Maven

Autori:
Goran Savić
Milan Segedinac

1. Izgradnja projekta

Razvoj softvera uključuje i korišćenje dodatnih tehnologija za podršku razvoju, kao što su automatizovana izgradnja projekta ili kontrola verzija programskog koda. Kada je reč o automatizovanoj izgradnji projekta (eng. *build*), deo operacija pri kreiranju softvera se može automatizovati. Neke od operacija koje se mogu automatizovati su dobavljanje biblioteka programskog koda, kompajliranje, spajanje i organizovanje koda (eng. *packaging*), postavljanje koda u okruženje za korišćenje, npr. na veb server (eng. *deployment*), izvršavanje testova itd. Moguće je i automatizovano u nizu izvršiti sve operacije koje su deo životnog ciklusa razvoja softvera.

Postoji veliki broj do sada razvijenih alata za automatizovanu izgradnju projekta. Ovde ćemo prikazati Apache Maven, kao trenutno najpopularniji alat za izgradnju projekta u Java aplikacijama. Treba pomenuti i da se u poslednje vreme umesto Maven alata u Java aplikacijama sve češće koristi alat Gradle, kao noviji alat.

2. Maven

Apache Maven je namenjen izgradnji i upravljanju Java projektima. Za razliku od drugih alata čiji je fokus generičnost, Maven je ciljno uže fokusiran tako da obezbedi dobru podršku standardnom životnom ciklusu Java projekata. U skladu sa tim, postoji ugrađena podrška za standardne delove ovog ciklusa, kao što su kompajliranje, pokretanje testova, kreiranje jar fajla, itd. Nestandardne funkcionalnosti se mogu realizovati kao *plugin*.

Svaka softverska jedinica (projekat ili biblioteka) se prema Maven terminologiji naziva artefakt (eng. *artefact*). Svaki artefakt ima svoj identifikator, a srodni artefakti se organizuju u grupe. Artefakt je određen svojim koordinatama koje čine:

- identifikator artefakta
- identifikator grupe kojoj artefakt pripada
- verzija artefakta

Maven zahteva da Java projekat sledi sledeću predefinisanu strukturu:

- `src/main/java`
  - izvorni kod aplikacije
- `src/main/webapp`
  - izvorni kod veb aplikacije
- `src/test/java`
  - izvorni kod Java testova
- `src/main/resources`
  - dodatni fajlovi koje koristi izvorni kod
- `src/test/resources`
  - dodatni fajlovi koje koriste testovi
- `pom.xml`
  - konfiguracioni fajl koji opisuje Maven projekat
  - definiše kako će Maven da upravlja projektom

Za izgradnju projekta, važan je fajl *pom.xml* u kojem se opisuje projekat i specificiraju podaci potrebni za njegovu automatizovanu izgradnju. Pogledajmo primer ovog fajla.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>vp.izgradnjaProjekta</groupId>
  <artifactId>mavenPrimer</artifactId>
  <version>1.3</version>

  <packaging>jar</packaging>

  <name>Maven Primer</name>

  <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>1.5.4.RELEASE</version>
      <relativePath/>
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>6.0.6</version>
    </dependency>

  </dependencies>

</project>
```

Obzirom da je i specifikacija pom.xml fajla prolazila kroz izmene, tag *modelVersion* označava verziju *POM* modela koju koristimo. Nakon toga definisane su koordinate projekta. Maven će pri izgradnji projekta izvršiti automatsko pakovanje fajlova u format koji specificiramo u tagu *packaging*. U prikazanom primeru, rezultat izgradnje će biti *jar* fajl.

Pom fajl može da nasleđuje konfiguraciju iz roditeljskog pom fajla korišćenjem *parent* taga. Ovo je slučaj sa prikaznim POM fajlom, koji preuzima konfiguraciju artefakta *spring-boot-starter-parent*. Tag *relativePath* objasnćemo kasnije pri objašnjenju automatskog dobavljanja zavisnih biblioteka.

Maven fajl može da definiše proizvoljan broj *properties* promenljivih koje dodatno opisuju projekat. Npr. u primeru je naznačeno da se koristi UTF-8 za enkodovanje teksta u izvornim fajlovima.

Tag *dependencies* definiše zavisnosti projekta, što je u nastavku detaljnije objašnjeno.

Upravljanje zavisnostima

Zavisnostima smatramo sve biblioteke koje projekat koristi da bi realizovao funkcionalnosti. Npr. za projekat iz prethodnog primera, potrebne su klase koje realizuju vezu sa MySQL sistemom za upravljanje bazom podataka. Ove klase se nalaze u biblioteci `mysql-connector-java`. Maven omogućuje automatsko dobavljanje ovakvih biblioteka. Dobavljanje se zasniva na tome da postoji javni Maven repozitorijum koji skladišti ove biblioteke. Najčešće se koristi centralni Maven repozitorijum na adresi http://central.maven.org/maven2/, koji sadrži sve popularne Java biblioteke. Svaka biblioteka u sebi sadrži Maven POM fajl. Da bismo automatski dobavili biblioteku, neophodno je u okviru taga *dependency* specificirati koordinate biblioteke kako su definisane u njenom POM fajlu. Koordinate biblioteke se mogu pronaći pretragom repozitorijuma. Ako koristimo centralni Maven repozitorijum, pretraga se može izvršiti korišćenjem internet pretraživača ili na internet stranici na adresi https://mvnrepository.com/repos/central.

Biblioteka koju Maven automatski dobavi iz udaljenog repozitorijuma biće uskladištena u lokalni repozitorijum. Na Windows operativnim sistemima, podrazumevana lokacija za lokalni repozitorijum je `C:\Users\<Username>\.m2\repository`. Ako na lokalnom računaru imamo više projekata koji koriste iste biblioteke, programski kod biblioteka će biti uskladišten samo jednom, u lokalnom Maven repozitorijumu. Svi projekti će samo referencirati biblioteke iz lokalnog repozitorijuma, čime se pojednostavljuje upravljanje bibliotekama.

Osim biblioteka referenciranih u tagu *dependencies* POM fajla, Maven će automatski dobaviti i sve biblioteke od kojih referencirane biblioteke zavise. Ovakve zavisnosti nazivamo tranzitivne zavisnosti. Sve zavisnosti koje projekat sadrži formiraju stablo zavisnosti (eng. *dependency tree*). Pri tome, neke od zavisnosti iz stabla je referencirao projekat direktno, a neke su tranzitivno referencirane.

Osvrnimo se još jednom na gore pominjani tag *parent*. Kao što smo objasnili, ovim tagom takođe se referencira druga biblioteka, od koje je potrebno preuzeti konfiguraciju. Podrazumevano će biblioteka biti tražena u lokalnom folderu, u folderu koji je roditeljski folder ovom *POM* fajlu. Ukoliko se roditeljska biblioteka ne nalazi u lokalnom repozitorijumu, potrebno je dobaviti iz udaljenog repozitorijuma, što se definiše praznim tagom *relativePath*.

Maven životni ciklus izgradnje projekta

U prethodnom delu teksta smo objasnili kako se može specificirati na koji način Maven vrši izgradnju projekta. Pogledajmo sada kako se sama izgradnja projekta inicira. Maven vrši izgradnju projekta prolazeći kroz predefinisane faze životnog ciklusa projekta. Pri tome, u svakoj fazi se može izvršiti određen broj automatizovanih operacija, koje se u Maven terminologiji nazivaju ciljevi (eng. *goals*). Na primer, ciljevi koji se izvršavaju su kompajliranje, izvršavanje testova, pakovanje, kopiranje fajlova, ...

Postoje tri predefinisana ugrađena životna ciklusa

- *default* - sadrži faze vezane za izgradnju i postavljanje projekta
- *clean* - uklanjanje fajlova i foldera koje je Maven kreirao tokom izgradnje projekta
- *site* - generisanje projektne dokumentacije

Pogledajmo *default* ciklus kao najvažniji ciklus koji se i najčešće izvršava. Ciklus se sastoji od sledećih faza:

- *validate* - provera da li su sve informacije o projektu dostupne i ispravne
- *process-resources* - kopiranje resursa projekta na destinaciju za izgradnju
- *compile* - kompajliranje izvornog koda
- *test* - izvršavanje jediničnih testova
- *package* - pakovanje kompajliranog koda u format za distribuciju
- *integration-test* - postavlja paket u okruženje u kojem se mogu izvršiti integracioni testovi
- *verify* - provera da li je paket ispravan
- *install* - instaliranje paketa u lokalni repozitorijum da bude dostupan drugim lokalnim projektima
- *deploy* - postavljanje paketa u udaljeni repozitorijum

Kao što smo rekli, za svaku fazu vezan je jedan ili više ciljeva koji se izvršavaju u toj fazi. Ciljevi su definisani u određenom *plugin*-u. Npr. u *compile* fazi se izvršava *compile* cilj iz *plugin*-a *Compiler*. Postoji javni repozitorijum *plugin*-ova odakle Maven automatski preuzima *plugin* kada je potreban u projektu.

Iniciranje izgradnje projekta vrši se navođenjem faze životnog ciklusa koju treba izvršiti. Pri tome, **automatski se izvršavaju sve faze koje prethode navedenoj fazi u životnom ciklusu**. Npr. ako u komandnoj liniji izvršimo naredbu `mvn package`, izvršiće se operacije iz faze *package*, ali i svih faza koje joj prethode u *default* životnom ciklusu.

Podrška za Maven u Eclipse IDE

Eclipse IDE ima ugrađenu podršku za Maven kroz *plugin* pod nazivom m2e. Ovaj *plugin* omogućuje kreiranje novog Maven projekta, koji će kreirati projekat u skladu sa strukturom koju *maven* zahteva. Takođe, projekat će sadržati *pom.xml* fajl sa podrazumevanim sadržajem. Slično, moguće je operacijom *import* ubaciti postojeći Maven projekat u Eclipse.

Za Eclipse Maven projekat, kada se vrši snimanje *pom*.xml fajla, automatski će iz udaljenog repozitorijuma biti preuzete biblioteke koje POM fajl referencira. Kompajliranje fajlova će Eclipse automatski izvršiti pri snimanju fajla.

Pored preuzimanja biblioteka i kompajliranja, koje Eclipse automatski vrši, možemo pokrenuti proizvoljnu fazu iz životnog ciklusa projekta. Ovo se vrši desnim klikom na pom.xml fajl i izborom stavke Run As iz padajućeg menija. U ovoj stavci, može se pokrenuti neka od ponuđenih faza ili izborom opcije Maven build ... uneti proizvoljna faza koja se izvršava. Još jednom napominjemo da će izvršavanje faze izvršiti i operacije iz svih faza koje joj prethode u životnom ciklusu izgradnje.