Masarykova univerzita

Fakulta informatiky



Monitoring Esper serverů pomocí JMX

Diplomová práce

**Bc. Eduard Tomek  
2015**

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

**Vedoucí práce**: RNDR. Filip Nguyen

Poděkování

Na tomto místě bych rád poděkoval RNDr. Filipu Nguyenovi za cenné rady potřebné k vytvoření této práce a za vedení této práce.

**Shrnutí**

Cílem této diplomové práce je vytvoření aplikace pro monitoring Esper serverů pomocí JMX. V práci jsou popsány…

Klíčová slova

Esper, JMX, monitoring, komplexní zpracování událostí, CEP

Obsah

[1 Úvod 7](#_Toc418665841)

[2 Esper 8](#_Toc418665842)

[2.1 Zpracování událostí 8](#_Toc418665843)

[2.1.1 Události 9](#_Toc418665844)

[2.1.2 Zpracování komplexních událostí 10](#_Toc418665845)

[2.2 Esper 11](#_Toc418665846)

[2.3 Verze Esperu 12](#_Toc418665847)

[2.4 Fungování Esperu 12](#_Toc418665848)

[2.4.1 Atributy událostí 13](#_Toc418665849)

[2.4.2 Dynamické atributy událostí 13](#_Toc418665850)

[2.4.3 Dotazování 14](#_Toc418665851)

[2.4.4 Proudy událostí 15](#_Toc418665852)

[2.4.5 Datová okna 16](#_Toc418665853)

[2.5 Reprezentace událostí 16](#_Toc418665854)

[2.5.1 POJO události 18](#_Toc418665855)

[2.5.2 Události java.util.Map 19](#_Toc418665856)

[2.5.3 Události reprezentované polem objektů 20](#_Toc418665857)

[2.5.4 XML události org.w3c.dom.Node 20](#_Toc418665858)

[2.6 Sběr výkonnostních statistik v Esperu 21](#_Toc418665859)

[2.6.1 Systémové metriky 21](#_Toc418665860)

[2.6.2 Metriky dotazů 22](#_Toc418665861)

[2.6.3 Sběr metrik 23](#_Toc418665862)

[3 Monitoring 25](#_Toc418665863)

[3.1 Monitoring Java aplikací 25](#_Toc418665864)

[3.2 JMX standard 26](#_Toc418665865)

[3.2.1 Architektura 27](#_Toc418665866)

[3.2.2 Managed Bean (MBean) 28](#_Toc418665867)

[3.2.3 MBean server 28](#_Toc418665868)

[3.2.4 JMX klient 29](#_Toc418665869)

[3.3 Vizualizace monitorovaných dat 30](#_Toc418665870)

[3.4 Příbuzné práce 31](#_Toc418665871)

[4 Monitoring Esper serveru 32](#_Toc418665872)

[4.1 Aplikace pro monitoring Esper systému 32](#_Toc418665873)

[4.2 JMX server pro Esper 33](#_Toc418665874)

[4.3 JMX klient 33](#_Toc418665875)

[4.4 Vizualizace 33](#_Toc418665876)

[5 Testování aplikace 34](#_Toc418665877)

[5.1 Uživatelské rozhraní 34](#_Toc418665878)

[5.2 Test výkonnosti při různých reprezentacích událostí 34](#_Toc418665879)

[5.2.1 POJO 34](#_Toc418665880)

[5.2.2 Map 34](#_Toc418665881)

[5.2.3 Pole objektů 34](#_Toc418665882)

[5.2.4 XML 34](#_Toc418665883)

[5.3 Test výkonnosti při různých způsobech odběru výsledků dotazu 34](#_Toc418665884)

[5.3.1 Subscriber 34](#_Toc418665885)

[5.3.2 Listener 34](#_Toc418665886)

[5.3.3 Iterator 34](#_Toc418665887)

[6 Budoucí práce 35](#_Toc418665888)

[7 Závěr 36](#_Toc418665889)

[8 Seznam obrázků 37](#_Toc418665890)

[9 Literatura 38](#_Toc418665891)

[10 Příloha A 40](#_Toc418665892)

1. Úvod

Úvodní slova…

1. Esper

Esper je nástroj pro zpracování komplexních událostí.

* 1. Zpracování událostí

Zpracování událostí není žádný nový obor. Toto téma je zkoumané již více než šedesát let. Vývoj zpracování událostí probíhal přirozeně od aplikací v komunikačních systémech, zpracování informací, až po podnikové systémy. Během šedesáti let vznikly čtyři hlavní směry, z nichž každý se specializuje na zpracování událostí jiného typu. Jde o diskrétní událostní simulace, sítě, databáze a middleware.

Simulace pomocí zpracování událostí znamená jejich využití k řízení kroků vedoucích k předpovědi chování systému. Toto je jedno z prvních využití událostí a událostmi řízených výpočtů. Základní myšlenkou je použití počítačového programu, který imituje modelovaný systém. Protože celý systém je většinou jako jeden celek příliš složitý, bývá rozdělen do několika částí, které mezi sebou komunikují právě pomocí událostí a ovlivňují se tak navzájem. Tento přístup je využit například v předpovědích počasí, kdy je geografická oblast rozdělena na více částí o obsahu několika desítek km2 a pro každou takovou část existuje vlastní simulace specializovaná na tuto oblast.

Technologie zpracování událostí byla také vyvinuta jako základní část počítačových sítí. Zprvu ale nebylo vůbec zřejmé, že princip fungování sítí je vlastně zpracování událostí.

V 80. letech 20. století byla technologie zpracování události vyvinuta pro použití v databázích. V tradiční databázi existují statická data a pomocí dotazů lze tato data získávat. Databáze analyzuje přítomná data a vrátí výsledek. Tedy výsledek je aktuální pouze v okamžiku dotazu, po jeho skončení může být již stav v databázi jiný a tedy výsledek již neplatný. Aktivní databáze fungují na principu časových oken, a protože jsou výsledky spjaty s časem jejich vyhodnocení, pro daný čas jsou vždy aktuální.

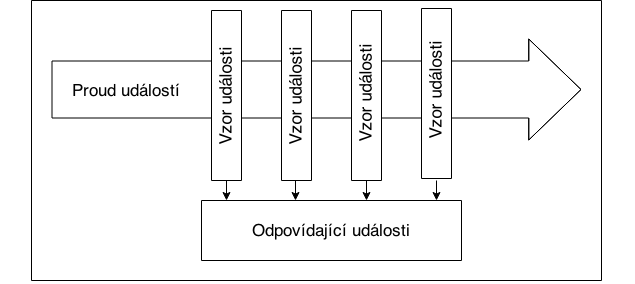
Posledním směrem využívání technologie zpracování událostí je middleware. Jeho vývoj započal na konci 80. a pokračoval během 90. let 20. století. Smyslem bylo pokusit se modularizovat síťové vrstvy a komunikační protokoly, které se začínaly stávat kritickou infrastrukturou a propojit tyto moduly dohromady. Middleware měl za úkol skrýt události spojené s přenosem zpráv a namísto toho nabídnout rozhraní vyšší úrovně. Uživatelé nemusí vědět jak je jejich komunikace zpracovávána. Zjednodušené rozhraní pro příjem a poslání zprávy je zde dostatečné. [1][2][3]

* + 1. Události

Událost je cokoliv co se stane nebo je uvažováno, že se stane. Pokud se například auto blíží ke křižovatce, kde na semaforu svítí červené signalizační světlo, lze si představit, co by se stalo, kdyby auto nezastavilo, ale protože nakonec zastaví, tyto události se neuskuteční. I přesto, že tyto události nenastaly, pořád se jedná o události. Takové události jsou nazývány virtuální události.

Události nejsou jen typem zprávy. Ve speciálním případě tomu tak může opravdu být, ale většinou obsahují dodatečné informace. Událost obsahuje zejména informaci o času, kdy nastala. Pokud aktivita trvá nějaký čas, pak by událost měla obsahovat i informaci o době trvání reprezentovanou pomocí času začátku a konce aktivity. Událost tedy nepopisuje jen aktivitu, kterou reprezentuje, ale i kdy nastala, jak dlouho trvala a jaký má vztah k ostatním událostem.

Události jsou často seskupovány do proudů událostí. Proud událostí je lineárně uspořádaná sekvence událostí vznikajících ve specifikovaném čase. Některé z prvních komerčně úspěšných CEP aplikací zpracovávaly proudy událostí. Tyto systémy nevyužívaly informaci o vzájemném vztahu událostí, ale pouze data, která události obsahovala. [1][4][8]



1. Schéma získávání odpovídajících událostí z proudu událostí [4]
   * 1. Zpracování komplexních událostí

Zpracování komplexních událostí (Complex Event Processing - CEP) je dalším krokem v aplikacích zpracování událostí. Komplexní událost je taková událost, která může být ovlivňována ostatními událostmi, a to i v případě kdy tyto události nenastanou. CEP označuje sadu konceptů a principů pro zpracování událostí a zároveň i metody k implementování těchto konceptů. Tento termín se stal také označením pro produkty, které zpracovávají vysokoúrovňové události.[1]

CEP splňuje dvě základní podmínky pro dynamické a vysoce škálovatelné systémy. První z nich je oddělení poskytovatelů a přijímačů informací. Stejně tak jako poskytovatelé nemusejí znát svoje přijímače, tak ani přijímače totiž nemusejí mít znalost o zdroji přijímaných dat. Druhou podmínkou je, že CEP systémy nejen že zprostředkovávají informace ve formě událostí mezi poskytovateli a přijímači, ale také podporují detekci vztahů mezi nimi.[5]

CEP systémy zahrnují mnoho různých technologií. Tyto systémy se obyčejně potýkají s několika proudy událostí. Tyto události musí být neměnné. Událost nemůže být modifikována ani smazána. Události mohou být pouze kopírovány se změněnými atributy a kopie je potom novou událostí. Události v každém proudu mohou přicházet z různých zdrojů. Systémy mohou přijímat jednotky událostí za hodinu i tisíce událostí za sekundu. Techniky zpracování takových událostí se proto liší podle povahy událostí a cílů vlastního zpracování. [1][2]

Základními technikami zpracování událostí jsou:

Filtrování – snaha zbavit se většiny nežádoucích událostí pomocí jednoduchých kritérií. Filtrování bývá většinou prvním krokem při zpracování proudu událostí.

Adaptace – vstupní události jsou transformovány do formátu vyžadovaného systémem pro zpracování událostí.

Upřednostnění – ohodnocení důležitosti přicházejících událostí a upřednostnění událostí s vyšší prioritou. Události s vysokou prioritou jsou takové, které je třeba okamžitě zpracovat.

Výpočet nad daty události – v tomto kroku se nad daty provádějí výpočty. CEP dovoluje použití jakéhokoliv algoritmu. Výsledky výpočtů jsou použity dalšími kroky zpracování události.

Detekce vzoru události – vstupní proud událostí je prohledáván na shodu s definovanými vzory. Vzor události je šablona, která může odpovídat množině událostí stejného nebo různých typů.

Detekce a zpracování výjimek – zpracování událostí může také vést k detekci chyby nebo anomálie, nazývanou výjimka.

Abstrakce vzoru události – pokud je detekován vzor událostí, může být žádoucí reakce na tuto skutečnost vytvoření nové události nebo více událostí. Tím může dojít ke zjednodušení dalšího zpracování vypuštěním nepotřebných dat nebo sumarizací důležitých detailů událostí. [1]

* 1. Esper

Esper od firmy Epsertech je sofistikovaný CEP systém, který integruje proudy událostí s analýzou událostí. Je nesmírně adaptivní na jakýkoliv typ aplikace a může být použit buď jako součást nějakého balíku software, nebo jako samostatný systém. Oba přístupy poskytují stejnou funkcionalitu.

V Esperu mohou být události a v nich obsažená data definovány. Událostí může být například obyčejná instance Java třídy. Tato instance potom může být vložena do proudu událostí. Proudy událostí jsou použity ke strukturování potenciálně obrovského počtu příchozích událostí a dotazy lze díky nim aplikovat více s důrazem na výsledek. Pro strukturování jsou použity filtry, které mohou události zahodit, anebo je přesunout do jiného proudu. Použitím těchto filtrů může architektura proudů usnadnit analýzu událostí. Také můžou být vytvořeny analyzační proudy, které vytvářejí další strukturu různou od struktury přicházejících a odcházejících událostí.

Události v proudech mohou být analyzovány pomocí dotazů. Definované dotazy hledají shodu s jejich vzorem. Jazyk dotazů je založen na jazyku SQL. Díky tomu je použití Esperu vcelku jednoduché pro osoby běžně zvyklé na práci s dotazy nad relačními databázemi. Přestože je jazyk dotazů založen na SQL, existuje v něm několik nových klíčových slov a funkcí. Ty jsou důležité kvůli proudové povaze dat v Esperu. Například bylo přidáno klíčové slovo pro jednotku času, díky kterému lze nastavit čas, po který se může vzor vyskytnout. Databázové dotazy nemusejí být časově omezeny díky tomu, že databázové systémy mohou mít obrovskou kapacitu. Proud je ale nekonečný a dotaz nad proudem, který není limitovaný časovým obdobím, musí brát v potaz všechny události. Takové dotazy se ale v CEP přístupu používají jen zřídka. Většinou se vzor musí objevit v proudu událostí během nějakého definovaného časového období. [4][9]

* 1. Verze Esperu

Pod pojmem Esper se rozumí základní balík funkcí obsahující kompletní CEP systém. Esper je vyvinut v jazyce Java pod licencí GNU GPL v2[[1]](#footnote-1). Existuje také ekvivalentní produkt NEsper[[2]](#footnote-2) pro platformu .NET.[10]

EsperIO je funkcionalita pro použití se systémem Esper s implementovanými vstupně výstupními funkcemi. Jedná se zejména o možnost připojení databáze, možnost práce s formáty s formáty CSV, XML nebo JSON. Dále také o Spring JMS[[3]](#footnote-3), AMQP[[4]](#footnote-4), HTTP a Socket adaptéry. Při použití tohoto dodatečného balíku je možné také kromě ostatních variant (2.5) reprezentovat události objekty typu Apache Axiom[[5]](#footnote-5).[12]

EsperHA (Esper High Availability) je balík speciálně upravený pro potřeby vysoké dostupnosti systému. EsperHA je také optimalizován pro vysoké rychlosti vzniku událostí a obrovské objemy dat. Jeho stav je průběžně zálohován, aby při případném výpadku mohlo dojít k obnovení předchozího stavu a nedošlo ke ztrátě informací. Jedná se o samostatnou aplikaci, pro jejíž spuštění je třeba zakoupení licenčního klíče.[11]

Nejvyšší verzí je EsperEE (Esper Enterprise Edition). Je to nejobsáhlejší verze ze všech uvedených. Jeho součástí je EsperHA. Navíc je zde ještě obsažena komponenta nazvaná EsperHQ[[6]](#footnote-6). Jedná se o vizualizaci a správu pomocí webového prohlížeče. Jako v případě EsperHA se jde o samostatnou aplikaci, na kterou je třeba zakoupit licenci.[13]

* 1. Fungování Esperu

Esper funguje jako klasická databáze převrácená naruby. Místo ukládání dat a spouštění dotazů nad nimi, Esper ukládá dotazy a data jimi procházejí. Události mají atributy, které nesou informace o události (2.4.1) a jsou sdružovány do proudů (). Je možné vytvářet EPL dotazy (2.4.3) pro získání událostí ze systému.

* + 1. Atributy událostí

Atributy uchovávají stav událostí. Atributy mohou být jednoduché, indexované, mapované nebo vnořené. Podle typu použitého atributu se také mění syntax dotazů.

|  |  |  |  |
| --- | --- | --- | --- |
| Typ události | Syntaxe | Příklad | Popis |
| Jednoduchá | name | sensorId | Atribut, který má jednu hodnotu. |
| Indexovaná | name[index] | sensor[0] | Indexovaný atribut obsahuje uspořádanou kolekci objektů (všechny stejného typu), které mohou být jednotlivě získány uvedením jejich indexu. |
| Mapovaná | name(‘key‘) | sensor(‘light‘) | Asociované pole je použito u mapovaných událostí. Objekty v kolekci musí být stejného typu. |
| Vnořená | name.nestedName | sensor.value | Vnořený atribut je atribut, který je atributem atributu události. |

1. Typy atributů událostí

Různé typy atributů je možné také kombinovat, což může být výhodné u událostí se složitější strukturou. Proto je validní i výraz *person.address(‘home‘).street[0].*

* + 1. Dynamické atributy událostí

Dynamické atributy jsou takové atributy, které nemusejí být známy v době kompilace dotazů, ale až při běhu programu. Pomocí speciální syntaxe je možné, aby dotazy obsahovaly i hodnoty těchto atributů.

|  |  |
| --- | --- |
| Typ události | Syntax |
| Dynamická jednoduchá | name? |
| Dynamická indexovaná | name[index]? |
| Dynamická mapovaná | name(‘key‘)? |
| Dynamická vnořená | name?.nestedPropertyName |

1. Dynamické typy atributů událostí

Výsledek dotazu na dynamické atributy je vždy typu *java.lang.Object*. Pokud dynamický atribut neexistuje, pak je vrácená hodnota *null.*

Ve spojení s dynamickými atributy je potom ještě možné využít funkce:

* *cast* – např. *cast(count, int) –* přiřazuje dynamickému atributu typ,
* *exists* – např. *exists(item.serviceName?) -* pokud dynamický atribut existuje, pak vrátí true, jinak false,
* *instanceof* – např*. instanceof(price?, double, float) -* zjistí zda typ atributu je některý z předaných typů,
* *typeof* – např. *typeof(timestamp?)* vrátí textovou reprezentaci typu atributu.
  + 1. Dotazování

Pro dotazování slouží v Esperu dotazovací jazyk na bázi SQL - EPL (Event Processing Language). EPL poskytuje základní funkčnost jako *SELECT, FROM, WHERE, GROUP BY* a *ORDER BY*. Místo tabulek se pracuje s proudy událostí a místo řádků ve výsledku jsou výsledkem samotné události. Protože události se skládají z dat, je možné v EPL využít také spojování proudů (JOIN), filtrování a agregaci.

Jednoduchý EPL dotaz vypadá například takto:

select avg(price) from StockTick.win:time(30 sec) where symbol='IBM'

Dotaz získává průměrnou změnu ceny akcií (*select avg(price) from StockTick*) společnosti IBM (*where symbol='IBM'*) za posledních 30 vteřin (*win:time(30 sec)*) (2.4.5).

Výsledky dotazů lze získávat trojím způsobem. Buď pomocí Listenerů, Subscriberu nebo vytvořením Iteratoru nad výsledky.

|  |  |  |
| --- | --- | --- |
| Jméno | Metody nad EPL dotazem | Popis |
| Listener | addListener, removeListener | Pro použití listeneru je třeba implementovat rozhraní UpdateListener nebo StatementAwareUpdateListener*.* Listenery přijímají EventBeany(co to je), obsahující výsledky dotazu. |
| Subscriber | setSubscriber | Subscriber je Java objekt s definovanou metodou update, která je zavolána v případě nového výsledku EPL dotazu.  K jednomu EPL dotazu může být přiřazen maximálně jeden Subscriber. Je ale možné použít Subscriber a k tomu ještě více Listenerů. |
| Iterator | safeIterator, iterator | Pokud aplikace nevyžaduje průběžné doručování nových výsledků EPL dotazů, je možné iniciovat získání dat z klientské strany. Zavoláním safeIterator nebo iterator dojde k získání všech dosavadních výsledků od posledního volání metod pro získání Iteratoru. |

1. Možnosti pro získání výsledků dotazů
   * 1. Proudy událostí

Zpracování událostí v Esperu je nepřetržité povahy. Události jsou proto sdružovány do proudů. Existují základní dva druhy proudů událostí. Jsou to vstupní proud a (Insert Stream) a výstupní proud (Remove Stream).

Vstupním proudem jsou chápány události, které vstupují do datových oken. Události ze vstupních proudů jsou odebírány Listenery jako nové události (první argument metody update z rozhraní UpdateListener).

Do výstupního proudu se dostanou ty události, které právě naopak vstupního datové okno opouštějí. V Listenerech jsou tyto události reprezentovány jako staré událostí (druhý argument metody update z rozhraní UpdateListener).

* + 1. Datová okna

Události v systému při běhu postupně přibývají. Proto jsou dotazy zřídkakdy prováděny nad všemi událostmi. Mnohem častěji je využito datových oken (Data Windows). Existují dvě základní limity na velikost datových oken a to jsou počet a čas.

Okna, jejichž velikost je závislá na počtu událostí (Length Windows) jednoduše vytvoří strukturu pevné dané délky pro uchovávání událostí. Od vzniku okna jsou do této struktury postupně přidávány události, dokud nedojde k jejímu zaplnění. Pokud je již vyčerpána celá kapacita okna, pak je nejdříve uložená událost v okně vyřazena z této struktury a místo ní do okna přibude nová událost. Vyřazená událost je vložena do výstupního proudu (2.4.4).

select \* from MyEvent.win:length(5)

Uvedený EPL dotaz využívá velikostní okno s nastavenou velikostí 5. Při nově příchozí události do zaplněného okna, je nejdříve vložená událost vyřazena z okna a nová událost vložena do okna.

Časová okna (Time Windows) fungují na podobném principu jako velikostní okna, jen neuchovávají předem daný počet událostí, ale uchovávají události vložené maximálně před definovanou dobou.

select \* from MyEvent.win:time(5 sec)

Události starší než 5 sekund jsou vyřazeny z okna a vloženy do výstupního proudu.

Jak časová, tak velikostní okna existují ještě v dávkových variantách (*time\_batch*, *length\_batch*). V tomto případě nejsou Listenery upozorňovány při každé nové události, ale dávkově po naplnění okna. Dále lze ještě v Esperu použít i kombinaci obojího, tedy dávkové okno limitované jak časem, tak velikostí (*time\_length\_batch*).

* 1. Reprezentace událostí

Událost v Esperu je neměnný záznam výskytu nějaké akce v minulosti. Atributy záznamu uchovávají informace o vlastnostech události. V Esperu mohou být události reprezentovány více způsoby, proto není potřeba vždy vytvářet pro každý typ události odpovídající třídu, která ji bude reprezentovat.

|  |  |
| --- | --- |
| Java třída | Popis |
| java.lang.Object | Jakýkoliv POJO[[7]](#footnote-7) objekt splňující JavaBean podmínky. |
| java.util.Map | Implementace rozhraní java.util.Map. Klíče mapy jsou názvy atributů události, ke kterým jsou přiřazeny hodnoty. |
| java.lang.Object[] (pole objektů) | Události reprezentované polem objektů obsahují pouze hodnoty vlastností události. |
| org.w3c.dom.Node | Model XML dokumentu (DOM) |
| org.apache.axiom.om.OMDocument nebo OMElement | XML API pro proudové zpracování (StAX) – Apache Axiom (poskytuje verze EsperIO) (2.3). |
| Aplikační třídy | Plug-in reprezentace přes rozšířené rozhraní |

1. Možné reprezentace událostí v Esperu

Všechny možné reprezentace mají tyto společné vlastnosti:

* podporují možnost vnořených, indexovaných a mapovaných atributů (2.4.1), na počet vnoření není žádný limit,
* poskytují metadata typu události (včetně vnořených událostí),
* umožňují převedení událostí, jejich atributů, nebo vnořených událostí do nových událostí. Jedinou výjimkou je reprezentace pomocí Apache Axiom, která neumožňuje převádění atributů události, ale jen celé události,
* Object, Map a Object[] dovolují použití nadtypů v dotazech.

Chování Esper systému se neliší vzhledem ke zvolené reprezentaci (až na uvedené výjimky). Díky tomu má možnost více reprezentací tyto výhody:

* pokud aplikace již obsahuje nějakou reprezentaci událostí, pak není třeba události přetvářet do Java tříd,
* reprezentace jsou zaměnitelné, a proto při přechodu na jinou reprezentaci není třeba měnit EPL dotazy,
* v jednom dotazu je možno pracovat s více reprezentacemi – není třeba mít zvolenu pouze jednu reprezentaci.

Díky volbě je možné vědomě volit mezi jednoduchostí použití a výkonností podle povahy a událostí a náročnosti zpracování.

* + 1. POJO události

POJO události jsou instance objektů, které vystavují svoje atributy pomocí JavaBeans getterů. Esper podporuje pro reprezentaci událostí také třídy, které implementují rozhraní nebo rozšiřují nadtřídu. Potom i v EPL dotazech lze použít některé z implementovaných rozhraní nebo nadtřídu třídy události.

Třídy, které reprezentují události, by správně měly být neměnitelné. Vyplývá to z logiky věci, protože události jsou záznamy o výskytu nějaké akce v minulosti a minulost již změnit nelze. Nicméně pro Esper toto není kritický požadavek a systém dokáže pracovat s událostmi, které jsou měněny, jakkoliv to porušuje zásady CEP.

Metody *equals* a *hashCode* nemusí být v třídách implementovány a ani implementace těchto metod nijak nezmění chování systému Esper.

Atributy POJO událostí mohou být všech typů (jednoduché, indexované, mapované, vnořené).

Požadavky na metody u POJO událostí budou popsány pomocí příkladu třídy *NewEmployeeEvent*.

public class NewEmployeeEvent {

public String getFirstName();

public Address getAddress(String type);

public Employee getSubordinate(int index);

public Employee[] getAllSubordinates();

public Department getDepartment();

}

Pro jednoduché atributy je třeba getter metoda vracející hodnotu atributu. Metoda *getFirstName* tedy vrací hodnotu atributu *firstName* typu *String.*

Indexované atributy vyžadují buď metodu s argumentem typu int, vracející hodnotu atributu na zadaném indexu jako *getSubordinate*, nebo metodu bez argumentu vracející třídu implementující rozhraní *java.lang.Iterable* nebo pole obsahující všechny indexované hodnoty, jakou je metoda *getAllSubordinates*.

Pro mapované atributy je třeba metoda, která má argument typu *java.lang.String*, která vrací hodnotu atributu. Příkladem pro tento případ je metoda *getAddress*.

Aby se bylo možné dotazovat vnořené atributy, je třeba pro tyto atributy mít gettery. Příkladem je nejen metoda *getDepartment*, ale i metody *getAddress* a *getSubordinate*, které také vracejí vnořený objekt.

V dotazech záleží na velikosti písmen. Jména atributů a jejich gettery tedy musí splňovat Java standardy, kdy jméno getteru je jednoznačně odvozeno podle názvu atributu třídy.

|  |  |  |
| --- | --- | --- |
| Metoda | Jméno atributu | Příklad |
| getPrice() | price | select price from MyEvent |
| getNAME() | NAME | select NAME from MyEvent |
| getItemDesc() | itemDesc | select itemDesc from MyEvent |
| getQ() | q | select q from MyEvent |
| getQN() | QN | select QN from MyEvent |
| getqn() | qn | select qn from MyEvent |
| gets() | s | select s from MyEvent |

1. Jména metod a atributů podle Java standardu
   * 1. Události java.util.Map

Události mohou být také reprezentovány pomocí objektů implementujících rozhraní *java.util.Map*. Hodnoty atributů událostí jsou dostupné pomocí metody *get*, která má argument, jímž je název atributu.

Atributy Map událostí mohou být stejně jako v případě POJO událostí jednoduché, indexované, mapované nebo vnořené. Typy Map událostí mohou být registrovány buď staticky pomocí XML konfiguračního souboru, nebo dynamicky pomocí konfiguračního API.

Pokud má Map událost nadtřídy, potom jsou události přístupné veškeré atributy jejich nadtříd. V běžící aplikaci je možné přidávat do Map událostí další atributy. Nelze ale měnit typy již existujících atributů nebo hodnoty atributů upravovat. Pro upravení atributu Map události, je třeba celý typ odebrat a poté je možné pod stejným jménem přidat upravený typ s upravenou hodnotou. Potom co je typ Map události přidán do systému, může být také atributem jiné události typu Map nebo pole objektů.

Na příkladu je definována událost *AccountUpdate*, jejíž nadtřídou je událost *BaseUpdate* (je již definována někdy dříve).

Map<String, Object> accountUpdateDef =

new HashMap<String, Object>();

accountUpdateDef.put("accountId", long.class);

accountUpdateDef.put("name", String.class);

epService.getEPAdministrator().getConfiguration().

addEventType("AccountUpdate", accountUpdateDef,

new String[] {"BaseUpdate"});

Při takové hierarchii typů potom výsledkem dotazu *select \* from BaseUpdate* budou všechny události typu *BaseUpdate* i typu *AccountUpdate*. Pokud by byl stejně definován ještě typ *BankAccountUpdate*, jehož nadtypem by byl typ *AccountUpdate*, pak by ve výsledku tohoto dotazu byly i události tohoto typu.

* + 1. Události reprezentované polem objektů

Další možností reprezentace událostí jsou pole objektů. V takovém poli potom jsou pouze hodnoty atributů události. Stejně jako Map události, události reprezentované polem objektů eliminují nutnost použití Java tříd k reprezentaci událostí a umožňují jednodušší změny atributů událostí během běhu systému.

Dědičnost typů u událostí reprezentovaných pomocí pole objektů funguje stejně jako u Map událostí (2.5.2). Atributy těchto událostí mohou být jakéhokoliv typu (2.5).

Protože pole objektů uchovává pouze hodnoty atributů události, je třeba při registraci typu události další pole *String* objektů, obsahující názvy atributů. Typická registrace nového typu pak vypadá takto:

String[] propertyNames = {"name", "addressLine1", "history"};

Object[] propertyTypes =

{String.class, String.class, UpdateHistory.class};

epService.getEPAdministrator().getConfiguration().

addEventType("UpdatedFieldType", propertyNames, propertyTypes);

* + 1. XML události org.w3c.dom.Node

Události reprezentované třídou *org.w3c.dom.Node* podporují stejně jako ostatní typy reprezentací jednoduché, indexované, mapované a vnořené atributy ().

Při této reprezentaci není nutné uvádět XSD schéma dokumentu. Pak ale návratové hodnoty všech atributů událostí jsou typu *java.lang.String*. Pokud je XSD schéma součástí konfigurace, potom už typy atributů událostí jsou validovány vůči tomuto schématu a jsou typu uvedeného ve schématu.

Výhodou reprezentace událostí pomocí XML je možnost využití XPath výrazů v dotazech.

Tvorba XPath výrazů bude popsána pomocí příkladu:

<?xml version="1.0" encoding="UTF-8"?>

<Sensor xmlns="SensorSchema">

<ID>urn:epc:1:4.16.36</ID>

<Observation Command="READ\_PALLET\_TAGS\_ONLY">

<ID>00000001</ID>

<Tag>

<ID>urn:epc:1:2.24.400</ID>

</Tag>

<Tag>

<ID>urn:epc:1:2.24.401</ID>

</Tag>

</Observation>

</Sensor>

EPL dotaz na takovou událost může mít standardní SQL podobnou syntaxi, která je v této práci již několikrát použita nebo může být využito XPath výrazů.

|  |  |
| --- | --- |
| Dotaz na atributy | Ekvivalentní XPath výraz |
| Observation.ID | /Sensor/Observation/ID |
| Observation.Command | /Sensor/Observation/Command |
| Observation.Tag[0].ID | /Sensor/Observation/Tag[position = 0]/ID |

1. Dotaz na atributy a ekvivalentní XPath výrazy
   1. Sběr výkonnostních statistik v Esperu

Sběr výkonnostních statistik je schopnost, která dovolí Esper aplikaci přijímat detailní informace o vytížení systémem Esper a nebo jednotlivými dotazy. Tento sběr je ve výchozím nastavení vypnut. Tyto hodnoty – metriky jsou reprezentovány pomocí událostí stejných jako aplikační události. Proto je lze získat pomocí EPL dotazu, jako jakýmkoliv jiné události.

* + 1. Systémové metriky

Systémové metriky lze získat pomocí EPL dotazu:

select \* from com.espertech.esper.client.metric.EngineMetric

Získáme tak EngineMetric událost, jejíž atributy poskytují souhrnné informace o událostech v systému.

|  |  |
| --- | --- |
| Jméno | Popis |
| *engineURI* | URI instance Esper systému. V jednom JVM může běžet více instancí Esper systému, ovšem URI každé instance je vždy jedinečné. |
| *Timestamp* | Současný čas Esper systému. Čas v systému Esper je ve verzi 5.1.0 udržován s přesností na milisekundy a získává se pomocí *System.getCurrentTimeMillis().* |
| *inputCount* | Souhrnný počet vstupních událostí vzniklých od startu Esper systému. Vstupní události jsou definovány jako události zaslané aplikačními vlákny nebo insert into události. |
| *inputCountDelta* | Počet vstupních událostí vzniklých od posledního měření. |
| *scheduleDepth* | Počet nevykonaných plánů. EPL dotazy mohou být naplánovány v určitý čas pomocí klíčového slova *timer:schedule*. |

1. Atributy systémových metrik a jejich význam
   * 1. Metriky dotazů

Metriky dotazů obsahují informace o náročnosti zpracování jednotlivých EPL dotazů. Získat je lze pomocí:

select \* from com.espertech.esper.client.metric.StatementMetric

Pro každý registrovaný dotaz pak systém zvlášť zjišťuje metriky uvedené v následující tabulce.

|  |  |
| --- | --- |
| Jméno | Popis |
| *engineURI* | URI instance Esper systému. |
| *timestamp* | Čas, kdy byla metrika naměřena. |
| *statementName* | Jméno měřeného dotazu. Pokud není jméno definováno, je použito vygenerované jméno Esper systémem. |
| *cpuTime* | Procesorový čas spotřebovaný na vykonání dotazu v nanosekundách. |
| *wallTime* | Celkový čas spotřebovaný na vykonání dotazu v nanosekundách. |
| *numInput* | Počet vstupních událostí zpracovaných dotazem. |
| *numOutputIStream* | Počet událostí vložených do vstupního proudu (2.4.4) |
| *numOutputRStream* | počet událostí vložených do výstupního proudu (2.4.4) |

1. Atributy metrik dotazů a jejich význam

Absolutní hodnoty (cpuTime, wallTime, numInput, numOutputIStream, numOutputRStream) jsou vždy vzhledem poslednímu reportování metrik.

* + 1. Sběr metrik

Sběr metrik je ve výchozím nastavení Esperu vypnutý. Existují dva způsoby, jak konfigurovat sběr metrik. První možností je přidat konfigurační informace do konfiguračního souboru Esperu *esper.cfg.xml*.

<engine-settings>

<defaults>

<metrics-reporting engine-interval="5000"

statement-interval="5000" enabled="true" threading="true"/>

</defaults>

</engine-settings>

K nastavení sběru metrik slouží element *metrics-reporting*. Atributem *enabled* je reportování zapnuto. Pomocí atributů *engine-interval* a *statement-interval* se nastavuje frekvence reportování metrik v milisekundách. Ve výchozím nastavení je to 10 vteřin. Atribut *threading* je nastaven na *true* a Esper tak bude reportování zpracovávat ve vyhrazeném vlákně.

Druhou možností je nastavit tyto hodnoty programově v kódu programu. Pokud chceme měnit konfiguraci Esper serveru za běhu je třeba, aby konfigurace měla nějakou objektovou podobu. Konfiguraci lze získat zavoláním konstruktoru *com.espertech.esper.client.Configuration*. Z takto vytvořeného objektu potom lze zavolat metody *getEngineDefaults().getMetricsReporting()*, což vrátí instanci *com.espertech.esper.client.ConfigurationMetricsReporting*. Tato třída potom obsahuje metody *setEnableMetricsReporting*, *setEngineInterval*, *setStatementInterval* a *setThreading*, pomocí kterých lze nastavit stejné vlastnosti, jako pomocí XML konfiguračního souboru.

Typickým způsobem vytvoření událostí v Esper systému je *epService.getEPRuntime().sendEvent(event),* kde *epService* je instancí *com.espertech.esper.client.EPServiceProvider*. Při každém zavolání metody *sendEvent* Esper přijme událost a zpracuje ji. Součástí tohoto zpracování je právě i spuštění funkcionality (pokud je zapnuta), která zajišťuje vytváření *StatementMetrics* a *EngineMetrics* událostí, na které je potom možno se dotazovat. [9]

1. Monitoring

Aplikace v reálném světě zpracovávají velká množství informací. S tím souvisí mnohdy také vysoké nároky na výpočetní výkon zařízení, na kterých jsou provozovány. Aby tyto zdroje mohly být využívány efektivně, je třeba běh aplikací monitorovat. Pomocí monitoringu pak lze nalézt neefektivní části systémů nebo detekovat úzká místa v systémech.

* 1. Monitoring Java aplikací

Java aplikace mohou být jednoduše monitorovány pomocí vestavěné funkcionality. Jedná se to tzv. MBeans a MXBeans, které odpovídají JMX specifikaci (3.2). Tyto objekty jsou vystaveny na MBean nebo MXBean serverech. Lze je tak pomocí klientských aplikací číst a dále zpracovávat.

Pokud je v například Oracle JDK spuštěn Java program, pak JVM vystavuje MBeany a MXBeany z těchto balíčků:

* *JMImplementation* - Obsahuje informace o použitém JMX protokolu, jméno JMX serveru, výrobce použitého JDK, atd.
* *com.sun.management* - Z tohoto balíku jsou vystaveny dvě MBeany. První je *com.sun.management.DiagnosticCommandMBean*. Tato MBean obsahuje metody k ovládání Garbage Collectoru[[8]](#footnote-8), Java Flight Recorderu[[9]](#footnote-9) a další informace týkající JVM. Druhou MBean je *com.sun.management.HotSpotDiagnostic*, díky které je možné vytvořit výpis paměti (Heap Dump) nebo získat nebo upravit parametry JVM[[10]](#footnote-10).
* *java.lang* - Ve výchozím nastavení je *java.lang* zdaleka největší vystavený balík. Obsahuje MBeany k získání informací o nastavení GC, spotřeby operační paměti procesem, dostupné paměti, vytížení procesoru, běžících vláken a například načtených tříd do operační paměti.
* *java.nio* - Balík *java.nio* definuje vyrovnávací paměti. Na JMX rozhraní jsou vystaveny informace o jednotlivých skupinách vyrovnávací pamětí jako je využitá paměť, celková kapacita vyrovnávacích pamětí nebo počet vyrovnávacích pamětí ve skupině.
* *java.util.logging* - Obsahuje informace o logování JVM[[11]](#footnote-11). Těmito informacemi jsou jména vytvořených loggerů, jejich rodičovských loggerů a úroveň logování jednotlivých loggerů, které lze i nastavovat.

Jednou z klientských aplikací, která může číst MBeany a MXBeany je společně s Oracle JDK dodávaný program JConsole. JConsole lze spustit z *JDK\_HOME/bin* adresáře, kde *JDK\_HOME* je adresář, kde je nainstalován JDK. Pomocí JConsole lze monitorovat běh jak lokálního procesu, tak i procesu na vzdáleném stroji. Program získává data pouze z vystavených MBean a MXBean, které lze číst nebo spouštět jejich metody. Ze získaných informací, které se zde nacházejí, také graficky zobrazuje v čase využitou operační paměť, počet vláken, počet načtených tříd a vytížení procesoru procesem.

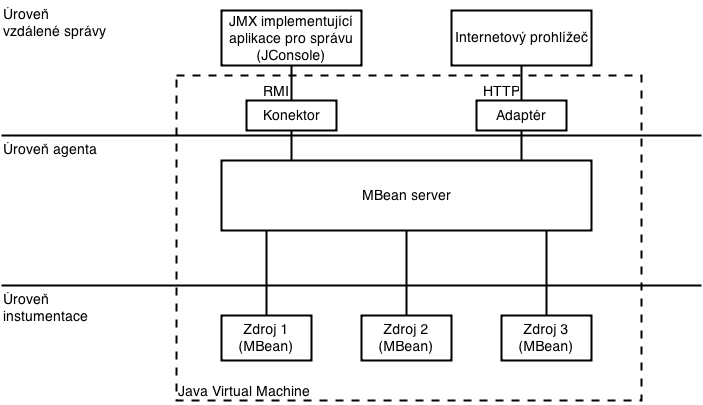
Novější možností pro monitoring JDK je aplikace jvisualvm. Je součástí Oracle JDK od Javy verze 6 aktualizace 7. Je založená na open-source projektu VisualVM[[12]](#footnote-12). Má veškerou funkcionalitu jako JConsole (čtení MBean a MXBean) a navíc ještě využívá dalších monitorovacích technologií jako Jvmstat[[13]](#footnote-13), Attach API[[14]](#footnote-14) a SA (Serviceability Agent)[[15]](#footnote-15). Díky tomu lze za běhu externě profilovat proces v JVM, vytvářet obrazy paměti a sledovat činnost jednotlivých metod. [14]

* 1. JMX standard

JMX (Java Management Extensions) technologie je standardní součástí platformy Java. JMX poskytuje jednoduchý způsob jak spravovat zdroje v aplikacích. JMX specifikace definuje architekturu, návrhové vzory, aplikační rozhraní a služby v programovacím jazyku Java pro monitorování a správu aplikací a sítí. Při použití JMX technologie je daný zdroj spojen jedním nebo více objekty známých jako MBean (Managed Bean) (3.2.2). Tyto MBeany jsou registrovány na speciálním serveru nazývaném MBean server (3.2.3).

* + 1. Architektura

Architektura JMX sestává ze tří základních vrstev. Jsou to úroveň instrumentace, úroveň agenta a úroveň vzdálené správy.



1. JMX architektura [15]

Na úrovni instrumentace jsou zdroje zastupovány MBeany. Díky instrumentaci může být zdroj spravován z vnějšku aplikace přes úroveň agenta. MBeany nevyžadují znalost JMX agenta kterým jsou zpracovávány. V úrovni instrumentace je také definován mechanismus upozorňování na změnu stavu MBeany (Notifikace) (3.2.2).

Úroveň agenta poskytuje specifikaci pro implementaci agentů. Agenti přímo ovládají zdroje a poskytují je aplikacím vzdálené správy. Agenti jsou většinou umístěni na stejném stroji jako zdroje, které kontrolují, ale nutně tomu tak být nemusí. Hlavní částí JMX agenta je MBean server, kde jsou registrovány MBeany. K implementaci agenta není třeba znát význam zdrojů, které bude spravovat. Agent dokonce ani nepotřebuje předem znát, které MBeany bude poskytovat. Stačí, když budou MBeany splňovat specifikaci.

Standardní způsob jak se připojit k JMX agentovi je přes JMX rozhraní založeném na technologii Java Remote Method Invocation (Java RMI)[[16]](#footnote-16). MBeany ale mohou být přistupovány i pomocí jiných protokolů jako Simple Network Management Protocol (SNMP)[[17]](#footnote-17) nebo Hypertext Transfer Protocol (HTTP).[14][15]

* + 1. Managed Bean (MBean)

Managed Bean (MBean) je Java objekt který implementuje konkrétní rozhraní a splňuje jisté návrhové požadavky. Toto rozhraní je určeno:

* atributy, ke kterým je možno přistoupit,
* operacemi, které je možné volat,
* oznámeními (Notifikacemi), které může vysílat,
* konstruktory MBean třídy.

MBeany zapouzdřují atributy a operace pomocí jejich veřejných metod, které jsou vystaveny pro aplikace pro správu. MBeany mohou podporovat oznámení při změně jejich stavu, díky kterým se MBean server postará o synchronizaci stavu MBean na připojených klientech. K tomu stačí aby MBean implementovala rozhraní *javax.management.NotificationBroadcaster.* Není však třeba vždy vytvářet vlastní implementaci a stačí být podtřídou třídy *javax.management.NotificationBroadcasterSupport*, která již rozhraní implementuje. Při případné změně stavu MBean pak stačí vytvořit instanci *javax.management.Notification* a odeslat ji metodou *sendNotification()* ze třídy *NotificationBroadcasterSupport*. [15]

* + 1. MBean server

MBean server je registrační místo pro MBeany, které mají být vystaveny pomocí JMX. Každý objekt, který je zde zaregistrovaný, se stává viditelným pro klientské aplikace. MBean server ale nikdy nevystavuje celé instance MBean, ale pouze jejich definované rozhraní.

import java.lang.management.ManagementFactory;

import javax.management.MBeanServer;

import javax.management.ObjectName;

…

MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();

ObjectName mbeanName = new ObjectName("com.example:type=Hello");

Hello mbean = new Hello();

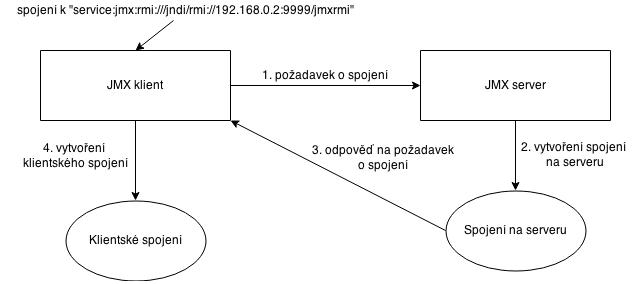
mbs.registerMBean(mbean, mbeanName);

Při registraci MBeany je jí třeba přiřadit unikátní jméno objektu. Klientské aplikace potom pro identifikaci objektů používají toto jméno. MBean server lze staticky získat z *java.lang.management.ManagementFactory.* Pak už stačí jen vytvořenou MBean na tomto serveru zaregistrovat a ta je tak již vystavena pomocí JMX pro klientské aplikace.

* + 1. JMX klient

Pro připojení k MBean serveru je třeba vytvořit klientskou část, která toto připojení bude obsluhovat.

JMX klient iniciuje připojení zasláním požadavku o spojení JMX serveru. JMX server vytvoří svoji část spojení a zašle odpověď s popisem tohoto přípojného bodu klientovi. Klient pak již jen vytvoří spojení na své straně.



1. Vytvoření spojení JMX klienta a serveru [15]

Prakticky je k navázání spojení třeba vytvořit instanci třídy *javax.management.remote.JMXServiceURL*. Poté stačí pomocí tohoto vytvořeného URL zavolat metodu *JMXConnectroFacotry.connect(JMXServiceURL serviceURL, Map<String,?> environment)*, která vyšle požadavek o spojení a vytvoří jej na straně klienta (). Z klientské části spojení pak lze získat objekt implementující *javax.management.MBeanServerConnection*, díky kterému lze potom jednoduše pracovat s MBean serverem*.*

JMXServiceURL url =

new JMXServiceURL(

"service:jmx:rmi:///jndi/rmi://192.168.0.2:9999/jmxrmi"

);

JMXConnector jmxc = JMXConnectorFactory.connect(url, null);

MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

K MBean vystaveným na JMX serveru lze přistupovat přes spojení na MBean server vytvořením MBean proxy. MBean proxy je lokální objekt na JMX klientu, který emuluje vzdálenou MBean. Vytvořením takové MBean proxy lze snadno manipulovat se vzdálenou MBean v lokálním kontextu. Pomocí getterů lze získat stav MBeany a pomocí volání ostatních metod lze tyto metody vyvolat na MBean serveru. MBean proxy se získá pomocí statické metody *JMX.newMBeanProxy*, jejíž argumenty jsou připojení na MBean server (instance *MBeanServerConnection*), jméno MBean zabalené do instance *javax.management.ObjectName*, třída rozhraní MBean a příznak nastavující, zda bude vrácená proxy implementovat rozhraní *javax.management.NotificationEmitter.*

ObjectName mbeanName = new ObjectName("com.example:type=Hello");

HelloMBean mbeanProxy = JMX.newMBeanProxy(

mbsc, mbeanName, HelloMBean.class, true

);

Pokud je na klientské straně třeba získat stav MBean, pak lze tedy zjistit z MBean proxy. MBean proxy ale nemusí být synchronizovaná, a tak může obsahovat neaktuální hodnoty. Aktuální hodnoty lze získat vytvořením nové instance MBean proxy, což ale nemusí být vždy efektivní.

Jednou z metod třídy *MBeanServerConnection* je metoda *addNotificationListener.* Pomocí této metody lze registrovat listener implementující rozhraní *javax.management.NotificationListener*, který bude odebírat notifikace o změně stavu MBean na serveru a v implementaci metody *handleNotification*, vhodně reagovat například aktualizací atributů MBean proxy. Tak je zajištěno, že MBean proxy objekt je vždy synchronizován s jeho předlohou na serveru. [15]

* 1. Vizualizace monitorovaných dat

Vizualizace je definována jako sdělování informací pomocí grafického znázornění. Ještě před vznikem písma bylo pro sdělování informací využíváno obrázků. Jednoduchý obrázek totiž dokáže člověku sdělit informace stejně dobře, jako psaný text. I když by ale v obou případech obrázek i text obsahovaly stejnou informaci, obrázky jsou pro člověka více pochopitelné díky tomu, že jsou zpracovávány paralelně na rozdíl od textu, u kterého je povaha procesu čtení sekvenční. [16]

* 1. Příbuzné práce

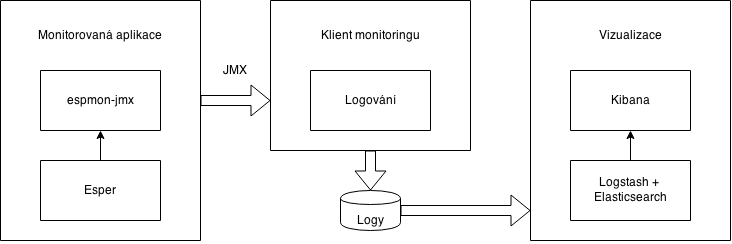
Sem napsat něco o všech těch jak free tak placených aplikací na monitoring Java aplikací. Nebo tuto sekci zrušit?

1. Monitoring Esper serveru

Cílem této diplomové práce je vytvoření aplikace, která bude sbírat výkonnostní metriky Esper serveru (2.6) a zveřejňovat je pomocí JMX (). Zároveň je třeba tato data odebírat a vhodně je graficky zobrazit ().

* 1. Aplikace pro monitoring Esper systému

Celá aplikace sestává z několika dílčích částí (modulů), z nichž každá část má svůj úkol a dohromady je tak zaručeno vše od získání metrik Esper systému, až po logování těchto dat. Všechny moduly jsou stejně jako Esper samotný napsány v jazyce Java s použitím nástroje Apache Maven[[18]](#footnote-18). Takto lze mít jednotlivé moduly součástí jednoho projektu a automaticky sestavovat jeho části podle potřeby.



1. Schéma aplikace pro monitoring Esper systému

Prvním modulem je modul espmon-jmx. Tento modul obstarává získávání systémových metrik a metrik dotazů ze systému Esper a poté je pomocí modulu Mbean serveru vystavuje na JMX rozhraní. Tyto MBeany jsou definovány v modulu espmon-mbean.

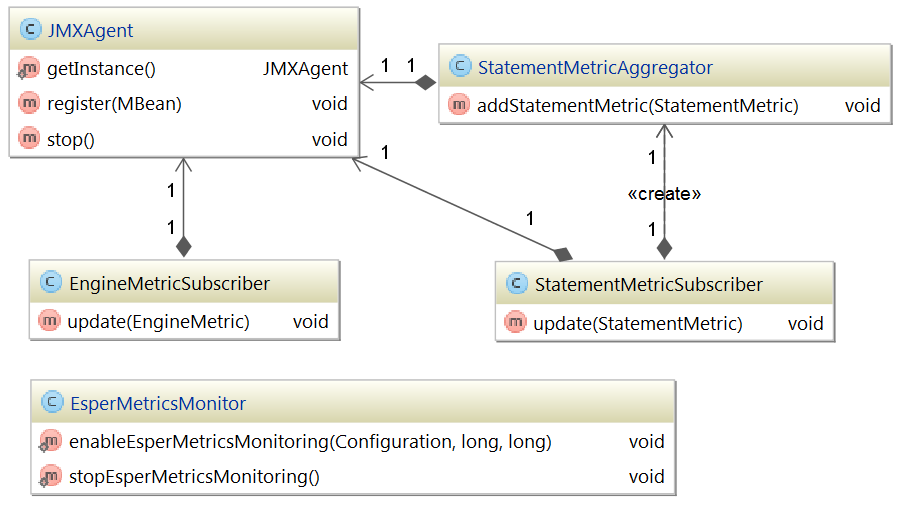
Modul espmon-client v sobě obsahuje JMX klienta, díky čemuž se může připojit JMX server a číst na něm registrované MBeany. MBeany jsou na serveru pravidelně aktualizovány a s každou takovou aktualizací klient loguje nový stav MBeany.

Takto pořízená data nejsou pro člověka dobře čitelná a proto je třeba je vhodně graficky interpretovat. Proto jsou tyto logy čteny systémem Logstash, díky kterému je potom možné je předat fulltextovému vyhledávači Elasticsearch. V tomto systému je jako plugin nasazen vizualizační a analyzační platforma Kibana, která je pro práci vyhledávačem Elasticsearch přímo navržena.

* 1. JMX server pro Esper

JMX server pro Esper je obsažen v modulu espmon-jmx. Tento modul obstarává sběr výkonnostních statistik Esper systému a vystavuje je pomocí Mbean serveru na JMX rozhraní.

Tento modul se skládá ze tří základních částí. První částí je třída *EsperMetricsMonitor*, díky které se dá z klientského kódu monitoring Esperu zapnout a vypnout. Další částí jsou Třídy *EngineMetricSubscriber* a *StatementMetricSubscriber*, jež jsou systémem Esper upozorňovány na změnu metrik. Poslední částí je třída *JMXAgent*, která má na starost publikování MBean pomocí MBean serveru.



1. Diagram tříd modulu espmon-jmx

JMX server pro Esper je samostatný Maven projekt a proto jej lze snadno použít. Pro použití stačí přidat do projektu sledovaného Esper systému přidat závislost právě na balík espmon-jmx nebo zabalený jar archív přidat na classpath.

V monitorované aplikace pak stačí funkcionalitu espmon-jmx zapnout. To se provede přidáním jediného řádku do již existujícího systému.

EsperMetricsMonitor.enableEsperMetricsMonitoring(conf, 5000, 5000);

Tím dojde k nastavení Esperu tak, aby začal sbírat výkonnostní metriky. Prvním argumentem metody *enableEsperMetricsMonitoring* je instance třídy *com.espertech.client.Configuration*, což je třída reprezentující konfiguraci v celém systému Esper. Další dva argumenty udávají intervaly vytváření systémových metrik a metrik dotazů. Oba dva argumenty jsou čísla typu *long*, a jejich přesnost je v milisekundách.

Potom je také třeba se na tyto metriky začít dotazovat, a proto je po správném nastavení systému Esper ještě třeba vytvořit příslušné EPL dotazy (2.6) a přiřadit jim subscribery. Po přiřazení tříd *EngineMetricsSubscriber* a *StatementMetricsSubscriber* k EPL dotazům je úvodní konfigurace dokončena.

Systém Esper pak periodicky (podle zvoleného intervalu), zveřejňuje výkonnostní metriky a metriky dotazů. Při takovém zveřejnění jsou automaticky spuštěny metody update ze subscriberů. Zpracování takové události v subscriberech spočívá zejména v převedení předané instance *com.espertech.esper.client.metric.EngineMetric* na MBean třídu *cz.muni.fi.esper.EngineMetric*. Tato MBean instance je tak předána JMX agentu pro registraci na MBean serveru zavoláním *jmxAgent.register(new EngineMetric(em)).*

Stejně jako v případě systémových metrik je tomu i v případě metrik dotazů, které systém Esper zveřejňuje pro každý vytvořený EPL dotaz. Zde ale ještě dochází k předání metriky dotazu třídě *StatementMetricAggregator.* Tato třída využívá toho, že metriky dotazů z jednoho intervalu mají shodný atribut *timestamp.* Díky tomu lze metriky z jednoho časového období agregovat a získat tak celkové vytížení systému Esper všemi EPL dotazy v systému. Tato souhrnná data jsou potom stejně jako data jednotlivých dotazů registrována na v třídě *JMXAgent* pod názvem *TotalStatementMetrics.*

Třída *JMXAgent* je navržena podle návrhového vzoru Singleton[[19]](#footnote-19). Při prvním zavolání statické metody *getInstance* dojde k inicializaci interní instance této třídy. Tato inicializace spočívá ve vytvoření MBean serveru a instance třídy *javax.management.remote.JMXConnectorServer*, ve kterém MBean server poběží. Důležitou metodou v této třídě je již zmíněná metoda *register*, jejíž pomocí na MBean serveru registrují vystavené MBeany. Pokud je již MBean nějakého typu vystavena, pak dojde pouze k její aktualizaci a vytvoření upozornění o změně stavu, na které poté mohou připojení klienti reagovat.

* 1. JMX klient

Popis klienta, obdobně jako server.

* 1. Vizualizace

Popis, jak jsou využity a nakonfigurovány technologie logstash, elasticsearch a kibana. + předvedení GUI a popis toho, co tam je a co to znamená.

1. Testování aplikace
   1. Test výkonnosti při různých reprezentacích událostí
      1. POJO
      2. Map
      3. Pole objektů
      4. XML
   2. Test výkonnosti při různých způsobech odběru výsledků dotazu
      1. Subscriber
      2. Listener
      3. Iterator

Iterator jen možná, protože tam mi esper neservíruje data, ale říkám si o ně sám, tak zatím nevím, jestli to bude porovnatelné.

1. Budoucí práce

Co ještě by se dalo vylepšit?

Možnost managementu klientů ze serveru (nastavení intervalu měření). Možnost opakovaného čtení konfiguračního souboru klienta a tak změna připojených serverů za běhu. Jednodušší nastavení vzhledu klienta. Asi tuto kapitolu vypustit, a dát tyto věci rovnou do závěru.

1. Závěr

Rekapitulace textu práce, zhodnocení splnění zadání, uvedení přínosu práce, uvést výhodu použití elasticsearch a tím tak univerzálnosti řešení a také ukládání dat pro pozdější potřebu. Uvést závěry z výsledků měření.

1. Seznam obrázků

Obr. 1 Schéma získávání odpovídajících událostí z proudu událostí [4] 9

Obr. 2 JMX architektura [15] 27

Obr. 3 Vytvoření spojení JMX klienta a serveru [15] 29

Obr. 4 Schéma aplikace pro monitoring Esper systému 32

Obr. 5 Diagram tříd modulu espmon-jmx 33

1. Seznam tabulek

[Tab. 1 Typy atributů událostí 13](#_Toc418704751)

[Tab. 2 Dynamické typy atributů událostí 14](#_Toc418704752)

[Tab. 3 Možnosti pro získání výsledků dotazů 15](#_Toc418704753)

[Tab. 4 Možné reprezentace událostí v Esperu 17](#_Toc418704754)

[Tab. 5 Jména metod a atributů podle Java standardu 19](#_Toc418704755)

[Tab. 6 Dotaz na atributy a ekvivalentní XPath výrazy 21](#_Toc418704756)

[Tab. 7 Atributy systémových metrik a jejich význam 22](#_Toc418704757)

[Tab. 8 Atributy metrik dotazů a jejich význam 23](#_Toc418704758)

1. Literatura

[1] LUCKHAM, David C. *Event processing for business: organizing the real-time enterprise.* Hoboken: NJ: Wiley, 2012. ISBN 978-0-470-53485-4.

[2] GOETZ, M. *Integration of Business Process Management and Complex Event Processing.* [online]. [cit. 11. března 2015]. Dostupné na: <http://www.itransparent.de/sites/default/files/ActiveVOS\_BPM\_Esper\_CEP\_Paper.pdf>

[3] LUCKHAM, David C. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Boston: Addison-Wesley, 2002.

[4] LAZO, D. *Osworkflow : A Guide For Java Developers And Architects to Integrating Open-Source Business Process Management*. Birmingham, UK: Packt Pub, 2007, str. 135-161 (Chapter 7: Complex Event Processing). ISBN 9781847191526.

[5] BUCHMANN, A., KOLDEHOFE, B. *Complex Event Processing*. [online]. [cit. 10. března 2015]. Dostupné na: <http://nirvana.informatik.uni-halle.de/~molitor/webcms/pdf/it0905.pdf>

[8] ETZION, O., NIBLETT, P. *Event Processing in Action.* Stamford, USA: Manning Publications Co., 2011. ISBN 9781935182214.

[9] Esper Reference [online]. [cit. 19.3.2015]. Dostupné na:

<http://www.espertech.com/esper/release-5.2.0/esper-reference/html/index.html>

[10] About Esper [online]. [cit. 12. března 2015]. Dostupné na: <http://www.espertech.com/esper/>

[11] Esper HA [online]. [cit. 12. března 2015]. Dostupné na: <http://www.espertech.com/products/esperha.php>

[12] EsperIO Refference [online]. [cit. 12. března 2015]. Dostupné na: <http://www.espertech.com/esper/release-5.2.0/esperio-reference/pdf/esper\_adapter\_reference.pdf>

[13] Esper Enterprise Edition [online]. [cit. 12. března 2015]. Dostupné na: <http://www.espertech.com/products/esperee.php>

[14] Monitoring and Management of the Java Virtual Machine [online]. [cit. 21. března 2015]. Dosutpné na:

<https://docs.oracle.com/javase/tutorial/jmx/overview/javavm.html>

[15] Java Management Extensions (JMX) Specification, version 1.4 [online]. [cit. 2. Dubna 2015]. Dostupné na: <http://docs.oracle.com/javase/7/docs/technotes/guides/jmx/JMX\_1\_4\_specification.pdf>

[16] WARD, M., GRINSTEIN G., KEIM D. *Interactive data visualization: foundations, techniques and applications, Second Edition*. Boca Raton: CRC Press, 2015.

1. Příloha A

1. GNU General Public Licence v2. Více na https://www.gnu.org/licenses/gpl-2.0.html [↑](#footnote-ref-1)
2. About NEsper. Více na http://www.espertech.com/esper/about\_nesper\_dotnet.php [↑](#footnote-ref-2)
3. Spring integrace JMS (Java Message Service) pro zjednodušení práce s JMS API. Více na http://docs.spring.io/spring/docs/current/spring-framework-reference/html/jms.html [↑](#footnote-ref-3)
4. Advanced Message Queuing Protocol. Více na https://www.amqp.org/ [↑](#footnote-ref-4)
5. XML model objektů. Více na http://ws.apache.org/axiom/ [↑](#footnote-ref-5)
6. HTML5 a JavaScript webový klient. Více na http://www.espertech.com/products/esperhq.php [↑](#footnote-ref-6)
7. Plain Old Java Object je obyčejný Java objekt, který neimplementuje ani nerozšiřuje jiné třídy nebo rozhraní a ani neobsahuje žádné anotace. [↑](#footnote-ref-7)
8. GC - Způsob automatické správy operační paměti běžícího procesu. Více na http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html [↑](#footnote-ref-8)
9. JFR – produkuje detailní informace o běhu Java aplikací. Více na http://www.oracle.com/technetwork/java/embedded/resources/tech/java-flight-rec-on-java-se-emb-8-2158734.html [↑](#footnote-ref-9)
10. VM Options. Více na http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html [↑](#footnote-ref-10)
11. Java logování. Více na https://docs.oracle.com/javase/7/docs/api/java/util/logging/package-summary.html [↑](#footnote-ref-11)
12. Nástroj k sledování detailních informací o JVM. Více na https://visualvm.java.net/ [↑](#footnote-ref-12)
13. Nástroje ke sledování výkonnosti a spotřebě prostředků JVM. Více na http://www.oracle.com/technetwork/java/jvmstat-142257.html#Tools [↑](#footnote-ref-13)
14. Rozšíření poskytující mechanismus pro připojení k JVM. Více na https://docs.oracle.com/javase/7/docs/technotes/guides/attach/index.html [↑](#footnote-ref-14)
15. Ladící nástroj běžicí mimo Java proces. Více na https://visualvm.java.net/saplugin.html [↑](#footnote-ref-15)
16. Rozhraní, díky kterému mohou být metody vzdálených objektů volány z jiné JVM. Více na http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html [↑](#footnote-ref-16)
17. Standard pro správu zařízení na počítačových sítích fungujících na Internet Protokolu [↑](#footnote-ref-17)
18. Nástroj pro správu, řízení a automatizaci buildů aplikací primárně určený pro jazyk Java. Více na https://maven.apache.org/. [↑](#footnote-ref-18)
19. Návrhový vzor díky kterému lze omezit počet vytvořených instancí na jednu instanci. Více na http://www.oodesign.com/singleton-pattern.html [↑](#footnote-ref-19)