Anyframe Cache Plugin

Version 1.1.0

I.Introduction

cache plugin은 Spring 3.1부터 제공되는 Cache 추상화 기능을 활용하여, Terracotta의 Ehcache [http://ehcache.org/]를 간편하게 사용하는 방법을 가이드하기 위한 샘플 코드와 이를 활용하는데 필요한 참조 라이브러리들로 구성되어 있다.

Installation

Command 창에서 다음과 같이 명령어를 입력하여 cache plugin을 설치한다.

```
mvn anyframe:install -Dname=cache
```

installed(mvn anyframe:installed) 혹은 jetty:run(mvn clean jetty:run) command를 이용하여 설치 결과를 확인해볼 수 있다.

Plugin Name	Version Range
query [http://dev.anyframejava.org/docs/anyframe/plugin/optional/query/1.6.0/reference/htmlsingle/query.html]	2.0.0 > * > 1.4.0

II.Cache

Cache Service는 Spring 3.1 부터 제공되는 Cache 추상화 기능과 Terracotta의 Ehcache [http://ehcache.org/]를 기반으로 개발되었으며, 상태를 공유할 수 있는 객체를 cache하는 기능을 제공하는 서비스이다. 변경이 자주 일어나지 않지만 사용빈도가 높고 생성하는데 비용이 많이 드는 객체일 경우, Cache를 이용하면 다음과 같은 장점을 얻을 수 있다.

- 자주 접근하는 데이터를 매번 데이터베이스로부터 fetch할 필요가 없으므로 오버헤드가 줄어든다.
- 객체를 매번 생성하지 않기 때문에 메모리를 효율적으로 사용할 수 있다.

1.Configuration

Spring Framework 3.1 버전부터 트랜잭션과 유사한 형태로 Spring 기반의 어플리케이션에 Cache 기능을 투명한 형태로 적용하는 기능을 지원한다. Cache 추상화 기능은 기존 코드 수정을 최소화하고, Cache 기능을 일관성있게 사용할 수 있도록 지원하는 기능이다. 자바 메소드에 대한 Cache 기능 수행시, 동일 파라메터 값으로 수행된 메소드 (CPU 또는 IO 처리에 많은 비용이 소요되는 메소드(expensive method)의 경우 매번 재실행하지 않고 동일한 결과 값을 얻을수 있도록 Cache 처리된 값을 반환할 수 있도록 한다. 단, 동일 파라메터 값으로 반복 실행시 동일한 결과 값이 반환되는 메소드에만 사용할 수 있음에 유의하도록 한다.

Cache 추상화 기능을 사용하기 위해서 개발자는 다음의 2가지 작업을 수행해야 한다.

1. Cache 선언-Cache 적용 대상 메소드와 Cache 정책을 정의한다.

2. Cache 설정-Caching 대상을 저장하여 사용할 Cache 저장소에 대한 설정을 정의한다.

Cache 서비스는 Spring Framework의 다른 서비스와 마찬가지로 Cache 구현체를 Spring Framework에서 제공하는 것이 아니고 추상화하여 제공하고 있다. 추상화로 인해서 개발자가 Cache 관련 로직을 작성할 필요는 없어지며, 실제 Cache Data를 저장할 저장소 또한 Spring에서 제공하지 않는다. 즉 Cache Data를 저장할 실제 저장공간은 별도로 필요하게 된다. 현재 Spring Framework에서는 기본적으로 2가지 구현체(JDK의 java.util.concurrent.ConcurrentMap 기반 또는 Ehcache 기반)를 제공하고 있으며 이외 다른 cache stores/provider가 필요한 경우 추가하여 사용할 수 있다.

1.1.Declarative annotation-based caching (선언적인 Annotation 기반 캐싱)

@Cacheable과 @CacheEvict 2가지 Java Annotation을 통해서 캐싱 대상 메소드를 정의할 수 있다.

@Cacheable은 cache population(생성)을, @CacheEvict는 cache eviction(제거)을 수행하도록 한다.

1.1.1.@Cacheable annotation

Cache 적용 대상 메소드를 선언할 때 사용한다. 메소드 결과가 Cache 저장소에 저장되고 동일한 파라미터로 메소드를 호출할 경우 메소드를 실제로 재실행시키지 않고 Cache 저장소에 저장된 값을 반환한다.

가장 간단히 Cache 이름을 설정하여 선언할 수 있다.

```
@Cacheable("books")
```

Cache는 기본적으로 key-value 형태로 Cache 저장소에 저장된다.

- Default key Generation (기본 키 생성 방식)

  Cache 저장소에 저장된 메소드 결과값을 호출할때 캐시를 찾을 적절한 키가 필요하므로 아래와 같은 알고리즘을 사용한다.

  - 메소드 파라미터가 없으면 0을 키로 사용한다.
  - 메소드 파라미터가 1개면 파라미터 인스턴스를 키로 사용한다.
  - 메소드 파라미터가 2개 이상이면 모든 파라미터의 hash값으로 계산된 키를 사용한다.

- 다른 기본키 생성기를 사용하려면, org.springframework.cache.KeyGenerator 인터페이스를 구현하도록 한다.

- Custom Key Generation Declaration (커스텀 키 생성 선언)

Cache 적용 대상 메소드의 파라미터가 여러개인 경우 Cache 키로 사용할 파라미터를 SpEL을 사용하여 설정할 수 있다.

메소드 파라미터 중에 Cache 관련된 파라미터가 일부이고 나머지 파라미터의 경우 메소드 로직 내에서 사용되는 경우에 사용한다.

```
@Cacheable(value="movie", key="#movieId")
public Movie findMovie(String movieId)
```

- Conditional caching (조건적인 캐싱)

  condition 속성의 조건에 맞는 경우에만 Cache 저장소에 저장해야 하는 경우가 있을 때 사용한다.

  condition 속성 내에 SpEL 표현식을 사용하여 true 혹은 false 결과값이 나오도록 조건문을 작성할 수 있다.

```
@Cacheable(value="movie", condition="#movieId.length < 16")
public Movie findMovie(String movieId)
```

- Available caching SpEL evaluation context (사용 가능한 캐싱 SpEL 평가식 컨텍스트)

명칭	사용 위치	설명	예제
methodName	root 오브젝트	호출되는 메소드명	#root.methodName
method	root 오브젝트	호출되는 메소드	#root.method.name
target	root 오브젝트	호출되는 대상 오브젝트	#root.target
targetClass	root 오브젝트	호출되는 대상 클래스	#root.targetClass
args	root 오브젝트	대상을 호출할 때 사용되는 인자(배열)	#root.args[0]
caches	root 오브젝트	현재 메소드가 실행될 때에 대응되는 캐시의 집합(컬렉션)	#root.caches[0].name
argument name	평가식 컨텍스트	메소드 인자명. 만약 이름을 사용할 수 없다면(예: 어떠한 debug 정보도 없는 경우), 인자명은 a<#arg> 로 대치하여 사용가능하며 #arg 는 인자의 인덱스의 약자를 나타낸다(0부터 시작).	iban 또는 a0 (별칭으로 p0 또는 p<#arg> 표기를 사용할 수 있다).

1.1.2.@CachePut annotation

메소드가 항상 수행되고 그 결과값 또한 cache에 저장된다. 즉, @cacheable와 다르게 항상 메소드를 수행하여 cache 정보를 업데이트한다.

```
@CachePut(value = "genre", key = "#genre.genreId")
public Genre updateAndGet(Genre genre)
```

1.1.3.@CacheEvict annotation

Cache 추상화는 캐시 생성 기능과 삭제 기능을 함께 제공한다. 삭제 기능은 변경된 Cache 데이터나 사용하지 않는 Cache 데이터를 Cache 저장소에서 제거할 때 사용된다. @Cacheable과 반대로 cache eviction 삭제를 실행하는 메소드에 설정한다. cache로부터 데이터를 제거하도록 하는 메소드이다.

allEntries 설정 추가 시 키값에 해당하는 캐시 엔트리 하나만 삭제하는 것이 아닌 전 cache 저장소 삭제 기능을 제공한다. 리턴타입이 없는 void 메소드에 대해서도 @CacheEvict 를 사용할 수 있다.

```
@CacheEvict(value = "genreList", allEntries=true)
public void remove(String genreId)
```

메소드 실행된 이후에 캐시 삭제 기능이 수행되는 것이 디폴트이나, beforeInvocation=true 설정을 통해 메소드 실행 전에 캐시 삭제 기능을 수행시킬 수도 있다.

1.1.4.@Caching annotation

1개의 동일한 메소드에 여러 annotation(@Cacheable, @CachePut, @CacheEvict)을 적용할 수 있다.

```
@Caching(evict = { @CacheEvict(value = "genre", key = "#genre.genreId"),
        @CacheEvict(value = "genreList", allEntries = true) })
public void update(Genre genre) throws Exception {
```

1.1.5.Enable caching annotations

cache annotation을 정의한다하더라도 자동으로 관련 action들이 수행되지는 않는다. Spring framework 의 다른 기능들처럼 선언적인 방법으로 기능을 enable 혹은 disable 시킬 수 있다. <cache:annotation-driven/> 설정이 바로 그것이다. 이 namespace는 caching behavior가 AOP를 통해 어플리케이션에 영향을 줄 수 있는 다양한 옵션을 설정할 수 있게 한다. 이 설정은 <tx:annotation-driven/>과 목적 면에서 매우 유사하다.

- <cache:annotation-driven/> 설정 어트리뷰트 표

속성	기본값	설명
cache-manager	cacheManager	사용할 캐시 매니저의 이름. 캐시 매니저의 이름이 기본값(cacheManager) 가 아닌 경우에만 필수이다.
mode	proxy	기본 모드인 "proxy"는 annotation된 bean을 Spring 의 AOP 기능을 통하여 proxy 처리한다. (proxy의 의미처럼, proxy를 통해 호출되는 메소드 호출에 대해 적용된다.) 반면 "aspectj" 모드는 Spring의 AspectJ 캐싱 aspect 를 적용 대상 클래스에 주입하게 되는데, 메소드 호출시 적용하기 위해서 대상 클래스의 bytecode 를 수정한다. 단, AspectJ 는 클래스패스내에 spring-aspects.jar 가 필요하며, load-time weaving(또는 complie-time weaving) 기능이 활성화되어 있어야 한다.
proxy-target-class	false	proxy mode 에서만 사용가능. @Cacheable 이나 @CacheEvict 이 annotation 된 클래스에 대하여 어떤 유형의 캐싱 프록시를 생성할 지 제어한다. 만약 proxy-target-class 속성 값이 true 이면 클래스 기반의 프록시가 생성되며, false 이거나 값이 생략되어 있으면 JDK interface 기반의 프록시가 생성된다.
order	Ordered.LOWEST_PRECEDENCE	@Cacheable 과 @CacheEvict 이 annotation 된 bean 에 적용될 cache advice 의 순서를 정의한다. 순서를 정의하지 않으면, AOP 서브시스템이 advice 의 순서를 결정하도록 한다는 의미이다.

<cache:annotation-driven/> 사용 시 유의점

<cache:annotation-driven/>은 동일한 application context 상에 있는 beans 들을 대상으로 @Cacheable/@CacheEvict를 찾는다. 즉 WebApplicationContext 에 <cache:annotation-driven/>를 설정했다면 controller bean들을 대상으로만 @Cacheable/@CacheEvict를 찾는다는 것이다.(service bean들은 찾지 않음)

- Method visibility

Proxy를 사용하는 경우 @Cache* annotation들을 public 메소드에만 적용해야 한다. 만약 protected/private/package-visible 메소드에 annotation을 적용한 경우 에러가 발생하지는 않지만 설정된 caching settings들이 동작하지 않는다. 이런 경우 AspectJ를 사용하여 bytecode 자체를 변경시켜서 사용하도록 한다.

1.1.6.Custom annotation

템플릿 메카니즘처럼 중복된 Cache Annotation 설정(key, condition)을 제거하고 spring package를 소스 코드 내에서 사용하지 않을 수 있다. @Cache* annotation을 meta-annotation으로 사용해서 새로운 annotation을 만들고 이 custom annotation을 실제 메소드에 적용시킬 수 있다. 캐시 관련 중복 선언을 제거할 때 유용하게 사용될 수 있다.

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
@Cacheable(value="movie", key="#movieId")
public @interface SlowService {
}
```

위에서 보는바와 같이 @Cacheable 로 annotation된 SlowService 자체를 annotation 으로 정의하면 아래의 코드를

```
@Cacheable(value="movie", key="#movieId")
public Movie findMovie(String movieId)
```

다음과 같이 대체할 수 있다.

```
@SlowService
public Movie findMovie(String movieId)
```

@SlowService 가 Spring annotation이 아니더라도, 컨테이너가 자동적으로 런타임시 선언 내용을 인식한다. 단, annotation-driven 이 활성화되어 있어야 한다.

1.2.Declarative XML-based caching (선언적인 XML 기반 캐싱)

자바 코드에 annotation을 설정하지 않고 XML 설정을 통해 Cache 기능을 선언하여 사용할 수 있다. 메소드에 annotation 설정을 하는 것 대신에 target method를 정의하고 캐싱 기능 설정을 할 수 있다. 선언적인 트랜잭션 관리 advice 설정하는 것과 매우 유사하다.

```
<!-- the service we want to make cacheable -->
<bean id="movieService"
 class="org.anyframe.plugin.cache.moviefinder.service.impl.MovieServiceImpl"/>

<!-- cache definitions -->
<cache:advice id="cacheAdvice" cache-manager="cacheManager">
  <cache:caching cache="movie">
    <cache:cacheable method="findMovie" key="#movieId"/>
    <cache:cache-evict method="remove" all-entries="true"/>
  </cache:caching>
</cache:advice>

<!-- apply the cacheable behaviour to all BookService interfaces -->
<aop:config>
  <aop:advisor advice-ref="cacheAdvice" pointcut="execution(*
 org.anyframe.plugin.cache.moviefinder.service.impl.MovieServiceImpl.*(..))"/>
```

```
</aop:config>
...
// cache manager definition omitted
```

1.3.Configuring the cache storage (Cache 저장소 설정)

Cache 추상화 기능은 기본적으로 2가지 저장소(storage)를 제공한다. 하나는 JDK ConcurrntMap이고 다른 하나는 Ehcache 라이브러리이다. 이것들을 사용하기 위해서는 단순히 적절한 CacheManager를 선언하면 된다. CacheManager는 Cache들을 관리하는 엔티티이고 storage로 부터 이 Cache들을 가져오기 위해 사용된다.

1.3.1.JDK ConcurrentMap 기반 Cache

JDK ConcurrentMap 기반 Cache 구현체는 org.springframework.cacahe.cocurrent 패키지에 있다. ConcurrentHashMap을 캐시 저장소로 사용하는 구현체다.

Cache 저장소로 ConcurrentHashMap 사용

- 장점:cache scales well, 매우 빠름
- 단점:관리 기능, persistence capabilities, eviction contract 제공하지 못함

```
<bean id=cacheManager" class="org.spring....cache.support.SimpleCacheManager"
  <property name="caches">
    <set>
      <bean class="org...ConcurrentMapCacheFactoryBean" p:name="default"/>
  ...
```

1.3.2.Ehcache 기반 Cache

Ehcache 구현체는 org.springframework.cache.ehcache 패키지에 있다.

```
<bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager"
 p:cache-manager-ref="ehcache"/>

<!-- Ehcache library setup -->
<bean id="ehcache" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"
 p:config-location="ehcache.xml"/>
```

ehcache 특화된 설정은 모두 ehcache.xml 파일에 작성한다.

1.3.3.Dummy Cache

실제 backing cache를 설정하지 않고 cache 정의를 한 경우 런타임 시에 에러없이 테스트 등을 수행하기 위해서 간단한 dummy cache를 사용할 수 있다. 실제 캐싱 기능이 동작하지 않는다. (매번 캐시 대상 메소드가 수행된다.) addNoOpCache property설정을 통해 수행시킬 수 있다.

```
<bean id="cacheManager" class="org.springframework.cache.support.CompositeCacheManager">
  <property name="cacheManagers"><list>
    <ref bean="jdkCache"/>
    <ref bean="gemfireCache"/>
    </list></property>
  <property name="addNoOpCache" value="true"/>
```

```
</bean>
```

1.4.Plugging-in different back-end caches(다른 백엔드 캐시 플러그인해서 사용하기)

Backing store로 사용되는 Cache 제품은 매우 많다. 이들을 사용하기 위해서 CacheManager와 Cache 구현체를 제공해야 한다. Spring에서 제공하는 AbstractCacheManager 클래스를 이용해서 실제 매핑할 때 자주 반복되는 보일러플레이트 코드를 줄여서 개발할 수 있다.

2.Samples

다음은 CacheService의 속성 설정 및 테스트 코드에 대한 예제이다.

- **Configuration**

  다음은 CacheService에서 사용할 Cache의 속성을 정의한 context-cache.xml 의 일부이다. 아래 속성 정의 파일 내용에 따르면, cacheManager 속성으로 EhCacheCacheManager 가 정의되어 있다. 이 때 EhCache 를 세부 설정하기 위한 ehcache.xml 설정은 별도의 ehcache.xml 에 정의되어 있다.

```
<!-- enable declarative annotation-based caching -->
<cache:annotation-driven />

<!-- configuring the cache storage -->
<bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager"
  p:cache-manager-ref="ehcache" />

<!-- Ehcache-based Cache -->
<bean id="ehcache" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"
  p:config-location="classpath:cache/ehcache.xml" />
```

- **TestCase**

  다음은 앞서 정의한 속성 설정을 기반으로 하여 Cache 내에 특정 데이터를 저장하고 추출하는 테스트케이스 코드의 일부이다.

```
public void manageGenre() throws Exception {
  GenreService.LOGGER.info("------ manageGenre test ------");

  // 1. create a new genre
  Genre genre = new Genre();
  genre.setName("Western");
  genreService.create(genre);

  // 2. assert - create
  String genreId = genre.getGenreId();
  genre = genreService.get(genreId);
  assertNotNull("fail to fetch a genre", genre);
  assertEquals("fail to compare a genre name", "Western", genre.getName());

  // 3. update a name of genre
  String name = "Western " + System.currentTimeMillis();
  genre.setName(name);
  genreService.update(genre);

  // 4. assert - update
  Genre updatedGenre = genreService.get(genreId);
  assertNotNull("fail to fetch a genre", updatedGenre);
  assertEquals("fail to compare a updated name", name, updatedGenre.getName());

  // 5. update a name of genre and get an updated genre
  String newName = "Western " + System.currentTimeMillis();
  genre.setName(newName);
  Genre updatedAndCachedGenre = genreService.updateAndGet(genre);
  assertNotNull("fail to fetch a genre", updatedAndCachedGenre);
  assertEquals("fail to compare a updated name", newName,
        updatedAndCachedGenre.getName());

  Genre cachedGenre = genreService.get(genreId);
  assertNotNull("fail to fetch a cached genre", cachedGenre);
```

```
  assertEquals("fail to compare a updated name of a cached genre", newName,
 cachedGenre.getName());

  // 6. remove a genre
  genreService.remove(updatedGenre.getGenreId());
}
```

3.Resources

- 참고자료
  - Ehcache Home [http://ehcache.org//]
  - Ehcache Configuration Guide [http://ehcache.org/documentation/configuration/configuration]