VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

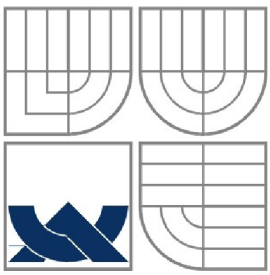
VYUŽITÍ DATABÁZE H2 V JBOSS APLIKAČNÍM SERVERU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

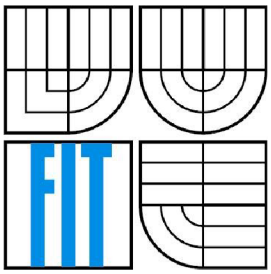
AUTOR PRÁCE
AUTHOR

MICHAL PENČIKOV

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VYUŽITÍ DATABÁZE H2 V JBOSS APLIKAČNÍM SERVERU

USE H2 DATABASE IN JBOSS APPLICATION SERVER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL PENČIKOV

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK LETKO

BRNO 2010

Abstrakt

JBoss aplikační server (AS) je open source implementace standardu Java EE. Jeho aktuálně používaná databáze pro potřeby testování, HSQL, má některá podstatná omezení, proto existují požadavky na využití jiných databází, použitelných při testování JBoss AS. H2 je relativně nová relační databáze, napsaná čistě v Javě. Tato práce popisuje, jak lze JBoss AS nakonfigurovat, aby místo HSQL databáze používal H2 databázi. Dále uvádí výsledky testů, provedených po této rekonfiguraci a identifikuje problémy, které bude nutno vyřešit, aby JBoss AS mohl používat databázi H2.

Abstract

The JBoss application server (AS) is an open source implementation of the Java EE standard. Currently, the JBoss AS uses the HSQL database system for testing purposes. This database system has certain limitations. Therefore, there is a need to use another database system for testing of JBoss AS. H2 is a relatively new relational database entirely written in Java. This thesis describes configuring JBoss AS so it is able to use the H2 database instead of HSQL database. The thesis then provides JBoss AS test results obtained by running tests after reconfiguration and identifies issues that need to be solved before deployment of H2 database.

Klíčová slova

Java EE, Java, JBoss, databáze, H2, aplikační server, Hypersonic, HSQL

Keywords

Java EE, Java, JBoss, database, H2, application server, Hypersonic, HSQL

Citace

Penčikov, M.: *Využití databáze H2 v JBoss aplikačním serveru*, bakalářská práce, Brno, FIT VUT v Brně, 2010

Využití databáze H2 v JBoss aplikačním serveru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zdeňka Letko.
Další informace mi poskytl pan Jiří Pechanec.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Penčikov
19. května 2010

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Zdeňku Letkovi za trpělivost, vstřícnost a ochotu a Ing. Jiřímu Pechancovi taktéž za vstřícnost, ochotu a kvalitní odborné vedení.

Obsah

1 Úvod

Dnešní doba, co se týče vývoje software, pomalu směřuje k trendu open source, neboli otevřenému kódu. Vývojáři mohou s radostí nahlédnout do vnitřností svého oblíbeného softwaru a něco se přiučit, či využít nově nabyté znalosti například k vytvoření zásuvného modulu. Nicméně, i obyčejní lidé dnes využívají open source produktů, převážně jako nástroje pro vlastní potřebu (zdrojový kód je nezajímá). Hlavní výhodou je pro ně cena – za většinu open source nástrojů nemusejí platit.

Nejen veřejnost však dnes používá open source řešení. Mnoho firem používá tyto produkty hlavně kvůli snížení výdajů za licence na drahý komerční software, a to je vcelku relevantní ukazatel toho, že se kvalita otevřeného softwaru dosti zvýšila. Úspěšné produkty kolem sebe shlukují rozsáhlé komunity, skládající se z odborníků a vývojářů i obyčejných uživatelů, a mnohé mají i zázemí v podobě společnosti, která projekt zaštiťuje. Některé společnosti si tímto způsobem, na otevřených řešeních, vybudovaly zdroj svých příjmů – díky pro tuto oblast vhodně zvolenému obchodnímu modelu.

Tato bakalářská práce využívá dvou takových open source řešení. Prvním je JBoss aplikační server (AS), jenž zaštiťuje firma Red Hat Inc. JBoss AS kolem sebe shromáždil komunitu JBoss.org, která se na jeho vývoji aktivně podílí. Red Hat pak poskytuje placenou podporu této technologie. Druhé řešení se nazývá H2 a je to relativně nový relační databázový systém. V současné době používá JBoss AS pro testovací účely databázi s názvem Hypersonic SQL (HSQL). Ta má však některá podstatná omezení. Existuje proto požadavek na použití jiné databáze za účelem dodatečného testování a z toho plynoucí možnosti nalézt některé potenciální chyby, které by při testování s HSQL pravděpodobně prošly s úspěchem. Účelem této práce je tedy připojit databázi H2 do aplikačního serveru, a pak v jeho rámci otestovat její využití, případné problémy pak zdokumentovat.

Práce se ze začátku zabývá popisem použitých technologií a seznamuje čtenáře se znalostmi obecnými i konkrétními, potřebnými k provedení projektu. Dále provádí analýzu řešení, jejich implementaci a vyhodnocuje výsledky testování zakomponované databáze. V závěru se zmíní o jejím významu a budoucnosti projektu s případnými rozšířeními.

2 Použité technologie

V následujících kapitolách jsou popsány technologie, důležité pro dostatečné pochopení a provedení tohoto projektu. Ze začátku se se zaměříme na technologii Java, kde si vysvětlíme mimo jiné rozdíl mezi Javou, využívající virtuální stroj, a programovacími jazyky jako je třeba C. Potom plynule přejdeme ke konkrétnější technologii ze stejné rodiny, s názvem Java Enterprise Edition. Popíšeme si funkce tohoto standardu, konkrétní technologie v jeho rámci definované a způsob použití databází. Navazující technologií je aplikační server JBoss, jenž je implementací standardu Java EE, a tudíž poskytuje všechny služby a funkce popsané v předchozí kapitole. V těchto kapitolách se dozvíme jeho strukturu a konkrétní specifika implementace. Navíc si popíšeme silný nástroj tohoto produktu s názvem testsuite. Nakonec se seznámíme s nástrojem, jenž poskytuje funkce pro automatickou správu projektů, Ant.

2.1 Java

Pojem Java může označovat kromě stejnojmenného programovacího jazyka též softwarovou platformu. *Platforma Java* (Java platform)[1] je množina softwarových produktů a průmyslových specifikací, které společně poskytují systém pro vývoj softwaru a jeho nasazení na různých počítačových platformách.

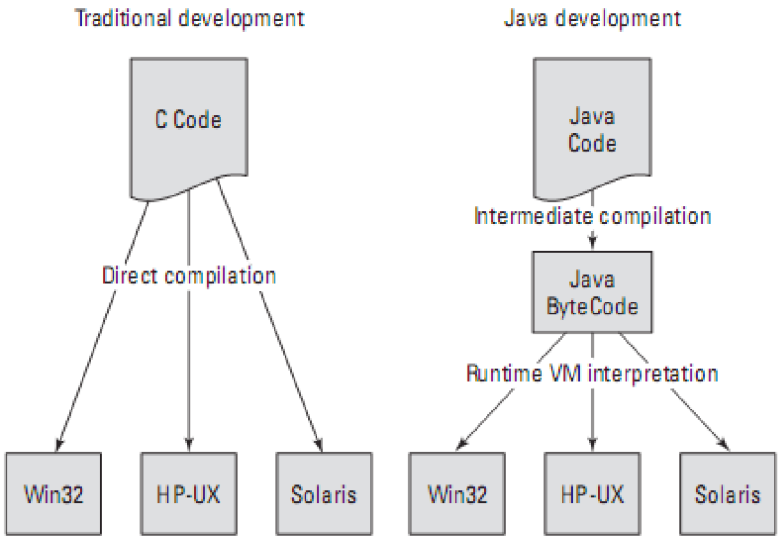
Vývojové prostředí Javy, tj. všechny potřebné programy pro plnohodnotnou práci s Javou pod správou firmy Sun, se nazývá *Java Development Kit* (JDK)[1][1]. Součásti JDK jsou mimo jiné běhové prostředí pro programy v Javě *Java Runtime Environment* (JRE)[1], překladač, debugger, nástroj javadoc[2], nástroj pro vytváření archivů jar, disassembler souborů class a další. JDK se vyvíjí společně s jazykem a platformou Java a s každou novou verzí je vydána další verze JDK[3].

JRE je běhové prostředí pro programy v Javě. Jeho hlavní součástí je *Java Virtual Machine* (JVM)[4], ta je založena na koncepci tzv. virtuálního stroje (virtual machine, VM). Virtuálním strojem se v tomto případě myslí počítačový software, který vytváří virtualizované prostředí mezi platformou počítače (hardwarem či operačním systémem počítače) a programem na něm spuštěným. Aplikaci psanou pro virtuální stroj je pak možné provozovat bez nutnosti modifikace platformně závislých částí kódu na platformě, pro kterou existuje verze virtuálního stroje. Překlad programu v jazyce Java tedy neprobíhá přímo do spustitelného souboru, tj. do strojového jazyka počítače, ale do mezikódu nazývaného bajtkód (*bytecode*). JVM pracuje s tímto mezikódem a provádí jej interpretací. Na Ilustraci 1 je názorně vidět funkce bajtkódu, který je prostředkem pro platformní nezávislost.

1 JDK je zdarma dostupné na http://java.sun.com/.

2 Nástroj pro automatické generování dokumentace ze zdrojového kódu, více na http://java.sun.com/j2se/-javadoc/.

3 Aktuálně (květen 2010) je nejnovější verzí JDK 6u18 (http://java.sun.com/javase/downloads/).



Ilustrace 1: Srovnání Javy a kompilovaných jazyků (zdroj: [3])

Obecným problémem interpretace je její pomalost oproti kompilovaným jazykům. Tento problém se v Javě v současné době řeší použitím kombinace tzv. *just-in-time* (JIT) kompilátorů a algoritmu *hot-spot*. JIT kompilátor, po analýze a výběru časově kritických míst algoritmem hot-spot, v reálném čase (za běhu) po optimalizaci překládá tyto části kódu do strojového jazyka konkrétní platformy, na které je JVM spuštěna. Program pak běží rychleji, mnohdy srovnatelně s rychlostmi kompilovaných programů.

Součástí JRE je také *Java Core API* (Application Programming Interface, Aplikační programové rozhraní)[2]. Je to množina veřejných knihovních tříd, napsaných v jazyce Java, které jsou součástí každé instalace tohoto produktu. Tyto třídy jsou k dispozici pro použití každým programem, psaným v Javě, jedná se tedy o standardní knihovnu tříd, která se musí vyskytovat v každém prostředí, kde se Java používá. Knihovny tříd jsou v javě uspořádávány do tzv. balíků (*packages*)[2]. Balík je obvykle reprezentován adresářem, který obsahuje přeložené zdrojové soubory. Vnořováním adresářů vzniká hierarchie balíků. Jména adresářů pak odpovídají jménům balíků. Systém balíků slouží k vytváření nových prostorů jmen (*name-spaces*) a umožňuje přehledné vytváření knihoven tříd a rozhraní.

Java byla firmou Sun Microsystems poprvé představena veřejnosti v roce 1995. Od té doby se specifikace platformy i samotného jazyka mnohokrát pozměnily. Java jako platforma se dělí na tři hlavní větve: Java Micro Edition, Java Standard Edition a Java Enterprise Edition.

Java Standard Edition (*Java SE*, dříve *J2SE*) označuje základní platformu pro vývoj desktopových aplikací. Zahrnuje kolekci API, která byla vyvíjena od první verze Javy a postupně rozšiřována. Kromě JRE obsahuje sadu balíků (knihoven), z nichž ty základní poskytují podporu pro práci se vstupy a výstupy, komunikaci s operačním systémem, pokročilé matematické funkce, práci se síti, aj. Ty speciální pak umožňují vytvářet applety (kap. 2.2), JavaBeans[1], grafické rozhraní pro

aplikace, ale také podporují protokoly pro komunikaci se vzdálenými objekty a aplikacemi, přístup k SQL databázím, práci s XML[4], poskytují podporu zabezpečení a další.

Java Micro Edition (*Java ME*, dříve *J2ME*) je větví platformy Java, uzpůsobená pro vývoj softwaru pro mobilní zařízení a vestavěné systémy. Kvůli omezeným paměťovým a výpočetním zdrojům těchto koncových zařízení se omezuje standardní sada knihoven z Java SE. Java ME poskytuje několik specifikací, popisujících sady knihoven a prvků VM, které se dále dělí na profily. Tyto profily umožňují a pomáhají vývojářům vybrat si vhodný programový model pro konkrétní zařízení. V další kapitole je popsána platforma Java Enterprise Edition.

2.2 Java Enterprise Edition

Java Enterprise Edition (*Java EE*, *dříve J2EE*)[3] je průmyslový standard a soubor specifikací, mající za cíl umožnit vývoj a poskytnout běhové prostředí pro vícevrstvé podnikové aplikace (multi-tier enterprise applications). Specifikace definují logické rozdělení na aplikační komponenty a role zastoupené napříč celým životním cyklem aplikace (správu aplikace po nasazení do ostrého provozu nevyjímaje). Základ pro Java EE tvoří platforma Java SE. Java EE navíc poskytuje služby jako jsou nástroje pro persistenci dat, řízení transakcí, zabezpečení aj.

Při vývoji podnikových aplikací pod Java EE se často používá model distribuované vícevrstvé aplikace. Aplikační logika je rozdělena na *komponenty* (components) logicky podle funkcí, které mají být poskytovány. Java komponenta je složena ze tříd a rozhraní, sdružených do jednoho celku, poskytující konkrétní funkcionalitu. Komponenta je typicky navržena jako nezávislý celek k použití do různých aplikací. Java EE definuje čtyři typy komponent: klientská aplikace, Java applet, webová komponenta a serverová komponenta. Všechny však mají jednu společnou vlastnost – běží v prostředí nazývaném *kontejner* (container). Kontejner musí poskytnout běhové prostředí pro komponenty, mechanizmus identifikace a schopnost zpracovat datový formát souboru, použitý pro nasazení (*deployment*[5]) komponent do provozu a standardní služby, jichž komponenty využívají.

Komponentově orientovaná abstrakce, použitá v Java EE, nabízí zásadní přednosti před dodnes používaným a populárním objektově orientovaným modelem. U objektového schématu je programátor nucen starat se jak o řídicí logiku aplikace, tak o v zásadě komplexní způsob používání kolekce podpůrných služeb (transakční zpracování, perzistence objektů do databáze, paralelní zpracování, a jiné). Komponenty naopak obsahují jen aplikační logiku; podpůrné služby jsou k aplikaci připojeny prostřednictvím příslušného kontejneru v okamžiku nasazení do provozu. Komponenty tedy s okolím komunikují prostřednictvím veřejného rozhraní služeb, poskytovaného příslušným kontejnerem.

Jak již bylo zmíněno výše, existují tyto čtyři typy komponent.

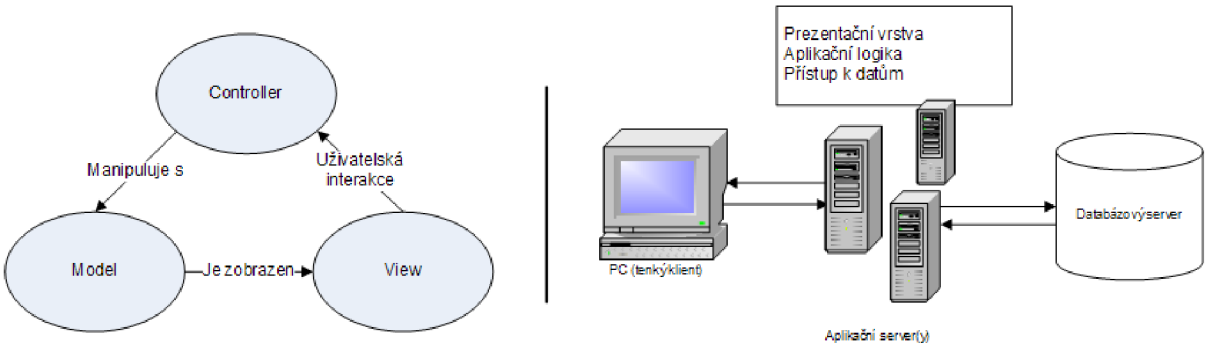
- *Klientské aplikace (klienti)* jsou obvykle samostatné aplikace napsané v jazyce Java. Jsou spouštěny v JVM a schopné využívat standardních služeb Java EE k přístupu k dalším komponentám v rámci rozdílné vrstvy. Jednotlivé služby jsou klientovi k dispozici většinou skrze lokální instalaci Java SE, či jsou dodané přímo v distribuci samotné aplikace.

4 XML specifikace aj. na http://www.w3.org/XML.

5 Pro v tomto kontextu těžce přeložitelné slovo „deploy, deployment“ budu používat krkolomně pojem „nasadit, nasazení“ (např. aplikace do provozu). Pro pojmy „undeploy, undeployment“ budu používat překlad „stáhnout, stažení“ (z provozu).

- *Applety* jsou podobné klientské aplikaci, avšak jsou spouštěny v prostředí webového prohlížeče (či jiné aplikace, podporující applety). Mnoho webových prohlížečů má v sobě vestavěné JVM. Pomocí zásuvného modulu (Java plugin) lze vynutit použití konkrétní verze JVM.
- *Webové komponenty* jsou komponenty na straně serveru, obecně užívané k předvedení prezentační vrstvy uživateli. Existují dva typy webových komponent: Java Server Pages (JSPs) a Java servlety. JSPs jsou velmi zjednodušeně podobné HTML stránkám, avšak obsahují i kód v Javě. Servlety jsou třídy v jazyce Java, využívající Java I/O API k produkci HTML stránek. Oba dva typy lze použít k produkci i jiných typů formátu.
- *Serverové komponenty* jsou k mání ve formě *Enterprise JavaBeans* (EJBs). Jsou spravovány v EJB kontejneru, a používány hlavně při implementaci aplikační logiky.

Role, jež Java EE definuje, jsou ty, zastoupené během vývoje a provozu aplikace. Přestože jsou rozčleněné, v praxi je tendence zastoupení vícero rolí jedním útvarem (organizací, firmou, …). Standard rozlišuje mezi poskytovatelem implementace (části) specifikace Java EE, dodavatelem hotových komponent a dodavatelem pomocných nástrojů (pro monitoring, zjednodušení vývoje aj.). Dále definuje roli sestavovatele aplikace (assembler), odpovědného za sestavování více komponent do jednoho celku pro distribuci, roli odpovědnosti za konfiguraci aplikace pro běh na konkrétní platformě (deployer) a administrátora, jenž využívá nástrojů například k monitoringu běhu a zlepšení výkonu aplikace.



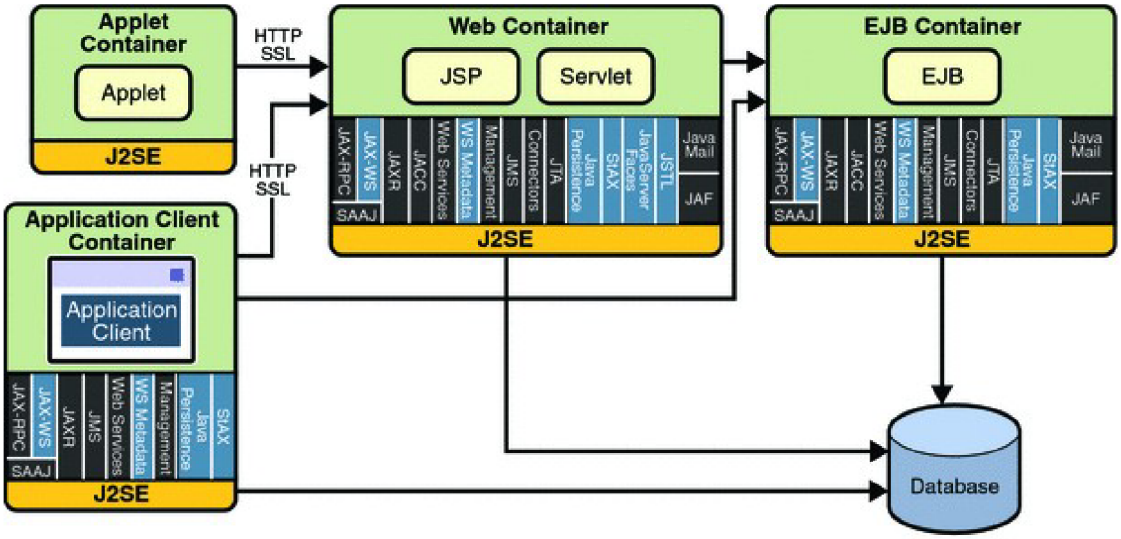
Ilustrace 2: Schéma vzoru *model-view-controller* (MVC) a 3-vrstvé architektury

Java EE navíc doporučuje použití návrhového vzoru *model-view-controller* (MVC)[3]. MVC poskytuje schéma pro oddělení prezentační logiky (*view*), aplikační logiky (*control*) a dat (*model*). Jiný model, použitý pro implementaci serverových komponent (EJB), se nazývá třívrstvá architektura. První vrstva, klientská, se skládá z klientských aplikací jako je webový prohlížeč či nezávislý klient. Druhá vrstva (aplikační) je hostitelem pro distribuované aplikace a typicky je nazývána *aplikační server*. V něm jsou obsaženy kontejnery. Třetí vrstva (Enterprise Information System, EIS) obsahuje systém pro správu zdrojů, databáze, systém pro podporu transakčního zpracování aj. Vzhledem k výše zmíněným informacím lze tedy mluvit i o n-vrstvé architektuře, kde n je číslo, jež může nabývat hodnoty i větší než 3. Softwarový architekt tedy může rozdělit informační systém do libovolného počtu vrstev, vycházejícího z potřeb aplikace a možností výpočetních prostředků, na nichž bude aplikace provozována. Všechny vrstvy jsou pak schopny běžet na rozdílných strojích, i když to tak zpravidla nebývá a více vrstev bývá provozováno v rámci jedné platformy, např. v rámci aplikačního serveru. Rozdělení aplikace na vhodný počet vrstev pomáhá

vypořádat se s řadou případů, při jejichž řešení obvykle vznikají problémy. Příklady takových případů je udržovatelnost aplikace, její konzistence, flexibilita, bezpečnost, schopnost propojení s jinými systémy, vypořádání se se zvýšenou zátěží na systém aj.

Java EE je otevřená specifikace, a proto ji může implementovat v podstatě každý. Aplikačních serverů je dnes hojně, a pokud má konkrétní implementace nosit nálepku „J2EE compliant“ (kompatibilní s J2EE), je nutné, aby prošla sadou testů s příhodným názvem J2EE Compatibility Test Suite. Rozhodnutí pro použití technologie J2EE má potenciál vyvarovat vytvoření závislosti na jednom konkrétním poskytovateli (tzv. *vendor lock-in*) aplikačního serveru. V případě nespokojenosti se současnou implementací není problém přecházet mezi „J2EE compliant“ řešeními a provozovat tu, která vyhovuje konkrétním potřebám.

2.2.1 Přehled služeb a technologií v Java EE



Ilustrace 3: API platformy Java EE (zdroj: [7])

V následujícím textu budou shrnuté základní technologie a API, popsané ve specifikacích Java EE. Na obrázku (Ilustrace 3) lze pozorovat dostupnost jednotlivých API v typech Java EE kontejnerů. Informace uvedené v této kapitole jsem čerpal ze zdrojů [3,6,7].

Komponenta Enterprise JavaBeans (EJB), či *enterprise bean*, je prvek, jenž samostatně či společně s dalšími enterprise beans vytvářejí obchodní logiku aplikace na straně serveru. Existují dva typy enterprise beans: session bean a message-driven bean. *Session bean* představuje dočasnou konverzaci s klientem. Jakmile klient ukončí komunikaci, je session bean ukončena také a data uvnitř obsažená jsou ztracena. *Message-driven bean* představuje fúzi prvků session bean a schopnosti naslouchat zprávám (*message listener*). Umožňuje tak komponentě asynchronně přijímat zprávy.

Technologie *Java Servlet* rozšiřuje možnosti serveru, který komunikuje s aplikačním klientem na principu dotaz-odpověď. I když je model Java Servlet obecně použitelný, valná většina servletů spadá do kategorie obsluhy Hypertext Transfer Protocol (HTTP) požadavků v rámci webového serveru. Požadavek na zpracování servletem přijme server ve formě HTTP požadavku. Web server reaguje spuštěním servletu s případnými parametry a výsledek vrátí (většinou ve formě Hypertext

Markup Language (HTML) stránky) zpět klientu. *JavaServer Pages* (JSPs) se stejně jako Servlety týkají dynamických webových stránek. Umožňují vkládat zlomky kódu přímo do textových souborů. JSP stránka je textový dokument, který obsahuje 2 typy textu: statická data (například HTML stránka) a JSP elementy, které dynamicky generují další části dokumentu. *JavaServer Faces* (JSF) je další technologie, pomocí níž lze zdynamizovat obsah webu. Tento framework obsahuje sadu grafických komponent, ve kterých je obsažena podpora pro validaci vstupu, zachycení událostí aj. Pro převedení komponent do HTML či jiných značkovacích jazyků se používá tzv. *custom renderer*, jehož záměnou za jiný lze docílit generaci do odlišného výsledného prezentačního rozhraní. JSF je obvykle doplněn o custom renderer pro převod komponent uživatelského rozhraní do HTML.

Java Message Service (JMS) je standard, který poskytuje aplikacím možnost vytvářet, zasílat, přijímat a číst zprávy. Umožňuje tak distribuovanou asynchronní komunikaci mezi komponentami, která je navíc spolehlivá. Transakční zpracování dat při přístupu k databázi se děje pomocí *Java Transaction API* (JTA). Pro práci s elektronickou poštou pak slouží *JavaMail*.

Důležitou součástí Java EE specifikace je podpora webových služeb (*web services*). Organizace pro vývoj webových standardů W3C připravila standardizovaný způsob popisu webových služeb[6], Web Service Description Language (WSDL), jenž v jazyce Extensible Markup Language (XML) popisuje nabízenou službu a způsob přístupu k této službě. *Java API for XML Web Services* (JAX-WS) definuje klientské API pro přístup k těmto webovým službám, jakožto i techniku nasazení webových služeb. *Java Architecture for XML Binding* (JAXB) poskytuje vhodnou metodu, jak provázat XML schéma s reprezentací programu v jazyce Java. *Java API for XML Registries* (JAXR) umožňuje přistupovat ke komerčním či univerzálním registrům metadat přes internet.

Mnohdy je potřeba připojit k budovanému informačního systému jiný systém či třeba databázi. K tomuto účelu, tzv. integraci podnikových aplikací (enterprise application integration, EAI), slouží *J2EE Connector Architecture*. K perzistenci dat pomocí mapování objektů do relační databáze (object-relational mapping, ORM) slouží *Java Persistence API* (JPA).

Java Naming and Directory Interface (JNDI) je významná služba v Java EE. Tato adresářová služba umožňuje provázat data, objekty, soubory či služby s konkrétním jménem, a pomocí něj je pak zpětně vyhledat či zjistit, že byla položka adresáře změněna. Autorizaci (ověření, zda je uživatel opravdu ten, kdo tvrdí, že je) a autentizaci (ověření, zda má autorizovaný uživatel oprávnění provést akci, o jejíž provedení usiluje) zajišťuje služba *Java Authentication and Authorization Service* (JAAS).

Jedna z důležitých technologií, vyskytující se nejen v Java EE, ale i v Java SE, je *Java Management Extensions* (JMX). JMX poskytuje standardizovaný a ve své podstatě jednoduše použitelný systém pro monitoring a správu prostředků jako jsou aplikace, zařízení či služby. Pomocí JMX technologie lze prostředky spravovat dynamicky, v době jejich vytvoření, instalace a provozu. JMX poskytuje možnost spravovat i JVM. Prstředky řídíme pomocí Java objektů zvaných Managed Beans, zkráceně MBeans. Tyto MBeans se zaregistrují do tzv. MBean serveru, který společně se sadou služeb pro správu MBeans tvoří prostředek pro přímé ovládání zdrojů a umožňují správu i klientům vzdálených aplikací.

6 Viz http://www.w3.org/TR/ws-arch/.

2.2.2 Java EE a databáze

Vzhledem k námětu této práce je důležité se důkladněji zaměřit na téma využití databází v technologii Java EE. Lze říci, že většina netriviálních aplikací používá nějaký typ databáze (ve smyslu datového úložiště) kvůli potřebě perzistence dat. Ve velké většině případů je toto datové úložiště typu relační databáze. Java EE obsahuje technologii *Java Database Connectivity* (JDBC), což je kolekce API, která umožňuje práci s databázemi – nahrávat databázové ovladače, vytvářet spojení a dotazy, vracející výsledná data z databáze. Typ databáze se neomezuje na relační; JDBC abstrahuje obecné funkce pro práci s databází na sadu tříd, jejichž implementaci nazýváme databázový ovladač. Existuje-li ovladač pro konkrétní typ databáze, lze JDBC pro práci s ní použít.

První, co musí aplikace udělat, když pracuje s databází, je vytvořit spojení. Vzhledem k tomu, že JDBC je abstrakcí, nemusí vývojář vytvářet sokety či používat jiné třídy pro přímou práci se sítí. Aplikační server (poskytující služby a API pro uživatelské komponenty, tzv. middleware) poskytne toto spojení prostřednictvím třídy `DataSource` (datový zdroj), jež je součástí implementace databázového ovladače. Datové zdroje jsou preferovaným prostředkem pro získání spojení k datovému úložišti – při jeho použití lze zaměnit použitá databáze aniž by bylo třeba úpravy kódu klientské aplikace.

`DataSource` objekty podporují tzv. *connection pooling*, neboli uchování spojení ve společném fondu, ze kterého tato spojení poskytují a po použití klientem následně vrací, čímž mimo jiné odpadá nutnost připojování k databázi a vytváření nového spojení pokaždé, kdy je vyžadováno. Administrátor aplikačního serveru vytvoří instanci `DataSource` a přiřadí ji jméno pomocí služby JNDI. Poté může více aplikací přistupovat k této databázi pomocí jména datového zdroje.

Připojení k databázi, představované objektem typu `Connection`, získané pomocí datového zdroje, umožňuje uživateli využít transakční zpracování nad databází a transakci řídit. Změny nad databází nebudou provedeny, dokud se nezavolá metoda `commit`. Naopak, pokud je vyvoláno `rollback`, změny nad databází jsou zahozeny a všechny zámky na databázi, držené tímto připojením, jsou uvolněny.[3,6]

2.3 JBoss aplikační server

JBoss aplikační server (*JBoss AS*) je open source implementací standardu Java EE. Počátky tohoto projektu sahají do roku 1999. Projekt vznikl jako implementace Enterprise JavaBeans (EJB) části specifikace Java EE. Roku 2003 se vydání pod označením JBoss AS 3 stalo již kompletním a populárním J2EE aplikačním serverem. Vývoj JBoss AS přešel v roce 2006 pod křídla společnosti Red Hat, Inc., přičemž jeho popularita stále roste, nejen díky dostupnosti zdrojových kódů a svobodné licenční politice. Mnoho jeho součástí je vyvíjeno v podobě samostatných projektů a je také schopno běžet samostatně, bez nutnosti integrace do aplikačního serveru, či je možné je integrovat do aplikací a aplikačních serverů, různých od JBoss AS.[5]

JBoss AS je možné získat zcela zdarma, a to stažením z internetových stránek komunity JBoss[7]. Binární verze je po rozbalení z archivu možné rovnou používat, zdrojové kódy celého aplikačního serveru si pak uživatel může sám upravit a jednoduše aplikaci sestavit. Zdrojové kódy i

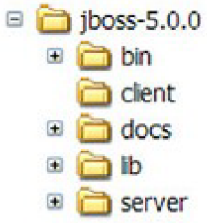
7 Ke stažení na http://labs.jboss.com/jbossas/downloads.

jiných projektů lze dalším způsobem získat z SVN[8] repositáře projektu JBoss (pro veřejný přístup je zřízen anonymní SVN repositář). V současnosti (květen, 2010) je ve vývoji verze JBoss AS 6, následující specifikaci Java EE 6. Vzhledem k tomu, že posledním finálním (GA, general availability) a „Java EE 5 compliant" vydáním je verze 5.1.0, rozhodneme se řešení implementovat právě pro toto vydání. Pokud tedy budeme mluvit o JBoss AS, budeme mít na mysli JBoss AS 5.1.0 GA[9]. Pro sestavení projektu lze použít nástroj Ant (viz kap. 2.4).

Pokud mluvíme o struktuře JBoss AS, mluvíme v podstatě o jeho adresářové struktuře, protože v prakticky celá jeho konfigurace se provádí na úrovni souborového systému (kopírování či mazání balíků aplikace na určených místech, kupříkladu). Proto se tímto tématem zaobírá další podkapitola.

2.3.1 Adresářová struktura serveru

Po sestavení projektu je JBoss AS téměř připraven ke spuštění. Všechny soubory potřebné ke stratu serveru jsou umístěny v jednom adresáři, vytvořeném sestavovacím skriptem. Adresář lze překopírovat do libovolného umístění ve struktuře souborového systému. Na toto umístění je třeba nastavit proměnnou prostředí `JBOSS_HOME` a poté již můžeme server spustit.



Ilustrace 4: adresářová struktura JBoss AS

V adresáři `bin` jsou umístěny všechny binární soubory a skripty, potřebné ke startu serveru, i další skripty, používané pro různé jiné účely. Adresář `client` obsahuje knihovny, které umožňují uskutečnit vzdálené volání serverových služeb. Tito tzv. vzdálení klienti, běžící vně JVM aplikačního serveru, obvykle volají EJB, webové služby či využívají JMS fronty zpráv, jedoucí v rámci aplikačního serveru. Adresář `docs` obsahuje soubory DTD a XML schémata, vzorové příklady konfiguračních souborů, licence k různým knihovnám, zahrnutým do JBoss AS aj. V adresáři lib jsou umístěny knihovny jádra aplikačního serveru. Adresář `server` je stěžejní část serverové struktury. Zde jsou umístěny adresáře, které nazýváme *serverové konfigurace*. Při spuštění JBoss AS se standardně spouští konfigurace v adresáři `default`. Výběr spouštěné konfigurace lze ovlivnit přepínačem `-c`. Konfigurace obsahuje sadu služeb a aplikací, jež jsou při startu spuštěny.

V adresáři konfigurace nalezneme několik podadresářů, z nichž ty nejvýznamnější jsou `lib` a `deploy`. Adresář `lib` obsahuje knihovny sdílené všemi službami a aplikacemi v rámci konfigurace. Zde se ukládají knihovny, které bude využívat více než jedna aplikace. Do adresáře `deploy` se nasazují služby a aplikace. JBoss AS umožňuje tzv. *hot deployment*, neboli nasazení aplikace za běhu serveru. Tento typ nasazení aplikace se provede jednoduchým nakopírováním balíku aplikace do

8 Více o SVN na http://subversion.apache.org.

9 Repositář pro přístup pomocí *svn* je http://anonsvn.jboss.org/repos/jbossas/tags/JBoss_5_1_0_GA.

adresáře `deploy` a vyčkáním, až aplikační server službu či aplikaci detekuje. Stažení aplikace spočívá v odstranění balíku aplikace z adresáře `deploy`.

2.3.2 JBoss a databáze

Datové zdroje (viz kap. 2.2.2) lze v JBoss AS definovat nejen přímo v kódu pomocí JDBC API. Častější způsob je vytvoření, a následné vydání, souboru *-ds.xml, tzv. *deskriptoru* (data source descriptor). Můžeme pro název souboru použít jakékoliv jméno, za předpokladu, že bude končit příponou *-ds.xml*. Datový zdroj může být nakonfigurován jako (1) zdroj, který se neúčastní JTA transakcí, pak jako (2) zdroj, jenž podporuje distribuované transakce na lokální úrovni a poté i (3) jako zdroj pro podporu transakcí, s nimiž lze využít distribuovanou transakci mezi více běžícími aplikačními servery. JBoss AS je distribuován s open source databází *Hypersonic SQL* (HSQL), používanou jako implicitní databázi. V ostrém provozu se však doporučuje využít jiného databázového systému[10].

```
<datasources>

   <local-tx-datasource>
      <jndi-name>DefaultDS</jndi-name>
      <connection-url>jdbc:hsqldb:${jboss.server.data.dir}/hypersonic/localDB
      </connection-url>
      <driver-class>org.hsqldb.jdbcDriver</driver-class>

      <user-name>sa</user-name>
      <password></password>

      <depends>jboss:service=Hypersonic,database=localDB</depends>
   </local-tx-datasource>

   <mbean code="org.jboss.jdbc.HypersonicDatabase"
     name="jboss:service=Hypersonic,database=localDB">
     <attribute name="Database">localDB</attribute>
     <attribute name="InProcessMode">true</attribute>
   </mbean>

</datasources>
```

Text 1: Konfigurace datového zdroje pro HSQL (hsqldb-ds.xml).

2.3.3 Konfigurace datového zdroje

V Text 1 můžeme vidět zredukovaný obsah souboru `server/default/deploy/hsqldb-ds.xml`, který obsahuje konfiguraci datového zdroje pro databázi HSQL. Na tomto příkladu si vysvětlíme význam základních elementů, které se při konfiguraci datového zdroje mohou vyskytnout.

Kořenový element `<datasources>` definuje typ deskriptoru, datový zdroj. Uvnitř se v tomto případě nacházejí dva vnořené elementy, a to `<local-tx-datasource>` a `<mbean>`. První element značí, že se jedná o datový zdroj podporující distribuované transakce v rámci lokálního aplikačního serveru. Druhý element, `<mbean>`, určuje MBean, použitelnou pro konfiguraci služeb, kterých datový zdroj využívá. Níže je uveden význam dalších elementů.

10 Více informací na http://kbase.redhat.com/faq/docs/DOC-15194#cf.

- Mapování tohoto zdroje na JNDI jméno, zde konkrétně „DefaultDS“, zajistí element `<jndi-name>`. Tento datový zdroj je tedy dostupný v globálním prostoru jmen pod názvem „DefaultDS“.
- Pro připojení k databázi je použit obsah elementu `<connection-url>` pomocí JDBC driveru, definovaného v elementu `<driver-class>`.
- Element `<depends>` obsahuje název služby, na které je inicializace datového zdroje závislá. Služba pro připojení k databázi se nespustí dříve, než závislá Mbean, identifikovaná svým jménem (atribut `name`) a plně kvalifikovaným názvem třídy (atribut `code`).

Element `<mbean>` může obsahovat i několik elementů `<attribute>`, představujících hodnoty atributů pro inicializovanou MBean.

2.3.4 JBoss Testsuite

JBoss Testsuite, obvykle nazývaný jen *testsuite*, je sada JUnit[8] testů, distribuovaná s každým vydáním (zdrojových kódů) JBoss AS. Z toho plyne, že pokud chceme spustit testy, musíme mít distribuci se zdrojovým kódem projektu. Tyto unit testy vyžadují spuštěnou instanci JBoss AS s kontejnerem, ve kterém se testy provádějí. Nemusejí však být součástí modulu konkrétních kontejnerů, ve kterých testované služby sídlí. Testy jsou umístěny v adresáři `testsuite` v kořenové složce zdrojových souborů JBoss AS.

Při sestavení a spuštění testsuite jako celku není třeba spouštět instanci aplikačního serveru, lépe řečeno, při spouštění testsuite nesmí běžet žádná instance JBoss AS, protože testsuite si sám spouští instance potřebných konfigurací aplikačního serveru[11]. Tento proces je plně automatizovaný, od přeložení testsuite, přes spouštění serverů a na nich jednotlivých testů, až po vygenerování dokumentu s výsledky celého procesu. Předem je potřeba jen přeložit a sestavit JBoss aplikační server[12].

2.4 Ant

Pod názvem *Ant*[8] se ukrývá open source skriptovací nástroj pro provedení procesu sestavování (*build process*) aplikace, napsaný v Javě. Sestavovací skripty jsou psány v XML, a jsou založeny na tzv. úlohách (*tasks*), jejichž základní sada je součástí instalace Antu a umožňuje vývojáři automatizovat prakticky všechny procesy v rámci vývoje softwaru, kompilací a spuštěním unit testů počínaje a síťovým nasazením aplikace či emailovou notifikací členů vývojového týmu konče.

Tento nástroj má v základu pomoci vývojáři transformovat zdrojový kód, přidružené knihovny a další soubory, které dohromady tvoří vyvíjený projekt, do balíku, způsobilého k uvolnění pro uživatele. Navíc, a to je důležité, umožňuje tento proces provádět pravidelně a opakovaně.

Jak již bylo řečeno, Ant skript je psán v XML. Popisuje, jak sestavit jeden a právě jeden *projekt*. Projekt se skládá z *cílů* a *úloh*. Tyto tři entity daly vzniknout třem typům elementů, z nichž se build skript skládá.

---

11 V případě spouštění jediného vybraného testu je nutné instanci (či více instancí) serveru předem nastartovat.

12 Pro překlad JBoss AS spusťte ve složce `build` příkaz `./build.sh`, pro spuštění testů ve složce `testsuite` příkaz `./build.sh tests`.

Element *project* identifikuje projekt, *target* označuje cíl, jehož chceme dosáhnout. Takový cíl může být kompilace celého či části kódu, spouštění unit testů či připravení aplikace do balíku k uvolnění pro zákazníka aj. *Task* jako typ elementu umožňuje definovat jednu konkrétní činnost, kterou chceme provést, např. kompilaci konkrétního zdrojového kódu pomocí překladače Java, spuštění testu pomocí JUnit, či vytvoření JAR archivu pomocí nástroje *jar*. Úlohy jsou implementované třídami v jazyce Java, což z Antu dělá lehce rozšiřitelnou technologii.

Ant přichází se základní a stěžejní vestavěnou sadou a volitelnou sadou task elementů. Vestavěné elementy task jsou typickou součástí Antu a nevyžadují žádnou speciální konfiguraci. Volitelné elementy task jsou spravované Ant týmem a dodávané s Antem, avšak jsou závislé na přítomnosti externí knihovny. Například, element *junit*, který spouští testy JUnit, vyžaduje zvlášť doplnění o externí knihovnu JUnit. Existují také elementy task třetích stran. Ty jsou pod správou externích dodavatelů a poskytují podporu pro další nástroje v Javě i jiné nástroje. Jako příklad lze uvést podpora Subversion tasks, která je poskytována komunitou Tigris, vývojovým týmem Subversion.

2.4.1 Jednoduchý Ant skript

```
<project name="projectname" default="compile">
      <path id="lib.classpath" location="lib/lib.jar" />
      <target name="init">
            <mkdir dir="build" />
      </target>
      <target name="compile" depends="init">
            <javac srcdir="src" destdir="build"
                  classpathref="lib.classpath" />
      </target>
      <target name="clean">
            <delete dir="build" />
      </target>
</project>
```

Text 2: Příklad souboru build.xml, skriptu pro nástroj Ant

Nyní přistoupíme k vysvětlení jednoduchého příkladu skriptu pro Ant (viz Text 2). Tento soubor se obvykle jmenuje `build.xml` a pokud v terminálu napíšeme `ant` a potvrdíme (a existuje-li v aktuálním adresáři soubor `build.xml`), začne se skript provádět.

Atribut `name` kořenového elementu `<project>` definuje jméno projektu, volitelný atribut `default` pak určí, který z cílů se bude implicitně provádět, pokud uživatel nezadá jinak.

Elementy `<target>` oddělují jednotlivé cíle. Mezi počáteční a koncový element `<target>` se vkládají konkrétní činnosti, elementy typu task. V příkladu můžeme vidět `<mkdir>`, který vytváří

zadaný adresář, element `<delete>`, který adresář a jeho obsah maže a element `<javac>`, jenž kompiluje zdrojové kódy a ukládá je do cílového adresáře.

Někdy požadujeme, aby se před provedením určitých činností provedly nejdříve jiné činnosti, protože to tak logika věci vyžaduje. Například, před kompilací a nakopírováním výsledných class souborů do cílového adresáře požadujeme, aby byl cílový adresář vytvořen. Toho lze docílit vytvořením přímé závislosti konkrétního cíle na cíli jiném, a to doplněním o atribut `depends`. V příkladu se cíl „init“ vždy provede před cílem „compile“ právě díky uvedení tohoto atributu.

Poslední dva elementy, které v souvislosti s build skriptem zmíním, jsou elementy `<path>` a `<property>`. Ant vyniká prací se strukturami pro popis cesty. Jednou z možností, jak popsat cestu k souboru, je použití elementu `<path>`. Atribut `location` udává umístění cílového souboru či složky. Jak lze vidět u elementu `<javac>`, odkaz na `id` cesty je proveden pomocí atributu `classpathref`. Element `<property>` v ukázkovém skriptu není, avšak je silným prostředkem pro vyvarování se duplicity dat a zároveň zlepšení čitelnosti kódu.

Ant je, mluvíme-li v kontextu vývoje aplikací v jazyce Java, hojně využívaný velmi širokou veřejností vývojářů, podporovaný prakticky každým soudobým IDE a lze jej provozovat na všech platformách s podporou JVM.[8]

2.5 H2 databáze

Databáze *H2* je open source databáze, napsaná čistě v Javě. Vývoj byl zahájen v květnu 2004 a poprvé byla vydána ke konci roku 2005. Její autor, Thomas Mueller, je též původním autorem databáze Hypersonic SQL (HSQL), nativní databáze v JBoss AS. H2 má být následníkem zmíněné HSQL (H2 má značit Hypersonic 2), nicméně nevychází z její implementace, její kód je napsán od začátku znovu.

K H2 databázím se lze připojovat třemi způsoby, v módech embedded, server a mixed. V *embedded* módu otevírá aplikace databázi ve stejné JVM jako je spuštěna pomocí JDBC. Nevýhodou je, že databázi lze otevřít v jeden okamžik jen v jedné JVM. Nicméně, neexistuje limit na počet souběžných databázových spojení. Při použití *server* módu je spuštěn databázový server a aplikace se prostřednictvím serveru souběžně připojují ke stejné databázi. Taktéž neexistuje limit na počet otevřených databází či otevřených spojení. Server mód je pomalejší než embedded, protože přenos dat je prováděn prostřednictvím TCP/IP. V *mixed* módu první aplikace, vytvářející spojení s databází, ji otevírá v embedded módu, nicméně zároveň spouští i server, aby ostatní klienti, jedoucí v jiných JVM či procesech, mohli přistupovat ke stejným datům. Lokální spojení je tak rychlejší, zatímco vzdálená spojení pomalejší. Všechny tři módy podporují jak *perzistentní*, tak „*in-memory*“ databáze (in-memory je typ rychlé databáze bez perzistence). V tabulce 1 jsou předpisy URL pro připojení k databázím v jednotlivých módech, in-memory nevyjímaje.

Mód	Předpis URL
Embedded	`jdbc:h2:[file:][<path>]<databaseName>`
Server	`jdbc:h2:tcp://<server>[:<port>]/[<path>]<databaseName>`
In-memory	`jdbc:h2:mem:<databaseName>`

Tabulka 1: URL pro připojení k databázi

H2 je kompatibilní se standardem ANSI SQL, nicméně se snaží být kompatibilní i s jinými databázemi – lze ji spustit v tzv. *kompatibilním módu*. Kompatibilní mód pro konkrétní databázi má za následek, že H2 emuluje chování databáze, v jejímž kompatibilním módu ji spouštíme, ne však ve všech aspektech. Kompatibilní mód existuje pro databáze DB2, Derby, HSQL, MS SQL Server, Oracle a PostgreSQL.

H2 má mnoho dalších užitečných vlastností a funkcí, mezi něž patří třeba podpora clusteringu, Multiversion Concurrency Control, 2-phase-commit a podpora distribuovaných transakcí, optimalizace dotazů, mnohé bezpečnostní prvky a mnoho dalších. Vzhledem k tomu, že tyto prvky nejsou podstatné pro řešení projektu, odkazuji proto na [1,9].

3 Implementace řešení

Poté, co jsme nabyli znalosti v sekci o použitých technologiích, se můžeme pustit do splnění zadání práce. Je požadováno, abychom se pokusili překonfigurovat JBoss AS tak, aby pod ním bylo možné provozovat databázi H2 a v případě, že se vyskytnou problémy, zdokumentovat je pro další zkoumání. V další kapitole se budeme zabývat analýzou problému a možnými řešeními, jak tento postup provést.
V předchozích kapitolách jsme se seznámili s technologiemi, potřebnými k pochopení a provedení této práce. Téma jednoduše řečeno požaduje, abych s pomocí doposud nabytých znalostí dokázal přijít se způsobem, jakým vyzkoušet, zda databázi H2 lze provozovat pod aplikačním serverem JBoss, případně získat seznam problémů, které se při provozování vyskytly. V příštích kapitolách se budeme zabývat návrhem řešení tohoto problému a jeho praktickou implementací.

3.1 Analýza a návrh

Jakým způsobem provést ověření, do jaké míry lze využít H2 v JBoss AS? Jaká se nabízejí řešení? Půjdeme-li na to selským rozumem, dojdeme k jednoduchému způsobu otestování využití. Toto řešení lze popsat v několika bodech.

- Je nutné vytvořit aplikaci, určenou pro provoz na Java EE aplikačním serveru. Vzhledem k potřebě ověření využití databáze, je důležité, aby tato aplikace databázi používala.
- V aplikaci použít databázi H2.
- Vyzkoušet funkčnost aplikace.

Řešení má v zásadě několik nedostatků. V prvé řadě chybí způsob, jak efektivně zhodnotit výstup provozu aplikace, kromě tvrzení, že aplikace „funguje“ či „nefunguje“. Poté chybí možnost porovnání provozu s jinými databázemi. To lze řešit konfigurací jiné databáze a jejím nahrazením za H2. Nicméně stále zde setrvává problém prokazatelnosti a dostatečným možnostem identifikovat chyby či problémy v případě, že nastanou. JBoss AS má však v rukávu velmi silný prostředek, který nám umožní databázi otestovat a případné nedostatky a chyby zaznamenat a zkoumat. Nazývá se testsuite (viz kap. 2.3.4).

Testsuite je nástroj pro otestování celého aplikačního serveru, přičemž v rámci unit testů je mnohokrát spouštěna databáze v různých módech, s níž se pak dále pracuje. Proto je testsuite ideální volbou „aplikace“, či lépe, modulu aplikačního serveru, který lze využít pro otestování funkčnosti H2 v rámci JBoss AS. Nebudeme tedy nahrazovat H2 jinou databází, ale právě naopak, zaměníme implicitní databázi v JBoss AS, Hypersonic SQL, za H2. Následným spuštěním testů pak získáme výsledky, jež lze porovnat s výsledky testů, spuštěných na „čisté“ instalaci JBoss AS (instalaci beze změn v užité databázi).

3.2 Úprava datového zdroje

Naším cílem je pozměnit konfiguraci datového zdroje tak, aby používal databázi H2 místo HSQL (příklad viz kap. 2.3.3, Text 1). Ve složce `docs/examples/jca` lze nalézt příklady

konfigurací datových zdrojů pro různé databázové systémy, H2 však ještě není příliš rozšířená, proto zde pro ni příklad přiložen není. Musíme si tedy vystačit s dosavadními znalostmi. V podstatě je vhodné změnit minimum tak, aby aplikace, používající datový zdroj, nemusela být změněna a databáze se, pro potřeby testů, chovala stejně. V příkladu Text 3 můžeme vidět potřebné změny.

```
<datasources>
   <local-tx-datasource>

      <jndi-name>DefaultDS</jndi-name>
      <connection-url>jdbc:h2:${jboss.server.data.dir}/h2/localDB
      </connection-url>
      <driver-class>org.h2.Driver</driver-class>

      <user-name>sa</user-name>
      <password></password>

      <depends>jboss:service=H2,database=localDB</depends>
   </local-tx-datasource>

   <mbean code="cz.vutbr.fit.jboss.jdbc.H2Database"
     name="jboss:service=H2,database=localDB">
     <attribute name="Database">localDB</attribute>
     <attribute name="EmbeddedMode">true</attribute>
   </mbean>

</datasources>
```

Text 3: Datový zdroj pro H2.

V konfiguraci samotného zdroje se změnily elementy `<connection-url>` a `<driver-class>`. Toto je přirozeně základ, jenž bude potřeba změnit i ve všech jiných datových zdrojích, po kterých budeme chtít použití H2. Tímto způsobem nasměrujeme služby pro přístup k databázi k použití správného JDBC driveru a URL. Další důležitou změnou je obsah tagu `<mbean>`. Podstata tkví v atributu `code`, jenž odkazuje na třídu MBean, kterou chceme spustit. Jak lze z hodnoty atributu vyvodit, tato MBean není součástí distribuce JBoss, nýbrž je nutno ji vytvořit. Z jakého důvodu a jakým způsobem se dozvíme v následující kapitole. (Více informací o změně datových zdrojů lze najít v příloze 1.)

3.2.1 Managed bean pro H2 databázi

Připomeňme, že MBean, zkrácený výraz pro managed bean, je Java objekt, který představuje prostředek, jenž můžeme řídit a obsluhovat. Je jedním z klíčových prvků technologie JMX (viz kap. 2.2.1). MBean implementuje svoje řídicí rozhraní, skládající se z

- atributů, které můžeme číst a do nichž můžeme zapisovat, a
- operací, které můžeme vyvolat.

V data source deskriptoru pro HSQL má atribut `code` elementu `<mbean>` hodnotu „org.jboss.jdbc.HypersonicDatabase“. Podíváme-li se do zdrojového souboru této třídy[13], zjistíme, že poskytuje atributy (vlastnosti) jako Database, Port, Password, User, aj. pomocí metod typu „getter“ a „setter“. Jádrem této MBean jsou však metody `startService()` a `stopService()`, které fakticky spouští a uzavírají požadovanou databázi. Pokud tedy chceme emulovat chování HSQL pomocí H2, je nutné (vzhledem k tomu, že žádná taková k dispozici není) vytvořit MBean, jíž

13 Lze najít v `JBOSS_DIST/varia/src/main/org/jboss/jdbc/HypersonicDatabase.java`.

budeme moci pomocí atributů nastavit vlastnosti a pomocí stejnojmenných metod spouštět a ukončovat H2 databáze.

Ve výsledku je MBean pro H2 databázi velice podobá třídě MBean pro HSQL. Má stejně pojmenované metody a podobné atributy. Hlavní rozdíly jsou v některých atributech, ve spouštění serveru při požadavku na databázi se vzdáleným přístupem, výchozí nastavení hodnot atributů a přepínačích/příznacích předávaných při startu serveru či připojovaných k URL při vytváření spojení s databází. Zdrojový kód vytvořené MBean bude popsán v následující kapitole.

3.3 Proces záměny dat

Máme-li MBean pro H2 hotovou a připravenou, nezbývá už nic než začít měnit všechny výskyty konfigurace datových zdrojů HSQL za konfigurace pro H2. Abychom měli jistotu, že žádný výskyt neopomeneme, budeme provádět nahrazení na čistém zdrojovém textu aplikačního serveru JBoss[14].

Princip nahrazení spočívá v prostém nahrazení určitých částí textu za text jiný. Nahrazení lze provést aplikováním pravidel na ty zdrojové soubory aplikačního serveru, které v sobě obsahují řetězec s hodnotou, jíž chceme nahradit. Tato pravidla mají obecný tvar „*hledej v textu řetězec* r1 *a v případě nalezení jej nahraď řetězcem* r2“. Tabulka 2 obsahuje informace, potřené k tvorbě těchto pravidel, a to hledaný řetězec ve svém prvním sloupci a ve sloupci druhém pak požadovanou náhradu. Níže zběžně vysvětlíme význam řetězců pro HSQL i H2 (podrobnosti ke změnám, vztahující se k H2, viz kap. 2.5).

č.	r1	r2
1	`org.jboss.jdbc.HypersonicDatabase`	`cz.vutbr.fit.jboss.jdbc.H2Database`
2	`org.hsqldb.jdbcDriver`	`org.h2.Driver`
3	`hypersonic`	`h2`
4	`jdbc:hsqldb:hsql`	`jdbc:h2:tcp`
5	`jdbc:hsqldb:.`	`jdbc:h2:mem:`
6	`jdbc:hsqldb`	`jdbc:h2`
7	`:1701`	`:9092`
8	`jboss:service=Hypersonic`	`jboss:service=H2`
9	`name="InProcessMode"`	`name="EmbeddedMode"`
10	`name="Persist"`	`name="Persistent"`
11	`name="Port">1701`	`name="Port">9092`

Tabulka 2: Pravidla pro záměnu HSQL a H2 databázi.

- První pravidlo nahrazuje výskyty plně kvalifikovaného názvu třídy MBean, potřebnou ke spuštění a ukončení databáze. Obvykle se vyskytuje jako hodnota atributu `code` elementu `<mbean>` v deskriptoru datového zdroje.

14 Slovem „čistý“ zde myslíme nepřeložený, nesestavený systém, ve stejném stavu jako čerstvě po stažení distribuce. Pokud byly AS či *testsuite* již přeloženy, pomocí příkazu `ant -projecthelp`, spuštěném v instalační složce, lze zjistit, jakým způsobem překládaný projekt do původního stavu obnovit.

- Druhý řádek představuje plně kvalifikovaný název třídy, představující JDBC driver databáze.
- Třetí řádek představuje název adresáře, ve kterém je databáze uložena.
- Čtvrtý až sedmý řádek ovlivňuje URL, s níž se služba připojuje k databázi.
- Osmý řádek definuje, podle specifikace JMX, název objektu, který MBean identifikuje.
- Poslední tři řádky souvisejí s atributy MBean, nastavované v deskriptoru datového zdroje. V těchto případech je potřeba změnit název nastavovaného atributu, v případě portu je vhodné změnit i hodnotu.

Tyto (a další) změny lze popsat pravidly pro náhradu textu, existují však případy, ve kterých je vyžadována zvýšená pozornost ze strany uživatele. Tento případ nastává konkrétně v kódu Java tříd z testsuite, které nevyužívají deskriptor datového zdroje, a tudíž ani příslušnou MBean pro vytváření spojení s databází. Datový zdroj si samy pomocí JDBC API programově vytvářejí. Text 4 obsahuje ukázku takového kódu.

```
// Create startup arguments
String[] args =
      new String[] {
                  "-database", dbPath.toString(),
                  "-port", String.valueOf(port),
                  "-address", address,
                  "-silent", String.valueOf(silent),
                  "-trace", String.valueOf(trace),
                  "-no_system_exit", String.valueOf(no_system_exit),
      };

// Start server
ClassLoader cl = Thread.currentThread().getContextClassLoader();
Class clazz = Class.forName("org.hsqldb.Server", true, cl);
Method main = clazz.getMethod("main", new Class[] { args.getClass() });
main.invoke(null, new Object[] { args } );
```

Text 4: Spuštění serveru HSQL

V horní části příkladu inicializujeme pole řetězců, následně použitých jako argumenty pro vytvoření objektu typu `org.hsqldb.Server`. Ten představuje databázový server, po jehož spuštění lze přistupovat k databázi pomocí IP adresy a portu. Třída představující server u H2 je samozřejmě odlišného typu a není tedy žádným překvapením, že akceptuje jiné argumenty. V tomto konkrétním případě musíme nahradit „-database“ za „-baseDir“, „-port“ za „-tcpPort“, doplnit seznam argumentů o přepínač „-tcpAllowOthers“ a zbytek pro změnu odstranit úplně. Spuštění serveru pak provedeme jednoduchou konstrukcí, jak lze vidět na příkladu Text 5.

```
org.h2.tools.Server server = org.h2.tools.Server.createTcpServer(args);
server.start();
```

Text 5: Spuštění serveru H2

Proces kompletní záměny bude tedy následující:

1. Stažení JDBC ovladače pro H2 databázi (viz kapitola 2.5).
2. Stažení zdrojových kódů JBoss AS a jeho sestavení.
3. Kompilace zdrojových souborů a vytvoření knihovny `H2MBean.jar`. Tato knihovna bude obsahovat MBean námi vytvořenou pro použití v deskriptorech datových zdrojů.
4. Záměna databází, popsaná dříve v této kapitole.
5. Překlad a sestavení testsuite.
6. Propagace knihovny `H2MBean.jar` a JDBC ovladače pro H2 databázi, aby byly k dispozici testsuite i spouštěným instancím JBoss AS. Toho lze jednoduše dosáhnout nakopírováním těchto dvou knihoven do adresářů `lib` všech serverových konfigurací i přeloženého testsuite.
7. Nakonec můžeme spustit testy.

Pokud všechny body výše uvedené proběhly zdárně a sada testů testsuite proběhla bez chyb až do konce, pustíme se do srovnání a vyhodnocování výsledků testů. (Více informací o kompletním procesu záměny databází lze najít v příloze 2.)

3.4 Výsledky testů

Testy byly nejprve spuštěny na instalaci JBoss AS, která byla v původním stavu bez jakýchkoliv změn. Poté byly testy postupně spouštěny na překonfigurovaných instalacích v průběhu řešení problému. Následující statistické výsledky pozměněné instalace jsou výsledky testů s nejaktuálnějšími úpravami.

	Procentuální úspěšnost	Počet testů	Chybné testy (Errors/Failures)
Čistá instalace	99.50%	3973	20 (9/11)
Pozměněná instalace	98.67%	3972	53 (39/14)

I když jsou ve statistických výsledcích jen nejaktuálnější data, bylo by vhodné poznamenat, že počty chybných testů se po vytvoření definitivní sady pravidel pro záměnu textu ve zdrojových kódech JBoss AS příliš neměnil (řádově v jednotkách testů). Od doby ucelení pravidel byla snaha spíše procházet problémové zdrojové soubory (deskriptory datových zdrojů též, ale spíše zdrojové kódy testovacích případů, psaných v jazyce Java) a hledat v nich příčiny chyb. Vzhledem k těmto faktům tedy lze usuzovat, že většina kritických testů má s H2 databází nějaký problém a je třeba odborného náhledu. Na druhou stranu, vzhledem k vcelku malému počtu vznikajících chyb (na kterých se JBoss stabilně drží) lze tvrdit lze tvrdit, že existuje určitá poměrem k problémovým testům velká množina testů, které problémy neprodukují, a dají se tak považovat za úspěšné.

Nyní si popíšeme několik problémů, které při testech nastávají, a postup jejich řešení. Prvním bude výjimka

- `java.lang.ClassNotFoundException: org.h2.jdbc.JdbcSQLException (no security manager: RMI class loader disabled),`

která nastává v 6 případech. Po nějakém čase hledání po tématicky zaměřených internetových fórech a diskusních skupinách bychom mohli dojít k závěru, že se jedná o absenci objektu, který se stará o bezpečnost přístupu (SecurityManager). Nicméně, po nahlédnutí do logovacího souboru konkrétní serverové konfigurace, na které tento nepodařený test běžel, se dostáváme k obyčejné výjimce:

- `org.h2.jdbc.JdbcSQLException: URL format error; must be "jdbc:h2:{{.|mem:}[name] | [file:]fileName | {tcp|ssl}: [//]server[:port][,server2[:port]]/name}[;key=value...]" but is "jdbc:h2:tcp://localhost:9092" [90046-132]`

Tato výjimka v podstatě znamená, že na konci JDBC URL pro H2 databázi chybí lomítko a název konkrétní databáze, což pro H2 není validní URL. Pokud tedy začneme zkoumat zdrojový kód na výskyt takového typu URL (tedy končící hostname:URL bez názvu databáze), zjistíme, že takových výskytů je vcelku hodně (řádově desítky). HSQL podporuje takovou URL v případě, že je nastavený tzv. alias (zástupné jméno databáze, díky němuž se klienti k databázi připojují) na prázdný řetězec, a to v souboru (web)server.properties. Výskyt takového souboru je však ve struktuře JBoss nulový. Na mysl může někomu přijít, že se jedná o automatické připojení k jediné existující databázi, která je přes databázový server k dispozici. Po delší detektivní činnosti (odhadu datového zdroje, který by mohl být ten pravý) a následném doplnění požadovaného jména k URL zjistíme, že to na výsledek testů nemá vliv. Zjevně bude problém jinde. Problém tedy nebyl vyřešen.

Podobný příklad, produkujícím stejnou výjimku, nazveme záhada uživatelského jména a hesla. V některých testech se vytváří spojení s in-memory databází. Při vytvoření nové databáze má databáze vždy uživatelské jméno nastaveno na řetězec "sa" a heslo na prázdný řetězec. V testu se viditelně použijí k vytvoření spojení výše zmíněné hodnoty, produkují chybový text s obsahem, že uživatelské jméno nebo heslo není správné. Jiný problém, kdy se dokonce nekorektně ukončovala JVM, se vyřešil nainstalováním poslední verze JRE od Sunu.

Po průběhu testů při generování závěrečné zprávy do člověkem zpracovatelného formátu v některých případech nastane výjimka „`java.lang.OutOfMemoryError: GC overhead limit exceeded`“ s výpisem `BUILD FAILED`. Po pokusu generaci testů zopakovat se pravděpodobně stane to samé. Problém je pravděpodobně způsoben tím, že vývojáři testsuite nepočítali s příliš velkým objemem výstupních dat při koncovém zpracování. Řešení lze dosáhnout změnou parametru interpretu java, který udává maximální rozsah alokace operační paměti pro spuštěnou JVM. V souboru `testsuite/build.{sh|bat}` najděte hodnotu „-Xmx1024m“ a číslo 1024 změňte na vyšší (u mě se osvědčilo 2048, ale záleží na velikosti operační paměti).

O výsledcích testů a kompletní záměně databáze H2 za HSQL tedy můžeme prohlásit, že proběhla vcelku úspěšně, nicméně existují chyby a problémy, které je nutné vyřešit před jejím nasazením. (Všechny chyby lze najít na přiloženém DVD.)

4 Závěr

Cílem práce bylo pokusit se zjistit, zda a do jaké míry lze použít databázi H2 v aplikačním serveru JBoss. Zadání se zaměřilo na kompletní záměnu databáze Hypersonic SQL, v JBoss AS používanou jako implicitní databázi, za H2, protože sada unit testů, která testuje funkcionalitu celého aplikačního serveru, ve velké většině případů využívá právě Hypersonic SQL. Databázi se podařilo úspěšně zaměnit a výsledky testů byly porovnány a vyhodnoceny. Proces záměny databáze byl převážně automatizován a zdokumentován, proto lze tento postup poměrně jednoduše v budoucnu zopakovat. Cíl této práce byt tedy dosažen a zadání splněno.

Význam lze předpokládat pro projekt JBoss a komunitu kolem něj. Vzhledem k výsledkům srovnání testů databází H2 a HSQL lze tvrdit, že H2 můžeme používat v rámci JBoss AS. Byly však nalezeny chyby (chyb u testů H2 bylo více než chyb u testů HSQL), na které je nutno se před nasazením zaměřit a vyřešit. Práce může být též dobrým startovním bodem pro jiné využití H2 v JBoss AS i jeho subprojektech.

Navíc, na webových stránkách podpory pro JBoss[15] je jasně řečeno, že „Hypersonic (HSQLDB) by v žádném případě neměla být používána v ostrém provozu“. Mluví se zde o problémech s izolací transakce, neuvolňování použitých zdrojů, kvalitě perzistence a poškození dat při pádu, stabilitě pod velkým zatížením aj. Hlavním důvodem a zároveň výhodou použití HSQL je možnost konfigurace a následné použití bez nutnosti rozsáhlejších úprav při vývoji a testování, dále pak licenční politika HSQL. Vyvstává zde tedy idea nahradit v rámci projektu JBoss AS HSQL databází H2 kompletně.

15 Viz http://kbase.redhat.com/faq/docs/DOC-15194.

Literatura

[1] Flanagan, D.: *Java in a Nutshell, 5th Edition*. Sebastopol: O'Reilly, 2002. 992 s. ISBN 0-596-00773-6

[2] Herout, P.: *Učebnice jazyka Java*. České Budějovice: KOPP, 2007. 381 s. ISBN 978-80-7232-323-4

[3] McGovern, J., aj.: *Java 2 Enterprise Edition 1.4 Bible.* Indianapolis: Wiley, 2003. xxix, 976 s. ISBN 0-7645-3966-3

[4] Lindholm, T., Yellin, F.: *The Java™ Virtual Machine Specification (2nd Edition)*. Prentice Hall, 1999. 496 s. ISBN 978-0201432947.

[5] Jamae, J., Johnson, P.: *JBoss in Action: Configuring the JBoss Application Server.* Greenwitch: Manning, 2009. 496 s. ISBN 978-1933988023

[6] Mukhar, K., Zelenak, C., aj.: *Beginning Java EE 5: From Novice to Professional*. Apress, 2006. 672 s. ISBN 978-1-59059-470-4

[7] The Java EE 5 Tutorial [online]. [cit. 26.1.2010]. Dostupný z WWW: <http://java.sun.com/javaee/5/docs/tutorial/doc/>

[8] Smart, J., F.: *Java Power Tools*. Sebastopol: O'Reilly, 2008. 912 s. ISBN 978-0-596-52793-8

[9] H2 Database Documentation [online]. [cit. 31.1.2010]. Dostupný z WWW: <http://www.h2database.com/h2.pdf>

Seznam příloh