

Hibernate a paralelní kolekce

Často od kolegů slyším, že Hibernate je neefektivní a že se na hodně problémů nehodí. Protože jsem s ním už přes rok nepracoval a za tu dobu se hodně změnilo, rozhodl jsem se, že si udělám menší experiment – vyzkouším efektivitu při zpracování paralelních kolekcí.

Problém je následující: mám objekt klient, který obsahuje kolekci adres a a bankovních účtů. Takovýchto klientů mám deset tisíc a chci zobrazit (načíst) záznamy 1000 až 1050 se všemi adresami a kontakty. Problém pro O/R mapovací nástroj jako stvořený.

Ukažme si nástin použitých tříd (kompletní zdrojový kód naleznete [zde](#)). Nejprve třída Client

```
@Entity
@Table(name="client")
public class Client {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    @Column
    private Long personId;

    @Column
    private String name;

    @OneToMany(cascade=CascadeType.ALL, mappedBy="client")
    private Set<Account> accounts;

    @OneToMany(cascade=CascadeType.ALL, mappedBy="client")
    private Set<Address> addresses;

    public void addAddress(Address address)
    {
        if (addresses == null)
        {
            addresses = new HashSet<Address>();
        }
        addresses.add(address);
        address.setClient(this);
    }

    //zbytek implementace vynechán
}
```

Takové ty věci za zavináčem jsou anotace. Od verze Hibernate 3.2 už totiž můžeme definovat metadata pro Hibernate i pomocí anotací. Použité anotace jsou mimochodem stejné, jako ty, které definuje JPA. Tuto třídu mohu tedy bez problému použít i třeba s TopLinkem. Asi není nutné si použité anotace moc popisovat, jsou více méně samopopisné. (více informací [zde](#).)

Zdržel bych se jen u `mappedBy`. Tento atribut říká, že relace je obousměrná (`Client↔Address`) a že se o její ukládání postará druhá strana. Aby všechno fungovalo tak jak má, musí každá adresa, obsažená v kolekci adres, mít správně nastavenou referenci na klienta (pole `client`). To nám zde zajišťuje metoda `addAddress`.

Bylo mnohem jednodušší použít jenom jednostranný vztah (Client→Address). Adresa nepotřebuje vědět kterému klientovi patří. Musím se ale přiznat, že se mi to pomocí Hibernate nepodařilo. V [dokumentaci](#) se jenom dočtete: „A *unidirectional one-to-many association on a foreign key* is a very unusual case, and is not really recommended.“ K tomu není co dodat. Pojďme ale dál, třída Address (a obdobně i Account) vypadá následovně:

```
@Entity
@Table(name="address")
public class Address {
    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    @Column
    private String street;

    @Column
    private String streetNo;

    @Column
    private String municipality;

    @Column
    private String zipCode;

    @ManyToOne
    @JoinColumn(name="client_id")
    private Client client;

    //zbytek implementace vynechán
}
```

Také to není žádná věda, za zmínku stojí snad jen již zmiňovaný druhý směr asociace Client↔Address.

Třídy máme tedy hotové. Databázové schéma je také nasadě. Jedna tabulka pro klienta a po jedné tabulce pro adresu a účet. Cizí klíče na klienta jsou obsaženy právě těchto tabulkách.

Podívejme se na slibovanou výkonnost. Pokud neměním standardní nastavení a spustím

```
session.createQuery("from Client c order by personId asc").
    setFirstResult(1000).setMaxResults(50).list();
```

Hibernate vykoná následující SQL dotaz (spuštěno na MySQL, zjednodušeno):

```
select id, personId, name from client order by personId asc limit 1000, 50
```

Protože má Hibernate standardně zapnut lazy-loading kolekcí, jsou informace o adresách a účtech načteny, až když jsou potřeba. Tzn. až když na ně poprvé přistoupíme. Pokud budeme iterovat přes všechny klienty a číst jejich adresy a účty dostaneme

```
select client_id, id, accountNumber
    from account where client_id = 1000
select client_id, id, street, streetNo, municipality, zipCode
    from address where client_id=1000
```

```

select client_id, id, accountNumber
    from account where client_id = 1001
select client_id, id, street, streetNo, municipality, zipCode
    from address where client_id=1001

```

.
.
.

```

select client_id, id, accountNumber
    from account where client_id = 1049
select client_id, id, street, streetNo, municipality, zipCode
    from address where client_id=1049

```

Máme tu tedy klasický problém n+1 čtení (zde 2n+1) – pro padesát klientů máme 101 SQL dotazů. Otázka zní, jak to zefektivnit? Abych vás trochu napnul nejdříve ukáži špatná řešení.

Fetch join

Můžeme zkusit vyrazit s kanónem na vrabce. Co takhle použít následující dotaz:

```

session.createQuery("from Client c " +
    "left join fetch c.accounts " +
    "left join fetch c.addresses " +
    "order by personId asc").
    SetFirstResult(1000).
    setMaxResults(50).list();

```

Tím Hibernate v podstatě přikážeme, aby použil outer join a načetl všechno pomocí jediného dotazu:

```

select c.id c, ac.id, ad.id,
    personId, name,
    accountNumber,
    street, streetNo, municipality, zipCode
from client c
    left outer join account ac on c.id=ac.client_id
    left outer join address ad on c.id=ad.client_id
order by personId asc

```

Narážíme tu ale na dva problémy. První je, že ve výsledku máme pro každého klienta všechny kombinace adres a účtů

Client 0	Account 0	Address 0
Client 0	Account 0	Address 1
Client 0	Account 0	Address 2
Client 0	Account 0	Address 3
Client 0	Account 1	Address 0
Client 0	Account 1	Address 1
Client 0	Account 1	Address 2
Client 0	Account 1	Address 3
...
Client 0	Account n	Address 0

Client 0	Account 0	Address 0
Client 0	Account n	Address 1
Client 0	Account n	Address 2
Client 0	Account n	Address 3

To není nic, s čím by si Hibernate neporadil. Největší potíží ale je, že se v podobném výsledku nedá stránkovat. Neumíme totiž určit na kolikátém řádku leží požadovaný tisíc klient. Hibernate se to rozhodne řešit po svém. Napíše

WARNING: firstResult/maxResults specified with collection fetch; applying in memory!

Dává nám tím vědět, že se rozhodl načíst všechny klienty se všemi adresami a účty a požadované stránkování udělat v paměti. Kupodivu se mu všech 500 000 záznamů nevejde do paměti a dostaneme nehezky OutOfMemoryError. Zajímavé je, že pokud se o to samé pokusíme pomocí kritérií

```
session.createCriteria(Client.class).
    setFetchMode("addresses", FetchMode.JOIN).
    setFetchMode("accounts", FetchMode.JOIN).
    setFirstResult(1000).setMaxResults(50).
    addOrder(Order.asc("personId")).
    list();
```

Dostaneme stejný SQL dotaz jako prve, ale navíc doplněný stránkovací formulí `limit 1000, 50`. Což je ale naprosto špatně! Dostaneme chybný výsledek! (Pravděpodobně jde o chybu, ještě jsem se nestačil podívat, jestli je hlášena. Nicméně v JIŘE projektu se pár podobných chyb najde). Nebudu vás ale už víc napínat. Správné řešení je

Dávkové načítání

Upravíme třídu Client následovně

```
@Entity
@Table(name="client")
public class Client {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    @Column
    private Long personId;

    @Column
    private String name;

    @OneToMany(cascade=CascadeType.ALL, mappedBy="client")
    @BatchSize(size=10)
    private Set<Account> accounts;

    @OneToMany(cascade=CascadeType.ALL, mappedBy="client")
    @BatchSize(size=10)
    private Set<Address> addresses;

    //zbytek implementace vynechán
}
```

Pozornější z vás si jistě všimli nové anotace `@BatchSize`. Tou Hibernatu říkáme, že může i nadále kolekce načítat až na vyžádání. Pokud ale někdo bude chtít načíst kolekci adres pro prvního klienta, spolu s ní se načte devět dalších kolekcí, které ještě nebyly inicializovány (tzn. Hibernate už ví, že je možná bude potřebovat, ale ještě je nenačetl). Můžeme se tedy vrátit k původnímu HQL dotazu

```
session.createQuery("from Client c order by personId asc").  
    setFirstResult(1000).setMaxResults(50).list();
```

Dostaneme opět

```
select id, personId, name from client order by personId asc limit 1000, 50
```

a při čtení adres a účtů

```
select client_id, id, accountNumber  
    from account where client_id in (1000,1001,1002,1003,1004,...,1009)  
select client_id, id, street, streetNo, municipality, zipCode  
    from address where client_id in (1000,1001,1002,1003,1004,...,1009)  
.  
.  
.  
select client_id, id, accountNumber  
    from account where client_id in (1040,1041,1042,1043,1044,...,1049)  
select client_id, id, street, streetNo, municipality, zipCode  
    from address where client_id in (1040,1041,1042,1043,1044,...,1049)
```

Z $2n+1$ dotazů jsme to stáhli na $2n/10+1 = 11$ dotazů. Je zřejmé, že kdybychom do parametru `BatchSize` dali 50, stáhli bychom to až na 3 dotazy. V našem jednoduchém příkladě si to můžeme dovolit, v reálu bych radil trochu opatrnosti. Mohli bychom zbytečně načítat kolekce, které nakonec nebudeme potřebovat. Abych trochu zchladil vaši radost z toho jak to dobře dopadlo, musím vám říci, že ta užitečná anotace `BatchSize` už není součástí JPA. Je to rozšíření, které bude fungovat jen v Hibernate. Je možné, že ostatní O/R mapovací nástroje mají něco podobného, bohužel se to bude pravděpodobně jmenovat a používat jinak.

Pokud někomu vadí, že se kolekce stále načítají až na vyžádání, mohu poradit malý figl. Použijte toto:

```
List result =  
    session.createQuery("from Client c order by personId asc").  
        setFirstResult(fromRow).setMaxResults(toRow-fromRow).list();  
Hibernate.initialize(result);
```

Všechno v kolekci `result` bude nainicializováno. (*Updated 25.10.2006 – zde se nainicializuje jenom to co je v kolekci, adresy a účty ne. V tomto případě je použitý příkaz naprosto k ničemu.*)

Nakonec zase nějaké poučení na závěr (když já tak rád poučuji). Pokud jste si jisti, že potřebujete O/R mapování, tak se ho nebojte. Je spousta možností jak výkon vyladit. Před použitím je ale nutné se zamyslet a hlavně se podívat do ladících výpisů jaké SQL dotazy to vlastně vykonává. A hlavně pokud si nejste jisti v kramflecích stejně jako já, nebojte se udělat si malý experiment.

PS: Pokud máte nějaký zajímavý problém z oblasti O/R mapování, neváhejte mi ho napsat. Chtěl bych najít limity toho, co se dá pomocí Hibernate vyřešit. Hlavně se ovšem chci vytahovat, jak umím pomocí Hibernate vyřešit skoro všechno.