Architektura CRCE

KIV/SAR - semestrální práce

student: Radek VAIS, Zdeněk VALEŠ
mail: vaisr@students.zcu.cz, valesz@students.zcu.cz
datum: 1.1.2019

1 Zadání

CRCE[1] je komponentové úložiště vyvíjené v rámci výzkumu na Katedře informatiky a výpočetní techniky na Západočeské univerzitě v Plzni. Jeho hlavní vlastností je podpora kontroly kompatibility verzí různých OSGi komponent nebo webových api.

1.1 Motivace

Vývoj projektu probíhá již několik let a vystřídalo se na něm mnoho vývojářů, což způsobilo zanesení větších či menších architektonických dluhů. Projekt již můžeme klasifikovat jako velký. Existuje potřeba, aby střídající se vývojáři (např. v rámci bakalářských prací) snadno a rychle zprovoznili základní instalaci a získali tak prostředí, pro rychlý vývoj nových částí.

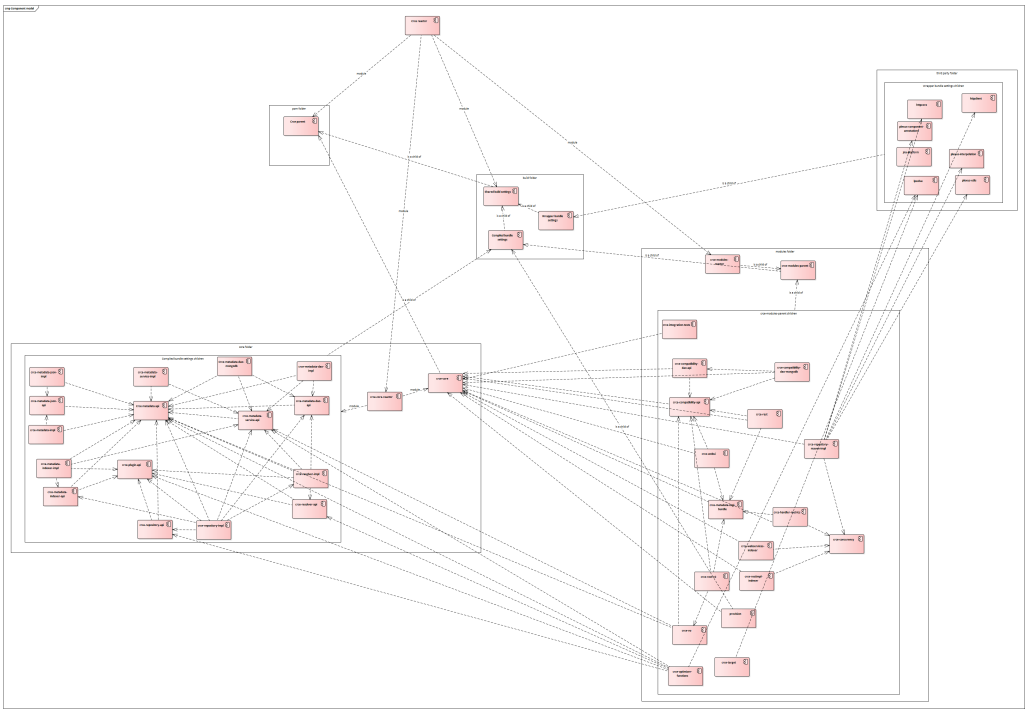
1.2 Cíle projektu

- Unifikovat proces sestavení aplikace a spuštění.
- Zdokumentovat současnou architekturu aplikace.
- Analyzovat závislosti modulů v projektu a ověřit, zda některé moduly mají být součástí jádra.
- Připravit Docker image pro vývojáře.

[1] Component Repository supporting Compatibility Evaluation

2 Analýza architektury

Celý projekt je rozdělen do čtyř hlavních modulů: sdílené nastavení sestavení, jádro, moduly a knihovny třetích stran. Ne zcela jasné je začlenění modulu (jedná se o modul provision) pro nasazení mezi moduly CRCE. Závislostí modulů na jádře pozorujeme dva typy. Prvním typem je závislost na agregačním modulu jádra, druhým je jmenovitá závislost na komponentách jádra. Tyto skutečnosti jsou patrné na celkovém diagramu na Obrázku 1.

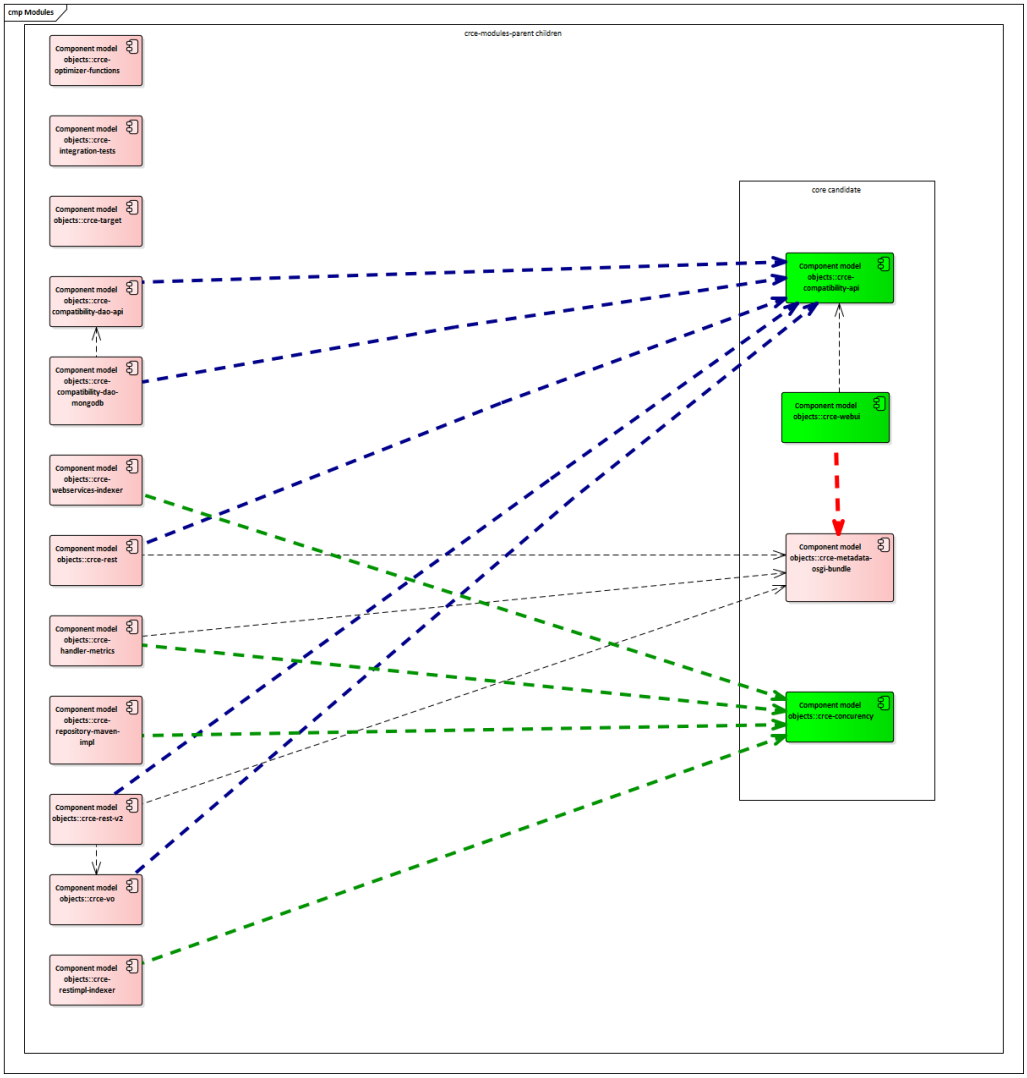


Obrázek 1: Diagram závislostí všech maven komponent projektu CRCE před provedenými úpravami.

Během analýzy modulu pro moduly CRCE jsme identifikovali několik kandidátů na přesun do jádra. Za kandidáty jsme volili takové komponenty, které využívá velká část ostatních modulů a zároveň tyto moduly mají závislosti do jádra. Komponenta webového rozhraní (crce-webui) tomuto pravidlu sice odporuje, představuje ale základní komponentu, bez které by úložiště nešlo použít. Přesouvané komponenty jsou: crce-compatibility-api, crce-webui, crce-concurency (graficky znázorněno na Obrázku 2.

Po identifikace kandidátů jsme z přesunu vyřadili komponentu metadata-osgi-bundle, kvůli nedostatečné obecnosti. Komponenta je schopna indexovat pouze OSGI komponenty – přesun do jádra by byl možný, pokud by uměla in-

dexovat obecné JAR. Komponenta crce-webui bohužel obsahuje silnou závislost na této komponentě, která brání přesunu crce-webui do jádra. Závislost spočívá v použití několika konstant, které jsou v definované v metadata-osgi-bundle. Navrhli jsme tedy přidání metody do rozhraní indexeru pro zjištění jaké elementy indexer zpracovává a silnou závislost jsme nahradili slabou závislostí (stejné konstatny jsme dodefinovali i v crce-webui).



Obrázek 2: Diagram původních závislostí uvnitř mudulu moduly CRCE. Zeleně jsou označeny přesouvané komponenty. Červeně je označena nevhodná závilsost.

3 Provedené změny

V projektu jsme provedli několik změn, které by měly vést k zjednodušení práce budoucích vývojářů. Pro přehlednost dodávám, že v této sekci pojem *modul* znační maven modul v kořenovém adresáři CRCE a pojem *komponenta* značí jeden z maven modulů obshující konkrétní implementaci (*crce-webui*, *crce-metadata-api*, ...).

3.1 Nový modul pro sestavení

Do kořene CRCE byl přidán nový modul *deploy*, do kterého byla přesunuta komponenta *provision*, používaná ke spouštění CRCE. Jsou zde agregovány všechny komponenty z jádra a komponenty z modulu *modules*, které jsou zachycené v *crce-default-modules*.

Nově přidaná komponenta *crce-default-modules* tedy slouží jako agregátor komponent z modulu *modules*, které mají být použité ve výsledné distribuci.

3.2 Přesun vybraných modulů do jádra

Do jádra byly z modulu *modules* přesunuty komponenty: *crce-compatibility-api*, *crce-webui* a *crce-concurency*. V případě *crce-webui* bylo nutné odstranit závislost na *crce-core* (jinak by se jednalo o kruhovou závislost) a nahradit ji jednotlivými závislostmi na vybraných komponentách z jádra.

V komponentě *crce-compatibility-api* bylo nutné přidat výchozí (prázdnou) implementaci rozhraní *Compatibility* a *Diff*. Pro tyto implementace byl přidán unit test.

Součástí přesunu komponent do jádra bylo přidání unit testu do *crce-metadata-osgi-bundle*, který dokazuje schopnost této komponenty indexovat pouze OSGI bundly a ne obecné JAR.

3.2.1 Nahrazení silné závislosti v komponentě crce-webui

Komponenta *crce-webui* obsahovala silnou závislost na komponentě *crce-metadata-osgi-bundle*. Tato závislost spočívala v použití několika konstant, označujících indexované elementy, definovaných v rozhraní *NsOsgiIdentity*. Konstanty byly použity ve třídě *ResourceServlet* k filtrování zobrazovaných OSGI bundlů.

Po dohodě s vyučujícím jsme tuto závislost nahradili slabou závislostí – konstatny stejného typu jsme dodefinovali i v komponentě *crce-webui*. Silná závislost byla odstraněna, nicméně pokud se v budoucnu změní komponenta

crce-metadata-osgi-bundle, bude nutné kzontrolovat, zda změna neovlivnila *crce-webui* a případné nesrovnalosti opravit.

Aby bylo podobným závislostem v budoucnosti zamezeno, rozhraní idnexeru bylo rozšířeno o metodu, která vrací elementy, jež je indexer schopen detekovat.

4 Dockerizace projektu

Do modulu *deploy* byl přidán Dockerfile, kterým je možné sestavit docker image obsahující spustitelnou verzi CRCE. Do takto spuštěného CRCE lze doinstalovávat nové OSGI komponenty a není nutné celý projekt znovu sestavovat.

Součástí dockerizace bylo také přidání konfigurace připrojení k databázi Mongo. To je nyní řešení proměnnou prostředí *mongo_connection*. V případě, že tato proměnná není nastavena, použije se výchozí hodnota, která je dostačující pro databáze běžící na localhostu a standardním portu. Výhodou tohoto přístupu je možnost konfigurovat connection string v Dockerfile a při změně pouze znovu sestavit image.

5 Uživatelská příručka

Provedené změny ovlivnily základní používání projektu při vývoji.

Pro sestavení na platformě Linux lze použít bash script *build.bash*, který automaticky spustí jednotlivé kroky sestavení pomocí nástroje Maven.

V případě, že databáze neběží na localhostu, nebo standardním portu, je možné nastavit připojení strkze proměnnou prostředí *mongo_connection*. Tato proměná by v případě použití měla obsahovat celý connection string. Pokud není nastavená, pro připojení k databázi bude použita výchozí hodnota: *mongodb://localhost:27017*.

5.1 Docker

K sestavení Docker image je možné použít Dockerfile, který se nachází v modulu *deploy*. Sestavení image je provedeno příkazem:

```
docker build . -t crce-docker
```

Výsledný image obsahuje distribuci Apache Felix 5.0.0, JDK 1.7 a všechny OSGI bundly z CRCE. Spuštění lze provést příkazem:

```
docker run -it \
   -p 8080:8080 \
   --add-host mongoserver:172.17.0.1 \
   -v /felix/deploy:/felix/deploy
   crce-docker \
```

Parametr *add-host* slouží k přidání záznamu do /etc/hosts v kontejneru a image se pak může k databázi běžící na hostovacím stroji dostat pomocí hostname *mongoserver*. IP adresa 172.17.0.1 je adresa, kterou docker přiřazuje hostovacímu stroji. Do adresáře */felix/deploy* na hostovacím stroji je možné vložit OSGI bundly a ty následně nainstalovat a spustit skrze Gogo shell. K běžící instanci je možné přistoupit na adrese *http://localhost:8080/crce*.

6 Závěr

V rámci projektu jsme naplnili cíl usnadnit novým vývojářům spuštění výchozí aplikace, především přípravou docker image. Nepodařilo se nám zjednodušit proces sestavení celé aplikace na jeden Maven příkaz nicméně pro prostředí Linux jsme vytvořili script, který úkony nutné pro sestavení provede automaticky. Dále jsme přesunuli několik modulů do jádra CRCE z důvodu jejich provázanosti s ostatními nebo nutnosti pro používání.