Alternativní databázové systémy (frameworky) pro EEG/ERP portál

MongoDB

Semestrální práce z KIV/DB2

Jméno: Jan Koreň

Osobní číslo: A11N0113P

E-mail: korenjan@students.zcu.cz

Datum: 23.5.2012

Obsah

Úvod

Relační databáze existují od 70. let minulého století a od té doby se jejich principy nijak výrazně nezměnily. Avšak návrh a vývoj aplikací a jejich vytížení během provozu se od té doby změnily zásadně. Proto s rozvojem agilních metodik vývoje software, cloud computingu a zvýšené interakce s uloženými daty vzrostla potřeba pro nové databázové systémy, které by byly pro tyto nové požadavky optimalizovány.

Jedním z těchto databázových systémů je MongoDB z rodiny NoSQL (Not only SQL) databází. Cílem tohoto dokumentu je popsat databázový systém MongoDB, porovnat jej se SŘBD Oracle, zjistit možnost koexistence s Oraclem a možnou integraci se Spring Data, vyzkoušet jej na vybraných testovacích datech a určit vhodnost nasazení pro potřeby EEG/ERP portálu.

Vlastnosti

MongoDB (pochází z anglického „humongous“ = ohromný) je open source dokumentově orientovaný NoSQL databázový systém napsaný v C++, za jehož vývojem stojí společnost 10gen. Ze všech aktuálně dostupných NoSQL databází je tento systém považován za nejpodobnější relačním databázím.

Základní jednotkou dat je v MongoDB **JSON dokument**, který je ukládán v binárním formátu, který se nazývá BSON (Binary JSON). V podstatě jde o **soubor dvojic klíč-hodnota**, které jsou od sebe odděleny znakem dvojtečky. Klíč určuje název položky, hodnota pak její obsah. Hodnotou však nemusí být jen jednoduché datové typy, jako jsou čísla a řetězce, nebo binární data jako například obrázky. Napravo od dvojtečky se mohou nalézat i pole hodnot nebo další páry klíč-hodnota – tzv. **vnořené dokumenty** (embedded documents). Dokumenty tak tedy vnořovat do jiných dokumentů a vytvářet tak hierarchické struktury.

Na následujícím příkladu JSON dokumentu vidíme informace o uživateli. První čtyři položky jsou prosté dvojice klíč-hodnota, následuje adresa ve formě vnořeného dokumentu a pole telefonních čísel, z nichž každé telefonní číslo je vnořený dokument:

```
{
     "jmeno" : "Petr",
     "prijmeni" : "Novák",
     "vek" : 25,
     "adresa" :
     {
         "ulice" : "Falešná 123",
```

```
        "mesto" : "Lhota",
        "stat" : "Česká republika",
        "psc" : "54321"
    },
    "telefonniCislo":
    [
        {
          "typ" : "domu",
          "cislo" : "212 555-1234"
        },
        {
          "typ" : "mobil",
          "cislo" : "646 555-4567"
        }
    ]
}
```

MongoDB je **horizontálně škálovatelná** databáze, tj. škálování je docíleno přidáním nových serverů. Není tedy potřeba vylepšovat stávající servery koupí drahého vysoce výkonného hardwaru a zbavovat se starých strojů.

Relační databáze jsou škálovatelné vertikálně pro zachování konzistence dat a umožnění transakcí, neškálují horizontálně (škálování joinu přes více serverů, podpora transakcí).

Srovnání s relačními databázemi

Struktura

Podobně jako Oracle nebo jiné relační databáze, které udržují databázová schémata, může mít i MongoDB více různých schémat, která se v terminologii MongoDB označují jako *databáze* (*database)*. Každá *databáze* může obsahovat více *kolekcí*. Kolekce je analogií tabulek z relačních databází. Každá kolekce obsahuje *dokumenty*, které můžeme přirovnat k záznamům v tabulce. Dokumenty se skládají z jedné nebo více *položek (fields)*, což je obdoba sloupců.

Mezi záznamy v tabulce v relačním světě a dokumenty v MongoDB existuje několik zásadních rozdílů, a to:

- V záznamu v tabulce je pro každý sloupec stanovena jen jedna hodnota daného datového typu. Dokument v MongoDB naproti tomu umožňuje uchovávat **více hodnot** (ve formě pole nebo vnořeného dokumentu) **pro jednu položku**.
- V relačních tabulkách je jasně dáno, jak mají vypadat záznamy, které v nich mohou být ukládány. Na úrovni kolekcí žádná takováto pravidla pro dokumenty definována nejsou. Zatímco relační databáze definují sloupce na úrovni tabulek, **dokumentově orientované**

databáze definují své položky na úrovni dokumentů. Kolekce je tedy v porovnání s tabulkou prostším kontejnerem, oproti tomu dokument uchovává daleko více informací než řádka tabulky.

Dynamické schéma

MongoDB má dynamické schéma, což znamená, že se schémata (resp. databáze v terminologii MongoDB) mohou vytvořit „za běhu“. Oproti relačním databázím zde není potřeba volat žádný příkaz pro tvorbu databáze.

Protože nejsme předem nijak vázáni, jaké dokumenty má která kolekce obsahovat, v jedné kolekci mohou být uloženy různorodé dokumenty. I když nás návrh datového modelu nutí, aby určitá struktura dokumentů pro danou kolekci byla dodržena, mít veškeré dokumenty v jediné kolekci je povolené, avšak z výkonnostních důvodů není doporučované. Strukturu uložených dokumentů je možné (oproti relačním databázím s pevnou strukturou tabulky) dynamicky měnit (např. přidáním nové položky) a to bez použití žádného příkazu, který by byl analogií k *„alter table“* nebo *„create table“*. Položky lze tak umístit do nových dokumentů, i když v těch starých ještě nebyly.

Dynamické schéma tak dělá z MongoDB ve srovnání s relačními databázemi databázi flexibilnější na změny požadavků a tedy vhodnější pro agilní vývoj. Nehledě na to, že změny ve velkých relačních schématech jsou drahé (pro zajímavost: společnost Craigslist spustila příkaz alter table nad velkou tabulkou určenou pro archivaci – operace byla dokončena za jeden měsíc).

Ovladače pro MongoDB

Pro práci s relačními databázemi je potřeba se naučit jazyk SQL. Kromě SQL se dnes už standardně využívá objektově-relační mapování (ORM, např. Hibernate) pro setření rozdílů mezi relačním a objektovým světem, které je však také nutno znát.

Pro MongoDB není ORM nutné, protože struktura dokumentu už odpovídá té objektové. Částečně je to i zásluha ovladačů, které pro MongoDB poskytují poměrně vysokou úroveň abstrakce. Tvorba datové vrstvy tak probíhá v programátorově oblíbeném programovacím jazyce. API ovladačů byla navržena tak, aby práce s dokumentem byla pro daný programovací jazyk co nejpřirozenější. V Javě tedy za využití Java ovladače pro MongoDB pracujeme s dokumenty jako s objekty.

Hlavní funkce MongoDB ovladače jsou následující:

- Generování Object IDs, která jsou uchovávána jako položka _id (primární klíč) pro každý dokument. 12-bytové Object ID je pro každý dokument unikátní a je vygenerováno v případě,

že si vývojář nevytváří vlastní a položka _id chybí. Více se o formátu generovaného primárního klíče lze dočíst v [1]

- Převod reprezentace dokumentů v daném jazyce do BSON a naopak.
- Komunikace s databází přes TCP socket za použití MongoDB komunikačního protokolu

Absence pohledů (views)

Pohledy známé z relačních databází v MongoDB nejsou.

Absence operace join

V MongoDB není operace *join.* To znamená, že si musíme join vytvořit sami, a to vytvořením dalšího dotazu pro nalezení souvisejících dat. Z tohoto důvodu jsou často data v dokumentech denormalizována a zodpovědnost za zachování konzistence dat se taktéž přenáší do aplikační vrstvy.

Absence operací join však není takovým handicapem, pokud vezmeme v úvahu, že hodnotou pro daný klíč může být pole hodnot. Pomocí hodnot uložených v poli tak můžeme vyjádřit relace 1:N nebo M:N. /Příklad/. Zajímavé je, že dotaz pracuje nezávisle na tom, zda je jako hodnota klíče skalární typ nebo pole.

Kromě polí jsou dále podporovány vnořené dokumenty, na které se můžeme dívat jako na „předjoinovaná“ data.

Chybí transakce

MongoDB nepodporuje komplexní transakce (pro mnoho dokumentů) s rollbackem. Avšak je zde podpora atomických operací na úrovni dokumentu pro modifikaci, insert a update elementů. Více se o atomických operacích lze dočíst na [2].

Žádná doménová integrita (chceck constraints)

Data na úrovni dokumentů nemohou mít definovaná žádná omezení. Omezení přípustných hodnot se proto řeší na úrovni aplikační logiky.

DBRef

DBRefs slouží k tomu, aby dokumenty z různých kolekcí na sebe mohly navzájem odkazovat. Jedná se o formálnější přístup zajištění referencí („cizí klíče“) mezi dokumenty. Pokud ovladač nalezne DBRef, automaticky načte referencovaný dokument. DBRef zahrnuje název kolekce a id referencovaného dokumentu. DBRefs se ale nevyužívají příliš často a přímo na oficiálních stránkách MongoDB [3] je doporučeno dát přednost přímým (manuálním) referencím. V praxi se pak rozhoduje, jestli související data uložit jako vnořené dokumenty a nebo je manuálně referencovat. Je důležité mít na paměti maximální povolenou velikost dokumentu, která je v současné době 16 MB. I tak se ale

vnořené dokumenty využívají, a to většinou pro nevelká data, která vždy chceme obdržet společně s rodičovským dokumentem (například komentáře k článku na blogu). Navíc v tomto případě ušetříme jeden dotaz pro provedení operace join, kromě toho je možné položky vnořeného dokumentu indexovat a dá se na ně i dotazovat.

Denormalizace

Další možnost, jak se vypořádat s absencí joinu v MongoDB, je denormalizovat data. Oproti relačním databázím jde zde o poměrně běžnou záležitost. Denormalizace se využívá především pro informace, které jsou přirozeně spjaté s nadřazeným dokumentem a na které se dotazujeme vždy společně s tímto dokumentem. Jedná se například o různé metainformace o uložených fotografiích.

Indexy

Indexy v MongoDB se koncepčně tolik neliší od těch v relačních databázích. V praxi platí, že se v MongoDB indexuje ve stejných případech jako by se indexovalo v relačních databázích. Jsou zde implementovány jako B-stomy. Pro každou kolekci se automaticky vytvoří index pro položku `_id`. Jinak musí být indexy explicitně deklarované stejně jako v relačních databázích mohou být indexy přidány později. Lze indexovat i klíče vnořených dokumentů za použití „tečka notace“. Je možné vytvářet složené indexy a tzv. řídké indexy (*sparse indexes*), které indexují jen ty dokumenty, které obsahují požadovanou indexovanou položku. Proto pokud použijeme řídký index pro řazení, některé dokumenty v kolekci nemusí být vráceny, protože ne všechny dokumenty musí tento index obsahovat.

Za zmínku dále stojí podpora *geospaciálních indexů* pro dotazy založené na určení polohy (hledání nejbližšího objektu v okruhu N kilometrů apod.). Z podpory geospaciálních indexů těží například FourSquare, který (nejen) proto provedl migraci dat o lokalitách a check-inech z relační architektury do MongoDB.

Kurzory

Dotaz v MongoDB vrací kurzor, se kterým můžeme dále manipulovat. Spuštění kurzorů je tedy odkládáno do doby, kdy je skutečně nutné. Přesný způsob zacházení s kursorem záleží na použitém ovladači (resp. pro který jazyk je ovladač určen). Ukázka práce s kursorem bude v tomto dokumentu později ukázána.

MapReduce

Map-reduce je jakousi obdobou příkazu `group by` ve světě NoSQL. Probíhá však ve dvou na sobě nezávislých krocích (*map* funkce a *reduce* funkce) a umožňuje tak využít paralelismu. Funkce *map* a *reduce* mohou být libovolně komplexní, a tak map-reduce nebízí širší možnosti agregovaných

dotazů než relační `group by`. Více se lze o map-reduce dozvědět např. na příslušné sekci na oficiálních stránkách MongoDB [3].

Sharding

Pokud už není možné uchovat indexy a pracovní množinu dat v paměti RAM, dochází k drastickému snížení výkonu databáze. Jde o jeden z případů, kdy je vhodné využít sharding.

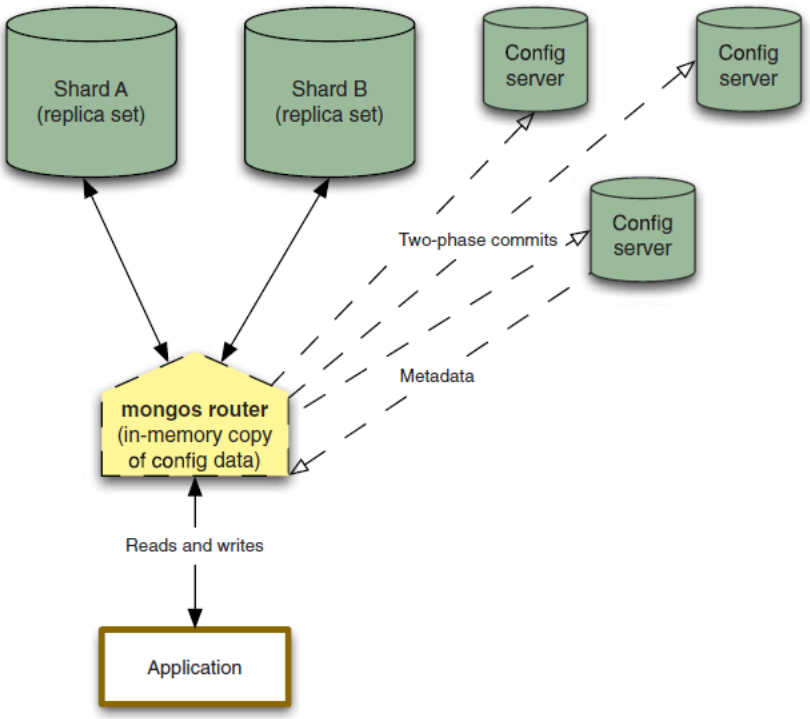
Pro sharding je také dlvod se rozhodnout, pokud kapacita disku na serveru pro ukládaná data nestačí nebo pokud chceme využít paralelismu a zapisovat data rychleji, než zvládne jediný proces *mongod*. Sharding je způsob, jakým je v MongDB realizováno horizontální škálování. Jde o rozdělění objemných dat v kolekcích do více databázových serverů podle určitého klíče. Tímto klíčem je tzv. *shard klíč,* který je dané kolekci přiřazen. Shard klíč určuje položky, podle kterých se distribuují data do jednotlivých databází.

Shard cluster v MongoDB distribuuje data do jednoho nebo více shardů. Každý shard je tvořen aspoň jedním serverem. Aby byla zachována dostupnost systému, musí být každý shard vždy online. Z tohoto důvodu se každý shard obvykle skládá z více serverů (každý z nich má spuštěn proces *mongod*). Tyto servery pak tvoří tzv. *replica set*. Každý server v replica setu obsahuje kopii všech dat pro daný shard a vždy jeden z nich je master (primární) a ostatní jsou slave (sekundární - repliky master serveru). Pro zajištění automatického failoveru jsou servery v shardu mezi sebou propojeny a vysílají si tzv. „heartbeat signál“, podle kterého repliky detekují havárii master serveru. Jsou schopny si mezi sebou zvolit nový master server využitím hlasovacího protokolu.

Všechny operace zápisu (insert, update, delete) v rámci jednoho shardu se pro zajištění konzistence dat provedou nejprve na primárním serveru, teprve pak se provádí asynchronní zápis na sekundárních serverech. U operace čtení je volba ponechána na aplikaci, číst se může z kteréhokoliv uzlu, avšak je možné, že data na slave uzlech vlivem asynchronních zápisů z master uzlu nemusí být aktuální.

MongoDB se také stará o automatické vyvažování zátěže na shardu. Pokud zátěž na některém shardu roste neúměrně oproti zbytku, data jsou redistribuována tak, aby byla zátěž na všech shardech opět vyrovnaná.

K jednotlivým shardům se přistupuje pomocí MongoS serverů. Na nich běží proces pro routování a koordinaci *mongos,* díky kterému se všechny uzly clusteru tváří navenek jako jeden logický celek. Konfigurace jednotlivých shardů je spravována a uchovávána v konfiguračních serverech.



Komponenty v MongoDB shard clusteru (zdroj: [1])

Sharding je tedy efektivní způsob, jak pro objemná data realizovat rychlé čtení a zápis. Více o shardingu včetně informací o jeho konfiguraci lze nalézt v [1], v [4] nebo v [5]

> MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. Automatic configuration is easy to deploy and new machines can be added to a running database.

Další omezení

BSON dokumety jsou v MongoDB limitovány 16 MB. Pro toto omezení existuje několik důvodů. Prvním důvodem je snaha, aby se předešlo dokumentům s vysokým stupněm zanoření, se kterými se hůře pracuje a které jsou ukázkou špatného návrhu datového modelu. Hluboce zanořené dokumenty je pak lepší ukládat do samostatných kolekcí. Dalším důvodem jsou výkonnostní hlediska. Na straně serveru se při dotazování velkého dokumentu musí celý tento dokument zkopírovat do bufferu, aby mohl být odeslán klientovi. Tato operace může být dosti drahá, zvlášť když klient nevyžaduje celý dokument.

Situace je jiná, pokud uchováváme velké binární soubory jako obrázky nebo video. Pokud je chceme uchovávat přímo v MongoDB, pak se pro tyto účely využívá GridFS, který rozděluje soubor na části po 256 KB a každou tuto část ukládá jako samostatný dokument. Více se lze o GridFS dozvědět např. na [6] nebo v [*1*].

Návrh schématu

Většinou jsou MongoDB systémy navrženy podobně jako systémy relační. Pokud bychom pro určitá data použili v relační databázi tabulku, v MongoDB pro tatáž data použijeme pravděpodobně kolekci. Můžeme ale data z podřazené kolekce uložit jako vnořené dokumenty k dokumentům v rodičovské kolekci. Odpověď na otázku, zda dokumenty nechat rozdělené v kolekcích nebo je vnořovat, není jednoznačná a záleží zde na povaze dat a na jejich velikosti, neboť musíme brát v úvahu limit 16 MB pro dokument. Neexistuje však žádné pevně dané pravidlo a konkrétní rozhodnutí je na vývojáři. Většinou se však dává přednost separaci do kolekcí, neboť se tento postup jeví obecně jako „čistší“.

Kdy použít MongoDB a kdy relační databáze

NoSQL databáze jsou charakteristické tím, že neobsahují funkcionality známé z relačních databází (např. z Oracle), které v praxi nejsou tolik využívány. Hodně webových stránek se obejde bez uložených procedur a funkcí, triggerů, nebo plné podpory ACID (a tedy podpory transakcí).

Pro volbu výsledné databáze musíme vyjít z požadavků pro aplikaci a uvědomit si některá omezení, která nám MongoDB přináší. Potřebujeme transakce? Můžeme riskovat ztrátu dat? Jaké jsou požadavky na výkon? Nutnost transakčního zpracování a nižší riziko ztráty dat hovoří jasně pro relační databáze. Kromě toho může záležet na požadavcích na volný prostor. Tady mají také relační databáze navrch, data uchovávaná v MongoDB totiž (především vlivem redundantních denormalizovaných dat) zabírájí obecně více místa.

Použití MongoDB bychom měli zvážit, pokud:

- Požadujeme zdroje CPU od několika strojů
- Máme velká kvanta archivovaných dat
- Struktura dat se rychle mění
- Jsme ochotni tolerovat možnou nekonzistenci dat

Relační databáze mají ze své podstaty problém s uchováváním semistrukturovaných dat, tj. dat, u nichž není pevně daná struktura. Takováto data lze uchovávat v tabulce typu jméno-hodnota, ale za cenu pomalého dotazování, nebo uložit celý dokument jako BLOB, což však znemožňuje dotazování na jednotlivé atributy.

NoSQL databáze nejsou v žádném případě náhrada zaběhnutých relačních databází. Místo toho je lepší se na ně dívat jako vhodnější řešení pro určité situace, se kterými si sice poradíme i v případě použití relačních databází, ale za cenu většího úsilí či ne příliš uspokojivého výsledku.

Z tohoto důvodu se nyní NoSQL databáze začínají využívat v kombinaci s relačními databázemi. Tento přístup se označuje jako *cross-store storage* (někdy též *polyglot storage*), o kterém je dále řeč v sekci o integraci MongoDB s frameworkem Spring Data.

MongoDB a možné případy užití

Následující část ukazuje možná vhodná použití MongoDB.

Společnost Craigslist využívá MongoDB pro **archivování starých příspěvků**. Využitím vlastnosti dynamického schématu je snadné v jedné kolekci uchovávat data, která se postupem času mění, a dotazovat se na ně.

Jako vhodné se jeví použití MongoDB k **uchovávání metadat** jako jsou štítky (tagy). Struktura těchto metadat je flexibilní a může být různá např. pro fotografie a pro videa. V jedné kolekci tak může být jak multimediální obsah, tak k němu přidružené metainformace, pro jejichž získání nepotřebujeme manipulaci s daty odpovídající operaci join. Pro vnořené dokumenty navíc můžeme vytvořit řídké indexy pro rychlejší vyhledávání podle štítků.

Spojení flexibility bezeschémové databáze, asynchronních a rychlých zápisů a *capped kolekcí* (jsou pevné velikosti, po naplnění se v nich mažou data podle strategie FIFO) našlo své využití v **žurnálování**.

Protože MongoDB pracuje rychle nad čtením a zapisováním malých objemů dat, uplatnila se tato databáze také v **online hrách** (např. firma Disney).

Guardian se potýkal s narůstající komplexitou databázového schématu, ORM mapování bylo dosti složité a navíc bylo v některých situacích od ORM upustit a napsat přímo SQL kód. Dalším problémem byla obtížná škálovatelnost RDBMS Oracle. Přechodem (zatím částečným) na MongoDB se vyřešila škálovatelnost, nasazení dynamického schématu pro správu článků se ukázalo jako výhodné, dají se nad ním vytvořit také komplexní dotazy a celkově se tak značně zvýšila produktivita práce. Z těchto důvodů Guardian uvažuje o kompletní migraci na MongoDB.

MongoDB má také potenciál pro **uchovávání XML dat**. Je ale nejprve nutný přesun dat z XML reprezentace do MongoDB (JSON). Pak mohou být jednotlivé položky dokumentu updatovány, indexovány i získány částečně. Manipulace s takovými daty tak může být rychlejší, jednodušší a robustnější.

Manipulace s dokumenty v MongoDB shellu

Manipulace s dokumenty zde MongoDB shellu zde nebude dopodrobna rozebírána, zájemce odkazuji na příslušné sekce v manuálu na oficiálních stránkách MongoDB [3]

Následující bodový přehled uvádí některá specifika MongoDB v této oblasti

Dotazování

- Kromě hledání podle položky má MongoDB podporu hledání dle regulárních výrazů.
- Dotazy mohou také obsahovat uživatelem definované javascriptové funkce.
- Na položky vnořených dokumentů se lze dotazovat, a to využitím „tečka notace“.
- Pro porovnávání se nepoužívají operátory `>, >=, <, <=,` ale `$gt, $gtr, $lt, $lte`.
- Pro práci s poli jsou dostupné operátory jako:
  - `$all` (nalézají se v poli všechny uvedené hodnoty?)
  - `$in` (nalézá se jedna z uvedených hodnot v hodnotě klíče?)
  - `$exists` (existuje položka s uvedeným klíčem?)
  - `$and` (splňuje dokument všechny uvedené podmínky?)
  - `$size` (vrací počet prvků pole)
  - A další.
- Podobně jako v MySQL je dostupná funkce `limit()`, která určuje maximální počet výsledků, které se mají vrátit. Je doporučeno ji používat všude, kde to je jen možné.

Update, hledání a mazání

- update() se standardně chová tak, že modifikuje první nalezený objekt (a celý jej nahradí)
- Jsou ale možné i tzv. **částečné updaty**, např.:

```
Db.lide.update({jmeno: 'Vašek'}, {'$set': {'vek': 50} });
```

- Dají se vybírat a vkládat položky polí:

```
Db.lide.update({jmeno: "Zuzka"}, {'$pull': {'zajmy: "horolezení"} });
Db.lide.update({jmeno: "Zuzka"}, {'$push': {'zajmy': "vaření"} });
```

- Pro další informace o operacích update(), find() a remove() odkazuji na referenční příručku [7].

Instalace MongoDB

Instalace MongoDB je velmi jednoduchá. Nejprve je nutné stáhnout si MongoDB pro příslušný operační systém. Jednotlivé distribuce MongoDB jsou dostupné ke stažení na oficiálních stránkách [3].

Stažený archiv rozbalíme do libovolného adresáře. V něm se nachází adresář `bin`, ve kterém vytvoříme soubor pojmenovaný např. `mongodb.config`. V tomto souboru specifikujeme cestu k datům, které MongDB uchovává, a to tímto způsobem:

```
dbpath=CESTA_K_DATŮM
```

Cestu k datům nemusíme nastavovat. V takovém případě se volí výchozí nastavení a data se ukládájí do \data\db. V obou případech ale musíme daný adresář vytvořit, protože jej MongoDB nevytvoří automaticky.

Další možností, jak zvolit jiné umístění pro své datové soubory, je pomocí parametru

```
--dbpath CESTA_K_DATŮM
```

ve spuštěném `mongod` (o tomto souboru viz dále)

Spuštění serveru

V domovském adresáři MongoDB, do kterého jsme rozbalili staženou distribuci, se nalézá adresář `bin`. V něm se nacházejí dva důležité soubory – databázový server `mongod` (`mongod.exe` pod Windows) a shell pro administraci `mongo` (`mongo.exe`).

Pro spuštění databáze stačí spustit `mongod.exe`. Vidíme, že je logována činnost serveru. Pro možnosti nastavení serveru poslouží při jeho spouštění příkaz `mongod --help`.

Chceme-li využít nastavenou cestu k datům ve vytvořeném souboru mongodb.config, spustíme MongoDB server mongod s parametrem

```
--config CESTA_K/mongodb.config
```

Vytvoření databáze

Pro spuštění shellu je potřeba spustit `mongo.exe`. Při výchozím bezparametrickém spuštění se mongo.exe připojuje k `mongod` serveru, běží na `localhost` a používá databázi pojmenovanou `test`. Pro zjištění dalších nastavení lze opět použít příkaz `--help`. Pro přepnutí databáze na databázi `eegmongo` napíšeme v shellu

```
use eegmongo
```

Stojí za povšimnutí, že jsme nikde explicitně nevytvářeli novou databázi. MongoDB ji vytvoří za nás, jakmile do ni vložíme nějaká data. Do té doby s ní MongoDB zachází jako s prázdnou kolekcí a nezobrazí se v seznamu databází (příkaz `show dbs`).

V shellu můžeme vidět seznam užitečných příkazů po zadání příkazu `help`.

Bezpečnost a autentikace

Pro základní zabezpečení databáze uživatelským jménem a heslem se stačí přepnout do žádané databáze a vytvořit pro ni nového uživatele:

```
use mojeDatabaze
db.addUser("novyUzivatel","noveHeslo")
```

Chceme-li si ověřit, že nový uživatel pro danou databázi skutečně existuje, můžeme vypsat seznam všech uživatelů pro aktuální databázi:

```
db.system.users.find()
```

Další informace o bezpečnosti a autentikaci databáze lze nalézt v odpovídající sekci na oficiálních stránkách MongoDB [8]

Alokace

Po vytvoření databáze alokuje MongoDB několik souborů na disku, ve kerých jsou uchovávány veškeré kolekce, indexy a další metadata jedné databáze. Tyto soubory jsou uloženy do adresáře uvedeného jako hodnota parametru `dbpath` při startu mongod. Nebyl-li parametr `dbpath` uveden, soubory se uloží do `/files/db`. Databázové soubory se jmenují podle databáze, ke které přísluší. Kolekce a indexy databáze jsou uchovávány v souborech končících číslem, které se s každým novým souborem o 1 zvětšuje. Velikost prvního takového souboru je 64 MB a každý následující dostane alokován dvojnásobek místa než jeho předchůdce, dokud se nedosáhne limitu 2 GB. Od začátku se alokující poměrně velké soubory proto, že se MongoDB snaží zajistit, aby co nejvíce dat bylo alokováno co fyzicky nejblíže (a tím operace nad nimi byly rychlejší).

Testování na podmožině EEG/ERP portálu

V EEG/ERP portálu se informace o scénářích EEG/ERP experimentů ukládají do XML. Způsob, jakým se uchovávají data v MongoDB, je dosti podobný způsobu ukládání dat v XML. Proto se nabízí otázka, zda není lepší ukládat data scénářů přímo jako dokumenty v MongoDB a mít tak větší kontrolu nad manipulací s daty.

Byla proto vytvořena jednoduchá aplikace založená na frameworku Spring, která spravuje data scénářů v kolekcích MongoDB, do kterých je také umožňuje z XML reprezentace převést.

Spring Data

Hlavním cílem tohoto projektu je usnadnění vývoje aplikací založených na Springu s využitím jiných technologií pro uchovávání dat, mezi které patří i NoSQL databáze. Jednou z hlavních myšlenek je využití generických rozhraní pro přístup k uloženým datům. Metody uvedené v rozhraní stačí pojmenovat dle určitých konvencí a jejich implementace je pak vygenerována frameworkem automaticky.

Pro databázový systém MongoDB je integrace se Spring Data možná a byla vyzkoušena na zmíněné webové aplikaci. V následujícím textu bude tento proces integrace s využitím kódu aplikace popsán a budou zde také ukázány některé zajímavé možnosti, kterými nám Spring Data ulehčuje práci.

Konfigurace

Předpokládá se, že aplikace používá Maven a že vše potřebné pro běh Spring MVC je již nastaveno. V případě, že tomu tak není, mohou kromě sposty veřejně dostupných materiálů o dané problematice dobře posloužit i konfigurační XML soubory ukázkové aplikace.

Podpora MongoDB

Do *pom.xml* přidáme pro podporu Spring Data pro MongoDB následující závislost:

```
<dependency>
      <groupId>org.springframework.data</groupId>
      <artifactId>spring-data-mongodb</artifactId>
      <version>${aktuální_verze} </version>
</dependency>
```

Aktuální verze byla v době psaní tohoto dokumentu *1.0.1.RELEASE*.

Pro propojení MongoDB se Springem je dále potřeba přidat do kořenového elementu springovského konfiguračního souboru definici Mongo XML schématu:

```
xmlns:mongo="http://www.springframework.org/schema/data/mongo"
...
xsi:schemaLocation=
"http://www.springframework.org/schema/data/mongo
http://www.springframework.org/schema/data/mongo/spring-mongo-
1.0.xsd"
```

Nově dostupný jmenný prostor nám umožňuje nastavit připojení k MongoDB databázi. Nutným předpokladem pro připojení k databázi je mít instanciovanou třídu *com.mongodb.Mongo*. Tu lze „podstrčit“ IoC kontejneru Springu v konfiguračním XML:

```
<mongo:mongo id="mongo"
                 host="${db.mongo_host}"
                 port="${db.mongo_port}"/>
```

Tato třída slouží pro nastavení různých parametrů připojení. Vlastní připojení k MongoDB databázi obstarává *MongoFactory*, resp. jedna z jejích implementací:

```
<mongo:db-factory id="mongoDbFactory"
                  dbname="${db.mongo_dbname}"
                  username="${db.mongo_username}"
                  password="${db.mongo_password}"
                  mongo-ref="mongo"/>
```

Proměnné pro parametry připojení k databázi se v projektu nachází se v

`/src/main/resources/database.properties`

V našem konfiguračním XML se dále musíme zasadit o vytvoření instance *MongoTemplate* Jedná se o důležitou třídu, které poskytuje funkcionalitu pro komunikaci s MongoDB (CRUD a mapování Java objektů na MongoDB dokumenty).

```
<bean id="mongoTemplate"
class="org.springframework.data.mongodb.core.MongoTemplate">
     <constructor-arg name="mongoDbFactory"
ref="mongoDbFactory"/>
     <property name="writeResultChecking" value="LOG"/>
</bean>
```

Nyní určíme v našem XML kontextu, kde má Spring hledat repozitáře:

```
<mongo:repositories base-
package="cz.zcu.kiv.eegmongo.repository"/>
```

Spring pak prochází tento balík a jeho podbalíky a hledá tu rozhraní dědící od *Repository* (nebo některých jeho potomků). Pro každé nalezené rozhraní pak vygeneruje beanu, která obsahuje konkrétní implementaci metod dotazů nad úložištěm. Standarně se tyto beany jmenují stejně jako rozhraní, jen začínají malými písmeny.

Kompletní konfigurace pro MongoDB se v ukázkové nachází v

`/src/main/resources/mongodb-context.xml`

Nastavení podpory cross store

V případě, že chceme z různých důvodů využít pro chovávání dat daného objektu více databázových systémů, například část dat uchovávat v Oraclu a část dat v MongoDB, hovoříme o tzv. cross store.

Ve Spring Data se o podporu cross store stará pro MongoDB samostatný modul. Je proto potřeba do Mavenu přidat následující závislost:

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-mongodb-cross-store</artifactId>
    <version>${spring.data.mongo.version}</version>
</dependency>
```

Podpora cross store ve Spring Data se také neobejde bez AspectJ aspektů. Do *build* sekce pom.xml je potřeba přidat *AspectJ Maven Plugin.* Čtenáře z důvodu obsáhlosti kódu odkáži na příslušné řádky v pom.xml.

V ukázkové aplikaci se nachází konfigurační XML soubor pro nastavení JPA *jpa-context.xml*. Pro JPA úložiště je zde nastaveno uchovávání dat v databázi Oracli. Stojí zde za povšimnutí zejména tyto řádky:

```
<jpa:repositories base-
package="cz.zcu.kiv.eegmongo.repository">
     <jpa:repository id="userRepository"/>
</jpa:repositories>
```

JPA úložiště zde musela být definována explicitně, vlivem uchovávání rozhraní úložišť pro MongoDB ve stejném balíku dochází k určitým konfliktům a aplikaci by tak jinak nebylo možné přeložit.

Nakonec je třeba nastavit v konfiguračním XML podporu cross store a aspektů:

```
<bean
class="org.springframework.data.mongodb.crossstore.MongoDocumen
tBacking" factory-method="aspectOf">
     <property name="changeSetPersister"
ref="mongoChangeSetPersister"/>
</bean>

<bean id="mongoChangeSetPersister"
class="org.springframework.data.mongodb.crossstore.MongoChangeS
etPersister">
     <property name="mongoTemplate" ref="mongoTemplate"/>
     <property name="entityManagerFactory"
ref="entityManagerFactory"/>
</bean>
```

Další nastavení

Je možné nastavit podporu Log4j pro MongoDB (viz referenční příručka [9]) nebo podporu QueryDSL. QueryDSL je Maven plugin, který představuje způsob, jak vytvářet typově bezpečné dotazy. Více se o něm lze dozvědět na oficiálních stránkách [10] a o možnostech jeho nastavení například na [11].

Použití

MongoDB

Doménové třídy, jejichž objekty chceme ukládat v MongoDB, označíme anotací *@Document*. Tato anotace sice není povinná (objekt by byl mapován správně i bez této anotace), umožňuje však classpath scanneru najít a předpřipravit si doménové objekty a vzít si z nich potřebná metadata. Proto její nepoužití znamené pomalejší běh aplikace, pokud běží poprvé.

Další anotaci *@Id* lze využít pro mapování anotovaného atributu na *_id* dokumentu v MongoDB. Téhož dosáhneme, pokud atribut pojmenujeme *id.* V ostatních případech se o generování povinného *_id* stará MongoDB driver (mapování mezi žádným atributem objektu pak pochopitelně není).

Existuje řada dalších anotací (@Indexed pro indexování dané property, @DBRef dává na vědomí, že je položka uchovávána v jiné kolekci), jejichž kompletní přehled lze nalézt kupříkladu v referenční příručce [9].

Data objektů jsou ukládány v kolekcích, jejichž jméno standardně odpovídá názvu třídy s tím rozdílem, že první písmeno je malé.

Repozitáře (rozhraní) pro manipulaci s dokumenty dědí v ukázkové aplikaci od *MongoRepository*. Kromě základních operací poskytovaných tímto rozhraním mohou být v repozitáři definovány dotazy následujícími dvěma způsoby:

- Využít mechanismu parsování názvů metod. Spring Data se stará o převod názvu metody (za dodržení daných konvencí) do příslušného JSON dotazu. Více se o tomto způsobu lze dozvědět v [9].
- Anotací *@Query* pro deklaraci metody specifikovat konkrétní dotaz (více informací opět v [9]). V ukázkové aplikaci je například uživatel hledán podle id takto:

```
@Query("{ 'userId' : ?0 }")
public List<T> findByTheUserId(String userId);
```

Cross-store

V aplikaci je předvedeno cross-store na uchovávání dat o uživateli, jehož jméno a příjmení se ukládají v relační databázi Oracle a další nepovinné informace jako dokument v MongoDB. Uvedený příklad nemá větší praktický význam a slouží hlavně pro předvedení funkčnosti a možností.

Ve třídě *User* je použito kromě standardních JPA anotací pro objektově relační mapování také anotace @RelatedDocument:

```
@RelatedDocument
private UserInfo userInfo;
```

Ta říká, že tento objekt nemá být uložen v relační databázi, ale v kolekci MongoDB jako dokument. Spring Data se pak již o další vzájemné postará automaticky.

Mapování vztahu 1:N nebo M:N pro třídu a její objekt s anotací @RelatedDocument se nepodařilo realizovat a jeho podpora nebyla zjištěna.

O aplikaci

Zhotovená aplikace ukazuje možnosti MongoDB v kombinaci se Spring Data a cross-store perzistentním úložištěm. Řeší především uchovávání dat z XML (EEG scénáře) v odpovídající dokumentové podobě.

Aplikace umožňuje registrovat uživatele a ke každému uživateli spravovat data XML scénářů, přičemž existuje více typů scénářů. Data z XML byla parserem SAX převedena do své objektové podoby, což otevřelo dveře k použití Spring Data repozitářů pro jejich uchování v MongoDB a snazší manipulaci. Nevýhodou tohoto přístupu je, že je nutné znát předem strukturu daného typu scénáře pro převod na objekt a pro každý nový scénář se pak musí vytvářet nový parser a nová objektová reprezentace pro tento scénář.

Uživatel při nahrávání scénáře musí zvolit jeho typ pro správné uložení do MongoDB. Je ale možné v případě absence požadovaného typu scénáře zvolit tzv. generický typ scénáře. Protože ale nemáme dopředu žádnou informaci o struktuře dat v tomto souboru, nelze jej parsovat podobným způsobem jako známé typy scénářů, což má za následek velmi omezené možnosti dotazování na jednotlivé položky. Celý XML dokument scénáře převeden do formátu JSON využitím knihovny Staxon. JSON dokument je pak uložen jako jediná položka dokumentu v kolekci.

Závěr

Databázový systém MongoDB byl v tomto dokumentu popsán a srovnán s relačními databázemi. Byly zde nastíněny některé možnosti využití MongoDB a jedna z těchto možností – náhrada za XML – byla prakticky vyzkoušena na vzorových datech na ukázkovém demo projektu. V projektu byla navíc předvedena integrace s frameworkem Spring Data a byla také předvedena jeho podpora cross-store pro koexistenci MongoDB se SŘBD Oracle.

Nepodařilo se však za použií cross-store pro objekt uchovávaný v Oracle ukládat k němu příslušnou kolekci dokumentů v MongoDB.

Větším zádrhelem pro uchovávání XML dat je převod XML do JSON. Pro každý typ XML dokumentu byly vytvořeny třídy pro jejich parsování pomocí SAX, které převádějí data v XML do objektů navržených pro konkrétní typy scénářů. Tyto objekty je pak už možné voláním patřičných metod tříd repozitářů automaticky uložit jako JSON (resp. BSON) do odpovídající kolekce. Je však nutné znát předem strukturu ukládaných dat. Na XML data, jejichž struktura není známa, se ve své současné podobě prakticky není možné dotazovat.

Dokumenty v MongoDB se tak ukázaly jako možná alternativa uchovávání dat EEG/ERP scénářů. Nabízí v kombinaci se zde uvedenými technologiemi vysoký komformt pro manipulaci s uloženými daty. Jako největší překážku však vidím získání objektové reprezentace pro XML data a obecně převod z XML tak, aby se s daty ze všech možných scénářů dalo rozumně manipulovat.

Zdroje

[1] BANKER, Kyle. *MongoDB in action*. Shelter Island, NY: Manning, c2012, xxi, 287 p. ISBN 19-351-8287-0.

[2] http://www.mongodb.org/display/DOCS/Atomic+Operations

[3] http://www.mongodb.org/

[4] http://www.mongodb.org/display/DOCS/Sharding+Introduction

[5] http://horicky.blogspot.com/2012/04/mongodb-architecture.html

[6] http://www.mongodb.org/display/DOCS/GridFS+Specification

[7] http://www.mongodb.org/display/DOCS/Manual

[8] http://www.mongodb.org/display/DOCS/Security+and+Authentication

[9] http://static.springsource.org/spring-data/data-document/docs/1.0.0.M3/reference/html/#mongodb.repositories.queries

[10] http://www.querydsl.com/

[11] http://www.petrikainulainen.net/programming/spring-framework/spring-data-jpa-tutorial-part-five-querydsl/