Anyframe Generic Plugin

Version 1.6.0

저작권 © **2007-2014** 삼성**SDS**

본 문서의 저작권은 삼성**SDS**에 있으며 **Anyframe** 오픈소스 커뮤니티 활동의 목적하에서 자유로운 이용이 가능합니다. 본 문서를 복제, 배포할 경우에는 저작권자를 명시하여 주시기 바라며 본 문서를 변경하실 경우에는 원문과 변경된 내용을 표시하여 주시기 바랍니다. 원문과 변경된 문서에 대한 상업적 용도의 활용은 허용되지 않습니다. 본 문서에 오류가 있다고 판단될 경우 이슈로 등록해 주시면 적절한 조치를 취하도록 하겠습니다.

I.Introduction

Anyframe의 generic 서비스는 Java5부터 지원하는 **Generics** 개념을 기반으로 개발된 것으로, 도메인 클래스 기반의 Service 인터페이스/구현 클래스, DAO 인어페이스/구현클래스(Hibernate/JPA, Query Service 지원) 등의 기본 CRUD 기능이 모두 구현된 클래스를 제공하고 있다. 어플리케이션 개발 시에는 Generic 서비스의 클래스들을 그대로 이용하거나 상속받아서 추가적인 기능을 개발할 수 있다. generic plugin은 이 generic 서비스를 활용하는 방법을 가이드하는 샘플코드와 참조라이브러리들로 구성되어 있다.

Installation

Command 창에서 다음과 같이 명령어를 입력하여 generic plugin을 설치한다.

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

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

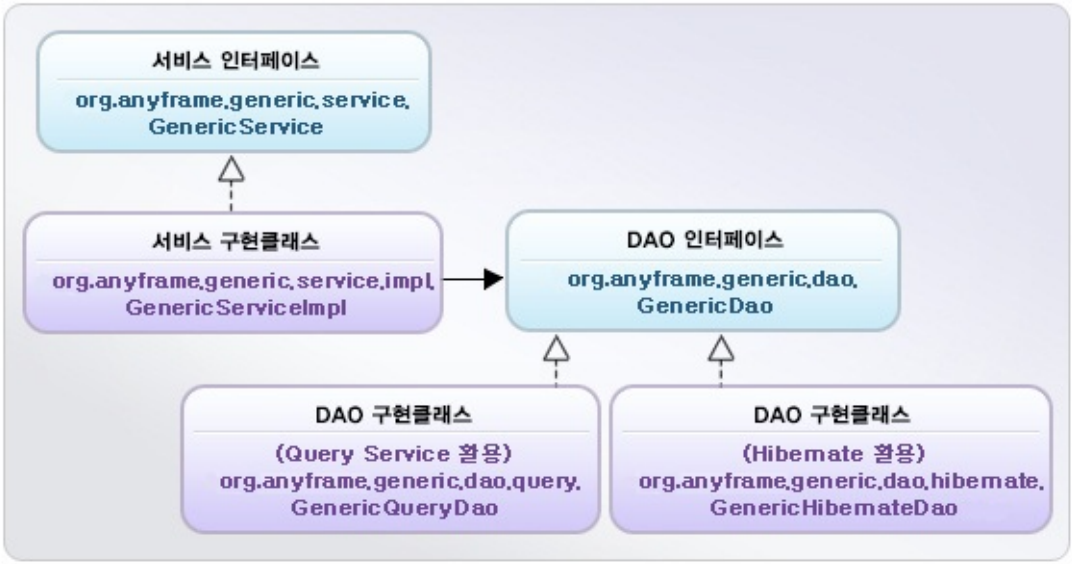
II.Generic

Generic Service는 AppFuse Framework [http://appfuse.org/display/APF/Home] (http://appfuse.org/display/APF/Home)의 개념과 소스 코드(템플릿 포함)를 참고하여 Anyframe에 맞게 수정되었다. 아래의 Generic Service 구성 클래스들을 활용한 샘플 코드들은 직접 작성하여 사용할 수도 있고, Anyframe IDE Editor의 CRUD Generation 기능을 통해 자동생성되는 코드를 사용할 수도 있다.

Generic Service를 이용하여 개발 시 다음과 같은 특징을 갖는다.

- 도메인 모델 객체를 중심으로 기본 CRUD 코드들을 쉽게 작성할 수 있다.
- 개발자가 Business Layer, Data Access Layer 코드를 작성하지 않고 Generic Service에서 제공하는 Service 클래스와 Dao 클래스들을 그대로 재사용하여 기본 CRUD 기능을 구현할 수 있다.
- 기본 CRUD 외의 부가 기능이 필요한 경우 Generic Service에서 제공하는 클래스를 상속받아서 부가 기능에 대해서만 추가 기능을 구현할 수 있다.
- DAO Framework으로 Hibernate/JPA, Query Service를 지원한다.

Generic Service 클래스들의 관계에 대한 구조도이다.



1.Domain 클래스 생성

Domain 클래스 생성 시 Serializable 인터페이스 구현은 필수 사항은 아니지만, Hibernate에서 persistent object가 HttpSession에 저장되거나 RMI를 이용해서 전달할 때는 반드시 필요하다.

1.1.Query Service 사용 시

다음은 Generic Plugin 설치로 추가된 Query Service DAO Framework 기반의 ~generic/domain/GenericMovie.java 의 일부이다. 이때 @Entity이나 @Table과 같은 Annotation 설정 없이 @Id(복합키의 경우, @EmbeddedId) Annotation 설정만 있으면 Query Service를 이용하여 Generic Service의 CRUD 기능을 사용할 수 있다.

```
public class GenericMovie implements Serializable {
    private String movieId;
    중략...
    @Id
    public String getMovieId() {
        return movieId;
    }

    public void setMovieId(String movieId) {
        this.movieId = movieId;
    }
}
```

2. Generic 서비스 사용 방법

Generic Service가 정의한 기능을 그대로 활용하는 방법과 원하는 기능을 추가로 구현해서 사용하는 두 가지 방법을 제공한다.

2.1.Generic 서비스 그대로 활용

제공되는 Generic 서비스를 재정의하지 않고 사용하는 방법이다. Container에 제공되는 설정 메타데이터 내에 정의해주고 사용하면 된다.

- DAO 속성 정의

  DAO Framework에 따른 정의 방식은 아래와 같다.

  - 다음은 Generic Plugin 설치로 추가된 Query Service DAO Framework 기반의 Spring 속성 정의 파일 context-generic.xml 의 일부이다. Query 서비스 기반의 경우 sessionFactory를 내부적으로 사용하기 때문에 inner bean으로 정의해준다.

    ```
    <bean id="genericGenreDao" class="org.anyframe.generic.dao.query.GenericQueryDao">
        <constructor-arg value="org.anyframe.plugin.generic.domain.GenericGenre" />
        <property name="queryService" ref="queryService" />
    </bean>
    ```

  - Hibernate 기반의 경우 sessionFactory와 dynamicHibernateService는 내부적으로 사용하기 때문에 inner bean으로 정의해준다.

    ```
    <bean
     id="genericHibernateGenreDao" class="org.anyframe.generic.dao.hibernate.GenericHibernateDao">
        <constructor-arg value="org.anyframe.plugin.generic.domain.GenericGenre"/>
        <property name="sessionFactory" ref="sessionFactory"/>
        <property name="dynamicHibernateService" ref="dynamicHibernateService" />
    </bean>
    ```

- Service 속성 정의

  다음은 Generic Plugin 설치로 추가된 Spring 속성 정의 파일 context-generic.xml 의 일부이다. 정의해 놓은 DAO Bean을 inner bean으로 injection 한다.

  ```
  <bean
   id="genericGenreService" class="org.anyframe.generic.service.impl.GenericServiceImpl">
      <property name="genericDao" ref="genericGenreDao" />
  </bean>
  ```

2.2.Generic 서비스 확장 활용

제공되는 Generic 서비스를 재정의해서 사용한다. DAO 와 Service 에 대해서 필요한 부분에 대해서 재정의해서 사용한다.

2.2.1.DAO 클래스

도메인 객체를 기반으로 신규 생성, 단건 조회, 목록 조회, 삭제, 데이터 존재 여부 확인에 관한 데이터 접근 로직을 제공한다.

리턴 값	메소드	역할	지원 여부
T	get(PK id)	단건조회	Query/ Hibernate
boolean	exists(PK id)	데이터 존재 여부 확인	Query/ Hibernate
void	create(T object)	신규 생성	Query/ Hibernate
void	update(T object)	수정	Query/ Hibernate
void	remove(PK id)	삭제	Query/ Hibernate
Page	getPagingList(SearchVO searchVO)	페이징처리가 된 목록 조회	Query/ Hibernate
Page	getPagingList(T object, int pageIndex)	페이징처리가 된 목록 조회	Query
List	getList(SearchVO searchVO)	목록 조회	Query/ Hibernate
List	getList(T object, int pageIndex)	목록 조회	Query

- **GenericDao**

  DAO인터페이스로 GenericDao<T, PK extends Serializable>Class를 이용한다. 여기서 T는 도메인 객체 타입으로 도메인 모델 클래스를 의미하고, PK는 도메인 객체의 Primary Key타입을 의미한다. 다음은 GenericDao 클래스에 정의되어 있는 단건조회,데이터 존재여부 확인, 저장, 삭제, 목록 조회와 관련한 메소드이다.

- **GenericQueryDao**

  GenericDao 인터페이스 클래스를 구현하는 구현 클래스 중 하나로 QueryService를 DAO Framework으로 결정한 경우에 사용한다. DAO 구현 클래스에서 Query Service를 이용하기 위해 GenericQueryDao 클래스와 QueryDaoUtils 클래스를 사용한다. QueryDaoUtils는 Primary Key 정보를 알아내는 Utility 클래스다.

- **GenericHibernateDao**

  GenericDao 인터페이스 클래스를 구현하는 구현 클래스 중 하나로 Hibernate를 DAO Framework으로 결정한 경우에 사용한다. Hibernate Framework 사용 시 목록 조회의 경우 개별 인자들을 포함하고 있는 VO 유형의 객체를 전달하는 방식의 조회 서비스는 제공하지 않는다.

<Samples>

DAO Framework으로 Query Service를 사용하기 위해 GenericQueryDao클래스를 상속받은 DAO 클래스를 생성한다. Query Service를 사용하기 위한 configuration은 Query Service 설정하기 [http://dev.anyframejava.org/docs/anyframe/plugin/optional/query/1.6.0/reference/htmlsingle/query.html#query_configuration]를 Hibernate Service를 사용하기 위한 configuration은 Hibernate Service 설정하기 [http://dev.anyframejava.org/docs/anyframe/plugin/optional/hibernate/1.6.0/reference/htmlsingle/hibernate.html#hibernate_hibernate_configuration]를 참조하도록 한다.

- 다음은 Query Service를 이용한 DAO 구현 클래스의 일부분이다. 오버라이드 하지 않은 메소드(단건조회, 데이터 존재여부 확인, 수정, 삭제)의 기능은 GenericQueryDao에 구현된 형태 그대로 사용한다. create 메소드는 오버라이드는 했으나 GenericQueryDao의 구문을 그대로 사용하고 있다. getPagingList 메소드는 오버라이드해서 구현했다. GenericQueryDao에서 제공하지 않는 Genre별 Movie 건수 구하는 신규 데이터 접근 로직을 구현하고자 countMovieListByGenre메소드를 추가 작성하였다.

```
@Repository("genericMovieDao")
public class MovieDao extends GenericQueryDao<GenericMovie,String> {

    @Inject
    IQueryService queryService;

    @PostConstruct
    public void initialize() {

        //Domain객체에 대해서 persistentClass에 세팅
        super.setPersistentClass(GenericMovie.class);
        super.setQueryService(queryService);
    }

    public void create(GenericMovie genericMovie) throws Exception{
     genericMovie.setMovieId("MV-" + System.currentTimeMillis());
        create("createGenericMovie", genericMovie);
    }

    public Page getPagingList(Movie movie, int pageIndex) throws Exception{
        return this.findListWithPaging("findMovieList", movie, pageIndex, pageSize,
            pageUnit);
    }

    public int countMovieListByGenre(String genreId) throws Exception{
    중략...
    }

}
```

- Hibernate Framework를 이용한 DAO 구현 클래스의 일부분이다. 오버라이드 하지 않은 메소드(단건 조회, 데이터 존재여부 확인, 수정, 삭제)의 기능은 GenericHibernateDao에 구현된 형태 그대로 사용한다. create 메소드는 오버라이드해서 구현한 경우이며 GenericHibernateDao에서 제공하지 않는 Genre별 Movie 건수 구하는 신규 데이터 접근 로직을 구현하고자 countMovieListByGenre메소드를 추가작성하였다.

```
@Repository("genericHibernateMovieDao")
public class MovieDao extends GenericHibernateDao<GenericMovie,String>{

    @Inject
    SessionFactory sessionFactory;

    @Inject
    IDynamicHibernateService dynamicHibernateService;

    @PostConstruct
    public void initialize(){
        super.setSessionFactory(sessionFactory);
        super.setPersistentClass(GenericMovie.class);
        super.setDynamicHibernateService(dynamicHibernateService);
    }

    public void create(GenericMovie genericMovie) throws Exception {
        super.getHibernateTemplate().save(genericMovie);
    }

    public int countMovieListByGenre(String genreId) throws Exception{
    중략...
    }
}
```

2.2.2.Service 클래스 생성

Service 클래스의 경우 Generic Service에서 제공하는 기본 CRUD 메소드 이외의 기능을 제공하는 경우나 기본 CRUD 메소드를 확장하여 사용해야 하는 경우 Service 구현 클래스를 생성하여 사용하도록 하고 기본 CRUD 메소드를 그대로 사용하는 경우 Service 구현 클래스를 생성하지 않는다.

리턴 값	메소드	역할
T	get(PK id)	단건조회
boolean	exists(PK id)	데이터 존재 여부 확인
void	create(T object)	신규 생성
void	update(T object)	수정
void	remove(PK id)	삭제
Page	getPagingList(SearchVO searchVO)	페이징처리가 된 목록 조회
Page	getPagingList(T object, int pageIndex)	페이징처리가 된 목록 조회
List	getList(SearchVO searchVO)	목록 조회
List	getList(T object, int pageIndex)	목록 조회

- 서비스 인터페이스는 GenericService<T, PK extends Serializable>를 사용한다. GenericService에는 신규 생성, 수정, 단건 조회, 목록 조회, 삭제, 데이터 존재 여부 확인에 관한 메소드가 선언되어 있다. 여기서 T는 도메인 객체 타입으로 도메인 모델 클래스를 의미하고, PK는 도메인 객체의 Primary Key 타입을 의미한다.
- 정의된 도메인 객체를 기반으로 신규 생성, 단건 조회, 목록 조회, 삭제, 데이터 존재 여부 확인에 관한 비즈니스 로직을 구현한다.
- 서비스 구현 클래스는 GenericServiceImpl을 사용하며 서비스 인터페이스를 상속받아 오버라이드한다.

<Samples>

- GenericService에 선언된 메소드 중 생성과 목록조회 메소드만 오버라이드 한 경우이다. 나머지 단건 조회, 데이터 존재 여부 확인, 저장, 삭제, 목록 조회 기능은 GenericService에 정의된 그대로 사용한다. GenericService에서 제공하지 않는 Genre별 Movie 건수 구하는 신규 비즈니스 로직을 구현하고자 countMovieListByGenre메소드를 추가하였다.

```
public interface MovieService extends GenericService<GenericMovie, String> {

    public void create(GenericMovie movie) throws Exception;

    public Page getPagingList(Movie movie, int pageIndex) throws Exception;

    public int countMovieListByGenre(String genreId) throws Exception;

    }
```

- GenericServiceImpl에 선언된 메소드 중 생성과 목록조회 메소드만 오버라이드 한 경우이다. 나머지 단건 조회, 데이터 존재 여부 확인, 저장, 삭제, 목록 조회 기능은 GenericServiceImpl에 정의된 그대로 사용한다. GenericServiceImpl에서 제공하지 않는 Genre별 Movie 건수 구하는 신규 비즈니스 로직을 구현하고자 countMovieListByGenre메소드를 추가 작성하였다.

```
@Service("genericMovieService")
public class MovieServiceImpl extends GenericServiceImpl<GenericMovie , String>{
    implements MovieService {
```

```
    @Inject
    @Named("genericMovieDao")
    MovieDao genericMovieDao;

    @PostConstruct
    public void initialize() {
        super.setGenericDao(MovieDao);
    }

    public void create(GenericMovie genericMovie) throws Exception {
        genericMovieDao.create(genericMovie);
    }

    public int countMovieListByGenre(String genreId) throws Exception {
        int count = genericMovieDao.countMovieListByGenre(genreId);
        return count;
    }

}
```

3.Test Case

다음은 Query 서비스를 사용하여 Generic Service를 Test하기 위한 코드이다.

- 다음은 Generic Plugin 설치로 추가된 Query Service DAO Framework 기반의 generic/moviefinder/service/GenericGenre.java의 일부로써 Generic 서비스를 그대로 활용한 케이스에 대한 Test Code이다.

```
public class GenreServiceTest {

    @Inject
    @Named("genericGenreService")
    private GenericService<GenericGenre, String> genericGenreService;

    @Test
    public void testGenreGetList() throws Exception {
        SearchVO searchVO = new SearchVO();
        List list = genericGenreService.getList(searchVO);

        Assert.assertEquals(10, list.size());
    }
}
```

- 다음은 Generic Plugin 설치로 추가된 Query Service DAO Framework 기반의 generic/moviefinder/service/GenericGenre.java의 일부로써 Generic 서비스를 확장한 케이스에 대한 Test Code이다.

```
public class MovieServiceTest {

    @Inject
    @Named("genericMovieService")
    private MovieService movieService;

    @Test
    @Rollback(value = true)
    public void manageMovie() throws Exception {

        // 1. create a new movie
        GenericMovie movie = getMovie();
        movieService.create(movie);

        // 2. assert - create
        movie = movieService.get(movie.getMovieId());
        assertNotNull("fail to fetch a movie", movie);
        assertEquals("fail to compare a movie title", "Shrek (2001)", movie
                .getTitle());

        // 3. update a title of movie
        String title = "Shrek 2 " + System.currentTimeMillis();
        movie.setTitle(title);

        // 4. assert - update
        movieService.update(movie);

        // 5. remove a movie
        movie = movieService.get(movie.getMovieId());
        assertNotNull("fail to fetch a movie", movie);
        assertEquals("fail to compare a updated title", title, movie.getTitle());

        // 6. assert - remove
```

```
        movieService.remove(movie.getMovieId());
        중 략...
    }
}
```

4.Resources

- 참고자료
  - AppFuse Framework [http://appfuse.org/display/APF/Home]
  - Generics in the Java Programming Language [http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf]