Anyframe Routing DataSource Plugin

Version 1.1.0

I.Introduction

Spring Framework에서는 AbstractRoutingDataSource라는 다중 DB 접근 방법을 제공한다. routingdatasource plugin에서는 AbstractRoutingDataSource를 활용하여 다중으로 정의된 DB를 동적으로 변경하여 접근하는 방법을 가이드 한다.

Installation

Command 창에서 다음과 같이 명령어를 입력하여 routingdatasource plugin을 설치한다.

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

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.RoutingDataSource

Spring Framework에서 제공하는 AbstractRoutingDataSource를 이용하여 동적으로 DB 접근 정보를 변경할 수 있다. routingdatasource plugin는 동일한 DB 스키마와 비즈니스 로직을 가지는 어플리케이션이 3개의 서로 다른 DB에 접근해서 데이터를 엑세스 하는 예제로 구성 되어있다.

1.기본 구현

1.1.CustomRoutingDataSource

다중 DB 접근을 위해서 가장 먼저 구현해야 할 것은 CustomDataSource이다. Spring에서 제공하는 AbstractRoutingDataSource를 상속받는 사용자 정의 DataSource를 정의 해야한다.

아래의 JAVA 코드는 AbstractRoutingDataSource를 상속받은 CustomRoutingDataSource이다.

```
import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;

public class CustomRoutingDataSource extends AbstractRoutingDataSource {
 protected Object determineCurrentLookupKey() {
  return TheaterInfoContextHolder.getTheater();
 }
}
```

Custom DataSource 클래스는 AbstractRoutingDataSource를 상속 받고 determineCurrentLookupKey 메소드를 구현 해야 한다. 이 때, return 하는 키 값은 저장된 lookup key값에 매칭 되어야 한다.

다음으로 CustomDataSource를 bean으로 등록 해야 한다. bean 등록시 앞서 determineCurrentLookupKey 메소드에서 리턴 받는 값에 따라 각각 다른 url의 DB에 접근할 수 있도록 Map에 key값을 등록 해야 한다. 아래의 예시 코드는 context-routingdatasource.xml 에 정의된 coutomDataSource bean 정의 내용이다.

```
<bean id="customDataSource"
 class="org.anyframe.plugin.routingdatasource.common.CustomRoutingDataSource">
   <property name="targetDataSources">
    <map key-type="java.lang.String">
     <entry key="ABCCinema" value-ref="ABCCinema" />
     <entry key="GoodMovieCinema" value-ref="GoodMovieCinema" />
     <entry key="MovieHouseCinema" value-ref="MovieHouseCinema" />
    </map>
   </property>
   <property name="defaultTargetDataSource" ref="ABCCinema" />
</bean>
```

그리고 key값에 해당하는 DataSource를 bean으로 정의한다. 각각의 DataSource에 공통적인 내용이 중복될 경우 parentDataSource를 등록한 후 parentDataSource를 상속받는 bean을 등록 하는 방법도 있다.

```
<bean id="parentDataSource" class="org.apache.commons.dbcp.BasicDataSource" abstract="true">
   <property name="driverClassName" value="#{contextProperties.driver}" />
   <property name="username" value="#{contextProperties.username}" />
   <property name="password" value="#{contextProperties.password}"/>
</bean>

<bean id="ABCCinema" parent="parentDataSource">
   <property name="url" value="#{contextProperties.url}" />
</bean>

<bean id="GoodMovieCinema" parent="parentDataSource">
 <property name="url" value="jdbc:hsqldb:hsql://localhost:9002/sampledb2nd" />
</bean>

<bean id="MovieHouseCinema" parent="parentDataSource">
 <property name="url" value="jdbc:hsqldb:hsql://localhost:9003/sampledb3rd" />
</bean>
```

DB 정보 변경시 유의사항

change-db 명령을 사용하여 DB 정보를 변경 할 때 사용자가 정의한 Custom DataSource 중 contextProperties로 정의한 default 정보는 자동으로 변경이 된다. 하지만 위에서 예시로 설명한 GoodMovieCinema, MovieHouseCinema의 경우 사용자가 직접 DB 정보를 수정해야 한다.

등록한 customDataSource를 사용하는 QueryService를 새롭게 정의한다.

```
<query:auto-config id="queryServiceRoutingDataSource" dbType="hsqldb" dataSource-
ref="customDataSource"/>
```

1.2.ContextHolder

다음으로 TheadLocal을 이용하여 DB를 결정하는 key 값을 정의하는 ContextHolder 클래스를 작성 해야한다. 이 클래스는 Thread 내에서 보관하고 있는 DB정보 key 값을 꺼내고, 새로운 key값을 지정하고, Thread를 clear 하는 세 가지 기능을 제공해야 한다.

아래의 Java 코드는 상영관에 따라 DB에 접근하기 위해 상영관 정보를ThreadLocal 에 저장하는 ThreaterInfoContextHolder Class이다.

```
import org.anyframe.util.ThreadLocalUtil;

public class TheaterInfoContextHolder {

 public static String getTheater() {
  return (String) ThreadLocalUtil.get("theater");
 }

 public static void setTheater(String theater) {
  ThreadLocalUtil.add("theater", theater);
 }

 public static void clearTheaterInfo() {
  ThreadLocalUtil.clearSharedInfo();
  ThreadLocalUtil.add("theater", null);
 }
}
```

아래의 Java 코드는 ContextHolder를 이용하여 서로 다른 DB에 접근하도록 구현된 Testcase이다. 최초 DB 정보를 입력하지 않고 List 조회 했을 경우 defaultTargetDataSource 로 정의한 내용이 출력되고, theater 정보를 정의한 후 조회 했을 경우, 해당 상영관 DB에서 데이터를 조회하여 콘솔에 출력하는 예제이다.

```
@Test
public void testRoutindDataSource() throws Exception{
 Movie movie = new Movie();
 Page resultPage = movieFinder.getPagingList(movie, 1);
 assertNotNull("page is not null", resultPage);

 System.out.println(resultPage.getList());

 TheaterInfoContextHolder.setTheater("GoodMovieCinema");
 Page resultGoodMovieCinemaPage = movieFinder.getPagingList(movie, 1);
 assertNotNull("page is not null", resultGoodMovieCinemaPage);

 System.out.println(resultGoodMovieCinemaPage.getList());
```

```
 ThaterInfoContextHolder.setTheater("MovieHouseCinema");
 Page resultMovieHouseCinemaPage = movieFinder.getPagingList(movie, 1);
 assertNotNull("page is not null", resultMovieHouseCinemaPage);

 System.out.println(resultGoodMovieCinemaPage.getList());

 ThaterInfoContextHolder.setTheater("ABCCinema");
 Page resultABCCinemaPage = movieFinder.getPagingList(movie, 1);
 assertNotNull("page is not null", resultABCCinemaPage);

 System.out.println(resultGoodMovieCinemaPage.getList());

 ThaterInfoContextHolder.clearThaterInfo();
}
```

routingdatasource plugin에서는 theater 정보를 화면에서 parameter로 넘겨주도록 정의되어있다. 따라서 개발자는 Controller에서 request.getParameter("theater") 를 이용하여 key 값을 받아와서 ContextHolder에서 setTheater 메소드를 이용하여 값을 정의하면 된다. 각각의 Controller의 모든 메소드에서 request.getParameter("theater") 구문을 중복적으로 사용하는 것을 Interceptor 등록을 통해 쉽게 처리할 수 있다. 아래의 예제는 Interceptor에서 ThaterInfoContextHolder.setTheater 메소드를 이용하여 key값을 정의하도록 작성 되어졌다.

```
public class ThreadCleanupInterceptor extends HandlerInterceptorAdapter{

 @Override
 public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
 handler) throws Exception{
  String theater = request.getParameter("theater");
  ThaterInfoContextHolder.setTheater(theater);

  return super.preHandle(request, response, handler);
 }

 @Override
 public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
 Object handler, Exception ex) throws Exception{
  ThaterInfoContextHolder.clearThaterInfo();
 }
}
```

여기서 주의할 점은 요청이 끝난 후 반드시 ThreadLocal clear 작업을 수행 해야 한다는 점이다. Interceptor에서는 afterCompletion 메소드에서 cleanTheaterInfo() 메소드를 호출하여 ThreadLocal clear 작업을 지시 하고 있다.

2.주의 사항

2.1.RoutingDataSource 사용 한계

RoutingDataSource를 사용하여 동적으로 DB를 변경할 때 아래와 같은 문제점이 존재하므로, 개발자는 한계점을 반드시 인지 하고 있어야 한다.

- 동일한 DB Schema 사용

  routingdatasource plugin은 동일한 DB 스키마, 비즈니스 로직을 가지는 어플리케이션이 여러개의 DB에 접근해서 데이터를 처리하는 어플리케이션이다. RoutingDataSource를 사용하여서 Domain 객체뿐 아니라 DAO, Service 코드까지 공유 해서 사용하도록 설계 되어져 있다. 개발자는 이 예제와 같은 방식으로 어플리케이션을 개발 하기 위해서는 우선 DB Schema가 동일 해야 한다는 사실을 명심해야 한다. 또한 DAO Framework가 제공하는 Paging 기능을 사용하기 위해서 동일한 종류의 DB들을 대상으로 개발 하도록 해야한다.

- 1 transaction for 1 request

  일반적으로 Connection이 맺어지는 시점은 AOP로 Service 단위로 transaction을 설정한다. 따라서 Transaction을 시작하기 위해 Service 진입점에서 Connection이 맺어지므로, 동적으로 DB를 변경하기 위해서는 그 전에 DB Flag가 설정 되어 있어야 한다. 따라서 DB Flag를 ThreadLocal을 사용하여 공유하도록 설계 되어있다. 최초 한번 맺은 Connection이 Thread 내에서 공유 되어 재사용 되므로 Service 중간에 Flag를 변경하여도 DB가 변경 되지는 않는다. routingdatasource plugin의 코드를 살펴보면 Service 레벨에서 DB Flag를 변경하지 않는 것도 그 이유이다.

- ThreadLocal Cleanup

  routingdatasource plugin은 DB Flag를 Thread 단위로 설정하여 사용하도록 설계 되어있다. ThreadLocal을 사용할 때 가정 유의해야 할 점은 ThreadLocal에서 원하는 정보를 사용한 이후, clear 작업을 수행해야 한다는 점이다. 일반적으로 WAS는 Thread Pooling을 하며 다음 request가 할당받은 Thread에 이전 데이터 찌꺼기가 남게되어 의도하지 않게 이전 값을 참조하는 문제가 발생할 가능성이 있다. 따라서 request 종료 시점에는 ThreadLocal에 설정한 값을 null로 강제 설정하고, cleanup 작업을 수행 해야한다. routingdatasource plugin에서는 이런 작업을 Interceptor의 afterCompletion 메소드에 ThreadLocal clear 작업을 명시하는 방법으로 이 문제를 처리하였다.

2.2.예제 테스트를 위한 설정

플러그인 설치 후 예제 코드를 실행하기 위해서는 다음 설정이 추가로 필요하다(context-routingdatasource.xml).

```
<bean id="parentDataSource" class="org.apache.commons.dbcp.BasicDataSource" abstract="true">
    <property name="driverClassName" value="#{contextProperties.driver}" />
    <property name="username" value="#{contextProperties.username}" />
    <property name="password" value="#{contextProperties.password}"/>
   </bean>

   <bean id="ABCCinema" parent="parentDataSource">
    <property name="url" value="#{contextProperties.url}" />
   </bean>

   <bean id="GoodMovieCinema" parent="parentDataSource">
    <property name="url" value="jdbc:hsqldb:hsql://localhost:9002/sampledb2nd" />
   </bean>

   <bean id="MovieHouseCinema" parent="parentDataSource">
```

```
    <property name="url" value="jdbc:hsqldb:hsql://localhost:9003/sampledb3rd" />
  </bean>
```

위의 설정에서 보듯이 GoodMovieCinema와 MovieHouseCinema를 위한 ABCCinema와 동일한 DB Schema를 가지고 있는 DB를 설정해주어야 한다.

그렇지 않으면 DB연결이 되지 않아 "영화 목록을 조회 할 수 없습니다."라는 메시지를 보게 될 것이다.

3.Resources

- 참고자료
  - SpringSource Team Blog [http://blog.springsource.com/2007/01/23/dynamic-datasource-routing/]