf\_path=$3

fi

if [ ! -f $package\_path/bin/dcmd\_agent ]; then

echo "[$package\_path/bin/dcmd\_agent] doesn't exist."

exit 1

fi

if [ 3 -eq $# ]; then

if [ ! -f $conf\_path/dcmd\_agent.conf ]; then

echo "[$conf\_path/dcmd\_agent.conf] doesn't exist."

exit 1

fi

fi

if [ -f $agent\_path/bin/dcmd\_agent ];then

rm -f $agent\_path/bin/dcmd\_agent

fi

if [ ! -d $agent\_path/bin ]; then

mkdir $agent\_path/bin

fi

cp $package\_path/bin/dcmd\_agent $agent\_path/bin/dcmd\_agent

ret=$?

if [ "0" -ne $ret ]; then

exit $ret

fi

cp -f $package\_path/bin/start.sh $agent\_path/bin

ret=$?

if [ "0" -ne $ret ]; then

exit $ret

fi

cp -f $package\_path/bin/stop.sh $agent\_path/bin

ret=$?

if [ "0" -ne $ret ]; then

exit $ret

fi

cp -f $package\_path/bin/install.sh $agent\_path/bin

ret=$?

if [ "0" -ne $ret ]; then

exit $ret

fi

chmod 755 $agent\_path/bin/dcmd\_agent

chmod 755 $agent\_path/bin/start.sh

chmod 755 $agent\_path/bin/stop.sh

chmod 755 $agent\_path/bin/install.sh

if [ 3 -eq $# ]; then

if [ -f $agent\_path/bin/dcmd\_agent.conf ];then

rm -f $agent\_path/bin/dcmd\_agent.conf

fi

cp $conf\_path/dcmd\_agent.conf $agent\_path/bin/dcmd\_agent.conf

ret=$?

if [ 0 -ne $ret ]; then

exit $ret

fi

else

if [ ! -f $agent\_path/bin/dcmd\_agent.conf ]; then

if [ -f $package\_path/bin/dcmd\_agent.conf ]; then

cp $package\_path/bin/dcmd\_agent.conf $agent\_path/bin/dcmd\_agent.conf

ret=$?

if [ 0 -ne $ret]; then

exit $ret

fi

fi

fi

fi

exit 0

##### Agent基于svn的install的任务脚本实现

具体参数www.ducter.net的install\_by\_svn任务脚本

* + 脚本参数：

svn\_user：svn的用户名

svr\_passwd：svn的用户口令

is\_os\_package：是否不同的OS版本采用不同的软件包

* + 脚本内容：

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

#install path

#--base\_path/project/service[/service\_pool]: #if [node\_multi\_pool==1],then [service\_pool] is used.

# |--bin --->service's bin path

# | |--start.sh --->start service

# | |--stop.sh --->stop service

# | |--install.sh --->install service

# |--other path -->other path for service

# |--.install

# | |--version --->service's version file for running

# | |--env\_conf --->service's pool's env conf

# | |--package --->history package path

# | | |--project\_service\_version.tar.gz

# | |--tmp --->temporary path

import os

import commands

import shutil

import sys

import traceback

import datetime

import platform

RESERVE\_PACKAGE\_NUM = 5

class DcmdInstall():

def \_\_init\_\_(self):

#get environ var

self.app\_name\_ = os.environ.get('DCMD\_APP\_NAME')

self.svr\_name\_ = os.environ.get('DCMD\_SVR\_NAME')

self.svr\_pool\_ = os.environ.get('DCMD\_SVR\_POOL')

self.svr\_version\_ = os.environ.get('DCMD\_SVR\_VERSION')

self.svr\_repo\_ = os.environ.get('DCMD\_SVR\_REPO')

self.svr\_user\_ = os.environ.get('DCMD\_SVR\_USER')

self.svr\_ip\_ = os.environ.get('DCMD\_SVR\_IP')

self.is\_svr\_update\_env\_ = True if "1"== os.environ.get('DCMD\_SVR\_UPDATE\_ENV') else False

self.is\_svr\_update\_tag\_ = True if "1"== os.environ.get('DCMD\_SVR\_UPDATE\_TAG') else False

self.is\_node\_multi\_pool\_ = True if "1"== os.environ.get('DCMD\_SVR\_NODE\_MUTIL\_POOL') else False

self.is\_svr\_process\_ = True if "1"== os.environ.get('DCMD\_SVR\_PROCESS') else False

self.svr\_env\_v\_ = os.environ.get('DCMD\_SVR\_ENV\_V')

self.svn\_user\_ = os.environ.get('DCMD\_TASK\_svn\_user')

self.svn\_passwd\_ = os.environ.get('DCMD\_TASK\_svn\_passwd')

self.is\_merge\_ = True if "1"== os.environ.get('DCMD\_TASK\_is\_merge') else False

self.is\_os\_package\_ = True if "1"== os.environ.get('DCMD\_TASK\_is\_os\_package') else False

self.svr\_result\_file\_ = os.environ.get('DCMD\_SVR\_RESULT\_FILE')

self.svr\_home\_ = os.path.join(os.environ.get('DCMD\_SVR\_PATH'), self.app\_name\_)

self.svr\_home\_ = os.path.join(self.svr\_home\_, self.svr\_name\_)

if self.is\_node\_multi\_pool\_:

self.svr\_home\_ = os.path.join(self.svr\_home\_, self.svr\_pool\_)

self.install\_path\_ = os.path.join(self.svr\_home\_, ".install")

self.svr\_package\_path\_ = os.path.join(self.install\_path\_, "package")

self.svr\_version\_file\_ = os.path.join(self.install\_path\_, "version")

self.svr\_env\_conf\_path\_ = os.path.join(self.install\_path\_, "env\_conf")

self.svr\_env\_conf\_version\_path\_ = os.path.join(self.svr\_env\_conf\_path\_, self.svr\_env\_v\_)

self.svr\_tmp\_path\_ = os.path.join(self.install\_path\_, "tmp")

self.svr\_tmp\_service\_path\_ = os.path.join(self.svr\_tmp\_path\_, self.svr\_name\_)

self.svr\_online\_version\_ = ""

self.svr\_packages\_ = []

self.install\_errmsg\_ = ""

self.install\_success\_ = False

self.os\_platform\_ = ""

self.svr\_package\_ = ""

def clean\_path(self, path):

print("Clear path:%s" % path)

package\_files = os.listdir(path)

for f in package\_files:

rm\_file = os.path.join(path, f)

self.\_remove(rm\_file)

def remove\_old\_package(self):

print("Remove old package file...............")

package\_files = os.listdir(self.svr\_package\_path\_)

packages = []

for f in package\_files:

if os.path.isdir(f): ##delete path in package path

rm\_file = os.path.join(self.svr\_package\_path\_, f)

shutil.rmtree(rm\_file)

continue

statinfo=os.stat(os.path.join(self.svr\_package\_path\_, f))

packages.append(((int)(statinfo.st\_ctime), f))

packages.sort(lambda x,y:cmp(x[0],y[0]))

while len(packages) > RESERVE\_PACKAGE\_NUM:

self.\_remove(self.svr\_package\_path\_, packages[0][1])

del packages[0]

#init the service path

def init\_service\_env(self):

print("Get host's os")

os\_type = platform.platform()

if "centos" in os\_type:

self.os\_platform\_ = "centos"

elif "Ubuntu" in os\_type:

self.os\_platform\_ = "ubuntu"

elif "redhat" in os\_type:

self.os\_platform\_ = "redhat"

else:

print("Unsupport os type: %s" % os\_type)

raise Exception("Unsupport os type: %s" % os\_type)

if "64bit" == platform.architecture()[0]:

self.os\_platform\_ += "64"

else:

self.os\_platform\_ += "32"

print("OS type:%s" % self.os\_platform\_)

if self.is\_os\_package\_:

self.svr\_package\_ = self.svr\_name\_ + "\_" + self.os\_platform\_ + "\_" + self.svr\_version\_ + ".tar.gz"

else:

self.svr\_package\_ = self.svr\_name\_ + "\_" + self.svr\_version\_ + ".tar.gz"

print("Check service home:%s" % self.svr\_home\_)

self.\_check\_and\_mk\_missed\_path(self.svr\_home\_)

print("Check service package path:%s" % self.svr\_package\_path\_)

self.\_check\_and\_mk\_missed\_path(self.svr\_package\_path\_)

print("Check service tmp path:%s" % self.svr\_tmp\_path\_)

self.\_check\_and\_mk\_missed\_path(self.svr\_tmp\_path\_)

print("Check env conf path:%s" % self.svr\_env\_conf\_path\_)

self.\_check\_and\_mk\_missed\_path(self.svr\_env\_conf\_path\_)

self.clean\_path(self.svr\_tmp\_path\_)

self.clean\_path(self.svr\_env\_conf\_path\_)

self.remove\_old\_package()

#check current version

def check\_version(self):

print("Check service [%s]'s version....." % self.svr\_name\_)

if os.path.isfile(self.svr\_version\_file\_):

print("Version file exists, file:%s." % self.svr\_version\_file\_)

with open(self.svr\_version\_file\_, 'r') as f:

self.svr\_online\_version\_ = f.read()

print("Online version is:%s" % self.svr\_online\_version\_)

if self.svr\_online\_version\_ == self.svr\_version\_:

print("Version is same, version:%s" % self.svr\_version\_)

return True

else:

print("Version is not same, running version:%s, install version:%s" % (self.svr\_online\_version\_, self.svr\_version\_))

return False

print("Version file doesn't exist, version file:%s" % self.svr\_version\_file\_)

return False

#create version file to save version

def create\_version\_file(self):

print("Save version[%s] to %s" % (self.svr\_version\_, self.svr\_version\_file\_))

with open(self.svr\_version\_file\_, 'w') as f:

f.write(self.svr\_version\_)

#remove current version file

def remove\_version\_file(self):

print("Remove current verision file:%s" % self.svr\_version\_file\_)

self.\_remove(self.svr\_version\_file\_)

#out result to out\_file

def output\_result(self):

if self.install\_success\_:

with open(self.svr\_result\_file\_, 'w') as f:

f.write("state=success\n")

f.write("err=")

else:

with open(self.svr\_result\_file\_, 'w') as f:

f.write("state=failure\n")

f.write("err=%s" % self.install\_errmsg\_)

#download service package

def download\_package(self):

save\_file = os.path.join(self.svr\_package\_path\_, self.svr\_package\_)

tmp\_file = os.path.join(self.svr\_tmp\_path\_, self.svr\_package\_)

is\_download = True

if os.path.isfile(save\_file):

if not self.is\_svr\_update\_tag\_:

print("Pachage file:%s exist, doesn't download" % self.svr\_package\_)

shutil.copyfile(save\_file, tmp\_file)

is\_download = False

else:

self.\_remove(save\_file)

if is\_download:

if os.path.isfile(tmp\_file):

self.\_remove(tmp\_file)

download\_path\_file = self.svr\_repo\_ + "/packages/" + self.app\_name\_ + "/" + self.svr\_name\_ + "/" + self.svr\_package\_

print("svn package file:%s" % download\_path\_file)

if (self.svn\_user\_ and self.svn\_passwd\_):

svn\_cmd = "svn export %s %s --username=%s --password=%s --no-auth-cache --non-interactive " % (download\_path\_file, os.path.join(self.svr\_tmp\_path\_,self.svr\_package\_), self.svn\_user\_, self.svn\_passwd\_)

elif (self.svn\_user\_):

svn\_cmd = "svn export %s %s --username=%s --no-auth-cache --non-interactive " % (download\_path\_file, os.path.join(self.svr\_tmp\_path\_,self.svr\_package\_), self.svn\_user\_)

else:

svn\_cmd = "svn export %s %s --no-auth-cache --non-interactive " % (download\_path\_file, os.path.join(self.svr\_tmp\_path\_,self.svr\_package\_))

if not self.\_run\_cmd(svn\_cmd):

self.install\_errmsg\_ = "Failure to download package, cmd:%s" % svn\_cmd

self.install\_success\_ = False

return False

shutil.copy(tmp\_file, save\_file)

if os.path.isdir(self.svr\_tmp\_service\_path\_):

self.\_remove(self.svr\_tmp\_service\_path\_)

unzip\_cmd = "tar -zxvf " + tmp\_file + " -C " + self.svr\_tmp\_path\_

if not self.\_run\_cmd(unzip\_cmd):

self.install\_errmsg\_ = "Failure to unpack package:%s" % tmp\_file

self.install\_success\_ = False

return False

remove\_cmd = "rm -f %s" % tmp\_file

if not self.\_run\_cmd(remove\_cmd):

self.install\_errmsg\_ = "Failure to remove tmp package:%s" % tmp\_file

self.install\_success\_ = False

return False

#find app's home path

if not os.path.isdir(self.svr\_tmp\_service\_path\_):

self.install\_errmsg\_ = "package's extract-path must be service'name"

self.install\_success\_ = False

return True

#download service config

def download\_config(self):

if not self.is\_svr\_update\_env\_:

print("Needn't download env config")

return True

print("tmp\_conf\_path:%s" % self.svr\_env\_conf\_path\_)

if os.path.isdir(self.svr\_env\_conf\_version\_path\_):

os.rmdir(self.svr\_env\_conf\_version\_path\_)

# download conf from svn

svn\_path = self.svr\_repo\_ + "/conf" + "/" + self.app\_name\_ + "/" + self.svr\_name\_ + "/" + self.svr\_pool\_ + "/" + self.svr\_env\_v\_

print("svn env-config path:%s" % svn\_path)

if (self.svn\_user\_ and self.svn\_passwd\_):

svn\_cmd = "svn export %s %s --username=%s --password=%s --no-auth-cache --non-interactive " % (svn\_path, self.svr\_env\_conf\_version\_path\_, self.svn\_user\_, self.svn\_passwd\_)

elif (self.svn\_user\_):

svn\_cmd = "svn export %s %s --username=%s --no-auth-cache --non-interactive " % (svn\_path, self.svr\_env\_conf\_version\_path\_, self.svn\_user\_)

else:

svn\_cmd = "svn export %s %s --no-auth-cache --non-interactive " % (svn\_path, self.svr\_env\_conf\_version\_path\_)

if not self.\_run\_cmd(svn\_cmd):

self.install\_success\_ = False

self.install\_errmsg\_ = "Failed to get conf from svn:%s, cmd:%s" % (svn\_path, svn\_cmd)

return False

return True

#start service

def start\_service(self):

start\_cmd = os.path.join(os.path.join(self.svr\_home\_,"bin"), "start.sh")

if not self.\_run\_cmd(start\_cmd):

self.install\_success\_ = False

self.install\_errmsg\_ = "Failed to start service, cmd:%s" % start\_cmd

return False

return True

#stop service

def stop\_service(self):

stop\_cmd = os.path.join(os.path.join(self.svr\_home\_,"bin"), "stop.sh")

if not os.path.isfile(stop\_cmd): return True

if not self.\_run\_cmd(stop\_cmd):

self.install\_success\_ = False

self.install\_errmsg\_ = "Failed to stop service, cmd:%s" % stop\_cmd

return False

return True

#install service

def install\_service(self):

install\_cmd = os.path.join(os.path.join(self.svr\_tmp\_service\_path\_,"bin"), "install.sh")

change\_mod = "chmod 755 " + install\_cmd

if not self.\_run\_cmd(change\_mod):

self.install\_success\_ = False

self.install\_errmsg\_ = "Failed to change install's mod, cmd:%s" % change\_mod

return False

if self.is\_svr\_update\_env\_:

install\_cmd = install\_cmd + " " + self.svr\_home\_ + " " + self.svr\_tmp\_service\_path\_ + " " + self.svr\_env\_conf\_version\_path\_

else:

install\_cmd = install\_cmd + " " + self.svr\_home\_ + " " + self.svr\_tmp\_service\_path\_

if not self.\_run\_cmd(install\_cmd):

self.install\_success\_ = False

self.install\_errmsg\_ = "Failed to install service, cmd:%s" % install\_cmd

return False

return True

#remove file or dir

def \_remove(self, df):

if os.path.isfile(df):

os.remove(df)

elif os.path.isdir(df):

shutil.rmtree(df)

#check and create missing subpath

def \_check\_and\_mk\_missed\_path(self, path):

if not os.path.isdir(path):

os.makedirs(path)

#runing system cmd

def \_run\_cmd(self, cmd):

ret = os.system(cmd)

ret >>= 8

return ret == 0

def main():

install = DcmdInstall()

#1.check version

try:

print("\n%s:STEP 0: init svr enviroment.......\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

install.init\_service\_env()

print("\n%s:STEP 1: Check version.............\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

if install.check\_version():

print "==>The package version to be installed is same as current versionm, install succeed."

install.install\_success\_ = True

install.output\_result()

return

print("\n%s:STEP 2: Download package............\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

if not install.download\_package():

install.output\_result()

return

print("\n%s:STEP 3: Download config file.........\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

if not install.download\_config():

install.output\_result()

return

print("\n%s:STEP 4: Stop current service........\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

if not install.stop\_service():

install.output\_result()

return

print("\n%s:STEP 5: Remove version file............\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

install.remove\_version\_file()

print("\n%s:STEP 6: Install service..............\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

if not install.install\_service():

install.output\_result()

return

print("\n%s:STEP 7: Start service...............\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

if not install.start\_service():

install.output\_result()

return

print("\n%s:STEP 8: Create service version file............\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

install.create\_version\_file()

print("%s:Success to install serivice.\n" % datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

install.install\_success\_ = True

except Exception, e:

print ("%s:Failed to install, Exception:%s" % (datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"), e))

traceback.print\_exc()

install.install\_errmsg\_ = "Failed to install for exception: %s" % e

install.install\_success\_ = False

install.output\_result()

if \_\_name\_\_ == "\_\_main\_\_":

main()

## 操作脚本

### 操作脚本说明

操作脚本相对与任务脚本，基本没有什么规则约束。需要注意的是操作脚本的参数传递。

* 操作指令名称：

操作指令名称为[a-z,A-Z,0-9,\_]范围内的字符。

* 操作指令script文件：

dcmd\_opr\_【操作指令名称】.script，其保存在dcmd\_center所在的服务器上，**存放位置由dcmd\_center配置文件的【common：opr\_script\_path】的参数项指定。**

* 操作指令的参数：

操作可以定义任意数量的参数，对于参数的值，agent会以DCMD\_OPR【参数名字】的环境变量传递给执行脚本。

参数名字有[a-z,A-Z,0-9,\_]范围内的字符构成。

传递的参数值，若包含【|】、【"】、【'】、【 】、【\r】、【\n】、【;】、【&】的字符，则系统会自动删除。

* 操作指令的输出：

脚本的stdout、stderr的输出，都会被作为操作的结果返回。

### 操作脚本例子：

#### 获取服务器运行进程信息脚本：ps

* + 脚本参数：
* filter：只提取含有filter指定字符串的进程
  + 此脚本内容：

#!/bin/sh

if [ -z "$DCMD\_OPR\_filter" ]; then

ps -ef

else

ps -ef|grep "$DCMD\_OPR\_filter"`

#### 获取服务器/proc下信息的脚本：proc\_info

* + 脚本参数：
* path：proc的路径
* filter：只显示包含filter内容的信息
  + 此脚本内容：

#!/bin/sh

proc\_path="/proc/""$DCMD\_OPR\_path"

if [ -f $proc\_path ]; then

if [ -z "$DCMD\_OPR\_filter" ]; then

cat $proc\_path

else

cat $proc\_path|grep "$DCMD\_OPR\_filter"

fi

elif [ -d $proc\_path ]; then

if [ -z "$DCMD\_OPR\_filter" ]; then

ls -1X $proc\_path

else

ls -1X $proc\_path|grep "$DCMD\_OPR\_filter"

fi

else

echo "Doesn't exist."

exit 1

fi

exit 0