리눅스 응용프로그램
배포 솔루션 개발지원

한국소프트웨어진흥원
공개SW기술지원센터

<Revision 정보>

일자	VERSION	변경내역	작성자
2007.10.16	0.1	초기 작성	김상운

목 차

1. 문서 개요

본 문서는 (주)티씨오솔루션 사의 요청으로 인해 자사에서 출시하고자 하는 리눅스 클라이언트용 배포 솔루션의 기능중 Pull방식 배포 기능을 구현개발을 위한 참고 문서이다.

가. 문서의 목적

다음과 같은 세부적인 목적을 달성하기 위하여 작성되었다.

0 리눅스 상에서 패키지 배포 예제 소스 제공

0 관련 개발 환경 정의

0 관련 개발자들에게 관련 문서 배포

나. 본 문서의 사용방법

다음과 같은 방법으로 사용할 수 있다.

0 본 문서는 일반 리눅스 클라이언트에 적용될 에이전트(Agent)에 한정하고 있음

0 본 문서에 포함된 소스는 GPL기반으로 공개 예정이므로 향후 자사 제품을 비공개하기 위해서는 별도의 개발 필요

다. 참고사항

0 본 소스로 개발된 프로그램은 python 2.4+ 및 rpm 4.3 기반에서 적용가능하며 본 문서는 한글과컴퓨터 아시아눅스 데스크탑 3.0 기반으로 개발 및 테스트되었음

0 일부 배포판중에서 본 문서에서 설명하는 개발도구 또는 라이브러리가 포함되어 있지 않을 경우 본 문서의 내용이 지원되지 않을 수 있으므로 참고바람

2. 개발지원 요청내역

가. 요청 사항

항목	요청내용	지원 내역	비 고
	리눅스 클라이언트용 응용프로그램 배포 솔루션 개발 지원	- python 2.4+ 및 rpm 4.3 이상의 환경에서 리눅스 클라이언트에 적용될 배포 에이전트 개발 지원 - 관련 예제 소스 지원	

나. 지원인력구성

담당	직급	성명	소속사	인력 구분	기간	지원내용	비고
솔루션 개발팀	과장	김상운	㈜한글과 컴퓨터	상주	2007.09.26~ 2007.10.15	리눅스 클라이언트용 에이전트 개발 지원	

3. 개발내역

가. 개발 소프트웨어 기본정보

-제품 이름 : 미정
-버전 : 0.05
-소스코드 언어 및 환경 : python 2.4 및 rpm 4.3

나. 개발 및 적용환경

-소프트웨어 환경 (한글과컴퓨터 리눅스 데스크탑 3.0)

커널	2.6.14 이상
X-Windows	Xorg 7.1 이상
Glibc	2.3 이상
GCC	3.0 이상
waxlib	1.2 이상

다. 개발내역

1) 개요

소프트웨어 배포를 위해서는 일반적으로 배포를 하는 대상(서버), 배포를 받는 대상(클라이언트) 및 관리(어드민)으로 일반적으로 구성된다.

배포의 방식에 있어서도 배포하는 대상이 주체가 되어 강제로 배포하는 강제배포방식(Push)과 배포를 받는 대상이 주체가 되어 수행하는 사용자배포방식(Poll) 방식으로 나뉘어지는데 , 참고로 레드햇의 up2date 등은 poll 방식에 해당한다.

현재 티씨오솔루션에서 개발하고자 하는 소프트웨어는 Push/Poll 두가지 방식을 모두 지원하는 소프트웨어로서 현재까지 서버에서 pull/push 기법은 구현되어 있으나 클라이언트상에서 이들 이벤트에 맞추어 각 배포판별로 커널에 상관없이 동작할수 있는 에이전트 개발이 필요로 하고 있다.

따라서 배포판과 문제없이 적용될수 있는 파이썬과 RPM을 이용하여 구현될 수 있도록 개발 중심을 맞추었으며 다음과 같은 분야의 소스가 제공되었다.

- RPM 업데이트 리스트 전송 및 파일전송

2) 관련 제공 소스

```
import sys
import gzip
import exceptions

from textwrap import wrap
from yum.yumRepo import YumRepository

try:
    from xml.etree import cElementTree
except ImportError:
    import cElementTree
iterparse = cElementTree.iterparse


class UpdateNoticeException(exceptions.Exception):
    pass
```

```
class UpdateNotice(object):
    def __init__(self, elem=None):
        self._md = {
            'from'          : '',
            'type'          : '',
            'title'         : '',
            'release'       : '',
            'status'        : '',
            'version'       : '',
            'pushcount'     : '',
            'update_id'     : '',
            'issued'        : '',
            'updated'       : '',
            'description'   : '',
            'references'    : [],
            'pkglist'       : [],
            'reboot_suggested' : False
        }

        if elem:
            self._parse(elem)

    def __getitem__(self, item):
        """ Allows scriptable metadata access (ie: un['update_id']). """
        return self._md.has_key(item) and self._md[item] or None

    def __str__(self):
        head = """
==============================================================================
======
  %(title)s
==============================================================================
======
  Update ID : %(update_id)s
    Release : %(release)s
       Type : %(type)s
     Status : %(status)s
     Issued : %(issued)s
""" % self._md
```

```
        if self._md['updated'] and self._md['updated'] != self._md['issued']:
            head += "     Updated : %(updated)s" % self_md


        bzs = filter(lambda r: r['type'] == 'bugzilla', self._md['references'])
        if len(bzs):
            buglist = "        Bugs :"
            for bz in bzs:
                buglist += " %s%s₩n₩t     :" % (bz['id'], bz.has_key('title')
                                                 and ' - %s' % bz['title'] or '')
            head += buglist[:-1].rstrip() + '₩n'


        cves = filter(lambda r: r['type'] == 'cve', self._md['references'])
        if len(cves):
            cvelist = "        CVEs :"
            for cve in cves:
                cvelist += " %s₩n₩t     :" % cve['id']
            head += cvelist[:-1].rstrip() + '₩n'

        desc = wrap(self._md['description'], width=64,
                    subsequent_indent=' ' * 12 + ': ')
        head += "Description : %s₩n" % '₩n'.join(desc)

        filelist = "        Files :"
        for pkg in self._md['pkglist']:
            for file in pkg['packages']:
                filelist += " %s₩n₩t     :" % file['filename']
        head += filelist[:-1].rstrip()

        return head

    def get_metadata(self):
        """ Return the metadata dict. """
        return self._md

    def _parse(self, elem):
        """ Parse an update element.
```

```
            <!ELEMENT update (id, synopsis?, issued, updated,
                              references, description, pkglist)>
                <!ATTLIST update type (errata|security) "errata">
                <!ATTLIST update status (final|testing) "final">
                <!ATTLIST update version CDATA #REQUIRED>
                <!ATTLIST update from CDATA #REQUIRED>
        """
        if elem.tag == 'update':
            for attrib in ('from', 'type', 'status', 'version'):
                self._md[attrib] = elem.attrib.get(attrib)
            for child in elem:
                if child.tag == 'id':
                    if not child.text:
                        raise UpdateNoticeException("No id element found")
                    self._md['update_id'] = child.text
                elif child.tag == 'pushcount':
                    self._md['pushcount'] = child.text
                elif child.tag == 'issued':
                    self._md['issued'] = child.attrib.get('date')
                elif child.tag == 'updated':
                    self._md['updated'] = child.attrib.get('date')
                elif child.tag == 'references':
                    self._parse_references(child)
                elif child.tag == 'description':
                    self._md['description'] = child.text
                elif child.tag == 'pkglist':
                    self._parse_pkglist(child)
                elif child.tag == 'title':
                    self._md['title'] = child.text
                elif child.tag == 'release':
                    self._md['release'] = child.text
        else:
            raise UpdateNoticeException('No update element found')

    def _parse_references(self, elem):
        """ Parse the update references.

            <!ELEMENT references (reference*)>
```

```
            <!ELEMENT reference>
                <!ATTLIST reference href CDATA #REQUIRED>
                <!ATTLIST reference type (self|cve|bugzilla) "self">
                <!ATTLIST reference id CDATA #IMPLIED>
                <!ATTLIST reference title CDATA #IMPLIED>
        """
        for reference in elem:
            if reference.tag == 'reference':
                data = {}
                for refattrib in ('id', 'href', 'type', 'title'):
                    data[refattrib] = reference.attrib.get(refattrib)
                self._md['references'].append(data)
            else:
                raise UpdateNoticeException('No reference element found')

    def _parse_pkglist(self, elem):
        """ Parse the package list.

            <!ELEMENT pkglist (collection+)>
            <!ELEMENT collection (name?, package+)>
                <!ATTLIST collection short CDATA #IMPLIED>
                <!ATTLIST collection name CDATA #IMPLIED>
            <!ELEMENT name (#PCDATA)>
        """
        for collection in elem:
            data = { 'packages' : [] }
            if collection.attrib.has_key('short'):
                data['short'] = collection.attrib.get('short')
            for item in collection:
                if item.tag == 'name':
                    data['name'] = item.text
                elif item.tag == 'package':
                    data['packages'].append(self._parse_package(item))
            self._md['pkglist'].append(data)

    def _parse_package(self, elem):
        """ Parse an individual package.

            <!ELEMENT package (filename, sum, reboot_suggested)>
```

```
                <!ATTLIST package name CDATA #REQUIRED>
                <!ATTLIST package version CDATA #REQUIRED>
                <!ATTLIST package release CDATA #REQUIRED>
                <!ATTLIST package arch CDATA #REQUIRED>
                <!ATTLIST package epoch CDATA #REQUIRED>
                <!ATTLIST package src CDATA #REQUIRED>
            <!ELEMENT reboot_suggested (#PCDATA)>
            <!ELEMENT filename (#PCDATA)>
            <!ELEMENT sum (#PCDATA)>
                <!ATTLIST sum type (md5|sha1) "sha1">
        """
        package = {}
        for pkgfield in ('arch', 'epoch', 'name', 'version', 'release', 'src'):
            package[pkgfield] = elem.attrib.get(pkgfield)
        for child in elem:
            if child.tag == 'filename':
                package['filename'] = child.text
            elif child.tag == 'sum':
                package['sum'] = (child.attrib.get('type'), child.text)
            elif child.tag == 'reboot_suggested':
                self._md['reboot_suggested'] = True
        return package


class UpdateMetadata(object):
    def __init__(self):
        self._notices = {}
        self._cache = {}
        self._repos = []

    def get_notices(self):
        """ Return all notices. """
        return self._notices.values()

    notices = property(get_notices)

    def get_notice(self, nvr):
        """
        Retrieve an update notice for a given (name, version, release) string
```

```
        or tuple.
        """
        if type(nvr) in (type([]), type(())):
            nvr = '-'.join(nvr)
        return self._cache.has_key(nvr) and self._cache[nvr] or None

    def add(self, obj, mdtype='updateinfo'):
        """ Parse a metadata from a given YumRepository, file, or filename. """
        if not obj:
            raise UpdateNoticeException
        if type(obj) in (type(''), type(u'')):
            infile = obj.endswith('.gz') and gzip.open(obj) or open(obj, 'rt')
        elif isinstance(obj, YumRepository):
            if obj.id not in self._repos:
                self._repos.append(obj.id)
                md = obj.retrieveMD(mdtype)
                if not md:
                    raise UpdateNoticeException()
                infile = gzip.open(md)
        else:
            infile = obj

        for event, elem in iterparse(infile):
            if elem.tag == 'update':
                un = UpdateNotice(elem)
                if not self._notices.has_key(un['update_id']):
                    self._notices[un['update_id']] = un
                    for pkg in un['pkglist']:
                        for file in pkg['packages']:
                            self._cache['%s-%s-%s' % (file['name'],
                                                      file['version'],
                                                      file['release'])] = un

    def __str__(self):
        ret = ''
        for notice in self.notices:
            ret += str(notice)
        return ret
```

```
def main():
    def usage():
        print >> sys.stderr, "Usage: %s <update metadata> ..." % sys.argv[0]
        sys.exit(1)

    if len(sys.argv) < 2:
        usage()

    try:
        print sys.argv[1]
        um = UpdateMetadata()
        for srcfile in sys.argv[1:]:
            um.add(srcfile)
        print um
    except IOError:
        print >> sys.stderr, "%s: No such file:₩'%s₩'" % (sys.argv[0],
                                                          sys.argv[1:])
        usage()

if __name__ == '__main__':
    main()
```

라. 향후개선(지원요청)내역

- RPM, dep, gz 등 다양한 포맷 지원
- 본 소스는 티씨오솔루션의 개발에 참조하기 위해 만들어진 소스이며 향후 요청에 따라 확대 개발 예정