Programowanie obiektowe

Wykład 15: Przygotowanie i udostępnianie aplikacji

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.3
4 marca 2021

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Programowanie obiektowe

Wykład 15: Przygotowanie i udostępnianie aplikacji

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.3
4 marca 2021

Przygotowanie pliku do dystrybucji

- Java zachęca nas do tworzenia oprogramowania poprzez kreację wielu wyspecjalizowanych klas.
- Powoduje to powstawanie wielu małych plików tworzących rozbudowaną hierarchię katalogów.
- Utrudnia to dystrybucję oprogramowania i zaciemnia proces jego uruchamiania.
- W celu ułatwienia dystrybucji kompresuje się całą strukturę klas i umieszcza w pliku JAR.

Plik JAR

- Plik Java Archive (JAR) jest skompresowanym plikiem ZIP.
- Struktura pliku
  - binarne pliki klas i interfejsów
  - manifest opisujący działanie aplikacji
  - inne zasoby niezbędne do jej uruchomienia
- Do jego wytworzenia stosuje się polecenie *jar*.

Tworzenie pliku JAR

Tworzenie pliku JAR

```
jar cf JARNazwaPliku Pliki_do_dodania
```

- c tworzenie archiwum
- f generowanie pliku JAR

Manifest

- Manifest jest specjalnym plikiem zawierającym informacje o zawartości archiwum JAR.
- Archiwum może zawierać tylko jeden manifest META-INF/MANIFEST.MF.
- Manifest zawiera pary „nagłówek: wartość”.
- Narzędzie *jar* tworzy domyślny manifest.

Domyślny manifest

```
Manifest-Version: 1.0
Created-By: 1.7.0_06 (Oracle Corporation)
```

Modyfikacja manifestu

- Aby zmodyfikować manifest potrzebny jest plik z informacjami, które mają być dodane do manifestu

Modyfikacja manifestu

```
jar cfm plik_jar dodatkowe_pliki_manifestu
```

- Zawartość manifestu musi być kodowana w UTF8.

Zasoby

- Plik JAR może zawierać zasoby do wykorzystania w aplikacji
  - pliki obrazów i pliki dźwiękowe,
  - pliki z etykietami interfejsu,
  - inne pliki binarme
- Zasoby mogą być wczytane w czasie wykonywania programu

Wczytanie zasobu z katalogu klasy

```
URL url = ResourceTest.class.getResource("mini.jpg");
Image img = new ImageIcon(url).getImage();
```

Określenie klasy głównej

- Jeżeli plik JAR zawiera aplikacje należy wskazać klasę główną uruchamianą przy starcie aplikacji.
  - Klasa z metodą main.
  - Applet.
  - *Main-Class: classname*.

Określenie klasy głównej

```
Manifest-Version: 1.0
Created-By: 1.7.0_06 (Oracle Corporation)
Main-Class: MyPackage.MyClass
```

- W manifeście możemy także określić uprawnienia jakie powinna otrzymać aplikacja i zamieścić informacje o autorze.

Uruchamianie aplikacji

- Aplikację uruchamiamy poleceniem *java*

Uruchamianie aplikacji

```
java -jar MyProgram.jar
```

- Na maszynie docelowej musi być zainstalowane środowisko Java SE Runtime Environment.
- Manifest musi określać klasę główną.
- Istnieją narzędzia przekształcające JAR w aplikacje środowiskowe
  - Launch4J,
  - IzPack,
  - Jar Bundler.

Podpisywanie plików JAR

- Pliki JAR, które będą szeroko dystrybuowane muszą być podpisane cyfrowo.
- Podpis cyfrowy daje użytkownikowi gwarancję, że
  - Kod nie został zmodyfikowany od momentu podpisania.
  - Kod pochodzi od firmy, która go podpisała.
  - Rzetelność podpisu została zweryfikowana przez zaufany ośrodek autoryzacji
- Kosz uzyskania certyfikatu zależy od zasad dystrybucji oprogramowania
  - OpenSource Code Signing: około 100 zł / rok
  - Standard Code Signing: około 550 zł / rok

Narzędzia do podpisywania

jarsigner Narzędzie do podpisywania i weryfikacji plików JAR.

keytool Zarządza parami kluczy (prywatny-publiczny) i certyfikatami.

Tworzenie certyfikatu

- W pierwszym kroku konieczne jest wygenerowanie klucza prywatnego z użyciem *keystore*

```
C:\j2sdk1.4.1_07\bin>keytool -genkey -keyalg RSA -keysize 1024 -alias cunizetowski
Enter keystore password:  moje_haslo
What is your first and last name?
  [Unknown]:  Certacy Unizetowski
What is the name of your organizational unit?
  [Unknown]:  Moja Firma
What is the name of your organization?
  [Unknown]:  Oddzial w Moja Firma
What is the name of your City or Locality?
  [Unknown]:  Szczecin
What is the name of your State or Province?
  [Unknown]:  Zachodniopomorskie
What is the two-letter country code for this unit?
  [Unknown]:  PL
Is CN=Certacy Unizetowski, OU=Moja Firma, O=Oddzial w Moja Firma, L=Szczecin, ST=Zachodniopomorskie, C=
  [no]:  yes

Enter key password for <cunizetowski>
        (RETURN if same as keystore password):
```

Rysunek 1: Generowanie certyfikatu [UNIZETO, 2017]

Łańcuch certyfikatów

- Poprawnie wygenerowany klucz będzie zawierał informacje o wszelkich powiązanych certyfikatach

```
C:\j2sdk1.4.1_07\bin>keytool -list
Enter keystore password:  moje_haslo

Keystore type: jks
Keystore provider: SUN

Your keystore contains 3 entries

certumca, 2005-07-22, trustedCertEntry,
Certificate fingerprint (MD5): 2C:8F:9F:66:1D:18:90:B1:47:26:9D:8E:86:82:8C:A9
certumlevel_1, 2005-07-22, trustedCertEntry,
Certificate fingerprint (MD5): A1:42:3D:0A:27:16:ED:DC:2E:94:81:29:D6:3B:98:52
cunizetowski, 2005-07-22, keyEntry,
Certificate fingerprint (MD5): 9F:67:DB:A3:83:49:E6:73:E9:7C:BE:61:EE:91:8F:89
```

Rysunek 2: Łańcuch certyfikatów [UNIZETO, 2017]

Podpisywanie kodu

- Korzystając z narzędzia *jarsigner* podpisujemy kod

Podpisywanie kodu

```
jarsigner plik_jar nazwa_klucza
```

- Możemy teraz zweryfikować poprawność podpisu

Weryfikacja podpisu

```
jarsigner -verify -verbose -certs plik_jar
```

```
smk    20348 Sat Dec 06 19:57:34 CET 2003 src/Notepad.java

      X.509, CN=Certacy Unizetowski, O=Oddzial w Moja Firma, C=PL (cunizetowski)

      X.509, CN=Certum Level I, O=Unizeto Sp. z o.o., C=PL (certumlevel_1)
      X.509, CN=Certum CA, O=Unizeto Sp. z o.o., C=PL (certumca)
      X.509, CN=Certacy Unizetowski     , OU=Moja Firma, O=Oddzial w Moja Firma,
 L=Szczecin, ST=Zachodniopomorskie, C=PL


  s = signature was verified
  m = entry is listed in manifest
  k = at least one certificate was found in keystore
  i = at least one certificate was found in identity scope

jar verified.
```

Rysunek 3: Weryfikacja podpisu [UNIZETO, 2017]

Proguard

- Proguard jest narzędziem do optymalizacji i zaciemniania kodu.
- Zmniejsza i zabezpiecza pliki wyjściowe.

Wywołanie

```
proguard @myconfig.pro
```

Plik konfiguracji

```
-injars       myapplication.jar
-outjars      myapplication_out.jar
-libraryjars  <java.home>/lib/rt.jar
-printmapping myapplication.map

-keep public class com.example.MyMain {
      public static void main(java.lang.String[]);
}
```

Aplety

- Java umożliwia tworzenie aplikacji osadzonej na stronach internetowych - apletów.
- Kod apletu był pobierany z serwera na maszynę użytkownika i uruchamiany lokalnie pry użyciu plug-in do przeglądarki.
- Ze względu na rozwój technologii i kwestie bezpieczeństwa ta metoda dystrybucji kodów nie jest już dłużej stosowana.
- Chcąc dystrybuować oprogramowanie poprzez sieć możemy stosować *Java Web Start*.

Java Web Start

- Java Web Start jest technologią uruchamiania aplikacji bezpośrednio z Internetu.
  - Aplikacja jest udostępniana za pomocą przeglądarki.
  - Aplikacja jest wykonywana poza środowiskiem przeglądarki.
  - Do poprawnego działania aplikacja musi być podpisana.

Plik JNLP

- Do skorzystania z JWS konieczne jest utworzenie pliku opisującego działanie aplikacji.
- Opis jest dostarczany w formacie Java Network Launch Protocol (JNLP).
- Jest to plik XML z rozszerzeniem jnlp zawierający informacje o uruchamianym programie i jego dostawcy.
- W tej technologi są udostępniane przykłady na stronach Oracle.

Przykład pliku JNLP

Dynamic Tree Demo

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+" codebase=
"https://docs.oracle.com/javase/tutorialJWS/samples/deployment/DynamicTreeDemo"
    href="dynamictree_webstart.jnlp">
    <information>
        <title>Dynamic Tree Demo</title>
        <vendor>Dynamic Team</vendor>
    </information>
    <resources>
        <!-- Application Resources -->
        <j2se version="1.7+"
              href="http://java.sun.com/products/autodl/j2se"/>
        <jar href="DynamicTreeDemo.jar"
            main="true" />

    </resources>
    <application-desc
         name="Dynamic Tree Demo Application"
         main-class=
           "webstartComponentArch.DynamicTreeApplication"
         width="300"
         height="300">
     </application-desc>
     <update check="background"/>
</jnlp>
```

Wskazówki dla deweloperów

- Podpisuj certyfikatem z uznanego centrum certyfikacji.
- Żądaj minimalnego zakresu dostępu.
- Optymalizuj pliki JAR i powiązane zasoby.
- Używaj protokołu weryfikacji wersji i używaj pobierania aktualizacji w tle.
- Upewnij się, że klient ma odpowiednią wersję Java Runtime Environment

Tworzenie wersji kodu

- Jeżeli tworząc kolejne wersje kodu będziemy tworzyli osobne i opisane pliki JAR to możemy zautomatyzować pobieranie nowych wersji.
  - DynamicTreeDemo__V1.0.jar
- Możemy też sprawić, aby nowe wersje aplikacji były pobierane w tle.

Sprawdzanie aktualności kodu

```
<resources>
    <!-- Application Resources -->
    <j2se version="1.6+"
        href="http://java.sun.com/products/autodl/j2se"
            max-heap-size="128m" />
    <jar href="DynamicTreeDemo.jar"
        main="true" version="1.0"/>
    <jar href="SomeOther.jar" version="2.0"/>
    <property name="jnlp.versionEnabled"
        value="true"/>
    <!-- ... -->
</resources>
<update check="background"/>
```

Sprawdzanie wersji JRE

- Sprawdzenie wersji JRE posiadanej przez użytkownika pozwala upewnić się, że aplikacja będzie mogła zostać poprawnie uruchomiona.
- Oracle oferuje narzędzie w JavaScript *deployJava*, które pozwala weryfikować wersję Javy.

Sprawdzanie wersji JRE

```
<script src="https://www.java.com/js/deployJava.js"></script>
<script>
    var url = "dynamictree_applet.jnlp";
    deployJava.createWebStartLaunchButton(url, '1.6.0');
</script>
```

Bibliografia

[UNIZETO, 2017] UNIZETO (2017).
Java code signing 1.4 – 6.0 – podpisywanie kodu.