Spring Hibernate, MVC

2018. 9. 18. 18:11

**데이터베이스 연동 방법

1.언어에서 제공하는 코드 만을 이용하는 방법(JDBC 코드 만을 이용해서 작성)

=>프로그램을 잘 만들면 성능은 가장 우수
=>개발 시간이 오래 걸릴 가능성이 높습니다.

2.프레임워크를 이용하는 방법

=>개발시간이 단축될 가능성이 높고 프로그램에 대한 깊은 지식이 없어도 성능이 나쁘지 않은 프로그램을 개발할 수 있습니다.
=>성능은 조금 떨어질 수 있고 프레임워크 변화에 민감합니다.

1)SQL Mapper: SQL 과 프로그래밍 언어의 분리, 자동 매핑, 예외를 좀 더 구체적으로 표현...
MyBatis가 대표적인 SQL Mapper Framework

2)ORM: 데이터베이스의 하나의 행과 객체를 직접 매핑, SQL 없이 작업 가능, 일반적으로 성능이 SQL Mapper 보다는 우수, 데이터베이스를 잘 알아야 합니다.
Java 에서는 Hibernate가 대표적입니다.
JPA 는 구현방식이고 Hibernate가 구현체 입니다.

**하이버네이트

=>Java ORM(Object Relation Mapping) Framework
=>Java 객체 1개와 테이블의 하나의 행을 매핑시켜 동작시키는 프레임워크
=>MyBatis 보다 성능이 우수한 편이지만 어렵습니다.
=>좋은 성능이 필요한 분야인 솔루션 개발에는 적합하지만 기간이 중요한 SI에는 부적합합니다.

**하이버네이트 연동 실습

1.하이버네이트 연동할 데이터 테이블 생성

--테이블 삭제

```
drop table goods;
```

-- 테이블 생성

```
create table Goods (
    code      number(5) not null,
    name     varchar2(50) not null,
    manufacture    varchar2(20),
    price     number(10) not null,
    primary key(code)
);
```

--샘플 데이터 입력

```
insert into Goods values(1, 'apple', 'korea', 1500);
insert into Goods values(2, 'watermelon', 'korea', 15000);
insert into Goods values(3, 'oriental melon', 'korea', 1000);
insert into Goods values(4, 'banana', 'Philippines', 500);
insert into Goods values(5, 'lemon', 'korea', 1500);
insert into Goods values(6, 'mango', 'taiwan', 700);


commit;


select * from Goods;
```

2.Simple Spring Maven(Spring을 사용하는 일반 Application 제작) 프로젝트를 생성

3.Java Version(1.8) 과 Spring Version(4.1.0) 을 변경

4.사용하고자 하는 라이브러리의 의존성을 설정 - pom.xml

=>oracle, spring-jdbc(데이터베이스 사용 시 무조건), spring-orm, hibernate(이미 포함된 상태)

1)oracle을 사용하는 경우는 repositories 를 추가해야 합니다.
=>dependencies 태그 외부에 작성

```
	<!-- 오라클을 사용하기 위해서 추가 -->
	<repositories>
		<repository>
			<id>codelds</id>
			<url>https://code.lds.org/nexus/content/groups/main-repo</url>
		</repository>
	</repositories>
```

2)의존성을 dependencies 태그 안에 작성

```
		<!-- 오라클 사용을 위한 설정 -->
		<dependency>
			<groupId>com.oracle</groupId>
			<artifactId>ojdbc6</artifactId>
			<version>11.2.0.3</version>
		</dependency>

		<!-- 스프링에서 데이터베이스를 사용할 때 설정 -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-jdbc</artifactId>
			<version>${spring-framework.version}</version>
		</dependency>

		<!-- 하이버네이트 사용을 위한 설정 -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-orm</artifactId>
			<version>${spring-framework.version}</version>
		</dependency>
```

5.Goods 테이블과 연동할 Domain 클래스를 생성

=>com.pk.domain.Good

```
package com.pk.domain;


public class Good {
	private int code;
	private String name;
	private String manufacture;
	private int price;


	public int getCode() {
		return code;
	}
	public void setCode(int code) {
		this.code = code;
```

```
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getManufacture() {
		return manufacture;
	}
	public void setManufacture(String manufacture) {
		this.manufacture = manufacture;
	}
	public int getPrice() {
		return price;
	}
	public void setPrice(int price) {
		this.price = price;
	}


	@Override
	public String toString() {
		return "Good [code=" + code + ", name=" + name + ", manufacture=" + manufacture + ",
price=" + price + "]";
	}
}
```

6.Goods 테이블과 Good 클래스를 매핑하기 위한 하이버네이트 설정 파일을 생성하고 작성

=>src/main/resources/hibernate/good.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">


<!-- 테이블과 객체 매핑 -->
<hibernate-mapping package="com.pk.domain">
	<!-- 테이블과 클래스를 연결 -->
	<class name="Good" table="Goods">
	<!-- 컬럼 이름 쓸 때 오라클은 대문자 나머지는 소문자 -->
	<!-- 기본키 연결 -->
	<id name="code" column="CODE"/>
	<!-- 나머지 컬럼 연결 -->
	<property name="name" column="NAME" />
	<property name="manufacture" column="MANUFACTURE" />
	<property name="price" column="PRICE" />


	</class>
</hibernate-mapping>
```

7.Spring Bean Configuration 파일을 추가해서 하이버네이트 사용 객체를 생성하는 코드를 추가

=>src/main/resources/applicationContext.xml
1)context, tx 네임스페이스 추가

2)bean 생성 코드 추가

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:tx="http://www.springframework.org/schema/tx"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd
		http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.1.xsd">


	<!-- 클래스에서 스프링 어노테이션을 사용할 수 있도록 해주는 설정 -->
	<context:annotation-config />
	<!-- 어노테이션이 설정된 클래스의 bean을 자동생성해주는 패키지 설정 -->
	<context:component-scan base-package="com.pk" />
	<!-- 트랜잭션 관련 어노테이션을 사용할 수 있도록 해주는 설정 -->
	<tx:annotation-driven/>


	<!-- 데이터베이스 접속정보를 저장하는 bean -->
	<bean id="dataSource"
		class="org.springframework.jdbc.datasource.DriverManagerDataSource">
		<property name="driverClassName"
			value="oracle.jdbc.driver.OracleDriver" />
		<property name="url"
			value="jdbc:oracle:thin:@192.168.10.101:1521:xe" />
		<property name="username" value="scott" />
		<property name="password" value="tiger" />
	</bean>


	<!-- 하이버네이트 설정 -->
	<bean
	class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
	<bean id="sessionFactory"
		class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
		<!-- 데이터베이스 접속 정보 -->
		<property name="dataSource"
			ref="dataSource" />
		<!-- 하이버네이트 설정 파일들의 위치를 지정 -->
		<property name="mappingResources">
			<list>
				<value>hibernate/good.hbm.xml</value>
			</list>
```

```
		</property>
		<!-- 데이터베이스 종류 설정 -->
		<property name="hibernateProperties">
			<value>
				hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
			</value>
		</property>
	</bean>


	<!-- 트랜잭션 적용을 위한 TransactionManager 객체 생성
	하이버네이트는 트랜잭션을 적용하지 않으면 예외를 발생시킵니다. -->
	<bean id="transactionManager"
	class="org.springframework.orm.hibernate4.HibernateTransactionManager">
		<property name="sessionFactory" ref="sessionFactory" />
	</bean>


</beans>
```

8.Hibernate를 사용할 Dao 클래스를 생성

```
package com.pk.dao;


import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


import com.pk.domain.Good;


//bean을 자동으로 만들어주는 어노테이션
@Repository
public class GoodDao {
	//동일한 자료형의 bean이 있는 경우 자동으로 주입받는 설정
	@Autowired
	//hibernate를 사용하기 위한 인스턴스 변수
	private SessionFactory sessionFactory;


	//메소드에서 예외가 발생하면 rollback 되고
	//예외가 발생하지 않으면 commit 되도록 해주는 어노테이션
	@Transactional
	//데이터를 삽입하는 메소드
	public void insertGood(Good good) {
		sessionFactory.getCurrentSession().save(good);
	}
}
```

9.main 메소드를 소유한 Main 클래스를 만들고 GoodDao를 가져와서 삽입하는 메소드 수행

```
import org.springframework.context.support.GenericXmlApplicationContext;


import com.pk.dao.GoodDao;
import com.pk.domain.Good;


public class Main {


	public static void main(String[] args) {
		GenericXmlApplicationContext context =
			new GenericXmlApplicationContext(
				"classpath:applicationContext.xml");


		GoodDao dao =
			context.getBean(GoodDao.class);


		Good good = new Good();
		good.setCode(7);
		good.setName("감귤");
		good.setManufacture("제주도");
		good.setPrice(700);


		dao.insertGood(good);


		context.close();


	}
}
```

****데이터 수정하기**

1.GoodDao 클래스에 데이터를 수정하는 메소드를 만들기

```
	@Transactional
	//데이터를 수정하는 메소드
	public void updateGood(Good good) {
		sessionFactory.getCurrentSession()
		.update(good);
	}
```

2.main 메소드 수정해서 실행하고 확인

```
GoodDao dao =
                context.getBean(GoodDao.class);


        Good good = new Good();
        good.setCode(7);
        good.setName("무화과");
        good.setManufacture("목포");
        good.setPrice(3000);


        dao.updateGood(good);
```

****전체 데이터 조회하기**

1.Dao 클래스에 전체 데이터를 조회하는 메소드를 만들기

```
@Transactional
    //전체 데이터를 조회하는 메소드
    public List<Good> list(){
        return (List<Good>)sessionFactory.
            getCurrentSession().
            createCriteria(Good.class).list();
    }


    @Transactional
    //code를 가지고 데이터를 조회하는 메소드
    public Good get(int code) {
        return (Good)sessionFactory.
            getCurrentSession().
            get(Good.class, code);
    }
```

2.main 메소드 수정해서 확인

```
public static void main(String[] args) {
        GenericXmlApplicationContext context =
            new GenericXmlApplicationContext(
                "classpath:applicationContext.xml");


        GoodDao dao =
            context.getBean(GoodDao.class);


        List<Good> list = dao.list();
        for(Good good : list) {
            System.out.println(good);
```

```
        }

        //데이터 1개 가져오기
        System.out.println("===============");
        System.out.println(dao.get(11));
        System.out.println(dao.get(12));
        context.close();

    }
```

****Spring MVC**

=>Spring 이 제공하는 MVC 패턴의 프로젝트
=>FrontController 패턴을 제공
=>FrontController는 직접 만들지 않고 Spring Bean Configuration 파일을 만들어서 생성
=>web.xml 파일에서 설정한 url 패턴과 서블릿 이름이 있을 때 서블릿이름-servlet.xml 파일로 WEB-INF 디렉토리에 만들면 됩니다.
이렇게 하면 url 패턴에 해당하는 요청이 왔을 때 서블릿이름-servlet.xml 파일에 있는 Controller 중에서 url을 처리할 수 있는 것을 찾아서 처리합니다.
=>클라이언트의 요청을 처리하는 Controller 를 생성할 때는 클래스를 하나 만들고 클래스 선언문 위에 @Controller를 추가하면 됩니다.

=>Controller에서 사용자의 요청을 처리하는 메소드

```
@RequestMapping("요청 URL")
public String 메소드이름(Model model){
    처리할 코드를 호출
    //request.setAttribute("키이름", 뷰에게 넘겨 줄 데이터);
    model.addAttribute("키이름", 뷰에게 넘겨 줄 데이터);
    //앞에서 만든 servlet.xml 파일의 ViewResolver 와 조합해서 실제 출력할 뷰 파일을 선택
    //기본은 forwarding
    return "뷰이름";
}
```

****하나의 요청을 만들고 그 요청을 Controller가 처리해서 결과 페이지로 출력하는 예제**

1.Spring MVC Project를 생성 - com.pk.mvc

2.Java Version(1.7 이상) 과 Spring Version(4.0.1 이상 - 4.1.0)을 변경

=>Spring MVC Project 에서는 src/main/webapp 디렉토리가 WebContent 디렉토리 입니다.

3.web.xml 파일의 모든 설정을 삭제

```
	<!-- 서블릿 클래스 등록 -->
	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-class>
	org.springframework.web.servlet.DispatcherServlet
		</servlet-class>
	</servlet>
	<!-- 서블릿 과 URL 매핑 -->
	<!-- 확장자가 do 인 요청이 오면 dispatcher-servlet.xml
	파일에서 만들어진 Controller 객체가 처리하도록 하는 설정 -->
	<servlet-mapping>
		<servlet-name>dispatcher</servlet-name>
		<url-pattern>*.do</url-pattern>
	</servlet-mapping>
```

4.애플리케이션의 시작 페이지로 사용할 파일을 webapp 디렉토리에 생성

=>index.jsp, index.html

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>시작 페이지</title>
</head>
<body>
	<a href="hello.do">클라이언트 요청</a>
</body>
</html>
```

5.클라이언트의 요청을 처리할 Controller 클래스를 생성하고 hello.do 요청을 처리

=>기본패키지.controller.DoController

```
package com.pk.mvc.controller;


import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
//Controller 클래스를 만들고 Bean을 자동생성하는 어노테이션
@Controller
public class DoController {
	//사용자의 요청을 처리하는 메소드
	@RequestMapping("hello.do")
	public String hello(Model model) {
		model.addAttribute("data", "Spring MVC");
		return "views/hello.jsp";
	}
}
```

6.WEB-INF 디렉토리에 Spring Bean Configuration 파일을 dispatcher-servlet.xml 파일로 생성하고 Bean을 자동으로 생성하는 패키지 설정을 추가

1)context 네임 스페이스 추가

2)코드 추가

```
<context:annotation-config />
<context:component-scan base-package="com.pk.mvc"/>
```

7.webapp 디렉토리에 출력할 파일을 생성하고 데이터를 출력합니다.

1)webapp 디렉토리에 views 디렉토리를 생성

2)views 디렉토리에 hello.jsp 파일을 생성하고 작성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>HELLO</title>
</head>
<body>
	data:${data}
</body>
</html>
```

> ****시작 페이지에서 hello.do 요청을 클릭했을 때 처리 과정**

hello.do 요청이 발생하면 dispatcher-servlet.xml 이 만든 FrontController가 요청을 받아서 자신이 만든 Controller 중에서 처리할 수 있는 Controller가 있는지 확인
없으면 404 에러이고 처리할 수 있으면 그 Controller에게 처리를 위임합니다.
Controller는 처리를 전부 수행하고 리턴되는 문자열과 Model에 저장한 데이터 이름과 데이터를 가지고 다시 dispatcher-servlet.xml 파일로 이동합니다.
그 파일에 ViewResolver 설정이 있으면 그 설정에 따라 리턴된 문자열과 조합해서 실제 출력할 파일을 결정하게 되고 ViewResolver 설정이 없으면 리턴된 문자열이 뷰의 위치가 됩니다.

> ****URL에서 파라미터가 아닌 URL의 일부분을 찾아오기**

=>Controller에서 @RequestMapping을 만들 때 /디렉토리이름/{변수명} 의 형식으로 만듭니다.
=>요청을 처리하는 메소드의 파라미터로 @PathVariable 자료형 변수명 을 추가해주면 메소드 내에서 변수가 값을 저장하고 있습니다.

1.index.jsp 파일에 요청을 생성

```
<a href="bloter/archives/319793">미래</a>
```

2.web.xml 파일에 bloter 라는 디렉토리가 포함된 url을 dispatcher-servlet.xml 파일이 처리할 수 있도록 서블릿 설정을 수정

```
<!-- 서블릿 클래스 등록 -->
	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-class>
	org.springframework.web.servlet.DispatcherServlet
		</servlet-class>
	</servlet>
	<!-- 서블릿 과 URL 매핑 -->
	<!-- 확장자가 do 인 요청이 오면 dispatcher-servlet.xml
	파일에서 만들어진 Controller 객체가 처리하도록 하는 설정 -->
	<servlet-mapping>
		<servlet-name>dispatcher</servlet-name>
		<url-pattern>*.do</url-pattern>
		<!-- bloter 디렉토리가 포함된 요청을 처리 -->
		<url-pattern>/bloter/*</url-pattern>
	</servlet-mapping>
```

3.Controller 클래스에 요청을 처리하는 코드를 추가

```
	//URL의 가장 마지막 부분을 분할해서 사용하
	@RequestMapping("/archives/{num}")
	public String archive(@PathVariable int num,
			Model model) {
		model.addAttribute("data", num);
		// bloter/archives 를 제거하기 위해서 ../../를 추가
		return "../../views/archives.jsp";
	}
```

4.webapp/views 디렉토리에 archives.jsp 파일을 만들고 실행하고 결과 확인

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>URL 뒷 부분 출력하기</title>
</head>
<body>
	번호:${data}
</body>
</html>
```