Caching

Goals

Understand Hibernate second-level in app caching. We will not be covering distributed cache.

The different caching providers

Powerful and easy to use caching functionality

Hibernate Caching

Hibernate features a second-level cache with a customizable cache provider.

Configured in `grails-app/conf/DataSource.groovy`

```
hibernate {
  cache.use_second_level_cache=true //default
  cache.use_query_cache=true
  cache.region.factory_class=
    'org.hibernate.cache.ehcache.EhCacheRegionFactory'
}
```

Caching instances

```
class Person {
    …
    static mapping = {
        cache true
    }
}
```

To configure ‘read-write’ cache that includes lazy properties:

```
cache usage: 'read-only', include: 'non-lazy'
```

Cache Usages

`read-only`: app only reads data

`read-write`: app updates data

`nonstrict-read-write`: app occasionally updates data

`transactional`: support for fully transactional cache providers such as JBoss TreeCache.

Caching Associations

```
class Person {
   String firstName
   static hasMany = [addresses: Address]
   static mapping = {
        table 'people'
        version false
        addresses column: 'Address', cache: true
    }
}
```

Caching Queries

Dynamic Finders

```
def person = Person.findByFirstName("Fred", [cache: true])
```

Criteria Queries

```
def people = Person.withCriteria {
    like('firstName', 'Fr%')
    cache true
}
```

Hibernate Caching

Works really well for read-only data

If you have domain classes that update, create, or delete frequently, query caching will often be slower than not caching. See

http://tech.puredanger.com/2009/07/10/hibernate-query-cache/

The app-info and hibernate-stats plugins make cache usage information available to determine if caching is helping your app

The Cache Plugin

```
compile ':cache:1.17' // installed by default
```

Adds Spring bean method call, controller action, and GSP page fragment and template caching to Grails applications.

Uses an in-memory implementation where the caches and cache manager are backed by a thread-safe

`java.util.concurrent.ConcurrentMap`

Cache Plugin

Fine for testing and possibly for low-traffic sites, but you should consider using one of the extension plugins if you need clustering, disk storage, persistence between restarts, and more configurability of features like time-to-live, maximum cache size, etc.

cache-ehcache

cache-redis

cache-gemfire

Configuration

Specify configurations in Config.groovy or *CacheConfig.groovy

- Cache implementation is very simple, so there are very few configurations.
- No time to live, overflow to disk, max cache size - If you need these settings use Ehcache
- Caches remain until the JVM is restarted or @CacheEvict is called

Cache DSL

Config.groovy

```
grails.cache.enabled = true  //turns on caching
grails.cache.config = {
    cache {
        name 'question'
    }
    cache {
        name 'answer'
    }
}
```

Creates 2 caches, one with the name “question” and one with “answer”.

Workshop

Enable Caching in config.groovy

Create 2 caches

- question
- answer

Annotations

`@Cacheable()` - check the cache for a pre-existing result, or generate a new result and cache it

`@CachePut()` - always store the result in the cache regardless of existing cache values

`@CacheEvict()` - flush the cache to force the re-evaluation of previously cached results

Service Method Caching

```
import grails.plugin.cache.CacheEvict
import grails.plugin.cache.CachePut
import grails.plugin.cache.Cacheable


class QuestionService {
    @Cacheable('question')
    def getQuestion(String questionId) {
        println "fetching question"
        def question = Question.get(questionId)
        return question
    }
    @CachePut(value='question', key='#question.id')
    void save(Question question) {
        question.save()
    }
    @CacheEvict(value='question', key='#question.id')
    void delete(Question question) {
       question.delete()
    }
}
```

Controller Action Caching

```
class QuestionController {
  @Cacheable('question')
   def lookup() {
      // perform some expensive operations
      println "called 'lookup'"
   }

   @CacheEvict(value='question', allEntries=true)
   def evict() {
      println "called 'evict'"
   }
}
```

Only works on methods, not closures

GSP Cache

Render a GSP template and caches the result

```
<cache:render template="myTemplate" model="[name: 'Some Value']"/>
```

Render a block of markup and caches the result

```
<cache:block key="${currentUser.id}">
    <!-- Any valid markup may be included here, including
    dynamic expressions, invoking other tags, etc.... →
</cache:block>
```

Specifying the key is optional

Workshop

Create a `QuestionService` with a `getQuestion` method that accepts a String id and returns a Question Object

The `QuestionService` should cache the get to the question cache

Modify the `Question.show` action to use the `QuestionService`