SADRŽAJ

Šta je JAR

JAR je egzekutabilna arhiva vezana za Javu. U suštini, JAR je jedan ZIP-ovan fajl. Možemo da promenimo ekstenziju fajla sa JAR na ZIP, i da ga otvorimo sa bilo kojim arhiverom. Takođe možemo I da napravimo JAR tako što napravimo ZIP,pa ZIP arhivi promenimo ime u JAR. Za rad sa JAR fajlom se najčešće koristi jartool koji dolazi zajedno sa Javom SDK. Evo pregleda osnovnih naččina rada sa JAR fajlovima:

- jar cf nazivFajla spisakFajlova - pravljenje
- jar tf nazivFajla - pregled
- jar xf nazivFajla – raspakivanje
- java -jar nazivFajla - pokretanje

Kreiranje JAR fajla

Videli smo da je naredba pomoću koje se kreira JAR fajl:

- jar cf nazivFajla spisakFajlova
- c – create, f file
- spisak fajlova – fajlovi razdvojeni razmakom
- ako u spisku fajlova stavimo direktorijum on će biti rekurzivno smešten u jar (kompletan njegov sadržaj sa pod folderima i njihovim sadržajem)
- Default manifest

JAR se od običnog ZIP-a razlikuje po jednoj datoteci, koja se zove manifest, u kojoj su smeštene dodatne informacije. Pomoću ove naredbe možemo da kreiramo JAR fajl sa default manifest fajlom u okviru JAR datoteke.

Dodatni parametri pri kreiranju

Pored parametara c i f, koje smo upoznali prethodno, postoji još nekoliko parametara koje možemo da upotrebimo na isti način kao i c i f:

- v – daje informacije o radu
- 0 – sprečava kompresiju
- M – sprečava pravljenje default manifesta
- m – prosleđuje se manifers

jar cmf mojManifest.mf nazivFajla spisakFajlova

- -C – podaci iz direktorijuma se smeštaju u root jar-a

Raspakivanje

- Raspakivanje konkretnog fajla ili fajlova

jar xf nazivFajla fajl1 fajl2

- Raspakivanje celog jar-a
- jar xf nazivFajla
- Biće bez upozorenja pregaženi fajlovi

Dodavanje fajlova

- Dodavanje fajlova se vrši naredbom

jar uf nazivJara fajlKojiDodajemo fajl2

- Parametri jar-a
  - u – update
  - t – sadržaj
  - x – raspakivanje

Default manifest fajl

- Jedan manifest u okviru JAR fajla
- META-INF/MANIFEST.MF
- Default manifest

Manifest-Version: 1.0

Created-By: 1.5.0_01 (Sun Microsystems Inc.)

- Default JAR je ne izvršni JAR

Ako želimo da napravimo izvršni fajl, onda moramo da kreiramo poseban manifest, koji dodajemo prilikom kreiranja JAR fajla.

Kreiranje manifest fajla

Možemo da kreiramo sopstveni manifest, i tom prilikom moramo da vodimo računa o nekoliko stvari:

- Text u okviru manifest fajla mora da bude UTF-8 enkodiran
- Tačka startovanja – ako želimo da bude izvršan
- Main-Class: nazivKlase
- Fajl mora da se završi novim praznim redom inače neće biti urađena zadnja linija u manifest fajlu
- Ako se tačka startovanja nalazi u nekom paketu
- Main-Class: paket.nazivKlase

Postupak primene JAR-a

Evo nekoliko koraka koje je neophodno ispoštovati kada kreiramo sopstvenu izvršnu datoteku:

- Napiše se manifest fajl sledeće sadržine

Main-Class: paket.StartnaKlasa

- Kreira se jar na sledeći način:

jar cmf mojManifest nazivJara spisakFajlova

- Može da se startuje program pozivom

java -jar nazivJara.jar

Classpath na drugi jar

Situacija koja može da se dogodi u izvršavanju JAR-ova je da se deo podataka nalazi u nekom drugom JAR-u. To se, recimo, dešava kod applet-a. U tom slučaju moramo u okviru JAR fajla da se pozovemo na klase iz nekog drugog JAR fajla.

- Dodavanje drugog jara u putanju prvog odnosno učitavanje biblioteke drugog jara
- Linija u okviru manifesta
- Class-Path: jar1naziv jar2naziv folder/jar3naziv
- Upotreba:
  - appleti koji imaju više jar fajlova
  - driver za bazu se nalazi u drugom jaru u odnosu na aplikaciju

Informacije vezane za pakete

- **Ako želimo da dodatne informacije vežemo za svaki pojedinačni paket u JAR-u**
- Izged Manifest fajla preuzeto od paketa java.util

Name: java/util/

Specification-Title: Java Utility Classes

Specification-Version: 1.2

Specification-Vendor: Sun Microsystems, Inc.

Implementation-Title: java.util

Implementation-Version: build57

Implementation-Vendor: Sun Microsystems, Inc.

Zaključavanje

U okviru JAR fajla možemo da uključimo dodatnu komandu, pomoću koje ćemo da zaključamo JAR.

- Zaključavanje JAR-a
- Zaključavanje paketa
- Sigurnosni mehanizam
- Garantuje da su sve klase koje se koriste u aplikaciji ili paketu iz istog JAR fajla

U slučaju da zaključavamo paket, to znači da sve klase koje se startuju u okviru tog paketa moraju da budu iz ovog JAR-a. Sigurnosno mehanizam zaključavanja JAR-a povećava sigurnost naše aplikacije, zato što neko spolja ne može da nam, preko interneta, podmetne neku drugu klasu da se izvrši.

Primer zaključavanja

- Primer kako se vrši zaključavane paketa

Name: paketi/paket1/

Sealed: true

Name: paketi/podpaket2/

Sealed: true

- Naziv foldera u kome se nalazi paket mora da se završi sa „/”

Potpisivanje i verifikacija

- Sigurnosni mehanizam
- Za potpisivanje koristimo privatni ključ poznat samo nama
- Za verifikaciju se koristi javni ključ koji proverava da li je korišćen konkretan privatni ključ prilikom potpisivanja
- Javni ključ se ugrađuje u JAR.

Ovaj sigurnosni mehanizam je veoma bitan, jer na ovaj način možemo da obezbedimo aplikaciju od potencijalnih zlonamernih izmena.

Potpisivanje JAR-a cilj

Pomoću ovog mehanizma potpisivanja fajlova možemo da ustanovimo da li je došlo do izmene bilo kog fajla, pa i samog JAR-a od trenutka kada je potpisan.

- Kreira se fajl sa extenzijom sf
- U ovom fajlu se nalaze potpisi svih fajlova u okviru JAR-a uključujući i sam sf.
- Pomoću ovog fajla možemo da ustanovimo da li je došlo do izmene bilo kog fajla pa i samog JAR-a od trenutka kad je potpisan.

Potpisivanje JAR-a pripreme

Da bismo mogli da izvršimo potpisivanje JAR datoteke, moramo da imamo pripremljene odgovarajuće informacije.

- Moramo da imamo private ključ.
- Privatni ključ i korsepondujući javni ključ moraju da budu smešteni u bazu zaštićenu šifrom.
- Baza je pod imenom „keystores”.
- U bazi može da se nalazi više parova ključeva.
- Svaki par ključeva ima svoj alias naziv pod kojim vodimo taj potpis

Potpisivanje JAR-a naredba

- Koristi se comandni alat „jarsigner”.

jarsigner jarFajl alias

- Nakon ove naredbe biće nam tražena šifra za bazu potpisa „keystores” i šifra za konkretan potpis.
- Gornja naredba će prepisati postojeći jar sa novim potpisanim jar fajlom.

Baza potpisa „keystores”

1. Default baza ključeva se nalazi u sakrivenom folderu korisnika „.keystore”
2. jarsigner može da ima još nekoliko parametara:

-keystore url - ako želimo da se ne koristi default keystores

-storepass password – ubacivanje passworda u komandu kako ne bi bili pitani za password baze potpisa

-keypass password – ubacivanje passworda u komandu kako ne bi bili pitani za password alias-a

Nastavak jarsigner

sigfile file – možemo da specificiramo nazive fajlova sa potpisima koji se završavaju na .SF i .DSA ako ne želimo da se koriste alias nazivi.

-signedjar file – Možemo da promenimo ime jar fajla koji dobijamo kao rezultat potpisivanja.

Verifikacija JAR-a

- Jar fajl se verifikuje od strane JRE (Java Runtime Enviroment)
- Verifikovanje se vrši automatski prilikom startovanja jar-a.
- Možemo da verifikujemo jar i bez pokretanja sledećom naredbom:

jarsigner -verify jar-file

- Tri moguće poruke:

jar verified

jar is unsigned

jarsigner: java.lang.SecurityException: invalid SHA1

Resursi u jar-u

1. Ako pristupamo resursima na klasičan način kao kad aplikacija nije u jar-u onda ti resursi moraju da se nalaze van jar-a.
2. Na primer hoćemo da učitamo sliku koja se nalazi u folderu images.
3. Ako hoćemo da je učitamo iz jara moramo van jara da imamo folder sa datom slikom
4. Ako hoćemo da i resursi budu u jar-u moramo na svakom mestu gde čitamo resurse da koristimo naredbu
5. cl.getResource("images/save.gif")

Ako se resursi nalaze van JAR-a, možemo da im pristupamo na klasičan način kao i kod obične aplikacije koju nismo napakovali u JAR. U slučaju da želimo da resursi budu sastavni deo JAR-a, moramo da koristimo naredbu getResource da bismo pristupili tim resursima.

Parametri JVM-a

Sve parametre JVM-a možemo da podelimo da tri grupe:

- standardni parametri
- važe za sve operativne sisteme, bez izuzetka
- ne standardni parametri

parametar počinje sa X

- može da mu se promeni značenje ili da se ukine u nekim od sledećih verzija Jave

1. Sistemski zavisni parametri
   - para metar počinje sa XX
   - mogućnosti parametara su vezani za OS na kome se izvršava, definišu neku konkretnu karakteristiku konkretnog OS.

Primer upotrebe parametara JVM

Na nekoliko sledećih primera prikazaćemo nekoliko karakterističnih primera upotrebe parametara JVM:

Na datom primeru su data dva karakteristična primera upotrebe parametara za startovanje aplikacije

java -Dfile.encoding=windows-1250 -classpath lib/pg80b1.308.jdbc3.jar -jar Magic.jar -debug

java -Xms480000000 -Xmx512000000 -Dsun.java2d.opengl=true app/Main

Putanje i verzije

-classpath=putanja - definiše putanju do klasa i jar-ova koji se koriste za rad programa.

-version=true– prikaže verziju proizvoda i završi sa radom

-showversion=true- prikaže verziju proizvoda i nastavi sa radom

-fullversion=true – prikaže verziju jave i završi sa radom

Niti i skupljac đubreta

-Dname=value – setuje se neki sistemski properti

-Xnoclassgc=true – isključuje skupljač đubreta

-XX:GCTimeRatio=<value> - setuje vrednost na koju se vrši skupljanje đubreta

-Xincgc= true – aktivira inkrementalni skupljač đubreta

-Xconcgc=true – postavlja višenitni skupljac đubreta

Parametri za rad sa memorijom

Jedna od kritičnih situacija sa Javom je upotreba memorije. S' obzirom na to da se Java izvršava u okviru virtuelne mašine koja mora unapred da zauzme određene memorijske resurse od strane operativnog sistema, ponekad može da se desi da količina zauzete memorije nije dovoljna za izvršavanje aplikacije. Ako je to slučaj, imamo dva parametra pomoću kojih možemo da podesimo količinu memorije:

- -Xms<size> - setovanje inicijalne vredonsti za memoriju koju će koristiti JVM. Vrednost se setuje u bitovima. Primer za 480 MB memorije dodeljene inicijalno JVM.

java -Xms480000000 app/Main

- -Xmx<size> - setovanje maksimalne vredonsti za memoriju koju će koristiti JVM. Vrednost se setuje u bitovima. Primer za 512 MB memorije dodeljene inicijalno JVM.

java -Xmx512000000 app/Main

JNI

XX:+AggressiveHeap=true – zahteva agresino korišećenje memorije. Koristi se maximalno memorije koliko je raspoloživo sistemu.

-Xcheck:jni=true – proverava da li u kodu postoji jni funkcije

-XX:+CheckJNICalls=true - proverava da li u kodu postoje pozivi jni funkcije

Prioriteti

-Xss<size> - setuje veličinu steka za thread-ove

-XX:JavaPriority1_To_OSPriority=<value> - mapira prioritet definisan u java programu na odgovarajući prioritet Operativnog Sistema. Ovom naredbom mapira se javin prioritet 1 na odgovarajući prioritet OS-a (definisano sa value). Postoji za svaki javin prioritet ovakav parametar JVM. Samo umesto 1 stoji odgovarajući javin prioritet (1-9). Kada ne postoji ovaj parametar JVM, onda se Java prioriteti preslikavaju na prioritete operativnih sistema 1na1.

Prioriteti niti

-XX:+UseThreadPriorities=true – koriste se prirodni prioriteti niti.

-Dsun.java2d.opengl=true - forsira da se za ispis na ekran koristi openGL

Linkovi za dalje učenje

1. http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html
2. http://netbeans.org/
3. http://developer.apple.com/java/faq/