Hibernate ve JPA'da Cache Kabiliyeti

Önbellek kabiliyeti, ORM kullanan uygulamaların fonksiyonlarını yerine getirmeleri için olmazsa olmaz bir şart değildir. Ancak veritabanındaki işlemleri çoğunlukla sorgu şeklinde olan, kendisine ait meta-verinin büyük kısmını veritabanında tutan uygulamalar için ilgili verinin önbelleğe alınması hissedilir performans artışı sağlayacaktır. Başka bir ifade ile önbellek sadece performans optimizasyonu ile ilgilidir.

Giriş

Bu yazıda sizlere Hibernate ve JPA'da cache kabiliyetlerinden bahsedeceğim. Cache işlemi genel anlamda uygulama tarafından ihtiyaç duyulan ve veritabanında bulunan verinin uygulamaya daha yakın bir yerde tutulmasıdır. Bu yer hafıza veya dosya sistemi olabilir. Bu açıdan bakıldığında cache kelimesine Türkçe, "önbellek" karşılığını oldukça yerinde bulduğumu belirtmeliyim. Yazımın devamında da cache yerine önbellek kelimesini kullanacağım.

Önbellek kabiliyeti, ORM kullanan uygulamaların fonksiyonlarını yerine getirmeleri için olmazsa olmaz bir şart değildir. Ancak veritabanındaki işlemleri çoğunlukla sorgu şeklinde olan, kendisine ait meta-verinin (referans-lookup- verileri, sabitler, yerelleştirme bilgileri gibi.) büyük kısmını veritabanında tutan uygulamalar için ilgili verinin önbelleğe alınması hissedilir performans artışı sağlayacaktır. Başka bir ifade ile önbellek sadece performans optimizasyonu ile ilgilidir.

ÖnBellek Temelleri

Önbellek ile çalışan uygulamalar için iki temel konu söz konusudur. Bunlardan ilki önbellekteki verinin güncelliği, diğeri ise veriye eş zamanlı erişimdir. Birbiri ile de yakından alakalı bu durumların daha iyi anlaşılması için öncelikle uygulamalar içerisindeki değişik önbellek yapılarını incelemek gerekir. Veri yoğun uygulamalarda genellikle üç farklı düzeyde işlem gören önbellek türü vardır. Bunlar:

1. Transaction Kapsama Alanında Önbellekleme
2. Süreç(process) Kapsama Alanında Önbellekleme
3. Küme (cluster) Kapsama Alanında Önbellekleme

Transaction kapsama alanında yapılan önbellekleme işleminde her "Unit of Work" (UOW) için ayrı bir önbellek söz konusudur. "Unit of Work'de nedir?" diyen arkadaşların kafalarındaki soru işaretlerini kaldırmak için Hibernate için kısaca "UOW eşittir Session" diyebiliriz. Her UOW boyunca tek bir önbellek kullanılır. Farklı UOW nesnelerinin, bu durumda farklı transaction'ların aynı ön belleğe erişmesinin önüne geçilmiş olunur. UOW düzeyinde önbellek ile "repeatable read" transaction izolasyon düzeyi sağlanır. Transaction kapsama alanındaki ön belleğe birincil önbellek adı da verilmektedir.

Süreç kapsama alanında oluşturulan bir önbelleğe ise farklı transaction'lardan eş zamanlı erişim söz konusu olabilir. Bu durumun da doğal olarak transaction izolasyonuna direkt etkisi olur. Küme kapsama alanında oluşturulan bir önbellek yapısında ise birden fazla sürecin veya bilgisayarın önbelleğe eş zamanlı erişimi olabilir. Kümelenmiş önbellek grubu veri tutarlılığını sağlamak için kendi aralarında bir tür haberleşme mekanizmasına ihtiyaç duyar. Önbelleğe atılan veri ağ üzerindeki her bir önbellek birimine yazılır. ORM çözümleri yukarıda sıraladığımız önbellek düzeylerinden bir veya daha fazlasını aynı anda kullanıma sunabilir. Süreç ve küme kapsama alanlarında oluşturulan önbelleğe ise ikincil önbellek adı verilmektedir. Aslında bu yazıda anlattığımız konuların büyük bir kısmı ikincil önbellekleri ilgilendirmektedir.

Transaction kapsama alanı boyunca aynı veritabanı kimliğine sahip entity'ler için önbellek aynı Java nesnesini döner. Süreç ve küme kapsama alanları boyunca ise aynı Java nesnesini dönmek uygulama düzeyinde kilit mekanizmaları gerektireceği için nesnelerin aynılığının, içerdikleri değerler üzerinden tespit edilmesi söz konusudur. Bu web uygulamaları açısından da istenilen bir durumdur. Çünkü web uygulamalarında da birden fazla web isteğinin, başka bir ifade ile birden fazla thread'in tek bir Java nesnesine erişmesi senkronizasyon gerektirecektir. Bu da uygulamaların

performanslarına olumsuz biçimde etki edecektir. Ayrıca uygulama düzeyinde bir kilit mekanizması veritabanlarında hali hazırda mevcut olan transaction senkronizasyon kabiliyetlerinin de sıfırdan tekrar hayata geçirilmesi demektir. Bu da çok tercih edilecek bir şey olmayacaktır.

Sonuç olarak süreç ve küme kapsama alanlarındaki önbellek kullanımlarında bir UOW'den erişilen entity, diğer bir UOW'den de farklı bir nesne referansı ile erişilebilecektir. Bu durumda da her iki UOW'de yapılan işlemlerin transaction izolasyonuna menfi etkileri kaçınılmazdır.

Ne Zaman ÖnBellek Kullanmalı?

Önbellekte problemsiz biçimde tutulabilecek verinin temel özelliklerini şu şekilde sıralayabiliriz.

1. Veri çok nadir değişir. Veri üzerinde yapılan işlem çoğunlukla sorgulama türündedir.
2. Veri uygulamaya özeldir, veritabanına uygulamanın kontrolü dışında legacy sistemlerden erişim söz konusu değildir.
3. Veri kritik öneme sahip değildir.

Önbellekte tutulması problem yaratabilecek verinin temel özellikleri ise şöyledir.

1. Veri sık sık güncellenir.
2. Veriye uygulama dışından da erişim söz konusudur.
3. Veri, finansal veri gibi, kritik öneme sahiptir.

Pek çok kurumsal uygulamada aşağıdaki niteliklere sahip türden veriyi görmek mümkündür.

1. Uygulama içerisinde az sayıda Java nesnesine karşılık gelen
2. Diğer nesnelerle M:1 türden ilişkiye giren
3. Verinin az veya hiç güncellenmediği

Bu tür veriye referans(lookup) veri adı da verilir. Referans veri süreç veya küme düzeyinde önbelleklenmeye en uygun türde veridir.

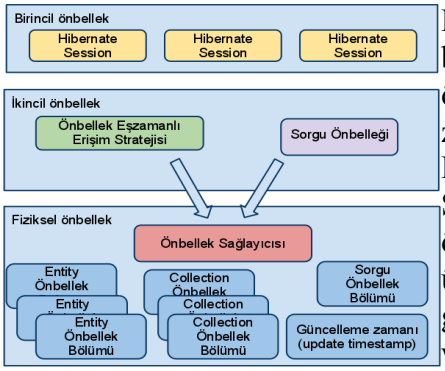
Yukarıda belirttiğimiz kriterler ışığında önbellekte tutulabilecek türden veri tespit edildikten sonra yapılması gereken ikinci adım bu verinin önbellekte hangi transaction izolasyon düzeyinde tutulacağına karar vermektir. Mevcut ikincil önbellek çözümleri bütün transaction izolasyon düzeylerini desteklemezler.

JPA 1.0

JSR 220 EJB 3.0 spesifikasyonun JPA bölümünde önbellek ile ilgili herhangi bir konuya değinilmemiştir. Bu nedenle önbellek kabiliyeti JPA 1.0 kullanan uygulamalar için tamamen ORM çözümüne özel biçimde ayarlanıp, kullanılabilmektedir.

Hibernate

Hibernate hem birincil hem de ikincil önbellek kabiliyetlerine sahiptir. Entity'lere karşılık gelen verinin doğrudan erişimi ve güncellenmesi boyunca geçerli bir önbellek kabiliyetinin yanında sorgu sonuçlarının da önbelleklenmesi mümkündür. Aşağıda genel hatları ile Hibernate önbellek mimarisini gösteren bir şekil mevcuttur.



Hibernate'de birincil önbellek zorunludur. Hibernate Session birincil önbellek rolünü üstlenir. Ömrü genellikle ya web isteği, ya da bir conversation boyuncadır.

İkincil önbellek ise isteğe bağlı olarak ayarlanır. Önbellek eş zamanlı erişim stratejisi, transaction izolasyon düzeyini kontrol eder. Sorgu önbelleği, sorguların da önbelleklenmesini sağlar.

Fiziksel önbellek herhangi bir önbellek çözümüne karşılık gelir. İşlemler önbellek sağlayıcısı üzerinden yürütülür. Fiziksel önbellek içerisinde her bir entity ve 1:M,M:N collection ilişkilerinin tutulduğu önbellek alanları vardır. Sorgu sonuçlarının önbellekte tutulması için ise hem sorgu önbellek bölümü, hem de güncelleme zamanı bölümü kullanılır. Veri önbellek bölümlerinde “disassembled” biçimde tutulur. Bunu Java serialization'a benzetmek mümkündür.

Hibernate'de ikincil önbellek süreç veya küme düzeyinde olabilir. Aynı SessionFactory nesnesinden oluşturulan bütün Hibernate Session nesneleri ortak bir ikincil ön belleği paylaşırlar. Entity ve collection ilişkilerinin ikincil önbellekte tutulup tutulmayacağı sınıf ve ilişki düzeyinde

belirlenebilir. Her bir sınıf ve collection ilişkisi için farklı türde eş zamanlı erişim stratejileri ayarlanabilir.

Eş Zamanlı Erişim Stratejileri

Hibernate entity sınıfları için dört farklı eş zamanlı erişim stratejisi sunar.

1. TRANSACTIONAL: Sadece JEE container içerisindeyken kullanılabilir. Repeatable read izolasyon düzeyine kadar erişim kontrolü sağlar.
2. READ_WRITE: Sadece süreç kapsamlı önbelleklemede kullanılabilir. Verinin zaman içerisinde çok sık değişmediği durumlar için uygundur. Veri üzerinde değişiklik yapılması ön belleğin güncellenmesini tetikler. Read committed izolasyon düzeyinde erişim kontrolü sağlar.
3. NONSTRICT_READ_WRITE: Bir önceki eş zamanlı erişim stratejisine benzer. Ancak veritabanı ve önbellek arasında herhangi bir tutarlılık gözetmez. Veri zaman aşımına uğrayana değin güncel kabul edilir. Bu nedenle güncelliğini yitirmiş verinin uygulamaya dönülmesi söz konusu olabilir.
4. READ_ONLY: Salt okunur veri için uygundur. Referans veri bu kategoriye girer. Salt okunur olarak işaretlenmiş entity sınıfları Hibernate Session tarafından "dirty-check" işlemine tabi tutulmazlar.

Örneğin az sayıda nesnenin olduğu, bunların nadiren güncellendiği, ve nesnelerin birden fazla kullanıcı tarafından paylaşıldığı bir durumda READ_WRITE erişim kontrolü kullanılması daha uygundur. Bu durumda nesneler üzerinde yapılan değişiklikler anında diğer kullanıcılara da yansıtılabilir. Eğer NONSTRICT_READ_WRITE erişim kontrolü kullanılırsa sadece zaman aşımı değerine bakılır. Bu da değişikliklerin anında diğer kullanıcılara yansıtılamamasına neden olabilir.

Önbellek Bölgeleri

Fiziksel önbellek içerisinde önbelleklemeye tabi tutulacak sınıflar, collection ilişkileri ve sorgular için oluşturulan bölgelerdir. Önbellek verisi bu bölgelerde tutulur. Veri "disassembled" formatta tutulur. Bu formata göre nesnenin, id ve ilkel property değerleri bölge içerisinde yer alır. Sınıf önbellek bölümlerinin isimlendirmesi sınıf tam adı ile yapılır. Collection önbellek bölümlerinin isimlendirmesi ise ilişkinin bulunduğu sınıfın tam adı + ilişkinin adı şeklinde olur. Her önbellek bölgesinin kendine has ayarları söz konusudur. Bu ayarlar fiziksel ön belleğin varsayılan değerlerinden de elde edilebilir.

Önbellek bölgeleri SessionFactory düzeyindedir. Eğer uygulama içerisinde birden fazla SessionFactory veya persistence unit kullanılması söz konusu ise bölge isimlerinde çakışmalar söz konusu olacaktır. Bunun önüne geçmek için hibernate.cache.region_prefix konfigürasyon property'si kullanılabilir.

EHCache ile İkincil Önbellek Kullanımı

Öncelikle ikincil önbellek sağlayıcısını belirleyen **hibernate.cache.provider_class** konfigürasyon property'si **hibernate.cfg.xml** içerisinde set edilmelidir.

```
<property
name="hibernate.cache.provider_class">
org.hibernate.cache.EhCacheProvider
</property>
```

Classpath'de **ehcache.xml** isimli bir dosya oluşturulur ve önbellek bölgeleri tanımlanır. Bunun en kolay yolu **ehcache.jar** içerisindeki **ehcache-failsafe.xml** dosyasının alınıp kopyalanmasıdır. Bu dosya içerisindeki varsayılan bölge tanımı kullanılarak ilgili bölge tanımları kolayca türetilebilir.

```
<cache name="examples.Foo"
  maxElementsInMemory="30"
  eternal="false"
  timeToIdleSeconds="0"
  timeToLiveSeconds="0"
  overflowToDisk="true"
  maxElementsOnDisk="1000"
  diskPersistent="false"
  diskExpiryThreadIntervalSeconds="120"
  memoryStoreEvictionPolicy="LRU"
/>
```

Yukarıda Foo sınıfı için bir önbellek bölge tanımı yapılmıştır. Şimdi önemli birkaç attribute'u inceleyelim. Eğer **maxElementsInMemory** değeri uygulama içerisinde var olabilecek Foo

nesnelerinin toplam sayısından fazla ise önbellekten çıkarma (evict) işlemi kendiliğinden devre dışı kalır. Ancak evict hala zaman aşımı parametreleri üzerinden devam etmektedir. Son erişimden itibaren zaman aşımını **timeToIdleSeconds** kontrol eder. Verinin ön belleğe ilk konmasından itibaren zaman aşımı ise **timeToLiveSeconds** ile kontrol edilir. Zaman aşımı üzerinden de evict işlemini devre dışı bırakmak isterseniz **eternal** attribute değerini true yapmanız yeterlidir.

Hibernate'in varsayılan ayarlarında hibernate.cfg.xml ve sınıflardaki önbellek tanımları ikincil ön belleği otomatik olarak devreye sokar. Eğer bu tanımlara rağmen ikincil ön belleğin devreye girmesi istenmiyor ise hibernate.cfg.xml içerisinde

```
<property
name="hibernate.cache.use_second_level_
cache">false</property>
```

yapılmalıdır. Şimdi de Foo sınıfı içerisindeki önbellek ayarlarına bakalım.

```
package examples;

@Entity
@Cache(usage=CacheConcurrencyStrategy.R
EAD_WRITE)
public class Foo {
...
@OneToMany(cascade=CascadeType.ALL)
@Cache(usage=CacheConcurrencyStrategy.R
EAD_WRITE)
private Set<Bar> bars = new
HashSet<Bar>();
...
}
```

@Cache anotasyonu sınıf düzeyinde ve ilişki düzeyinde kullanılmıştır. Foo sınıfı READ_WRITE erişim kontrolüne tabi tutulmuştur. Foo sınıfı içerisindeki bars 1:M ilişki üzerinde de @Cache anotasyonu ile önbellekleme yapılması istenmiştir. Bu durumda ehcache.xml içerisinde examples.Foo.bars önbellek bölgesi de oluşturulmalıdır.

Artık uygulama içerisinde Hibernate Session load ve get metotları ile Foo nesnelerine erişildiği vakit ikincil önbellek devrede olacaktır. Foo nesneleri üzerinde save,persist,update,merge ve delete işlemleri ise önbelleğin temizlenmesine neden olacaktır.

Hibernate Sorgularında İkincil Önbellek Kullanımı

Hibernate sorgularının önbellek kullanması için öncelikle hibernate.cfg.xml içerisine

```
<property
name="hibernate.cache.use_query_cache">t
rue</property>
```

eklenmelidir. Ancak bu tek başına yeterli olmayacaktır. Varsayılan durumda bütün sorgular önbelleklemeyi gözardı etmektedir. Bunun aşılması için **Query** ve **Criteria** nesnelerinin **setCacheable()** metodunu çağırarak değeri true olarak set etmek gerekir. Hibernate'i JPA üzerinden kullanırken ise JPA Query nesnesinin setHint metodu **setHint("org.hibernate.cacheable",true)** şeklinde çağrılmalıdır.

Hibernate sorgularının ikincil önbellekte tutulması ile ilgili iki tane bölüm vardır.

```
<cache
name="org.hibernate.cache.StandardQuery
Cache"
      ...
/>
<cache
name="org.hibernate.cache.UpdateTimesta
mpsCache"
      ...
/>
```

StandardQueryCache bölgesinde çalıştırılan sorgu ifadesi, parametreleri ile birlikte tutulur. Bunun yanında sorgu sonucu dönen ResultSet'de tutulur. Ancak entity nesnelerinin sadece id değerleri tutulur. Eğer sorgu sonucu scalar değerler içeriyor ise o zaman bütün değerler önbellekte tutulur. **UpdateTimestampsCache** ise Hibernate tarafından önbellekte tutulan sorgunun güncelliğini yitirip yitirmediğini kontrol etmek için kullanılır.

Hibernate sorgularının önbelleklenmesi sanılanın aksine çok fazla performans artışı sağlamayabilir. Öncelikle dikkat edilmesi gereken, önbellekte tutulacak sorguların aynı parametrelerle kaç defa çalıştırılacağıdır. Diğer bir durum ise sorgu sonucunda dönen entity nesnelerin güncellenme sıklığıdır. Eğer entity nesnelerin herhangi biri üzerinde Session save,persist,update,merge,delete işlemlerinden birisi yapılırsa sorgu ön belleği güncelliğini yitirecektir. Böyle bir durumda sorgu sonuçları tekrar veritabanından yüklenir.

CacheMode ile Session Düzeyinde Önbellek Kontrolü

Hibernate, CacheMode seçenekleri ile entity nesneler ile çalışırken Session düzeyinde ikincil önbellek ile etkileşimi kontrol etmeye izin verir. Mevcut CacheMode seçenekleri şu şekildedir.

1. NORMAL: Mevcut moddur. Veri önbellekten okunur, ön belleğe yazılır.
2. IGNORE: Veri okuma sırasında önbellekten okunmaz, doğrudan veritabanından okunur, güncelleme sırasında önbellekteki veri invalidate edilir.
3. GET: Veri okuma sırasında önbellek devrededir. Ancak veritabanından okunan veri önbellekte yoksa ön belleğe konmaz. Güncelleme önbellekteki veriyi invalidate eder.
4. PUT: Veri okuma sırasında önbellek devre dışıdır. Ancak okunan veri önbellekte yoksa ön belleğe yerleştirilir.
5. REFRESH: PUT moduna benzer. Tek farkı **hibernate.cache.use_minimal_puts** özelliğinin gözardı edilmesidir. Bu sayede önbellekteki veri her seferinde yenilenmiş olur.

NORMAL ve IGNORE dışındaki seçeneklerin çok fazla kullanım senaryosu yoktur.

JPA 2.0

JPA 2.0 ile gelen yeniliklerden birisi de ikincil önbellek desteğidir. Genel olarak yukarıda anlattığım Hibernate ile ikincil önbellek kabiliyetine çok benzer bir konfigürasyon ve kullanıma sahiptir. JPA'da ikincil önbellek kabiliyeti opsiyoneldir, persistence sağlayıcısı ikincil önbellek kabiliyetine sahip olmayabilir. Böyle bir durumda JPA 2 önbellek ayarları sessizce göz ardı edilir.

İkincil ön belleği ayarlamak için **persistence.xml** içerisindeki **shared-cache-mode** elemanının

1. NONE
2. ALL
3. ENABLE_SELECTIVE
4. DISABLE_SELECTIVE
5. UNDEFINED

değerlerinden birisine set edilmesi gerekir.

ALL değeri bütün entity sınıflar için ikincil ön belleği aktive eder. NONE değeri ise ikincil ön belleği devre dışı bırakır. ENABLE_SELECTIVE ve DISABLE_SELECTIVE değerleri ile **@Cacheable** anotasyonu ile birlikte kullanılır. Eğer shared-cache-mode değeri ENABLE_SELECTIVE ise sadece @Cacheable(true) anotasyonuna sahip entity sınıfları için önbellekleme aktif olur. DISABLE_SELECTIVE değerinde ise @Cacheable(false) ile işaretlenmeyen bütün entity sınıfları otomatik olarak önbelleklemeye tabi tutulur.

EntityManager düzeyinde set edilen **javax.persistence.cache.retrieveMode** ve **javax.persistence.cache.storeMode** property'leri vasıtası ile EntityManager ikincil önbellek etkileşimi kontrol edilebilir.

RetrieveMode find() metodu ve JPAQL sorguları sırasında EntityManager'ın önbellek ile ilişkisini düzenler. İki farklı değere sahiptir.

1. CacheRetrieveMode.USE: entity nesnenin önbellekten okunmasını söyler.
2. CacheRetrieveMode.BYPASS: önbelleğin göz ardı edilip verinin doğrundan veritabanından okunması gerektiğini belirtir.

StoreMode ise veritabanından okuma yapıldığında ve transaction'ın commit edildiği vakit EntityManager ile önbellek arasındaki etkileşimi yönetir. Üç farklı değere sahiptir.

1. CacheStoreMode.USE: veritabanından okuma yapıldığında veya transaction commit olduğundan verinin ön belleğe yazılmasını söyler. Halihazırda önbellekte bulunan bir veri ise herhangi bir şey yapmaz.
2. CacheStoreMode.BYPASS: verinin ön belleğe yazılmamasını sağlar.
3. CacheStoreMode.REFRESH: veritabanından okuma olduğunda veya transaction commit anında verinin ön belleğe yazılmasını söyler, ancak USE'dan farklı olarak veritabanından okuma yapıldığında önbellekte daha evvel mevcut olan verinin yenilenmesini sağlar.

RetrieveMode ve StoreMode property'lerinin devreye girmesi için öncelikle ikincil önbelleğin hem genel olarak hem de ilgili entity sınıf düzeyinde aktif olması gerektiği bilinmelidir.

JPA 2'de sorguların önbelleklenmesi ile ilgili ise herhangi net bir kabiliyet ortaya konmamıştır. Sorgularla ilgili olarak yine ORM çözümüne spesifik özelliklerin kullanılması söz konusudur.

Sonuç

Önbellek kabiliyeti uygulamanın fonksiyonlarını yerine getirebilmesi için olmazsa olmaz bir ORM özelliği değildir, ancak uygun yerlerde kullanıldığı vakit hissedilir bir performans artışı sağlayacağı da bilinmelidir. Ancak ikincil önbelleklerle çalışırken transaction izolasyon konusuna ve verinin güncelliğini yitirmesi durumlarına dikkat etmek gerekir. İkincil ön belleğin performans amaçlı kullanılmasının yanı sıra Hibernate ile çalışırken fetch stratejisinin belirlenmesinde ve lazy problemlerini çözmek için de kullanılabileceğini belirtmeliyim. Ancak bu Hibernate ve fetch stratejilerinin incelendiği bir başka yazının konusu olabilir.

Yazar: ODTÜ Bilgisayar Mühendisliği'nden 1999 yılında mezun olan Kenan Sevindik o dönemden bu yana pek çok büyük ölçekli kurumsal yazılım projesinde görev yapmıştır. Halihazırda kurumsal Java teknolojileri ile yazılım geliştirme, eğitim ve danışmanlık hizmeti sunan yazarımızın verdiği eğitimlere http://www.java-egitimleri.com adresinden ulaşabilirsiniz. Değişik ortamlarda teknoloji içerikli konuşma ve sunumlar da yapan yazarımız edindiği bilgi birikimi ve deneyimleri http://www.harezmi.com.tr/blog adresinden sanal alemde sizlerle paylaşmaktadır. Kendisi ile iletişime geçmek için ksevindik@gmail.com adresine mesaj gönderebilirsiniz.