VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

RYCHLÉ DOTAZOVÁNÍ NAD METADATY JAZYKA JAVA

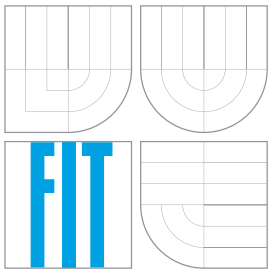
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VLADIMÍR FALTÝN

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

RYCHLÉ DOTAZOVÁNÍ NAD METADATY JAZYKA JAVA

FAST QUERIES OVER JAVA LANGUAGE METADATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VLADIMÍR FALTÝN

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. KŘIVKA ZBYNĚK, Ph.D.

BRNO 2015

Abstrakt

Výtah (abstrakt) práce v českém jazyce.

Abstract

Výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Klíčová slova v českém jazyce. dotazy, metadata, java, jazyk

Keywords

Klíčová slova v anglickém jazyce.

Citace

Vladimír Faltýn: Rychlé dotazování nad metadaty jazyka Java, bakalářská práce, Brno, FIT VUT v Brně, 2015

Rychlé dotazování nad metadaty jazyka Java

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana ...

.....
Vladimír Faltýn
16. ledna 2015

Poděkování

Zde je možné uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc.

Obsah

Kapitola 1

Úvod

Proč aplikaci vytváříme? Účelem této bakalářské práce je vytvořit co nejednoduší dotazovací jazyk pro dotazování nad metadaty jazyka Java. Jedná se o knihovnu, která by byla zakomponovaná do aplikací určených pro zkoumání Java kódu. Použití těchto aplikaci je migrace Java aplikaci, hledaní artefaktů v kódu, analýza aplikací a další použiti by se dala najít.

V první části si popíšeme použité technologii. Jejích vlastnosti, použití, licence, případně zajímavosti z historie jak vznikly jejích původ milníky a jejích aktuální stav.

Ve druhé kapitole si specifikujeme jak naše aplikace bude vypadat co dokáže a jaká bude mít omezení na základě analýzy. Srovnání s existující nebo podobnou aplikací pokud existuje. A nějaké příklady jak by aplikace mohla fungovat.

V návrhu se zaměříme na to jak postavit aplikaci aby fungovala. Navrhneme způsob spuštění aplikace a vložení dotazu. Dále pak návrh zapracování použitých technologii do aplikace. Poté provedeme návrh struktury aplikace rozdělení na menší části které spolu budou koordinovat.

V implementaci si zopakujeme jak jsme aplikaci navrhly a jaká by měla dodržovat specifika aby opravdu dělala to co chceme. Poté popíšeme jak jsme jí implementovali skutečně a kde jsme museli udělat odbočku od návrhu a co se změnilo ve specifikaci nebo co nemůžeme dodržet. Na konci kapitolo zmíníme systémové požadavky.

Náplní šesté kapitola je testování zde zjistíme jak aplikace skutečně funguje a případné problémy. Aplikaci podrobíme sérií testů které budou zaměřené na určité části, kde by se mohly vyskytnout nějaké problémy a tam kde chceme ukázat sílu aplikace. Tak aby testování nebylo zaujaté a obsáhl co nejširší škálu možností a zároveň ukázalo její klady a hranice.

V poslední kapitole zhodnotím dosažené výsledky aplikace v testech a také můj přínos s touto aplikací pro případné uživatele. Zde také uvedu jak by mohl pokračovat další vývoj.

Příloha CD s kódem a odkaz na elektronické umístění (https://github.com/Faltyn/Java-AST-query-language).

Kapitola 2

Použité technologie

Aplikace, kterou jsem v rámci bakalářské práce vytvořil, používá různé nástroje pro její funkčnost. Cílem této kapitole je stručný popis těchto nástrojů.

2.1 Jazyk Java

Java je Objektově orientovaný jazyk vyvinutý firmou Sun Microsystems představený v roce 1995. Jedná se o jeden z nejpoužívanějších jazyku na světě. Především kuli své přenositelnosti, aplikace se používají v různých zařízení čipové karty JavaCard, mobily Java ME, pro desktopy Java SE a distribuované systémy Java EE. Tyto technologie se dohromady nazývají platforma Java. V roce 2007 byli uvolněny zdrojové kódy. Java je vyvíjena jako open source.

Vlastnosti jazyka Java Java je objektově orientovaný jazyk až na výjimku osmi datových typu které nejsou kuli výkonnostním důvodům objekty (celá čísla, desetinná čísla, logické hodnoty a znaky).

Multiparadigmatický jazyk podpora vice jak jednoho programovacího paradigma. Paradigma způsob jak řeší určitý problém co nejefektivněji (př. Strukturované programování a imperativní programování).

Strukturované programování rozdělení algoritmu na dílčí části do metod jedná se o podmnožinu imperativního programování.

Imperativní programování algoritmus je posloupnosti příkazů určující přesný postup, jak danou úlohu řešit.

Reflexivní za běhu dokáže získat informace o objektu (název typu, druh typu, atributy, konstruktory, metody vnořené datové typy, anotace atd.). S informacemi získanými za běhu se dá déle pracovat volat konstruktory, přiřazovat typy atd..

Distribuovaný podpora aplikací v síti práce se soubory, klienty a servery.

Interpretovaný nevytváří se strojový kód ale mezi kód bajtkód tento kód je nezávislí na architektuře pc. Kód se poté spouští v JVM (Java Virtual Machine) emulačním stroji. V pozdějších verzích byl mezi kód kompilován do strojového kódu v čase spuštění (JIT – just in time compilation). Výrazné zrychlení běhu aplikace za cenu pomalejšího náběhu. V současnosti se používá HotSpot compiler převedení často používaného kódu do strojového plus další optimalizace. Existuje však i přímá hardwarová podpora namisto emulace v JVM (Java Virtual Machine).

Robustní je určen pro psaní vysoce spolehlivého softwaru, odstranění některých nebezpečných programátorských konstrukcí.

Bezpečný chrání počítač na kterém běží kód běh v JVM zabezpečí ochranu hardwaru tak i softwaru před jinými aplikacemi, které by mohly zapříčinit problémy.

Automatická správa paměti o kterou se stará garbage collector ulehčení práce programátorům o správu paměti se plně stará Java méně chyb způsobené unikem paměti.

Více úlohoví zpracování více vláken, paralelně běžící kód.

Historie Začátek se datuje roku 1991 James Gosling, Bill Joy, Mike Sheridan a Patrick Naughton započali na Stealth Project (tajný projekt) což měl být nějaký systém pro domácí spotřebiče později přejmenován na Green Porject (zelený projekt). Vedoucím týmu byl James Gosling, který začal pracovat na novém jazyku který pojmenoval Oak podle stromu který rostl před oknem jeho kanceláře. V roce 1995 ho přejmenovali na Java slangové označení pro kávu v Americe. 1996 první oficiální verze JDK (Java Development Kit) 1.0. Další verzí bylo JDK 1.1 které už obsahovalo vnořené třídy, reflexi, JavaBeans, JDBS, Java RMI, rozšíření AWT a JIT pro Microsoft Windows. Rok 1998 byl zlomový rozdělení Javy do tří částí J2SE , J2EE a J2ME. 2000 přidání HotSpot do JVM. V roce 2004 vyšla specifikace pro Java 5.0. Výčtový typ (angl. enum) se stalo plnohodnotnou třídou. Poslední vydanou verzí Javy je Java 8 2014.

2.2 Knihovna Procyon

Je metaprogramming nastroj pro generování a analýzu Java kódu. Vyvíjí ho Mike Strobel pod licenci The Apache Software License, Version 2.0. Název Procyon je odvozen od souhvězdí Procyon.

2.2.1 Core Framework

Core framework obsahuje podpůrné třídy, které jsou použité v API Pocyonu. To je vkládání řetězců a manipulace s kolekcí rozšíření, správa cest souborů, zamrznuti objektu a kolekcí a další podporu.

2.2.2 Reflection Framework

Reflection framework poskytuje bohatou reflexy a generování kódu aplikaci s plnou podporou pro generiku, zástupných znaků a další vysoko úrovňové typy jazyka Java. Je založen na .NET System.Reflection a System.Reflection.Emit API a je určen k mnoha nedostatkům jádra Java reflexe API, které nabízí poměrně limitovanou a těžkopádnou podporu pro generiku typu kontroly. K jeho generování kódu patří TypeBuilder, MethodBuilder a byte kód vysílač.

2.2.3 Expressions Framework

Expression framework poskytuje přirozenější způsob generování kódu. Spíše než požadování byte kódu vysílá přímo, jako procyon-reflection a další populární knihovny jako ASM. Procyon-expression umožňuje kódovou kompozici pomoci deklarativního výrazového stromu. Tyto expresivní stromy poté umožňují sestavit přímo na zpětné volání nebo s MethodBuilder. Procyon-expresion API je téměř přímý port System.Linq.Expression z .NET

Dynamic Language Runtime, minus dinamická podpora callsite (a mírnější pravidla pokud jde o typ převodu).

2.2.4 Compiler Toolset

Procyon-compilertools projekt je nedokončená práce která zahrnuje:

- Metadata tříd a byte kód kontrolu/manipulační zařízení na základě Mono.Cecil
- Optimalizace a dekompilator framework založený na ILSpy

Compiler sada nástrojů je v brzkém vývoji a změna je vyhrazena.

2.3 TinkerPop

TinkerPop je open source grafový výpočetní framework.

2.3.1 Blueprints

Blueprints je sbírka rozhraní, implementace, ouplementace a testovací sady pro modelování vlastnostního grafu. Blueprints je analogii JDBC, ale pro grafovou databázi. Jako takový poskytuje sadu rozhraní tak, aby vývojářům plug-and-play jejích grafové databázi backend. Navíc software je napsaný na vrcholu Blueprint funguje přes všechny Blueprints-enabled podporující grafovou databázi. V rámci TinkerPop softwaru, Blueprint slouží jako základ technologii pro:

Pipes

Pipes (potrubí) je framwork datových toků používaný v grafech procesu. Graf procesů se skládá z Pipe vrcholů spojený komunikačními hranami. Pipe implementuje jednoduchý výpočetní krok, který může být složený z jiných Pipe objektů, tak se vytvoří složitější výpočet. Tyto grafy toku dat umožňují rozdělení, sloučení, opakování. Obecně platí že umožňují transformovat data ze vstupu na výstup. Existuje mnoho Pipe tříd odvozených od hlavn Pipe.

Gremlin

Gremlin je jazyk grafů průchodů. může být využit v různých jazycích JVM.

Frames

Frames (rámy) umožňují Blueprint graf jako Java objekt. Umožňuje vytvoření rozhraní pro vrchol nebo hranu, tím se z ních stanou objekty. Rozhraní je anotované a tím umožňuje vytvořit předem připravené šablony pro společnou manipulaci s těmito objekty.

Furnace

Furnace (pec) je baliček grafových algoritmů pro Blueprint grafy. Existuje mnoho grafových algoritmů, které byli vyvinuty v průběhu dějin teorie a analýzy grafů. Mnoho z těchto algoritmů bylo navrženo pro neznačené single-relational grafy. Účelem pece je odhalit vlastnosti grafů (tj. připsat, multi-relational grafy) na single-relational grafové algoritmy. Kromě toho,

Furnace poskytuje různé implementace grafových algoritmů které jsou optimalizované pro různé grafové výpočetní scénáře – například single-machine grafy a distribuované grafy.

Rexter

Rexter je grafový server, který umožňuje přístup skrz binární protokol RexPro. HTTP webová služba poskytující standardní low-level služby GET, POST, PUT, DELETE metody, flexibilní rozšíření o model, který umožňuje plug-in jako vývoj pro externí služby (jako je adhoc grafové dotazy přes Gremlin), serve-side uložených procedůr napsaných v Gremlinu, a rozhraní na bázi prohlížeče Dog House. Rexster Console umožňující provést spuštění vzdálených scriptů vůči vnitřní konfiguraci grafů Rexster Servru.

2.4 AntLR

Uvidím jestli ho použiji.

Kapitola 3

Specifikace a analýza požadavků

Zadání BP: Rychlé dotazování nad metadaty jazyka Java

1. Seznamte se s nástroji pro získávání metadat a abstraktního syntaktického stromu jazyka Java (Java AST); doporučený je nástroj Procyon. Proveďte rešerši architektury a použití nástroje Procyon, kterou bude moci využít i komunita kolem nástroje.

2. Dále se seznamte s generátorem analyzátorů ANTLR a nástrojem WindUp využívajícím nástroj Procyon a vlastní reprezentaci metadat a Java AST.

3. Dle pokynů vedoucího navrhněte dotazovací jazyk vhodný pro rychlé získání informací z metadat zdrojových textů projektu v Javě. Dále navrhněte překlad vytvořeného jazyka do dotazovacího jazyka grafové databáze (např. Gremlin pro databázový systém Titan) nebo případně využijte přímo API této databáze.

4. Nástroj dle návrhu implementujte.

5. API vytvořeného nástroje zdokumentujte, testujte na minimálně 20 dotazech a navrhněte jeho integraci do WindUp.

3.1 Pro koho nebo co je aplikace vytvořena

Účelem knihovny potažmo aplikace je získání informací o zkompilovaném kódu a k jejímu dalšímu použití. Systém jakým by měla aplikace fungovat je dekompilace kódu získání metadat a AST, které se převedou do databáze grafu, kam se bude dotazovat pomocí vytvořeného jazyka pro dotazování. Tyto dotazy by měli být co nejednoduší, efektivní a intuitivní. Ale přitom by měli pokrýt co největší prostor pro složitější dotazy. Bude nutné v jazyku povolit vnořené dotazy kdy se do jednoho dotazu může na určité místo vložit dotaz vnitřní a to i více krát. Výsledkem dotazu bude odkaz nebo odkazy do grafu na danou třídu (tj. vždy se budeme dotazovat na třídy nebo třídu). Zde je několik případů užití co by měl daný jazyk zvládnout:

- Chci všechny třídy, které rozšiřují nebo jsou importovány do třídy "C"a zároveň tyto třídy implementují rozhraní který je rozšířen o interface "I".

- Chci všechny třídy, které implementují rozhraní I.

- Chci všechny třídy, které volají metodu z rozhraní I.

- Chci všechny třídy, které volají metodu z rozhraní, které rozšiřuje rozhraní I.
- Chci všechny třídy, které volají metodu daného jména z rozhraní, které rozšiřuje interface I s danými parametry.
- Chci všechny třídy, které volají metodu která implementuje interface I a ta je anotovaná anotací @A.
- Chci všechny třídy, které volají metodu která implementuje interface I a ta je anotovaná anotací @A s parametrem @A(foo="bar").
- Chci všechny třídy, které anotují anotaci @A.
- Chci všechny třídy, které importují pod třídu z třídy B.
- Chci všechny třídy, které importují pod třídu z třídy která implementuje pod rozhraní z rozhraní I.

3.2 Co bude umět aplikace

-== Obrázek: ? ==-

První co bude aplikace umět a co bude jejím základem tak je získání dat, tedy dekompilace nejjednodušší je najít už hotový nástroj, který to zvládne, jednou z požadavků na tento nástroj je aby byl aktuální. Specifikace Java se mění přidává se nová funkcionalita, opravují chyby a optimalizace pro zrychlení běhu. Aktuální verze je Java 8 vydaná v roce 2014. Výhodou takového nástroje by mělo být získání AST a metadat.

Druhou částí aplikace bude získání metadat k čemuž by měl posloužit dekompilátor (tuto funkcionalitu má samozřejmě i Java při zvolení správných parametru při spuštění aplikace). Metadata budou v surové podobě takže z nich musíme získat potřebné informace a uložit si je do přehledné struktury pro další použití. Z metadat budeme potřebovat názvy tříd, jejích vlastnosti, metody a cizí metody volané v té třídě.

Další částí je získání AST je to další z důležitých částí, ale tato aplikace pro svojí funkčnost prozatím stačí metadata avšak pro budoucí rozšíření nebo nemožnost získání některých dat by měl být nedílnou součástí této aplikace.

Předposlední částí aplikace je uložení získaných dat do úložiště nejlepší je použít nějakou z dostupných databází které již existují a jsou vhodné pro tento typ dat. Taková to databáze by měla být rychlá, jednoduchá, a schopná pracovat s velkým množstvím dat výhodou by bylo že data mají mezi sebou spojení které určuje směr vztahů mezi daty (tj. třídy, balíky, metody jejích umístění a vztahy: import, extend, implement atd.). Databází, která se nám nabízí je grafová databáze kde máme vrcholy a hrany. Vrcholy jsou úložiště dat s jejími vlastnostmi srovnatelné s jedním řádkem v tabulkové databázi. A hrany určují vztahy mezi jednotlivými vrcholy.

Poslední částí a zároveň cílem je vytvořit jazyk pro dotazování nad uloženými daty (tj. metadaty). Tento jazyk je dotazovací takže nějaká specifika musí mít. Nejlepší jak začít je prozkoumat existující dotazovací jazyky, jaký mají základ, poučit se z nedostatků a celkově ho analyzovat.

Jedním takovým jazykem je SQL. Z SQL jazyka nás zajímá pouze část dotazů do databáze, tedy pouze příkaz SELECT. Jak takový dotaz do databáze vypadá "SELECT "sloupc/sloupce"FROM "table"WHERE "podmínka"GROUP BY "seskupení"ORDER "seřazení". V první části říkáme co chceme z té tabulky v další odkud to chceme, poté za jakých

podmínek to chceme. V dalších částí pouze upřesňujeme jak výsledek bude vypadat jestli ho chceme seskupit a seřadit.

Další na řadě je analyzovat na co se budeme dotazovat několik případů bylo zmíněno už v předchozí sekci. Za prvé měli bychom se dotázat na vlastnosti třídy (např. název, typ , metody, anotace, pod třídy, rozšíření, rozhraní, importované třídy a volané metody). Za druhé dotazovat se může na více než jednu vlastnost pro komplikovanější dotazy. Užitečné by také bylo dotazy do sebe vkládat. Vhodné pro to abychom nemuseli psát několik dotazů, ale bude stačit jeden dotaz složený z několika. Další možností je dotazy skládat za sebe a postupně se budou vykonávat podle pořadí.

3.3 Popis užiti

Popis jednotlivých části co se bude dělat

Jak dekompilovat. Dekompilovat se bude pomocí nástroje Procyon je v brzkém vývoji, ale z dostupných nástrojů je nejaktuálnější a pro projekt bude dostačující. Problémem je nedostatečná dokumentace zdrojový kód je téměř bez komentářů, takže to bude boj, ale co se dá dělat.

Jak získat Metadata. Metadata získáme pomocí dekompilátoru Procyon. Výhodou získávání metadat je že není potřeba dekompilovat bajt kód tudýž je to rychlé. Výsledkem Procyonu je textový řetězec ze kterého budeme muset získat data pro další použití.

Jak získat AST. AST získáme taktéž z Procyonu tentokrát musíme dekompilovat. AST je ve stromové struktuře tak nám bude stačit když najdeme způsob jak ho projít a poté vytvořit z něho graf.

Uložení do databáze. Jako databázi jsem si vybral Titán jedná se o grafovou databázi která je rychlá přehledná a jednoduchá na pochopení. Data se do ní budou ukládat po řádcích přičemž jeden řádek bude jeden Vrchol. Tyto vrcholy se propojí hranami které popisují směr a vlastnictví nebo vztah objektů

Dotazování do databáze. Dotazovat se budeme prostřednictvím Jazyka který bude níže navržen. Dotaz se poté přeloží do dotazovacího jazyka grafové databáze.

Kapitola 4

Návrh Aplikace

-== Obrázek: Balíky a třídy ==- Aplikace je koncipovaná jako knihovna tudíž je rozdělená do více balíků. Každý balík bude zapouzdřovat jednu funkcionalitu o kterou si budou ostatní žádat (rozděl a panuj). Aplikaci jsem proto rozdělil do šesti částí první vrchní část "app"obsahuje nastavení, spuštění aplikace a 5 balíků ty popíšu níže:

4.1 Jádro

Jádro obsahuje pomocné algoritmy a výčtové typy které jsou použité ve více částech aplikace. Zde jsou popsaný třídy v jádře:

- Data.java představuje úložnou jednotku která uchovává informaci o jednom uzlu v grafu
- Tree.java je stromová struktura obecného rázu pro uchování dat uchovává instance Data.java.
- ProcessingData.java přesměrovává data z výstupu do proměnné. Procyon v základu všechny výstupy posílá na standardní výstup. Pomocí této třídy to změním a všechny informace se ukládají do proměnné.
- TypFile.java výčtový typ typu dat. Určuje jakého typu je třída v souboru.
- TypModifier.java výčtový typ typu modifikátor. Zde jsou uloženy všechny typy modifikátoru co jsou v jazyku Java.

4.2 AST

Tato třída není hotové je připravená pro další použití v případě potřeby obsahuje pouze jednu třídu a to k získání AST. Pokud bude třeba tak i implementují.

4.3 Metadata

Balík Metadata obsahuje dvě třídy které popisují získání metadat, následné analyzování metadat s uložením do Data.java a uspořádané do stromu Tree.java. -== Obrázek: parser-Metadat DKA ==-

4.4 Graph

Graph má za úkol vytvoření grafu obsahuje třídu ClassVertex.java a JarGraph.java.

ClassVertex.java má za úkol vytvořit z jedné java část grafu která bude napojena do grafu.

JarGraph.java analyzuje Jar soubor a postupně na něj vola ClassMetadata.java nebo ClassAST.java podle nastaven výsledkem jsou části grafu které se poté spojí do výsledné podoby.

4.5 QueryLanguage

-== Obrázek: Parser - Analyzátor - interpret QLang DKA ==- Dotazovací jazyk bude mít základ cestu kde bude hledat následovanou podmínkami, které budou na sebe navazovat.

Kapitola 5

Implementace Aplikace

Co jsme udelaly v predchozích kapitolách Co budu dělat v této kapitole

5.1 Konfigurace aplikace

SettingsQuery

5.2 Použití nastroje Procyon

Decompilace

5.2.1 Get AST

získaní AST

5.2.2 Get metadata

získaní Metadat

5.3 Použití TinkerPop

Uloženi do databáze

5.3.1 Blueprints

tvorba grafu

5.3.2 Gremlin

Dotazy do grafu

5.3.3 Pipes

Skladaní dotazu do rour

5.4 Systémové požadavky

JAVA VM atd. (Linux,Widle)

Kapitola 6

Testování

Testování přes platformy

6.1 Typy testování

Testování funkčnosti Jsou na předem připravené vstupy odpovídající výstupy?

Testování použitelnosti Funguje aplikace správně je dala by se napsat lépe, rychleji, přehledněji atd.

Testování neočekávaných vstupu Otestování erro stavu atd.

6.2 Výsledky testování

Výsledky testu padala aplikace je rychlá přehledná jednoduchá intuitivní?

Kapitola 7

Závěr

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků se zvlášť vyznačeným vlastním přínosem studenta. Povinně se zde objeví i zhodnocení z pohledu dalšího vývoje projektu, student uvede náměty vycházející ze zkušeností s řešeným projektem a uvede rovněž návaznosti na právě dokončené projekty.

Literatura