Excel@FIT 2015
http://excel.fit.vutbr.cz

Automatická koordinace a řízení procesů na platformě Java

Martin Janyš*

Abstrakt
Předmětem článku je téma odolnosti a stability webových aplikací se zaměřením na platformu Java. Řada existujících informačních systémů postavených nejen nad touto platformou se potýká s problémy, které narušují stabilitu aplikace. Tyto problémy pak mohou vyústit v plánovaný nebo neplánovaný výpadek, odstávku a následně i finanční nebo obchodní ztrátu v důsledku nefunkčnosti celé služby. Cílem bude ukázat problémy, se kterými se aplikace potýkají v provozním prostředí a jak je proaktivně řešit. Jako možná dílčí řešení zvýšení stability mohou být vhodná konfigurace JVM (Java Virtual Machine), analýza a oprava odhalených chyb a nebo technika na zvýšení stability nazývaná Sandboxing, které se věnuje tato práce. Pomocí této techniky je možné rozdělit aplikace do samostatných částí, které se nemohou ovlivnit. Zamezí se tak šíření chyb mezi částmi aplikace a tím zvýšíme stabilitu celé aplikace. Mezi cílové aplikace patří Java aplikace realizované za pomoci aplikačního rámce Spring. Do takto postavených aplikací lze zavést techniku Sanboxing vhodnou konfigurací, která zajistí, že běh aplikace bude rozdělen do určených částí, které budou automaticky testovány a případně restartovány. Aplikace se tak sama zotaví v postižených částech bez kompletního výpadku. Práce je realizována ve spolupráci s firmou IBAcz. Projekt nese jméno Java Capsules.

Klíčová slova: Java — Sandbox — Spring — Oddělování procesů — Webové aplikace — Stabilita

Přiložené materiály: *N/A*

*xjanys00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Práce se zabývá problémy stability, které se objevují ve webových aplikacích, jakými mohou být informační systémy a podnikové portály. Tyto systémy jsou často v podobě monolitické aplikace, běžící na jednom aplikačním serveru a v jednom procesu i když se jedná o několik samostatně nasazených aplikací. Nasazeným aplikacím není možno přidělovat systémové prostředky jednotlivě. Funkcionalita, která je z důvodu chyby schopna vyčerpat všechny přidělené systémové zdroje může způsobit výpadek celé aplikace. Pokud tento problém oddělíme od zbytku aplikace, chyba nepostihne celý systém. Zotavení může proběhnout také pouze v rámci zasažené části.

Na poli podnikových aplikací, přesněji řečeno portálů, se objevil princip *sandboxingu* a to v podání firmy Liferay [1]. Liferay poskytuje formou pluginu možnost využívat sandboxing portletů ve webové aplikaci. Tuto podporu lze využít pouze v kombinaci s touto platformou a v placené edici, která je určena firmám a podnikům (Enterprise edition). Proto na ně

většina institucí nemusí finančně dosáhnout.

Praktická část této práce staví na odlišných idejích. Aplikovaným principem na zvýšení stability je také bezpečnostní mechanizmus (*sandboxing*)[1]. Tato technika je schopna přispět k prevenci problémů, které budou blíže popsány dále v sekci 2. Volbou otevřené platformy Java a Spring si hledá místo a uplatnění v širší škále existujících aplikací mezi systémy využívající aplikační rámec Spring. Dále je kladen důraz na oddělení menší části aplikace než celé portletové aplikace. Implementace je navržena na oddělení libovolné části aplikace a nejmenším oddělitelným celkem je jeden objekt (Java *bean*).

Projekt této práce nese jméno *Java Capsules*. Realizace Java Capsules poskytuje nástroj na oddělování částí aplikací za účelem ochrany stability zbývajícího celku. Koncept je funkční a dle mého názoru a zkušeností i uplatnitelný. Jeho funkčnost je podložena testy výkonnosti (performance testing), které srovnávají propustnost aplikace s využitím techniky Sanboxing a bez ní. Realizace je hotova a jsou k dispozici i výsledky testů výkonu řešení. Jejich stručné zhodnocení je součástí kapitoly 5. Řešení přináší do aplikace zvýšení odolnosti, za cenu určité režie a zvýšení odezvu.

2. Problémy podnikových aplikací

Používaná terminologie je vztažena k systémům na platformě Java (přesněji *Java EE*), která se používá pro implementaci podnikových aplikací. V momentě, kdy budeme hovořit o aplikaci nebo systému, je zpravidla myšlena aplikace podniková nebo webová a informační systém běžící na *Java Virtual Machine*.

Mezi vybrané problémy patří trojice klíčových problémů, kterým se práce věnuje: stabilita, přidělování prostředků a škálovatelnost.

2.1 Stabilita

Vlastnosti každého nedistribuovaného systému jsou do značné míry ovlivněny nejslabším článkem. Proto může zejména výkon a stabilita jednoho systému záviset na takovém úzkém místě. Pokud se v aplikaci vyskytuje část, která odebírá pro svůj běh neúměrně velké množství výpočetního času nebo paměťového prostoru, zpravidla se tak děje na úkor ostatních subsystémů. Pokud tato problematická část vyčerpá zdroje, dostává se systém do potíží. V krajních případech může tato situace vést až k zastavení nebo pádu celého systému. Systém přestane komunikovat s okolím a je pro svůj účel nepoužitelný.

Pod pojmem stabilní aplikace budeme uvažovat aplikaci, která odpovídá na příchozí požadavky, a to dokonce i za nespecifikovaných podmínek se umí zotavit. Do nedefinovaného stavu se může aplikace dostat z různých důvodů. Tyto stavy nelze v naprosté většině případů řešit reaktivně, protože zpravidla nelze zabrané prostředky do systému vrátit nebo obnovit stav před poruchou. Aplikace v takovém chybovém stavu není schopna plnit svůj účel.

Stabilitu mohou ovlivňovat i produkty třetích stran, které nejsou pod naší správou. Může se jednat o dodavatele do stejné aplikace ale i o využité aplikační rámce a knihovny, které obsahují chyby.

Neočekávaný stav v jedné části systému může postihnout stabilitu systému jako celku. Restart celé aplikace potom vede k odstávce systému. Tato odstávka, ať už se jedná o manuální či automatické zotavení systému, může být pro zákazníka kritická a tvůrci aplikace mohou být vystaveni sankcím v rámci dohody o úrovni poskytovaných služeb (*SLA* - Service Level Agreement)[2]. Po spuštění může následovat kontrola funkčnosti a výpadek se dále prodlužuje.

Mezi časté problémy se řadí výpadek sítě a rozpojení komunikace, hardwarovou poruchu, přetečení zásobníku, vyčerpání počtu otevřených souborů nebo dosažení maximálního počtu vláken. Jedna z nejčastějších příčin pádu aplikace je způsobena vyčerpáním paměti (`Out of Memory Error`). Nebývá to ovšem případ, kdy bychom systému zapomněli nastavit dostatečný příděl operační paměti, ale jde o únik paměti (tzv. *memory leak*). Pády na nedostatek paměti bych ze svých zkušeností z praxe rozdělil na dvě skupiny. V prvním případě se jedná o *chybu na straně aplikace*, ať už naši nebo využívaného produktu. V druhém případě může být část systému dočasně přetížena aktuálním provozem. Příkladem takového případu může být rozesílání měsíčních vyúčtování, SMS kampaně apod., ale i v těchto případech není chyba v implementaci aplikace vyloučena. V obou případech budeme mít za cíl prostředí v co nejkratší době zotavit. V práci budu hledat řešení na předcházení pádu celé aplikace a na zotavení bez nutnosti odstávky celého systému.

2.2 Přidělování prostředků

Problém stability, který poukazuje na slabý článek v systému, je doprovázen dalším negativním faktorem. Tím je nemožnost nastavovat na konfigurační úrovni

[1] Sandbox je bezpečnostní mechanizmus, jehož principem je oddělení některých programů do samostatného prostředí tak, aby se zamezilo šíření chyb do ostatních oblastí systému. Je tak odstraněna hrozba pro okolní programy [2].

[2] SLA je portfolio metrik je stěžejním parametrem rozhrani mezi odběratelem a externím poskytovatelem infomatických služeb [3]

s vysokou úrovní abstrakce prostředky, které část aplikace smí využívat.

V rozsáhlém systému bývá také obvykle více dodavatelů, kteří se dělí o stejné zdroje, které jim jsou v rámci jedné aplikace poskytnuty na stejném hardware a na stejné JVM (Java Virtual Machine). Systém pak tyto prostředky přiděluje všem, jež si o ně požádají. Je tedy velice obtížné přesně vymezit a přidělit zdroje jednotlivým aplikacím v jednom systému. Pokud aplikace jednoho dodavatele vyčerpá všechny zdroje, bude tím postižen celý systém bez rozdílu. Není tak možné například pro dva dodavatele rozdělit operační paměť a čas procesoru na poloviny. V některých případech to ani nebude vhodné.

Prakticky bude v takovém případě rozumné shora omezit prostředky pro jednotlivé části aplikace různých dodavatelů. Tím dosáhneme zejména toho, že aplikace nebude mít možnost neúměrně čerpat prostředky na úkor jiné své části.

2.3 Škálovatelnost

Požadavky systému na zdroje se při jeho růstu a přidávání nové funkcionality mohou přirozeně zvýšit. Takto zvyšující se požadavky nemusí být možné z technických důvodů uspokojit v rámci jedné výpočetní jednotky. V takovém případě by bylo vhodné aplikaci rozdělit do vzájemně komunikujících celků. Na takový zásah není zpravidla architektura současného řešení připravena. Při vysoké složitosti kódu aplikace ani nemusí být takový úkon efektivně proveditelný.

V případě, že by bylo možné aplikaci rozdělit na komunikující celky, máme řešení na právě zmíněné problémy. Vznikl by tak komunikující distribuovaný systém, jehož odolnost, stabilita a přidělování prostředků je dána vlastnostmi nezávislých uzlů. Toho lze dosáhnout pomocí existujících technologií jako *JAX-WS*, *JAX-RS* nebo *JMS*. Tyto technologie definují komunikační rozhraní mezi systémy. Vyžadují však implementační zásah a znalost dané technologie a musí být integrovány na úrovni architektury a jejich dodatečné zavedení do komplexního systému nemusí představovat triviální úlohu. Společnou odpovědí na tyto problémy a omezení je právě moje práce — Java Capsule. Tento aplikační rámec zajistí automatické oddělení do procesů a potřebnou komunikaci. Pro využití je nutná minimální konfigurace a vše je zajištěno bez nutnosti změn současné implementace[3].

[3] Jediné omezení je serializovatelnost objektů. Jedná se však pouze o doplnění rozhraní `Serializable` k hlavičce třídy

3. Zajištění stability Java aplikací

Příčiny porušení stability tkví v širokém spektru možných scénářů, které mohou nastat. Tyto případy lze popsat jako neošetřené či nečekané stavy, do kterých se aplikace dostala jako výpadek na síti, neošetřené vstupy aplikace (například velikost vstupního souboru), vyčerpání vláken nebo paměti a mnoho dalších. Uniformní odpovědí na řešení takových problémů je návrat aplikace do známého stavu (restart) nebo v krajním případě obnovení záloh systému (revert). Cílem je oddělit aplikaci do samostatného procesu (sandboxu), který bude možné nezávisle pomocí *let it crash* [4] modelu, neboli nechme to spadnout, zotavit restartem.

3.1 Návrh

Řešení problému stability spočívá v rozdělení aplikace na celky, které budou běžet odděleně. Nejmenším oddělitelným celkem je objekt. Tím dosáhneme, že v případě výskytu problému zůstane zbytek aplikace neohrožen. Java Capsules je schopná ochránit aplikaci pouze před problémy, které se vyskytnou za běhu a jsou vláknově bezpečné, tedy nepracují například se statickými poli tříd.

Rozdělení aplikace na takové celky musí splňovat některá základní specifika.

1. Oddělené části aplikace musí být na sobě nezávislé.
2. Navržené řešení musí poskytovat mechanizmus pro automatické uvedení oddělených částí do konzistentního stavu (restart).
3. Zavedení mechanizmu oddělení musí být konfiguračního charakteru v rámci aplikace.

Pro účely oddělení připadá v úvahu vlákno, proces nebo samostatný hardwarový prvek. Vzhledem k tomu, že vlákno tvoří samostatný celek pouze z pohledu výpočtu na procesoru, není oddělení pomocí vláken vhodné. Použití samostatných hardwarových jednotek by porušilo bod 3, protože bychom, kromě samotné aplikace museli vytvořit architekturu výpočetních uzlů. Vhodným prostředkem pro oddělení celků je tedy jen proces. Pokud oddělenému celku dojde paměť neohrozí zbylou funkcionalitu, protože se chyba projeví pouze v rámci jednoho procesu[4].

Vybraným řešením bude tedy *rozdělení aplikace do více procesů a jejich automatická správa a koordinace* (*Sandboxing*). Pokud v aplikaci oddělíme části do samostatných procesů, nutně nás to zavede k nutnosti

[4] Vzhledem k dnešním hardwarovým možnostem neuvažujeme případ vyčerpání operační paměti. Každý Java proces má přidělenou svoji vlastní určenou paměť a její maximální limit

meziprocesové komunikace (Inter-Process Communication — *IPC*). Mezi základní *ipc* patří signál, socket, roura, sdílená paměť. Většina IPC je použitelná pouze v rámci jednoho stroje. Vzhledem k tomu, že z řešení nechceme vylučovat možnost běhu Sandboxu na jiném stroji, připadá v úvahu pouze *socket*.

Stav, kdy aplikace v odděleném procesu přestane fungovat, bude možné detekovat a podsystém zotavit bez nutnosti výpadku. Jednotlivé procesy je možné konfigurovat samostatně a tím je omezit. Takto dosáhneme jemnějšího dělení prostředků, než bylo možné v rámci monolitické aplikace, a navrhované řešení bude možné použít i pro vzdálené sandboxy, kdy hlavní systém a jeho podčást nemusí nutně běžet na stejném stroji.

Pro realizaci jsou stěžejní především 2 technologie z aplikačního rámce Spring. Jedná se o vzdálené volání [5] a aspektově orientované programovaní [6].

3.2 Princip řešení zvýšení stability

Při rozdělení aplikace na několik samostatných celků, které spolu komunikují, jsou vlastnosti celého systému určeny parametry jednotlivých částí nezávisle na ostatních.

Oddělený proces bude vytvářet sandbox a bude v něm spuštěn ekvivalent současné aplikace. Procesy spolu komunikují a volání jsou vykonávána v rámci sandboxu. V případě pádu části aplikace není ohrožen celý systém a je možné tento subsystém resetovat samostatně. Dosáhneme tak řešení problémů stability a zotavení (viz Stabilita 2.1).

Sandbox představovaný Java procesem je možné samostatně konfigurovat a omezit tak přístup ke zdrojům (viz Přidělování prostředků 2.2).

Dalším produktem řešení bude možnost definovat Sandboxy na různých strojích. Je to však pouze nadstavba nad samotným Sandboxem. Vybrané řešení tuto variantu podporuje nativně, protože socket je určen pro síťovou komunikaci. Speciální přístup si žádá pouze správa (start, restart) sandboxu. Takovým rozdělením lze zvýšit škálovatelnost aplikace (viz Škálovatelnost 2.3).

Pomocí oddělení běhu Java aplikace do samostatných procesů získáme vlastnosti systému, které řeší předchozí problémy. V systému se bude vyskytovat jeden hlavní uzel, který bude rozdělovat požadavky vydefinovaným odděleným částem.

4. Využití

4.1 Možnosti nasazení

Možnosti použití jsou úzce spojeny s úsilím, které je nutné vynaložit k nasazení takových Sandboxů. Nutná konfigurace musí tedy být tedy minimální. Samotný mechanizmus zavedení a udržování Sandoxů musí být sám o sobě velice stabilní, jinak by nemohl řešit problémy stability. Z těchto důvodů je vhodné zvolit v maximální míře existující řešení, která již poskytují náležitý standard. Možnosti nasazení musí pokrývat širokou škálu aplikačních domén. Použití Sandboxu je vhodné i v případě, kdy chceme použít nedůvěryhodný software (knihovnu).

Tyto prerekvizity mě vedly k výběru prostředků pro realizaci mechanizmu Sandboxu. Implementace je rozšířením aplikačního rámce `Spring`[5] a je s ním tedy velmi snadno integrovatelná. Podnikové aplikace jsou zpravidla postaveny nad vybraným aplikačním rámcem a právě Spring. Nutno podotknout, že Spring nenabízí prostředky pouze pro webové aplikace. Jeho spektrum užití je mnohem širší. Tento rámec je také neinvazivním způsobem integrovatelný do většiny ostatních webových rámců a technologií *java EE*.

4.2 Přínos

Zavedením rámce Java Capsule, který přináší do existující aplikace koncept Sandboxingu, získáváme zvýšení odolnosti a stability aplikace. Aplikaci lze ručně či automaticky spravovat po definovaných částech. V případě chyb za běhu aplikace je šíření chyby omezeno pouze na oddělenou část a zbytek aplikace zůstane ochráněn. Stabilita aplikace jako celku se tedy zvýší, protože jedna chyba nezpůsobí výpadek celého systému. V konfiguraci lze definovat libovolné uskupení objektů, tříd, balíčků nebo typů a jejich provádění přenášet do volání pod sandboxem. V případě, že tato volání způsobí chybu, správa sandboxu se o tom dozví a může tuto instanci sandboxu restartovat. Po startu je obnovena původní komunikace a systém funguje jako před chybou.

Na příkladech, které mám k dispozici lze pozorovat nejčastější chyby Java aplikací jako vyčerpání paměti, vyčerpání počtu vláken nebo ukončení (v reálném případě např. systémem). Jakmile je automaticky detekován výpadek aplikace je zotavena znovu spuštěním sandboxu. V praxi jsme se setkali s případy, kdy jedna služba na obsluhu zpráv obsahovala těžko odhalitelnou race condition, která znemožnila funkčnost služby, nebo s případem, kdy marketingová kampaň pravidelně přetížila systém natolik, že nemohl dále pracovat, a dalšími. V obou případech pak bylo nutné celý systém restartovat. Start takového systému pak znamenal nedostupnost okolo jedné hodiny ve špičce. Oba tyto případy by bylo možné řešit pomocí Java Capsules a vyhnout se tak odstávce celého systému.

[5] http://www.spring.io

Protokol	Čas jednoho volání v ms
—	2
RMI	546
—	49
Http Invoker	95
Burlap	108
Hessian	**87**
Hessian2	97
—	49
Binnary	**67**
Restful/binnary	79
Restful/json	84

Tabulka 1. Odezva — Java Capsules

4.3 Dosažené výsledky – odezva

Tabulka 1 ukazuje tři sady testů různých částí Java Capsules a jejich odezvu v milisekundách. Zavedení Java Capsules přináší do aplikace pochopitelnou režii při volání odděleného procesu. Z profilování běhu v Java Capsules kódu bylo zřejmé, že aplikace tráví 90–95% času právě síťovou komunikací. Výsledky odezvy zůstaly ve stejném řádu a například pro oddělení celé webové aplikace (jeden z režimů oddělení, aplikace komunikují na úrovni http požadavků) byla odezva zvýšena z 49 ms (aplikace bez Java Capsules) na 67 ms. Pro oddělení na úrovni objektů (bean, komunikace na úrovni invokaci metod) pak z 49 ms na 87 ms[6].

5. Závěr

Aplikace píší lidé. Proto v nich jsou nežádoucí problémy stability a také nadále budou. Lze se jim však bránit bezpečnostními technikami jako Sandboxing.

V současnosti je implementace hotova a otestována jednotkovým testováním a performance testováním webové aplikace s nasazeným řešením Java Capsules. Na ukázkových případech lze demonstrovat přínos zavedení. Klíčové je, že se veškeré úpravy existující aplikace týkají pouze konfiguračních souborů. Není nutné upravovat kód existující aplikace. Práce je realizována ve spolupráci se společností IBAcz, která se vývojem podnikových aplikací zabývá.

Ve své praxi jsem se již setkal se všemi druhy chyb, které byly v textu zmíněny. Java Capsules je aplikační rámec, který uživateli poskytne prostředky pro zvýšení stability a pro tento účel si jistě najde místo v problémových částech existujících řešení. Usuzuji tak i z ohlasu, který byl v komunitě po představení řešení týkající se stejného tématu od firmy Liferay.

[6] Nejednalo se o modelový příklad, ale o reálnou aplikaci, která byla vytvořena jako samostatný projekt.

Literatura

[1] *Using Liferay Portal 6.2*, chapter Sandboxing Portlets to Ensure Portal Resiliency. [online]. 2015 [cit. 2015-04-01], Liferay Inc.

[2] Cory Janssen. Sandboxing. [online]. 2010 – 2014 [cit. 2015-03-26].

[3] Pavel Učeň. *Metriky v informatice: jak objektivně zjistit přínosy informačního systému*. Grada, 2001.

[4] Benjamin Wootton. Increasing system robustness with a "let it crash" philosophy. [online]. 2013 [cit. 2015-04-01].

[5] *Spring Framework Reference Documentation*, chapter Remoting and web services using Spring. [online]. 2014 [cit. 2015-04-01], Pivotal Software.

[6] *Spring Framework Reference Documentation*, chapter Aspect Oriented Programming with Spring. [online]. 2014 [cit. 2015-04-01], Pivotal Software.