자바 스프링 프레임워크

1. 스프링 개요

1-1. 스프링 프레임워크

- 스프링 프레임워크는 주요 기능으로 DI, AOP, MVC, JDBC 등을 제공한다.

1-2. 스프링 프레임워크 모듈

- 스프링 프레임워크에서 제공하고 있는 모듈을 사용하려면, 모듈에 대한 의존설정을 개발 프로젝트에 XML 파일 등을 이용해서 개발자가 직접하면 된다.

스프링 모듈	기능
spring-core	스프링의 핵심인 DI(Dempenedncy Injection)와 IoC(Inversion of Control)를 제공
spring-aop	AOP 구현 기능 제공
spring-jdbc	데이터베이스를 쉽게(적은 양의 코드) 다룰 수 있는 기능 제공
spring-tx	스프링에서 제공하는 트랜잭션 관련 기능 제공
spring-webmvc	스프링에서 제공하는 컨트롤러(Controller)와 뷰(View)를 이용한 스프링 MVC 구현 기능 제공

1-3. 스프링 컨테이너(IoC)

- 스프링에서 객체를 생성하고 조립하는 컨테이너(container)로, 컨테이너를 통해 생성된 객체를 빈(Bean)이라고 부른다.

- 객체 생성 및 속성 데이터 작성(XML 문서) -> 스프링 컨테이너에서 객체 생성 및 조립(스프링 컨테이너 > 빈 생성 및 조립) -> 애플리케이션 구현(개발문서)

2. 개발 환경 구축

2-1. Java 설치

- JDK(Java Development Kit) 설치 : 개발자는 JDK가 있어야 개발을 할 수 있고, 단지 프로그램만을 사용하는 사용자라면 JRE만 설치되어 있으며 된다.

2-2. Path 설정

- javac.exe, java.exe(JVM 구동 명령 컴파일러)를 다른 디렉터리에서도 실행할 수 있도록 하기 위해 환경 변수(Path)에 bin 경로를 등록한다.

- JAVA_HOME 환경 변수 추가, Path 환경 변수에 JDK의 bin 디렉터리 추가

2-3. IDE(이클립스) 다운로드

- IDE(Integrated Development Environment : 통합 개발 환경) 설치 - eclipse

3. 스프링 프로젝트 생성

3-1. 프로젝트 생성

- New -> Project -> Maven Project -> Create a simple project(skip archetype selection), Use default Workspace location -> Group Id : spring4, Artifact Id : pjt03

3-2. pom.xml 작성

```
<dependencies>
 <dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>4.1.0.RELEASE</version>
 </dependency>
</dependencies>
```

```
<build>
 <plugins>
  <plugin>
   <artifactId>maven-compiler-plugin</artifactId>
   <version>3.1</version>
   <configuration>
    <source>1.8</source>
    <target>1.8</target>
    <encoding>utf-8</encoding>
   </configuration>
  </plugin>
 </plugins>
</build>
```

- 프로젝트의 JRE 라이브러리 버전이 메이븐 설정 파일에 명시되어 있는 버전과 일치하지 않아서 발생하는 것으로 프로젝트를 업데이트하라고 나온다.
- 우클릭 -> Maven -> Update Project

3-3. 폴더 및 pom.xml 파일의 이해

- pjt03 프로젝트 : 스프링 프로젝트 Root
- pjt03/src/main/java 폴더 : .java 파일 관리
- pjt03/src/main/resources 폴더 : 자원 파일 관리
- java 폴더의 경우 특별한 것은 없고, 앞으로 만들어지는 자바 파일들이 관리되는 폴더이다.
- resources 폴더의 경우 자원을 관리하는 폴더로 스프링 설정 파일(XML) 또는 프로퍼티 파일 등이 관리된다.
- java, resources 폴더는 스프링 프레임워크의 기본 구조를 이루는 폴더로 개발자는 이대로 폴더를 구성해야 한다.
- pom.xml 파일은 메이블 설정 파일로 메이븐은 라이브러리를 연결해주고, 빌드를 위한 플랫폼이다.
- pom.xml에 의해서 필요한 라이브러리만 다운로드 해서 사용한다.
- 라이브러리는 C:\Users\사용자\.m2\repository\org\springframework 경로에서 확인 가능하다. 참고로 모듈의 라이브러리 파일명은 artifactId + "-" + 버전 명 + ".jar"로 표시된다.

4. 처음해 보는 스프링 프로젝트

4-1. Java 파일을 이용한 프로젝트 실행

- New -> Class -> TransportationWalk

```
public void move() {
   System.out.println("도보로 이동합니다.");
}
```

- 똑같이 MainClass를 생성해준다.

4-2. 우선 따라 해 보는 스프링 프로젝트

- 스프링 방식의 '의존'을 이용하기 위해서는 Main에서 TransportationWalk 객체를 직접 생성하지 않고, 스프링 설정 파일(XML)을 이용해 보기로 한다. 가장 큰 차이점은 Java 파일에서 이용한 new 연산자를 이용하지 않고 스프링 설정 파일(XML)을 이용하는 것이다.
- New -> XML File -> applicationContext.xml

5. 또 다른 프로젝트 생성 방법

5-1. 폴더(java, resources)와 파일(pom.xml) 만들기

- 프로젝트 진행할 파일에 새폴더를 생성하고, pom.xml을 생성한다.

5-2. 이클립스에서 import 하기

- 우클릭 후 import -> Existing Maven Projects -> Root Directory에 경로 입력 -> Finish

6. DI(Dependency Injection)

6-1. DI(Dependency Injection)란?

- 배터리 일체형 -> 배터리가 떨어지면 장난감을 새로 구입해야 한다.

```
publicclassElectronicCarToy {

    privateBattery battery;

    publicElectronicCarToy() {
        battery= newNormalBattery();
    }
}
```

- 배터리 분리형 -> 배터리가 떨어지면 배터리만 교체하면 된다.

```
publicclassElectronicRobotToy {

    privateBattery battery;

    publicElectronicRobotToy() {
    }
    publicvoidsetBattery(Battery battery) {
        this.battery= battery;
    }
}
```

6-2. 스프링 DI 설정 방법

- 스프링 컨테이너 생성 및 빈(Bean) 객체 호출 과정 : 스프링 설정파일(applicationContext.xml) -> GenericXmlApplicationContext -> Spring Container(객체 getBean()) -> 빈(Bean) 객체를 필요로 하는 로직

- 클래스 구조 : MainClass -> 조립기(Java 언어 이용) : StudentAssembler(객체 생성 및 조립기), 조립기(XML 언어 이용) : applicationContext(객체 생성 및 조립기) -> service 클래스(ems.member.service) : StudentRegisterService(학생정보 등록), StudentSelectService(학생 정보 조회), StudentModifyService(학생 정보 수정), StudentDeleteService(학생 정보 삭제), StudentAllSelectService(전체 학생 정보 조회), EMSInformationService(시스템 정보) -> DAO 클래스(ems.member.dao) : StudentDao(학생 정보 관리)

7. 다양한 의존 객체 주입

7-1. 생성자를 이용한 의존 객체 주입

- publicStudentRegisterService(StudentDao studentDao) {

```
this.studentDao= studentDao;
```

```
}
 -> <bean id="registerService" class="ems.member.service.StudentRegisterService">
<constructor-arg ref="studentDao" ></constructor-arg>
</bean>
```

7-2. setter를 이용한 의존 객체 주입

```
 - publicvoidsetJdbcUrl(String jdbcUrl) {
this.jdbcUrl= jdbcUrl;
}
publicvoidsetUserId(String userId) {
this.userId= userId;
}
publicvoidsetUserPw(String userPw) {
this.userPw= userPw;
}
 -> <bean id="dataBaseConnectionInfoDev" class="ems.member.DataBaseConnectionInfo">
<property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />
<property name="userId" value="scott" />
<property name="userPw" value="tiger" />
</bean>
```

7-3. List 타입 의존 객체 주입

```
 - publicvoidsetDevelopers(List<String> developers) {
this.developers= developers;
}
 -> <property name="developers">
<list>
<value>Cheney.</value>
<value>Eloy.</value>
<value>Jasper.</value>
<value>Dillon.</value>
<value>Kian.</value>
</list>
</property>
```

7-4. Map 타입 의존 객체 주입

```
 - publicvoidsetAdministrators(Map<String, String> administrators) {
this.administrators= administrators;
}
 -> <property name="administrators">
<map>
<entry>
<key>
<value>Cheney</value>
</key>
<value>cheney@springPjt.org</value>
</entry>
```

<entry>
<key>
<value>Jasper</value>
</key>
<value>jasper@springPjt.org</value>
</entry>
</map>
</property>

8. 스프링 설정 파일 분리

8-1. 스프링 설정 파일 분리

- applicationContext.xml -> appCtx1.xml, appCtw2.xml, appCtx3.xml

8-2. 빈(Bean)의 범위

- 싱글톤(Singleton) : 스프링 컨테이너에서 생성된 빈(Bean) 객체의 경우 동일한 타입에 대해서는 기본적으로 한 개만 생성이 되며, getBean() 메소드로 호출될 때 동일한 객체가 반환된다.
- 프로토타입(Prototype) : 싱글톤 범위와 반대의 개념도 있는데 이를 프로토타입 범위라고 한다. 프로토타입의 경우 개발자는 별도로 설정을 해줘야 하는데, 스프링 설정 파일에서 빈(Bean) 객체를 정의할 때 scope 속성을 명시해 주면 된다.

9. 의존 객체 자동 주입

9-1. 의존 객체 자동 주입이란?

- 의존 객체 자동 주입이란? 스프링 설정 파일에서 의존 객체를 주입할 때 <constructor-org> 또는 <property> 태그로 의존 대상 객체를 명시하지 않아도 스프링 컨테이너가 자동으로 필요한 의존 대상 객체를 찾아서 의존 대상 객체가 필요한 객체에 주입해 주는 기능이다.
- 구현 방법은 @Autowired와 @Resource 어노테이션을 이용해서 쉽게 구현할 수 있다.
- 스프링 컨테이너 : Bean1, Bean2, Bean3 -> 자동 주입 -> Bean

9-2. @Autowired

- Bean -> 자동 주입(주입하려고 하는 객체의 타입이 일치하는 객체를 자동으로 주입한다.) -> Bean

9-3. @Resource

- Bean -> 자동 주입(주입하려고 하는 객체의 이름이 일치하는 객체를 자동으로 주입한다.) -> Bean

10. 의존 객체 선택

10-1. 의존 객체 선택

- 스프링 컨테이너 : Bean1, Bean2, Bean3 -> 자동 주입(-> Exception -> 동일한 객체가 2개 이상인 경우 스프링 컨테이너는 자동 주입 대상 객체를 판다하지 못해서 Exception을 발생시킨다.) -> Bean

10-2. 의존 객체 자동 주입 체크

- <!--<beanid="wordDao"class="com.word.dao.WordDao"> -->, @Autowired -> Exception -> @Autowired(required = false)

10-3. @Inject

- @Autowired와 거의 비슷하게 @Inject 어노테이션을 이용해서 의존 객체를 자동으로 주입을 할 수 있다.
- @Autowired와 차이점이라면 @Autowired의 경우 required 속성을 이용해서 의존 대상 객체가 없어도 익셉션을 피할 수 있지만, @Inject의 경우 required 속성을 지원하지 않는다.

- Bean -> 자동 주입(Exception, required 속성 X) -> Bean

11. 생명주기(Life Cycle)

11-1. 스프링 컨테이너 생명주기

- GenericXmlApplicationContext를 이용한 스프링 컨테이너 초기화(생성)
-> getBean()를 이용한 빈(Bean) 객체 이용
-> close()를 이용한 스프링 컨테이너 종료

11-2. 빈(Bean) 객체 생명주기

- 스프링 컨테이너 초기화(빈(Bean) 객체 생성 및 주입), 스프링 컨테이너 종료(빈(Bean) 객체 소멸)
- 빈(Bean) 객체의 생명주기는 스프링 컨테이너의 생명주기와 같이 한다.

11-3. init-method, destory-method 속성

12. 어노테이션을 이용한 스프링 설정

12-1-1. XML 파일을 Java 파일로 변경하기

- Application(.xml) -> container(Bean) => Application(.java) -> container(Bean)

12-2-1. Java 파일 분리

- .xml -> 분리 -> 1.xml, 2.xml, 3.xml
- .java -> 분리 -> 1.java, 2.java, 3.java

12-2-2. @Import 어노테이션

- .java -> 분리 -> 1.java, 2.java, 3.java -> import -> 1+2+3.java

13. 웹 프로그래밍 설계 모델

13-1. 웹 프로그래밍을 구축하기 위한 설계 모델

- Model1 : 브라우저(클라이언트) -> WAS(웹 어플리케이션 서버) : JSP<->Service&Dao -> 데이터베이스 -> WAS(웹 어플리케이션 서버) : JSP<->Service&Dao -> 브라우저(클라이언트)

- Model2 : 브라우저(클라이언트) -> WAS(웹 어플리케이션 서버) : Controller(<->View)<->Service<->DAO <- Model -> 데이터베이스

13-2. 스프링 MVC 프레임워크 설계 구조

- 브라우저(클라이언트) -> 요청 -> DispatcherServlet -> HandlerMapping, <-> HandlerAdapter (<요청처리> Controller), -> 처리결과를 출력할 view 선택 -> ViewResolver, -> 응답생성 -> View -> 응답(JSP)

13-3. DispatcherServlet 설정

- web.xml에 서블릿을 매핑 : WEB-INF 폴더의 web.xml 파일을 만들고, <servlet> 태그와 <servlet-mapping> 태그를 이용한다.

```
<servlet>
   <servlet-name>서블릿별칭</servlet-name>
   <servlet-class>서블릿명(패키지이름을포함한전체서블릿명)</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>서블릿별칭</servlet-name>
   <url-pattern>/맵핑명</url-pattern>
</servlet-mapping>
```

->

```
<servlet>
   <servlet-name>appServlet</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
   <servlet-name>appServlet</servlet-name>
   <url-pattern>/</url-pattern>
</servlet-mapping>
```

- DispatcherServlet(스프링 설정파일, 초기화 파라미터) -> servlet-context.xml 이용한 스프링 컨테이너(HandlerMapping, HandlerAdapter, ViewResolver) -> 초기화 파라미터에서 지정한 파일(servlet0context.xml)을 이용해서 스프링 컨테이너를 생성
- DispatcherServlet -> appServlet-context.xml 이용한 스프링 컨테이너(HandlerMapping, HandlerAdapter, ViewResolver) -> 초기화 파라미터에서 스프링 설정 파일을 지정하지 않은 경우 서블릿 별칭을 이용해서 스프링 컨테이너 생성

13-4. Controller 객체 - @Controller

- DispatcherServlet <- HandlerAdapter -> Controller
- servlet-context.xml <annotation-diven />

-> Controller 객체로 사용할 클래스 정의

```
@Controller
public class HomeController {
...
}
```

13-5. Controller 객체 - @RequestMapping

- http://localhost:8090/ch08/success

->

```
@RequestMapping("/success")
publicString success(Model model) {
   return "success";
}
```

13-6. Controller 객체 - Model 타입의 파라미터

- 개발자는 Model 객체에 데이터를 담아서 DispatcherServlet에 전달할 수 있다.
- DispatcherServlet에 전달된 Model 데이터는 View에서 가공되어 클라이언트한테 응답처리 된다.

```
@RequestMapping("/success")
publicString success(Model model) {
```

-> model.setAttribute("tempData", "model has data!!");

13-7. View 객체

- JSP 파일명 : /WEB-INF/views/success.jsp

13-8. 전체적인 웹 프로그래밍 구조

- 최초 사용자 요청(ex : http:/localhost:8090/ch08/success) -> DispatcherServlet ->

Controller(ex : @Controller 어노테이션이 적용된 클래스 검색) -> DispatcherServlet -> 사용자 요청에 해당하는 메서드 실행(ex : @RequestMapping("success") 어노테이션이 적용된 메서드 검색 및 실행) -> DispatcherServlet -> View 검색(ex : ViewResolver에 의해서 검색된 success.jsp 검색 alv 실행) -> View(ex : 브라우저에 JSP를 이용한 응답)

14. 스프링 MVC 웹 서비스 (1)

14-1. 웹 서버(Tomcat) 다운로드

- http://tomcatapache.org/에 들어가 zip 파일을 다운로드 하고, 압축 파일은 푼다.
- Window -> Show View -> Other -> Servers
- No servers are available. Click this link to create a new server... -> 해당 Tomcat 버전 선택 -> directory 선택 -> Finish

14-2. 웹 서버(Tomcat)와 이클립스 연동

- Use Tomcat installation, Publish module contextx to separate XML files
- HTTP/1.1의 Port Num : 8090

14-3. 이클립스에 STS(Spring Tool Suit) 설치

- Help -> Eclipse Marketplace... -> find -> sts -> install -> confirm -> I accept the terms of the license agreements -> Finish -> Restart Now

14-4. STS를 이용한 웹 프로젝트 생성

- Project Explorer 창에서 오른쪽 마우스 버튼을 클릭해서 New -> Other..을 선택한다.
- Select a wizard에서 spring으로 검색하면 spring과 관련된 프로젝트들이 보이고, Spring Legacy Project를 선택한다.
- Spring Legact Project에서 Spring MVC Project를 선택한다.
- 처음 Spring MVC Project를 생성하면 관련 파일을 다운로드 한다. Yes 버튼을 클릭하고 잠시 기다린다. 다운로드한 파일이 많으므로 다소 시간이 소요된다.
- 프로젝트 패키지 명을 입력한다. 패키지 명은 유니크한 이름으로 하고, 2단계 이상으로 해야 한다.
- 프로젝트 생성이 완료되면 Project Explorer 창에 프로젝트가 생성된 것을 확인할 수 있다. 그리고 간단한 예제가 기본적으로 들어있고, 실제 Tomcat 웹 서버를 이용해서 실행해 볼 수 있다. 프로젝트를 실행해 보기 위해서 프로젝트 이름 위에서 오른쪽 마우스를 클릭하고 Run As -> Run on Server를 클릭한다.
- 프로젝트를 실행하기 위해서 서버를 선택한다. 기존에 설정해 놓은 Tomcat 웹 서버를 이용한다. Next 버튼을 클릭한다.
- Configured에 프로젝트가 포함되며, Finish 버튼을 클릭하면 프로젝트가 Tomcat 웹 서버에서 실행된다.
- 브라우저에서 프로젝트가 실행된다. 기본적으로 탑재된 프로젝트는 서버의 시간을 출력해 주는 예제이다.

15. 스프링 MVC 웹 서비스 (2)

15-1. 프로젝트 전체 구조

- java 파일 : java 파일들이 위치한다. 주로 패키지로 묶어서 관리한다. 웹 애플리케이션에서 사용되는 Controller, Service, DAO 객체들이 위치한다.
- webapp : 웹과 관련된 파일들(스프링 설정 파일, JSP 파일, HTML 파일 등)이 위치한다.
- resources : JSP 파일을 제외한 html, css, js 파일 등이 위치한다.
- spring 폴더 : 스프링 컨테이너를 생성하기 위한 스프링 설정 파일이 위치한다.

- views 폴더 : View로 사용될 JSP 파일이 위치한다.
- pom.xml 파일 : 메인 레파지토리에서 프로젝트에 필요한 라이브러리를 내려받기 위한 메이븐 설정 파일

15-2. web.xml

- 사용자 -> 요청 -> DispatcherServlet -> HandlerMapping,
<-> HandlerAdapter <-> Controller,
<-> ViewResolver -> View
- 웹 어플리케이션에서 최초 사용자의 요청이 발생하면 가장 먼저 DispatcherServlet이 사용자의 요청을 받는다고 하였다. 따라서 개발자는 DispatcherServlet을 서블릿으로 등록해주는 과정을 설정해 주어야 한다. 그리고 사용자의 모든 요청을 받기 위해서 서블릿 맵핑 경로는 '/'로 설정한다.

15-3. DispatcherServlet

- 사용자 -> 요청 -> DispatcherServlet -> HandlerMapping(사용자의 요청에 부합하는 컨트롤러 검색),
<-> HandlerAdapter(사용자의 요청에 부합하는 컨트롤러의 메서드 실행 요청) <- ModelAndView 객체 이용 -> Controller(bussiness 로직 수행) -> Service -> DAO -> DB,
<-> ViewResolver -> View
- 사용자의 모든 요청을 DispatcherServlet이 받은 후 HandlerMapping 객체에 Controller 객체 검색을 요청한다. 그러면 HandlerMapping 객체는 프로젝트에 존재하는 모든 Controller 객체를 검색한다. HandlerMapping 객체가 Controller 객체를 검색해서 DispatcherServlet 객체에 알려주면 DispatcherServlet 객체는 다시 HandlerAdapter 객체에 사용자의 요청에 부합하는 메소드 검색을 요청한다. 그러면 HandlerAdapter 객체는 사용자의 요청에 부합하는 메소드를 찾아서 해당 Controller 객체의 메소드를 실행한다.
- Controller 객체의 메소드가 실행된 후 Controller 객체는 HandlerAdapter 객체에 ModelAndView 객체를 반환하는데 ModelAndView 객체에는 사용자 응답에 필요한 데이터 정보와 뷰 정보(JSP 파일)가 담겨있다. 다음으로 HandlerAdapter 객체는 ModelAndView 객체를 다시 DispatcherServlet 객체에 반환한다.

15-4. servlet-context.xml

- DispatcherServlet를 서블릿으로 등록될 때 contextConfigLocation 이름으로 초기화 파라미터를 servlet-context.xml로 지정하고 있는데 이때 지정된 servlet-context.xml 파일이 스프링 설정의 역할을 하는 파일이다.
- 스프링 설정 파일은 클래스로부터 객체(Bean)를 생성하고 조합하는 역할을 한다고 학습했다. servlet-context.xml에서도 마찬가지로 프로젝트에 필요한 객체(Bean)를 생성하고 조립한다.
- <resources mapping="/resources/**" location="/resources/" />
- /WEB-INF/views/ + Controller에서 받은 View 정보 : home + jsp = /WEB-INF/views/home.jsp
<beans:property name="prefix" value="/WEB-INF/views/" />
<beans:property name="suffix" value=".jsp" />

15-5. Controller(컨트롤러)

- DispatcherServlet -> HandlerAdapter -> Controller -> Service -> DAO -> DataBase
- Controller, Service, DAO : 사용자의 요청을 실제로 처리하는 객체들(실제로 개발자의 공수가 많이 투입된다.)
- @RequestMapping : 사용자로부터 '/success' 요청이 발생하면 success() 메서드를 실행,
- ("/success") : 사용자 요청에 대한 URL
- public String success(Model model) : success() 메서드 실행 후 뷰에서 활용되는 데이터를 담고

있는 객체
- return “success”; : 뷰로 사용되는 JSP 파일 이름

15-6. View(뷰)

- 메서드 반환값 -> return “home”;
- JSP 파일 -> /WEB-INF/views/home.jsp
- 사용자 응답 브라우저 -> http://localhost:8090/lec14/

16. STS를 이용하지 않은 웹 프로젝트

16-1. 스프링 MVC 웹 애플리케이션 제작을 위한 폴더 생성

- C:\spring\pjt\lec16Pjt001\src
- C:\spring\pjt\lec16Pjt001\src\main
- C:\spring\pjt\lec16Pjt001\src\main\java
- C:\spring\pjt\lec16Pjt001\src\main\webapp
- C:\spring\pjt\lec16Pjt001\src\main\webapp\resources
- C:\spring\pjt\lec16Pjt001\src\main\webapp\WEB-INF
- C:\spring\pjt\lec16Pjt001\src\main\webapp\WEB-INF\spring
- C:\spring\pjt\lec16Pjt001\src\main\webapp\WEB-INF\views

16-2. pom.xml 및 이클립스 import

- import -> import -> Existing Maven Projects
- Root Directory : C:\spring\pjt\pjt16, Projects : /pom.xml -> Finish

16-3. web.xml

- WEB-INF -> New -> Other.. -> web.xml -> Finish

16-4. 스프링 설정 파일(servlet-context.xml) 작성

- spring -> New -> Folder -> appServlet -> Finish
- appServlet -> New -> Other -> XML File -> servlet-context.xml -> Finish

16-5. root-context.xml 작성

- spring -> New -> Other -> XML File -> root-context.xml -> Finish

16-6. 컨트롤러와 뷰 작성

- 컨트롤러(HomeController.java)
- 뷰(home.jsp)

16-7. 실행

- 프로젝트 선택 -> Run As -> Run on Server

17. Service & Dao 객체 준비

17-1. 웹 애플리케이션 준비

- 웹 애플리케이션의 일반적으로 프로그램 구조 : 사용자 요청(브라우저) -> 프론트 컨트롤러(DispatcherServlet) -> 뷰(jsp 파일), 컨트롤러 -> 서비스 -> DAO(Data Access Object) -> Database
- New -> Project -> Spring Legacy Project -> Spring MVC Project -> Next -> 패키지 명 -> Finish -> Run As

17-2. 한글 처리

```
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
```

```
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
      <param-name>forceEncoding</param-name>
      <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

17-3. 서비스 객체 구현

- new 연산자를 이용한 서비스 객체 생성 및 참조

```
MemberService service = new MemberService();
```

- 스프링 설정파일을 이용한 서비스 객체 생성 및 의존 객체 자동 주입

```
<beans:bean id="service" class="com.bs.lec17.member.service.MemberService"></beans:bean>
```

-> @Autowired

```
MemberService service;
```

- 어노테이션을 이용해서 서비스 객체 및 의존 객체 자동 주입

```
@Repository("memService")
public class MemberService implements IMemberService {
```

-> @Resource(name="memService")

```
MemberService service;
```

17-4. DAO 객체 구현

- 어노테이션을 이용해서 DAO 객체 생성 및 의존 객체 자동 주입

```
@Repository
public class MemberDao implements IMemberDao {
```

-> @Autowired

```
MemberDao dao;
```

18. Controller 객체 구현 (1)

18-1. 웹 애플리케이션 준비

- .controller : 컨트롤러 객체
- .dao : DAO(Data Access Object)
- .service : 서비스 객페
- html : 웹 컴포넌트
- views : 뷰(JSP) 컴포넌트

18-2. @RequestMapping을 이용한 URL 맵핑

- 메소드에 @RequestMapping 적용

http://localhost:8090/lec18/memJoin -> memJoin() 실행

- 클래스에 @RequestMapping 적용

기능에 따른 컨트롤러 클래스 제작, 컨트롤러 안의 세부기능, 컨트롤러 안의 세부 기능

- @RequestMapping을 이용한 URL 맵핑
데이터 전송방식에 따른 method 속성 설정

18-3. 요청 파라미터

- HttpServletRequest 객체를 이용한 HTTP 전송 정보 얻기

```
ID : <input type="text" name="memId">
PW : <input type="password" name="memPw">
 ->
@RequestMapping(value="/memLogin", method=RequestMethode.POST)
public String memLogin(Model model, HttpServletRequest request) {
   String memId = request.getParameter("memId");
   String memPw = request.getParameter("memPw");
}
```

- @RequestParam 어노테이션을 이용한 HTTP 전송 정보 얻기

```
ID : <input type="text" name="memId">
PW : <input type="password" name="memPw">
 ->
@RequestMapping(value="/memLogin", method=RequestMethode.POST)
public String memLogin(Model model, @RequestParam("memId") String memId,
@RequestParam("memPw") String memPw) {
   Member member = service.memberSearch(memId, memPw);
}
```

- 커맨드 객체를 이용한 HTTP 전송 정보 얻기

memJoin.html -> Member.java -> MemberController.java

19. Controller 객체 구현 (2)

19-1. @ModelAttribute

- @ModelAttribute를 이용하면 커맨드 객체의 이름을 변경할 수 있고, 이렇게 변경된 이름은 뷰에서 커맨드 객체를 참조할 때 사용된다.
- 컨트롤러 -> 뷰

19-2. 커맨드 객체 프로퍼티 데이터 타입

- 데이터가 기조 데이터 타입인 경우

memberJoin.html -> Member.java

- 데이터가 중첩 커맨드 객체를 이용한 List 구조인 경우

memberJoin.html -> Member.java

19-3. Model & ModelAndView

- 컨트롤러에서 뷰에 데이터를 전달하기 위해 사용되는 객체로 Model과 ModelAndView가 있다. 두 객체의 차이점은 Model은 뷰에 데이터만을 전달하기 위한 객체이고, ModelAndView는 데이터와 뷰의 이름을 함께 전달하는 객체이다.

20. 세션, 쿠키

20-1. 세션(Session)과 쿠키(Cookie)

- Connectionless Protocol : 웹 서비스는 HTTP 프로토콜을 기반으로 하는데, HTTP 프로토콜은 클라이언트와 서버의 관계를 유지하지 않는 특징이 있다.

- 클라이언트 -> 1. 요청(Request) : 서버 연결 -> 서버 -> 2. 응답(Response) : 서버 연결 해제 -> 클라이언트

- Connectionless Protocol : 서버의 부하를 줄일 수 있는 장점은 있나, 클라이언트의 요청 시마다 서버와 매번 새로운 연결이 생성되기 때문에 일반적인 로그인 상태 유지, 장바구니 등의 기능을 구현하기 어렵다.

- 클라이언트 -> 1. 로그인 요청 -> 서버 -> 2. 로그인 성공 응답 -> 클라이언트 -> 3. 상품 주문 요청 -> 서버 -> 4. 로그인 유도 페이지 응답 -> 클라이언트

- 이러한 Connectionless Protocol의 불편함을 해결하기 위해서 세션과 쿠키를 이용한다. 세견과 쿠키는 클라이언트와 서버의 연결 상태를 유지해주는 방법으로, 세션을 서버에서 연결 정보를 관리하는 반면 쿠키는 클라이언트에서 연결 정보를 관리하는데 차이가 있다.

20-2. HttpServletRequest를 이용한 세션 사용

- 클라이언트 -> 1-1. 회원가입 요청 -> 서버 -> 1-2. 회원가입 응답 -> 클라이언트 -> 2-1. 로그인 요청 -> 서버 -> 2-2. 로그인 응답 : 세션 생성 -> 클라이언트 -> 3-1. 회원 정보 수정 요청 -> 서버 -> 3-2. 회원 정보 수정 응답 : 세션 이용 -> 클라이언트 -> 4-1. 회원 정보 삭제 요청 -> 서버 -> 4-2. 회원 정보 삭제 응답 : 세션 이용 -> 클라이언트

- 스프링 MVC에서 HttpServletRequest를 이용해서 세션을 이용하려면 컨트롤러의 메소드에서 파라미터로 HttpServletRequest를 받으면 된다.

20-3. HttpSession을 이용한 세션 사용

- HttpServletRequest와 HttpSession의 차이점은 거의 없으며, 단지 세션 객체를 얻는 방법에 차이가 있을 뿐이다.

- HttpServletRequest : 파라미터로 HttpServletRequest를 받은 후 getSession()으로 세션을 얻음.

- HttpSession : 파라미터로 HttpSession을 받아 세션 사용.

20-4. 세션 삭제

- 세션을 삭제하는 방법은 세션에 저장된 속성이 더 이상 필요 없을 때 이루어지는 과정으로 주로 로그아웃 또는 회원 탈퇴 등에 사용된다.

20-5. 세션 주요 메소드 및 플로어

- getId() : 세션 ID를 반환한다.
- setAttribute() : 세션 객체에 속성을 저장한다.
- getAttribute() : 세션 객체에 저장된 속성을 반환한다.
- removeAttribute() : 세션 객체에 저장된 속성을 제거한다.
- setMaxInactiveInterval() : 세션 객체의 유지시간을 설정한다.
- getMaxInactiveInterval() : 세션 객체의 유지시간을 반환한다.
- invalidate() : 세션 객체의 모든 정보를 삭제한다.

- 클라이언트 -> 1. 서버 연결 및 요청 -> 서버(2.setAttribute("member", mem), -> 세션(3. member 속성 저장)) -> 4. 응답 -> 클라이언트 -> 5. 요청 -> 서버(6. getAttribute("member") -> 세션(7. member 속성 반환)) -> 8. 응답 -> 클라이언트

20-6. 쿠키(Cookie)

- mallMain()에서 쿠키를 생성하고, 파라미터로 받은 HttpServletResponse에 쿠키를 담고 있다.

- 쿠키를 생성할 때는 생성자에 두 개의 파라미터를 넣어주는데 첫 번째 쿠키 이름을 넣어주고, 두 번째는 쿠키값을 넣어준다.

- @CookieValue 어노테이션의 value 속성은 쿠키 이름을 나타내는데, 만약 value에 명시한 쿠키가 없을 경우 익셉션이 발생한다. 익셉션을 막는 방법이 있는데, 바로 required 속성이다. Required 속성은 기본값으로 true를 가지고 있는데 required가 true인 경우 value 값에 해당하는 쿠키가 없으면 익셉견이 발생한다. 따라서 required 속성 값을 false로 설정해서 value 값에 해당하는 쿠키가 없어도 익셉션이 발

생하지 않도록 한다.

21. 리다이렉트, 인터셉트

21-1. 리다이렉트(redirect)

- 지금 페이지에서 특정 페이지로 전환하는 기능
- 회원 정보 수정 요청 -> 회원 인증? -> NO. 메인 페이지로 유도, YES. 회원 정보 수정 페이지로 유도
- modifyForm() -> 회원 인증> -> NO. return “redirect:/”;, YES. return “member/modifyForm”;

21-2. 인터셉터(interceptor)

- 리다이텍트를 사용해야 하는 경우가 많은 경우 HandlerInterceptor를 이용할 수 있다.
- Request -> DispatcherServlet -> HandlerInterceptor(인터페이스)(preHandle() -> Handler(Controller) -> postHandle() -> View -> afterCompletion()) -> Response

22. Database

22-1. 오라클 다운로드

- http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html

22-2. 오라클 설치

- 오라클 다운로드 한 압축 파일 압축 푼 후 setup.exe -> install -> Next -> accept -> 패스워드 설정 -> Next -> Install -> Finish

22-3. 계정 생성

- cmd 창에 들어가 sqlplus
- user-name, password 입력
- create user scott identified by tiger; -> User created.
- grant connect, resource to scott; -> Grant succeeded.
- exit

22-4. SQL developer

- http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html
- SQL developer를 실행하면 Java 경로를 설정해야 한다. 컴퓨터에 설치되어 있는 Java 경로를 설정한다. -> SQL developer가 실행된다. -> 실행되면 새로운 접속을 하기 위해서 왼쪽 접속 메뉴에서 오른쪽 마우스 버튼을 클릭해서 ‘새 접속’ 메뉴를 클릭한다.

23. JDBC

23-1. 기본 SQL

- member 테이블 생성

```
CREATE TABLE member(
 memId VARCHAR2(10) CONSTRAINT memId_pk PRIMARY KEY,
 memPw VARCHAR2(10),
 memMail VARCHAR2(15),
 memPurcNum NUMBER(3) DEFAULT 0 CONSTRAINT memPurNum_ck CHECK (memPurcNum < 3));
```

- member 테이블에 테스트 차원에서 ‘b’ 계정을 삽입한다.

INSERT INTO member(memId, memPw, memMail)
values('b', 'bb', 'bbb@gmail.com');

- member 테이블에서 'memId'가 'b'인 회원을 삭제한다.

DELETE FROM member WHERE memId = 'b';

- member 테이블의 모든 회원정보를 출력한다.

SELECT * FROM member;

- member 테이블을 삭제한다.

DROP TABLE member;

23-2. JDBC

- 드라이버 로딩 -> DB 연결 -> SQL 작성 및 전송 -> 자원 해제

24. JdbcTemplate

24-1. JDBC의 단점을 보완한 JdbcTemplate

- JDBC : 드라이버 로딩 -> DB 연결 -> SQL 작성 및 전송 -> 자원 해제
- JdbcTemplate : JdbcTemplate(드라이버 로딩, DB연결, 자원 해제) -> SQL 작성 및 전송

24-2. DataSource 클래스

- 데이터베이스 연결과 관련된 정보를 가지고 있는 DataSource는 스프링 또는 c3p0에 제공하는 클래스를 이용할 수 있다.
- 스프링 : org.springframework.jdbc.datasource.DriverManagerDataSource
- c3p0 : com.mchange.v2.c3p0. DriverManagerDataSource

25. 커넥션풀

25-1. c3p0 모듈의 ComboPooledDataSource

- com.mchange.v2.c3p0.ComboPooledDataSource

```
dataSource = new DriverManagerDataSoruce();
dataSource.setDriverClass(driver);
dataSource.setJdbcUrl(url);
dataSource.setUser(userid);
dataSource.setPassword(userpw);
```

25-2. 스프링 설정 파일을 이용한 DataSource 설정

```
<beans:bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
  <beans:property name="driverClass" value="oracle.jdbc.driver.OracleDriver" />
  <beans:property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />
  <beans:property name="user" value="scott" />
  <beans:property name="password" value="tiger" />
  <beans:property name="maxPoolSize" value="200" />
  <beans:property name="checkoutTimeout" value="60000" />
  <beans:property name="maxIdleTime" value="1800" />
  <beans:property name="idleConnectionTestPeriod" value="600" />
</beans:bean>
```