



데이터 모델 디자인

Table of Contents

- 데이터 모델 디자인
- GORM?
 - Hibernate의 단일 객체 CRUD
 - GORM의 단일 객체 CRUD

해당 샘플 프로젝트는 사용자 로컬에 hsqldb가 설치되어 있음을 가정하고 작성하였다. hsqldb는 이클립스의 플러그인으로 구동하거나 hsqldb 홈페이지에서 다운받아 구동할 수 있다.

[HSQldb](#)

GORM?

GORM(Grails' object relational mapping)은 Grails에서 구현된 ORM을 얘기한다. 실제 구현은 Hibernate를 사용하고 있기 때문에 사용자가 직접적으로 SQL을 사용할 필요가 적어 직관적으로 비즈니스 로직 구현에 집중할 수 있고 Groovy를 사용하지 때문에 자바에서 구현할때보다 개발량이 줄어든다.

GORM을 살펴보기전에 Hibernate 사전 학습 및 Orm Framework에 대해 이해를 하기 위해 간단한 단일 객체의 CRUD 샘플을 만들어 본 후 동일한 객체를 Grails & GORM으로 작성해보겠다.

Hibernate의 단일 객체 CRUD

해당하는 hibernate기반의 sample 프로젝트는 maven 기반으로 database는 hsqldb, 구동은 jetty로 하는 것으로 구현 및 테스트 되었다. 해당 프로젝트는 hibernate_cat.zip으로 압축하여 첨부파일로 올려져있다.

hibernate의 설정파일인 hibernate.cfg.xml의 내용은 아래와 같다.

```

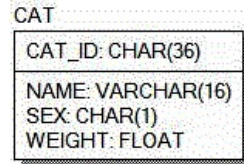
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
3      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
4  <hibernate-configuration>
5      <session-factory>
6          <!-- Database connection settings -->
7          <!-- JDBC connection pool (use the built-in) -->
8          <property name="connection.pool_size">10</property>
9          <!-- SQL dialect -->
10         <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
11         <!-- Enable Hibernate's automatic session context management -->
12         <property name="current_session_context_class">thread</property>
13
14         <!-- Echo all executed SQL to stdout -->
15         <property name="show_sql">true</property>
16
17         <!-- Drop and re-create the database schema on startup -->
18         <property name="hbm2ddl.auto">create</property>
19
20         <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
21         <property name="connection.url">jdbc:hsqldb:hsqldb://localhost/mydatabase</property>
22         <property name="connection.username">sa</property>
23         <property name="connection.password"></property>
24
25         <mapping class="com.fastsystem.sample.hibernate.model.Cat" />
26     </session-factory>
27 </hibernate-configuration>

```

기본적으로 일반적인 Database에 접속하는 다른 프레임워크와 다른 부분이 많이 없지만 Hibernate같은 경우는 개발자가 정의한 도메인에 따라서 데이터베이스 정의를 수정 및 삭제 후 재생성할 수 있는 기능이 있는데 **hbm2ddl.auto** 프로퍼티에 어떤 값을 주는데 따라서 개발자가 원하는 대로 동작시킬 수 있다. create, create-drop, update, validate 등이 있는데 자세한 내용 설명은 링크로 대체한다.

[하이버네이트의 hbm2ddl.auto에 update가 좋을까? validate가 좋을까?](#)

구현하려는 엔티티를 ERD로 표현하면 아래 이미지와 같다.



위의 Cat 엔티티를 Java로 구현하고 hibernate 문법을 적용하면 아래와 같은 소스가 된다.

```

1 package com.fastsystem.sample.hibernate.model;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.Id;
7 import javax.persistence.PrimaryKeyJoinColumns;
8 import javax.persistence.Table;
9
10 import org.hibernate.annotations.GenericGenerator;
11
12 @Entity
13 @Table(name = "CAT")
14 public class Cat {
15
16     @Id
17     @GeneratedValue(generator="UUIDGenerator")
18     @GenericGenerator(name="UUIDGenerator", strategy="com.fastsystem.sample.hibernate.util.UUIDGenerator")
19     @Column(name="CAT_ID", columnDefinition = "CHAR(36)", length = 36, nullable = false)
20     private String id;
21
22     @Column(name="NAME", columnDefinition = "VARCHAR(16)", length = 16)
23     private String name;
24
25     @Column(name="SEX", columnDefinition = "CHAR(1)", length = 1)
26     private char sex;
27
28     @Column(name="WEIGHT", columnDefinition = "float")
29     private float weight;
30
31     public Cat() {
32     }
33
34     public Cat(String catId) {
35         id = catId;
36     }
37
38     public String getId() {
39         return id;
40     }
41
42     /*
43     private void setId(String id) {
44         this.id = id;
45     }
46     */
47
48     public String getName() {
49         return name;
50     }
51
52     public void setName(String name) {
53         this.name = name;
54     }
55
56     public char getSex() {
57         return sex;
58     }
59
60     public void setSex(char sex) {
61         this.sex = sex;
62     }
63
64     public float getWeight() {
65         return weight;
66     }
67
68     public void setWeight(float weight) {
69         this.weight = weight;
70     }
71
72 }
```

hibernate의 객체와 데이터베이스와의 Mapping은 위에서 사용한 Annotation을 사용하거나 별도의 Mapping Xml을 정의하여 적용할 수 있다. 각각 방식에 대한 방법은 링크로 대체한다.

[XML Mapping](#)

[Annotation Mapping](#)

GORM의 경우 스크립트 언어 특성상 모델의 클래스명과 테이블명이 일치하거나 프로퍼티명과 컬럼명이 일치하는 경우 생략할수도 있다. 해당 경우는 GORM CRUD 샘플에서 살펴보자.

Cat 객체를 CRUD 테스트를 위해 Service, Util Class들을 아래와 같이 생성했다.

```

1 package com.fastsystem.sample.hibernate.service;
2
3 import java.util.Iterator;
4 import java.util.LinkedList;
5 import java.util.List;
6 import java.util.Map;
7 import java.util.Queue;
8
9 import org.hibernate.HibernateException;
10 import org.hibernate.Query;
11 import org.hibernate.Session;
12 import org.hibernate.Transaction;
13
14
15
16
17
18
19
20
21 import com.fastsystem.sample.hibernate.model.Cat;
22 import com.fastsystem.sample.hibernate.util.HibernateUtil;
23
24 public class CatService {
25
26     public CatService() {
27     }
28
29     public boolean createCat(String name, char sex, float weight) {
30
31         boolean resultBoolean = true;
32         Session session = null;
33         Transaction transaction = null;
34
35         try {
36
37             session = HibernateUtil.currentSession();
38             transaction = session.beginTransaction();
39
40             Cat cat = new Cat();
41             cat.setName(name);
42             cat.setSex(sex);
43             cat.setWeight(weight);
44
45             session.save(cat);
46             transaction.commit();
47
48             return resultBoolean;
49         } catch (HibernateException e) {
50
51             resultBoolean = false;
52
53             if(transaction != null) {
54                 transaction.rollback();
55             }
56
57             return resultBoolean;
58         } finally {
59             HibernateUtil.closeSession();
60         }
61     }
62
63
64
65     public Queue<Cat> getCatList(){
66
67         Session session = null;
68         Transaction transaction = null;
69         Queue<Cat> cats = new LinkedList<Cat>();
70
71         try {
72             session = HibernateUtil.currentSession();
73             transaction = session.beginTransaction();
74
75             @SuppressWarnings("unchecked")
76             List<Cat> list = session.createQuery("from Cat").list();
77
78             for (Iterator<Cat> iterator = list.iterator(); iterator.hasNext() ; ) {
79                 cats.add(iterator.next());
80             }
81
82             transaction.commit();
83
84         } catch (HibernateException e) {
85             e.printStackTrace();
86             if(transaction != null) {
87                 transaction.rollback();
88             }
89         } finally {
90             HibernateUtil.closeSession();
91         }
92         return cats;
93     }
94
95
96
97     public Cat getCat(String catId) {
98
99         Session session = null;
100         Transaction transaction = null;
101
102

```

```

103     Cat cat = null;
104
105     session = HibernateUtil.currentSession();
106     transaction = session.beginTransaction();
107
108     try {
109         //cat = (Cat) session.createQuery(" from Cat where cat_id = :catId ").setString("catId", catId).list().get(0);
110         cat = (Cat) session.get(Cat.class, catId);
111
112         transaction.commit();
113     } catch (Exception e) {
114         e.printStackTrace();
115         if(transaction != null) {
116             transaction.rollback();
117         }
118     } finally {
119         HibernateUtil.closeSession();
120     }
121
122     return cat;
123 }
124
125 public void modifyCat(Cat cat) {
126
127     Session session = null;
128     Transaction transaction = null;
129
130     session = HibernateUtil.currentSession();
131     transaction = session.beginTransaction();
132
133     try {
134         //session.createQuery(" update Cat set name = :name, sex = :sex, weight = :weight where cat_id = :catId ").setProperties(cat);
135         //cat = (Cat) session.get(Cat.class, cat);
136         session.update(cat);
137
138         transaction.commit();
139     } catch (Exception e) {
140         e.printStackTrace();
141         if(transaction != null) {
142             transaction.rollback();
143         }
144     } finally {
145         HibernateUtil.closeSession();
146     }
147 }
148
149 public void removeCat(String catId){
150
151     Session session = null;
152     Transaction transaction = null;
153
154     session = HibernateUtil.currentSession();
155     transaction = session.beginTransaction();
156
157     try {
158         //session.createQuery(" delete Cat where cat_id = :catId ").setString("catId", catId);
159         session.delete(session.get(Cat.class, catId));
160
161         transaction.commit();
162     } catch (Exception e) {
163         e.printStackTrace();
164         if(transaction != null) {
165             transaction.rollback();
166         }
167     } finally {
168         HibernateUtil.closeSession();
169     }
170 }
171
172 }
173
174

```

hibernate의 경우 sql을 작성하지 않아도 hibernate에서 sql을 생성하여 실행되지만 개발자가 직접 sql을 컨트롤 하고싶은 경우를 위해 hql과 Criteria Api를 제공한다. 위 Service 소스에 주석으로 막혀 있는 부분이 hql을 사용하여 구현한 부분이다. 위 문법 이외의 hql이나 Criteria에 대한 내용은 링크로 대체한다.

[Hibernate를 이용한 ORM 7 - HQL과 Criteria를 이용한 조회](#)

```

1 package com.fastsystem.sample.hibernate.util;
2
3 import org.hibernate.HibernateException;
4 import org.hibernate.Session;
5 import org.hibernate.SessionFactory;
6 import org.hibernate.boot.registry.StandardServiceRegistry;
7 import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
8 import org.hibernate.cfg.Configuration;
9
10
11 public class HibernateUtil {
12
13     private static final SessionFactory SESSION_FACTORY;
14
15     private static final ThreadLocal<Session> THREAD_LOCAL = new ThreadLocal();
16
17     static {
18         try {
19             Configuration cfg = new Configuration().configure("hibernate.cfg.xml");
20             StandardServiceRegistryBuilder sb = new StandardServiceRegistryBuilder();
21             sb.applySettings(cfg.getProperties());

```

```

22     sessionFactory = getSessionFactory();
23     StandardServiceRegistry standardServiceRegistry = sb.build();
24     SESSION_FACTORY = cfg.buildSessionFactory(standardServiceRegistry);
25 } catch (Throwable th) {
26     System.err.println("Initial SessionFactory creation failed" + th);
27     throw new ExceptionInInitializerError(th);
28 }
29
30 public static SessionFactory getSessionFactory() {
31     return SESSION_FACTORY;
32 }
33
34 public static Session currentSession() throws HibernateException {
35     Session session = THREAD_LOCAL.get();
36     if (session == null) {
37         session = SESSION_FACTORY.openSession();
38         THREAD_LOCAL.set(session);
39     }
40     return session;
41 }
42
43
44 public static void closeSession() throws HibernateException {
45     Session session = THREAD_LOCAL.get();
46     THREAD_LOCAL.set(null);
47
48     if (session != null) {
49         session.close();
50     }
51 }
52 }

```

위의 클래스는 hibernate 접속을 담당하는 util 클래스이다.

```

1 package com.fastsystem.sample.hibernate.util;
2
3 import java.io.Serializable;
4 import java.util.UUID;
5
6 import org.hibernate.HibernateException;
7 import org.hibernate.engine.spi.SessionImplementor;
8 import org.hibernate.id.IdentifierGenerator;
9
10 public class UUIDGenerator implements IdentifierGenerator{
11
12     public Serializable generate(SessionImplementor arg0, Object arg1) throws HibernateException {
13         return UUID.randomUUID().toString();
14     }
15 }

```

해당하는 클래스는 Cat 인스턴스를 생성할때 Pk인 Id를 생성해주는 클래스다.

실제 수행되는 화면은 아래와 같다.

생성

/hibernate_cat/create.jsp

| Cat Create | |
|--------------------------------------|-------|
| Cat Name | Caaat |
| Cat Sex | F |
| Cat Weight | 23 |
| <input type="button" value="Crate"/> | |

보기

/hibernate_cat/list.jsp

```

catName||catSex||catWeight
-----
Caaat||F||23.0
-----


```

수정, 삭제

/hibernate_cat/view.jsp

| Cat Modify | |
|------------|--------------------------------------|
| Cat Id | 613d3a2c-9ebb-4bf4-b92a-bafe5351d93d |
| Cat Name | Caaat |
| Cat Sex | F |
| Cat Weight | 23.0 |

상당히 간단한 model1 방식으로 구현되어 있으며 첨부파일로 올려져있는 hibernate_cat을 꼭 내려 받아 실제 구동을 통해 테스트해보기 바란다.

GORM의 단일 객체 CRUD

GORM의 경우도 위의 Cat 엔티티에 대한 CRUD를 구현해보겠다.

| CAT |
|-------------------|
| CAT_ID: CHAR(36) |
| NAME: VARCHAR(16) |
| SEX: CHAR(1) |
| WEIGHT: FLOAT |

grails의 도메인 정의는 'grails-app/domain/' 아래 정의해야 한다.

Cat이라는 파일로 Cat 엔티티를 도메인으로 정의해보겠다.

```

1 package grails_cat
2
3 class Cat {
4
5     String id;
6     String name;
7     String sex;
8     float weight;
9
10    static constraints = {
11        name blank: false, nullable: false, size: 1..16
12        sex blank: true, nullable: false, size: 1..1
13        weight blank: false, nullable: false
14    }
15
16    static mapping = {
17        id generator: 'uuid'
18        version false
19    }
20 }
```

위의 hibernate에서의 도메인 모델 정의와 비교할 때 소스코드가 상당히 많이 줄어든 것을 확인할 수 있다.

grails의 컨트롤러 정의는 'grails-app/controllers/' 아래 정의해야 한다.

```

1 package grails_cat
2
3 class CatController {
4
5     def scaffold = true //CRUD 화면을 자동으로 생성해준다.
6
7     //def index() { } //index 화면이 정의되어 있다면 동적으로 생성되지 않아 해당 정의를 막아야 정상적으로 리스트가 출력된다.
8 }
```

그리고 DataSource의 경우 hibernate와 마찬가지로 hsqldb를 사용하기 위해 grails-app/conf/DataSource.grovy 파일을

아래와 같이 수정한다.

```

1 dataSource {
2     pooled = true
3     driverClassName = "org.hsqldb.jdbcDriver"
4     username = "sa"
5     password = ""
6 }
7
8 hibernate {
```



```

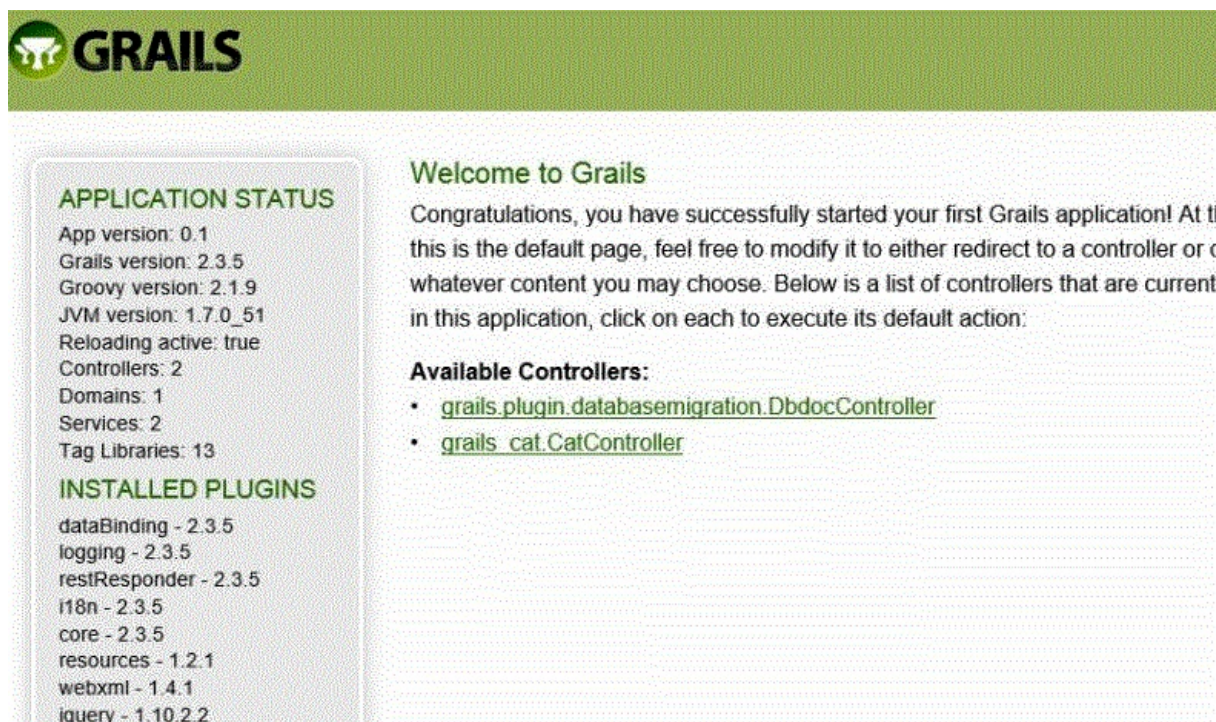
9      cache.use_second_level_cache = true
10     cache.use_query_cache = false
11     cache.region.factory_class = 'net.sf.ehcache.hibernate.EhCacheRegionFactory' // Hibernate 3
12     // cache.region.factory_class = 'org.hibernate.cache.ehcache.EhCacheRegionFactory' // Hibernate 4
13 }
14
15 // environment specific settings
16 environments {
17     development {
18         dataSource {
19             dbCreate = "create-drop" // one of 'create', 'create-drop', 'update', 'validate', ""
20             url = "jdbc:hsqldb:hsqldb://localhost/mydatabase:MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE"
21         }
22     }
23     test {
24         dataSource {
25             dbCreate = "update"
26             url = "jdbc:hsqldb:hsqldb://localhost/mydatabase:mem:testDb:MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE"
27         }
28     }
29     production {
30         dataSource {
31             dbCreate = "create"
32             url = "jdbc:hsqldb:hsqldb://localhost/mydatabase:prodDb:MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE"
33             properties {
34                 maxActive = -1
35                 minEvictableIdleTimeMillis=1800000
36                 timeBetweenEvictionRunsMillis=1800000
37                 numTestsPerEvictionRun=3
38                 testOnBorrow=true
39                 testWhileIdle=true
40                 testOnReturn=false
41                 validationQuery="SELECT 1"
42                 jdbcInterceptors="ConnectionState"
43             }
44         }
45     }
46 }

```

특별히 설명할 부분은 없지만 'dbCreate'의 경우 hibernate의 'hbm2ddl.auto'와 같은 옵션이라고 보면 된다.

해당 grails를 구동한 화면은 아래와 같다.

grails의 home



The image shows the Grails application home screen. On the left, there is a sidebar with the Grails logo and two sections: 'APPLICATION STATUS' and 'INSTALLED PLUGINS'. The 'APPLICATION STATUS' section lists various application details like version, JVM version, and reloading status. The 'INSTALLED PLUGINS' section lists installed plugins and their versions. The main content area on the right has a 'Welcome to Grails' message, a congratulatory paragraph, and a list of 'Available Controllers' with links to each controller.

APPLICATION STATUS

- App version: 0.1
- Grails version: 2.3.5
- Groovy version: 2.1.9
- JVM version: 1.7.0_51
- Reloading active: true
- Controllers: 2
- Domains: 1
- Services: 2
- Tag Libraries: 13

INSTALLED PLUGINS

- dataBinding - 2.3.5
- logging - 2.3.5
- restResponder - 2.3.5
- i18n - 2.3.5
- core - 2.3.5
- resources - 1.2.1
- webxml - 1.4.1
- jquery - 1.10.2.2

Welcome to Grails

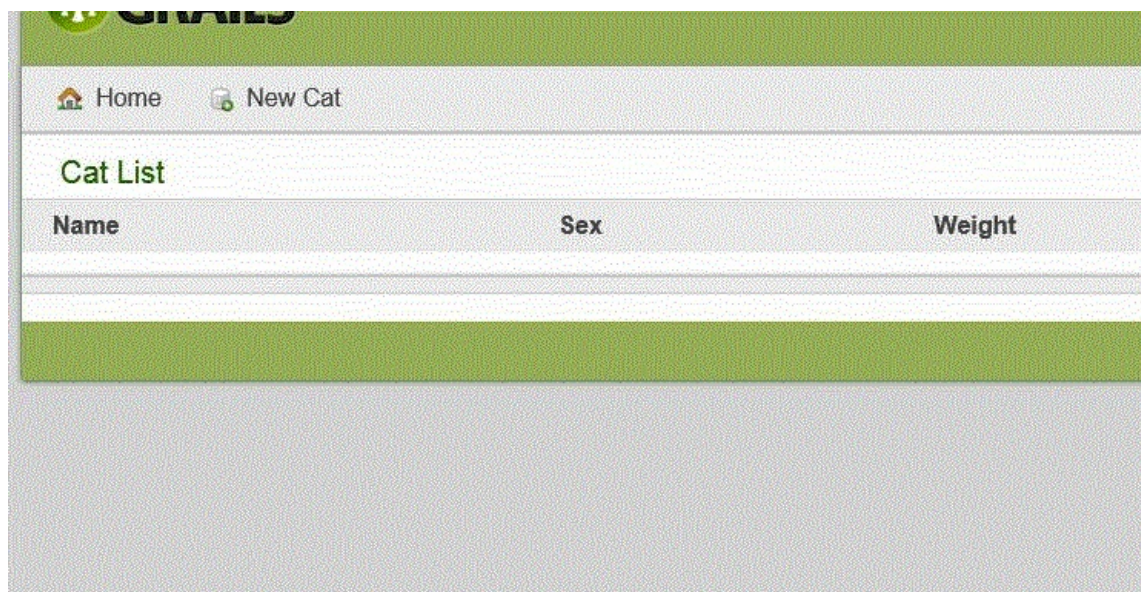
Congratulations, you have successfully started your first Grails application! At this time this is the default page, feel free to modify it to either redirect to a controller or to display whatever content you may choose. Below is a list of controllers that are currently available in this application, click on each to execute its default action:

Available Controllers:

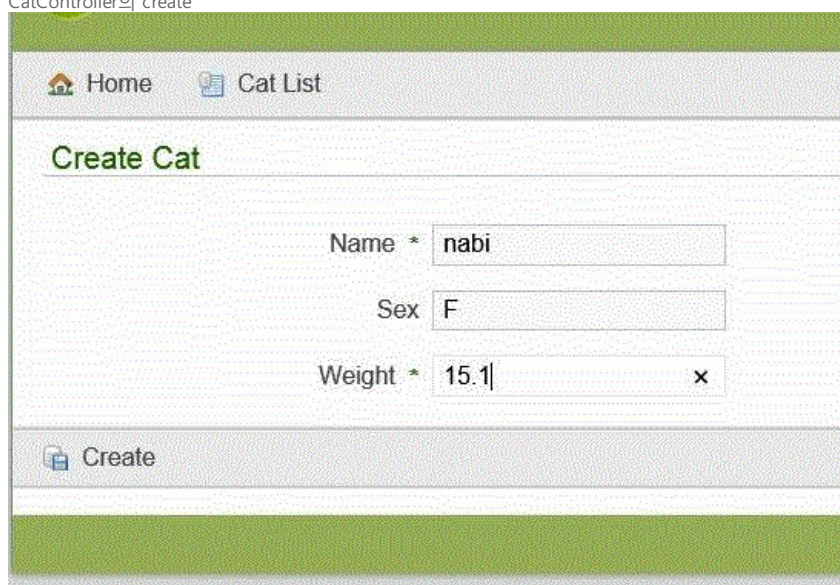
- [grails.plugin.databasemigration.DbdocController](#)
- [grails.cat.CatController](#)

CatController를 선택한다.

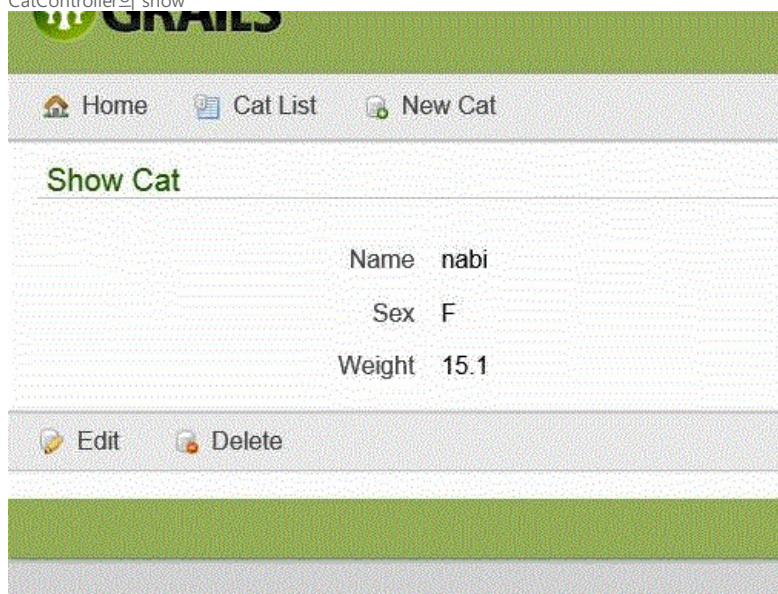
CatController의 list



CatController의 create



CatController의 show



CatController의 list

| Home New Cat | | |
|--|-----|--------|
| Cat List | | |
| Name | Sex | Weight |
| nabi | F | 15.8 |
| | | |
| | | |

단순히 도메인 모델만 설정했을 뿐인데 CRUD까지 모두 생성되었다. 하지만 GORM이 어디에 사용되었는지

확인을 Static scaffolding 기능을 사용하여 해당 소스를 생성해보겠다.

grails command에 아래와 같이 수행한다.

```
1 | grails generate-all grails cat.Cat
```

프로젝트명과 설정한 도메인에 따라서 변경될 수 있다.

해당 커멘드가 수행되면 아래와 같이 변경된 CatController를 확인할 수 있다.

```

1 | package grails_cat
2 |
3 |
4 |
5 | import static org.springframework.http.HttpStatus.*
6 | import grails.transaction.Transactional
7 |
8 | @Transactional(readOnly = true)
9 | class CatController {
10 |
11 |     static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]
12 |
13 |     def index(Integer max) {
14 |         params.max = Math.min(max ?: 10, 100)
15 |         respond Cat.list(params), model:[catInstanceCount: Cat.count()] //READ
16 |     }
17 |
18 |     def show(Cat catInstance) {
19 |         respond catInstance
20 |     }
21 |
22 |     def create() {
23 |         respond new Cat(params)
24 |     }
25 |
26 |     @Transactional
27 |     def save(Cat catInstance) {
28 |         if (catInstance == null) {
29 |             notFound()
30 |             return
31 |         }
32 |
33 |         if (catInstance.hasErrors()) {
34 |             respond catInstance.errors, view:'create'
35 |             return
36 |         }
37 |
38 |         catInstance.save flush:true //CREATE
39 |
40 |         request.withFormat {
41 |             form {
42 |                 flash.message = message(code: 'default.created.message', args: [message(code: 'catInstance.label', default: 'Cat'), catInsta
43 |                 redirect catInstance
44 |             }
45 |             '*' { respond catInstance, [status: CREATED] }
46 |         }
47 |     }
48 |
49 |     def edit(Cat catInstance) {
50 |         respond catInstance
51 |     }
52 |
53 |     @Transactional
54 |     def update(Cat catInstance) {
55 |         if (catInstance == null) {
56 |             notFound()
57 |             return
58 |         }
59 |
60 |         if (catInstance.hasErrors()) {
61 |             respond catInstance.errors, view:'edit'

```

```

62         response.sendRedirect(url); return url;
63     }
64
65     catInstance.save flush:true //UPDATE
66
67     request.withFormat {
68         form {
69             flash.message = message(code: 'default.updated.message', args: [message(code: 'Cat.label', default: 'Cat'), catInstance.id])
70             redirect catInstance
71         }
72         '*' { respond catInstance, [status: OK] }
73     }
74 }
75
76 @Transactional
77 def delete(Cat catInstance) {
78
79     if (catInstance == null) {
80         notFound()
81         return
82     }
83
84     catInstance.delete flush:true //DELETE
85
86     request.withFormat {
87         form {
88             flash.message = message(code: 'default.deleted.message', args: [message(code: 'Cat.label', default: 'Cat'), catInstance.id])
89             redirect action:"index", method:"GET"
90         }
91         '*' { render status: NO_CONTENT }
92     }
93 }
94
95 protected void notFound() {
96     request.withFormat {
97         form {
98             flash.message = message(code: 'default.not.found.message', args: [message(code: 'catInstance.label', default: 'Cat'), param
99             redirect action: "index", method: "GET"
100         }
101         '*' { render status: NOT_FOUND }
102     }
103 }
104 }

```

해당 소스를 보면 index,create,update,delete안에 hibernate에서 사용하는 메소드는 'list,save,delete'등이 사용된 것을 확인할 수 있다.

그리고 당연한 얘기지만 GORM도 HSQL을 수행할 수 있다.

<http://grails.org/doc/2.2.4/ref/Domain%20Classes/executeQuery.html>

해당 샘플은 grails_cat.zip으로 압축하여 첨부하였다.

GORM 관련 Api

[Domain Class Usage](#)

링크 목록

- [HSQLDB](http://www.hsqldb.org/) - http://www.hsqldb.org/
- [하이버네이트의 hbm2ddl.auto에 update가 좋을까? validate가 좋을까?](http://gyumee.egloos.com/viewer/2483659) - http://gyumee.egloos.com/viewer/2483659
- [XML Mapping](http://docs.jboss.org/hibernate/orm/3.3/reference/ko-KR/html/xml.html) - http://docs.jboss.org/hibernate/orm/3.3/reference/ko-KR/html/xml.html
- [Annotation Mapping](http://www.tutorialspoint.com/hibernate/hibernate_annotations.htm) - http://www.tutorialspoint.com/hibernate/hibernate_annotations.htm
- [Hibernate를 이용한 ORM 7 - HQL과 Criteria를 이용한 조회](http://javacan.tistory.com/entry/107) - http://javacan.tistory.com/entry/107
- <http://grails.org/doc/2.2.4/ref/Domain%20Classes/executeQuery.html> - http://grails.org/doc/2.2.4/ref/Domain%20Classes/executeUpdate.html
- [Domain Class Usage](http://grails.org/doc/2.2.4/ref/Domain%20Classes/Usage.html) - http://grails.org/doc/2.2.4/ref/Domain%20Classes/Usage.html