코드로 배우는 스프링 웹프로젝트

책 소개

Spring Framework를 사용해서 '웹 프로젝트'를 어떻게 진행하는지를 설명한다. 좀 더 구체적으로는 스프링으로 웹 프로젝트에서 사용되는 게시물 관리를 만들어 보는 것이 주된 목적이다. 모든 웹 프로젝트는 그 성격에 따라 구성과 구조가 다르기는 하지만 결과적으로는 게시물 관리 모듈의 집합체라고 볼 수 있다.

이번 개정판에서 스프링의 버전은 5.x 버전을 사용하고, 개발도구는 Spring Tool Suite(이하 STS) 혹은 Eclipse와 Maven으로 작성하며, 기존에 사용하던 XML과 Java Configuration과 어노테이션 기반의 설정을 이용한다. 최근 스프링 관련 예제나 프로젝트에서 XML 대신에 Java 설정을 이용할 때가 점점 증가하고 있으므로 이를 반영한다.

내용정리

Part 1 스프링 개발 환경 구축

1장 개발을 위한 준비

1.1 개발환경 설정

- Spring 버전에 따른 JDK 버전 호환성 체크

Spring	JDK
5	1.8이상
4	1.6이상
3	1.5이상

- Spring 버전에 따른 Tomcat 호환성 체크(아래 링크 표 참고)

  http://tomcat.apache.org/whichversion.html
- sts.ini(혹은 eclipse.ini) 파일편집

  맨 위에 다음 설정글 작성
  - Lombok라이브러리 오작동 예방

```
-vm
C:\Program Files\Java\jdk1.8.0_171\bin\javaw.exe
```

1.2 스프링 프로젝트 생성 (스프링 ver. 5)

- C:\Users\사용자명.m2\repository 에 라이브러리 설치됨
- 스프링 프로젝트 생성방식
  1. 처음부터 스프링 프로젝트를 지정하고 생성하는 방식
  2. Maven, Gradle 프로젝트를 생성한 후 프레임워크를 추가하는 방식
  3. 직접 프레임워크 라이브러리를 추가하는 방식
- p.31 상단 "스프링 기반프로젝트를 Maven기반으로 생성할 수 있습니다."
- pom.xml수정
  - Spring 버전수정 **3.1.1 --> 5.0.7**]

```
    <properties>
        <java-version>1.8</java-version>
        <org.springframework-
version>5.0.7.RELEASE</org.springframework-version>
        <org.aspectj-version>1.6.10</org.aspectj-version>
        <org.slf4j-version>1.6.6</org.slf4j-version>
    </properties>
```

  - Java버전수정 **1.6 --> 1.8**

```
          <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <compilerArgument>-Xlint:all</compilerArgument>
                    <showWarnings>true</showWarnings>
                    <showDeprecation>true</showDeprecation>
                </configuration>
            </plugin>
```

1.3 Tomcat을 이용한 프로젝트 실행 확인

- 프로젝트 실행시 발생하는 문제점 해결방법
  - .m2 폴더안 파일 모두 지우고 다시 이클립스 실행
  - Maven강제 업데이트
    - 프로젝트>마우스우클릭>Maven>Update Project
    - Force Update of Snapshots/Releases 체크

1.4 Lombok 라이브러리 설치

- 필요성 : 코드를 자동으로 작성해주는 기능 제공
- https://projectlombok.org 에서 다운로드
- `java -jar lombok.jar` 명령어 실행
- lombok설치시, 이클립스 설치된 폴더의 경로에 있는 eclipse.exe 지정해줌
- eclipse.exe가 위치한 폴더에 lombok.jar이 추가된 것을 확인
- test 와 lombok을 위한 의존성설치 (p.54)
- JUnit은 4.10 이상을 써야지 Spring 4.x 이상의 버전과 호환될 수 있다.

```
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId> <!--test를 위해 추가해야할 의존성
-->
            <version>${org.springframework-version}</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
```

1.5 Java Configuration을 하는 경우

- 필요성 : xml설정이 싫다, Java설정이 좋다.
- pom.xml 에서 plugin추가
  - `web.xml설정이 없다`는 설정

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>3.2.0</version>
  <configuration>
    <failOnMissingWebXml>false</failOnMissingWebXml>
</configuration>
</plugin>
```

- root-context.xml을 대신하는 Class작성(@Configuration 어노테이션)

```
package org.zeorck.config;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = {"org.zerock.sample"})//Bean객체 스캔
public class RootConfig {
    //여러 비즈니스 컴포넌트 Bean객체가 등록될 예정
}
```

- web.xml을 대신하는 Class작성

1. AbstractAnnotationConfig**DispatcherServlet**Initializer 추상클래스를 상속함
2. 3개의 추상메소드를 구현함

```
package org.zeorck.config;
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherSe
rvletInitializer;

public class WebConfig extends
AbstractAnnotationConfigDispatcherServletInitializer{

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return new Class[] {RootConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

2장 스프링의 특징과 의존성 주입

2.1 스프링 프레임워크의 간략한 역사

```
1. Enterprise Java Bean (EJB) 프레임워크에 반해 경량이며 빠르다
2. OOP구조를 뒷받침, 의존성 주입
3. 다른 프레임워크를 포용
4. 개발 생상성 증대와 개발도구 지원<hr>
```

2.2 의존성 주입 테스트

2.3 스프링이 동작하면서 생기는 일

- 스프링 xml파일의 namespace
- 스프링퀵스타트 p.56 참고

```
@Component
@Data
public class Restaurant {
    //방법1
    /*
    @Setter(onMethod_ =  {@Autowired})// lombok의 어노테이션을 굳이 쓰는 이유는?
```

```
    private Chef chef;
    */

    //방법2
    /*
    @Autowired//스프링에 종속적이게된다는 단점
    private Chef chef;
    */
  /*특히 테스트할때인데 단위 테스트를 하기위해 저 코드에서 PresonRepository를 Mock 객체로 끼워넣기위해서는 리플렉션을 쓰는수밖에 없다. 생성자도, 수정자(setter)도 없기때문이다. MockitoRunner 같은걸 이용하면 이 문제를 해결할 수 있기는한데 테스트하는 방식이 복잡해서 Runner를 여러개쓰는경우 SpringRunner와 MockitoRunner 를 같이 사용할 수가 없어서 곤란했던 적이 있다. 어디까지나 이는 실제 내 경험을 바탕으로 느낀 불편함이기때문에 이 외에도 몇가지 문제가 더 있을수있으나 근본적인 문제는 'POJO를 지향하는 스프링인데 필드주입때문에 스프링에 종속적이게 된다.' 라는 문제다.
*/


}
```

- @Setter(onMethod_ = {@Autowired}) vs @Autowired 차이점?
- 질문) @Data 가 있으면 @Setter가 필요없을 텐데.....?
  - 답) @Data 는 getter/setter, constructor, toString 모두 생성하지만, setter가 의존성 주입에 사용되길 원한다면 따로 @Autowired를 갖는 @Setter를 지정해준다.

- Lombok 관련
  - https://www.daleseo.com/lombok-popular-annotations/
  - ```
    @RunWith(SpringJUnit4ClassRunner.class)
    @ContextConfiguration(locations={"/app-config.xml", "/test-config.xml"})
    //@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-context.xml")
    @Log4j
    public class SampleTests {
        @Setter(onMethod_ = {@Autowired})
        private Restaurant restaurant;

        @Test
        public void testExist() {
            assertNotNull(restaurant);
            log.info(restaurant);
            log.info("------------------------------------");
            log.info(restaurant.getChef());
        }
    }
    ```

    `@RunWith` : `현 테스트 코드가 스프링을 실행하는 역할을 한다` 를 명시함

    `@ContextConfiguration` : 지정한 경로의 xml을 로딩해 Spring컨테이너에 bean객체를 생성한다.

    `@Log4j` : Lombok을 이용해서 로그를 기록하는 Logger변수를 생성해줌 따라서, 별도의 Logger객체 선언 없이도 `Log4j` 라이브러리와 설정이 존재한다면 바로 사용 가능

    `@Test` : JUnit이 테스트할 대상을 지정함
    ```

1. xml설정
    - xml경로를 `배열`로 지정 (**location속성**)
    - "file:src/main/webapp/WEB-INF/spring/root-context.xml" 지정 (**문자열 속성**)
2. java설정
    - classes속성으로 @Configuration 이 있는 클래스를 적용할 수 있다

2.4 스프링 4.3 이후

- 도서 SPRING QUICK START는 4.2 버전의 스프링을 사용함.
- Dependency Injection
    - p 네임스페이스 사용법 quick start (p.84)
- 4.3부터 추가된 기능들(lombok위주)
    - **생성자 직접 작성하지마~**

      ```
      @NoArgsConstructor
      @RequiredArgsConstructor// final이나 @NonNull인 필드 값만 파라미터로 받는 생성자
      @AllArgsConstructor
      ```

      사용 예)

      ```
      User user1 = new User();// @NoArgsConstructor
      User user2 = new User("dale", "1234");// @RequiredArgsConstructor
      User user3 = new User(1L, "dale", "1234", null);// @AllArgsConstructor
      ```

      (참고)@Data는 가운데 @RequiredArgsConstructor만 제공한다.
    - **생성자함수Injection의 개선**

      (Before) 생성자함수 + @autowired 조합으로 injection 진행

      (After) **생성자함수 만 작성해도 injection 암묵적으로 진행 (개선)**

      즉, @AllArgsConstructor 만 작성해서 생성자함수를 만들어주면 됨.

3장 스프링과 MySQL Database 연동

3.1 MySQL 설치

스키마 (데이터베이스) 'bizspring' 추가

포트 3306

3.2 MySQL WorkBench 설치

3.3 프로젝트의 JDBC 연결

굳이 jdbc.jar 파일을 다운받는 수고를 하지 말고 Maven의 도움을 받자

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.48</version>
        </dependency>
```

Test

```
package org.zeorck.sample;
import static org.junit.Assert.fail;
import java.sql.Connection;
import java.sql.DriverManager;
import org.junit.Test;
import lombok.extern.log4j.Log4j;
@Log4j
public class JDBCTests {
    String url = "jdbc:mysql://localhost/bizspring";
    static {
        try {
             Class.forName("com.mysql.jdbc.Driver");
        }catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }
    }

    @Test
    public void testConnection() {
        try(Connection con =
                DriverManager.getConnection(url, "root", "1234")){
            log.info(con);
        }catch (Exception e) {
            // TODO: handle exception
            fail(e.getMessage());
        }
    }
}
```

3.4 커넥션 풀 설정

- **필요한 의존성 3가지 (라이브러리)**

  https://gmlwjd9405.github.io/2018/05/15/setting-for-db-programming.html

  A. JDBC Driver - 통신 어댑터

  1. JDBC Driver는 자바 프로그램의 요청을 DBMS가 이해할 수 있는 프로토콜로 변환해주는 클라이언트 사이드 어댑터이다.
  2. DB마다 Driver가 존재하므로, 자신이 사용하는 DB에 맞는 JDBC Driver를 사용한다.
  3. DataSource를 JDBC Template에 주입(Dependency Injection)시키고 JDBC Template은 JDBC Driver를 이용하여 DB에 접근한다.

  B. Spring - JDBC (jdbc class)

  - 특징

1. Connection 열기와 닫기
2. Statement 준비와 닫기
3. Statement 실행
4. ResultSet Loop처리
5. Exception 처리와 반환
6. Transaction 처리

- JDBC Template 으로 이중 try ~ catch문을 안쓰도록 개선함
- 내부적으로 java.sql, javax.sql의 conn, stmt, resultSet 을 호출함.

C. DataSource

- JDBC `명세의 일부분`이면서 일반화된 `연결 팩토리`이다.
- DB와 관계된 `connection 정보`를 담고 있으며, bean으로 등록하여 sqlSessionFactoryBean을 생성할 때 인자로 넘겨준다. 이 과정을 통해 Spring은 DataSource로 DB와의 연결을 획득한다.
- DataSource는 JDBC Driver vendor(Mysql, Oracle 등) 별로 여러가지가 존재한다.
- DataSource가 하는 일
  - DB Server와의 기본적인 연결
    DB Connection Pooling 기능 (* 아래 참고)
    트랜젝션 처리
- DataSource의 구현 예시
  1. BasicDataSource
  2. PoolingDataSource
  3. SingleConnectionDataSource
  4. DriverManagerDataSource

보통의 방법 : properties에서 속성값을 취함

```
jdbc.driver = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql//localhost:3306/databaseName
jdbc.username = root
jdbc.password = password
https://gmlwjd9405.github.io/2018/05/15/setting-for-db-programming.html
```

```
<context:property-placeholder
location="com/spring/props/jdbc.properties"/>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
      <property name="driverClassName" value="${jdbc.driverClassName}"
/>
      <property name="url" value="${jdbc.url}" />
      <property name="username" value="${jdbc.username}" />
      <property name="password" value="${jdbc.password}" />
</bean>
```

설정 방법

방법1) 어플리케이션 측

- **XML 설정** root-context.xml

  (hikari DBCP를 이용하기 때문에 dataSource설정이 위의 코드와 살짝 다름에 유의)

```
    <!-- HikariCP configuration -->
    <bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
        <!-- 오라클인 경우 -->
        <!--<property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"></property>
        <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:XE">
</property> -->
        <property name="driverClassName" value="com.mysql.jdbc.Driver">
</property>
        <property name="jdbcUrl" value="jdbc:mysql://localhost/bizspring">
</property>
        <property name="username" value="root"></property>
        <property name="password" value="1234"></property>
    </bean>

    <!-- HikariCP DataSource -->
    <bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-
method="close">
        <constructor-arg ref="hikariConfig" />
    </bean>
```

- **java 설정** RootConfig.java

```
@Configuration
@ComponentScan(basePackages= {"org.zerock.sample"})
@MapperScan(basePackages= {"org.zerock.mapper"})
public class RootConfig {
      @Bean
      public DataSource dataSource() {
        HikariConfig hikariConfig = new HikariConfig();
        hikariConfig.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        hikariConfig.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:XE");
        hikariConfig.setUsername("book_ex");
        hikariConfig.setPassword("book_ex");
        HikariDataSource dataSource = new HikariDataSource(hikariConfig);
        return dataSource;
      }
}
```

Test.java

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-context.xml")
@Log4j
public class DBCPTests {
    @Setter(onMethod_ = { @Autowired })
    private DataSource dataSource;

    @Test
```

```
    public void testConnection() {
         //지양해야할 코드방식 (소스상에 개인정보가 있는건 좋지 않음)
         /* try(Connection con = DriverManager.getConnection(url, "root",
"1234")){
         * log.info(con); }catch (Exception e) { // TODO: handle exception
         * fail(e.getMessage()); }
         */
        try (Connection con = dataSource.getConnection()) {
            log.info(con);
        } catch (Exception e) {
            // TODO: handle exception
            fail(e.getMessage());
        }
    }
}
```

- JAVA Test

```
@RunWith(SpringJUnit4ClassRunner.class)
//@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-
context.xml")
@ContextConfiguration(classes= {RootConfig.class})
@Log4j
public class DBCPTests {
    @Setter(onMethod_ = { @Autowired })
    private DataSource dataSource;
    @Test
    public void testConnection() {
        try (Connection con = dataSource.getConnection()) {
            log.info(con);
        } catch (Exception e) {
            // TODO: handle exception
            fail(e.getMessage());
        }

    }
}
```

방법2) 서버 측 설정

- https://github.com/wonjin-do/Java-Handling-Web#DataBase

4장 MyBatis와 스프링 연동

4.1 MyBatis

- 개선점
  - Connection 직접 생성, 직접 close --> Connection 자동 close
  - PreparedStatement 직접 생성, 직접 close --> 내부적으로 PreparedStatement 처리
  - ? --> #{ }
  - ResultSet에서 칼럼을 각각 분석 --> 자동으로 객체생성

- 필요한 라이브러리 pom.xml
  - mybatis
    - mybatis-spring
  - spring-jdbc
    - spring-tx
- xml 설정 root-contex.xml
  - ```
    <bean id="sqlSessionFactory"
    class="org.mybatis.spring.SqlSessionFactoryBean">
            <property name="dataSource" ref="dataSource"></property>
        </bean>
    ```
- Java설정 RootConfig.java
  - 다음 코드를 RootConfig에 추가

```
@Configuration
@ComponentScan(basePackages= {"org.zerock.sample"})
@MapperScan(basePackages= {"org.zerock.mapper"})
public class RootConfig {


    @Bean
    public DataSource dataSource() {
        HikariConfig hikariConfig = new HikariConfig();

 hikariConfig.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        hikariConfig.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:XE");
        hikariConfig.setUsername("book_ex");
        hikariConfig.setPassword("book_ex");
        HikariDataSource dataSource = new
HikariDataSource(hikariConfig);
        return dataSource;
    }

    @Bean
    public SqlSessionFactory sqlSessionFactory() throws Exception {
        SqlSessionFactoryBean sqlSessionFactory = new
SqlSessionFactoryBean();
        sqlSessionFactory.setDataSource(dataSource());
        return (SqlSessionFactory) sqlSessionFactory.getObject();
    }
}
```

4.2 스프링과의 연동 처리

- Mapper인터페이스
```

```
package org.zerock.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {
    @Select("SELECT now() FROM dual")
    public String getTime();
}
```

- 위 인터페이스를 Bean등록 root-context.xml
  - namespace 로 mybatis-spring추가해야함.

```
    <mybatis-spring:scan base-package="org.zerock.mapper" />
```

- 자바설정일 경우 RootConfig.java

```
@MapperScan(basePackages = {"org.zerock.mapper"}) //클래스단에 설정
```

- Test

```
package org.zeorck.persistence;
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-
context.xml")
//@ContextConfiguration(classes= {RootConfig.class})
@Log4j
public class TimeMapperTests {
    @Setter(onMethod_ = { @Autowired })
    private TimeMapper timeMapper;

    @Test
    public void testGetTime() {
        log.info(timeMapper.getClass().getName());
        System.out.println("test");
        log.info(timeMapper.getTime());
    }
}
```

- 결론

  아래 코드처럼 connection을 가져오는 코드를 작성하지 않아도 됨. conn, stmt 관리를 MyBatis가 한번에 !!

```
        try (Connection con = dataSource.getConnection()) {
            log.info(con);
        } catch (Exception e) {
            // TODO: handle exception
            fail(e.getMessage());
        }
```

- xml 파일로 Mapper를 만드는 방식 두 가지
  - Spring 4.3 이전)

- mapper.xml 여러 장 작성

  --> mapperConfig 에 등록

  --> SqlSessionFactory Bean을 생성할 때, **dataSource**정보와 **mapperConfig.xml**을 input으로 준다.

- 4.3 이후)

  1. Mapper인터페이스에 대응하는 메소드 선언 (단, Annotation은 없다)

```
package org.zerock.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {

    @Select("SELECT sysdate FROM dual")
    public String getTime();

    public String getTime2();

}
```

  2. mapper.xml작성

     위치 : `Mapper인터페이스가 있는 패키지` 또는 `src/main/resources/org/zerock/mapper/`

     1. namespace 속성값은 **Mapper인터페이스의 패키지경로**
     2. sql 태그의 id 속성값은 **Mapper인터페이스에 있는 해당 메소드명**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.zerock.mapper.TimeMapper">

  <select id="getTime2" resultType="string">
  SELECT sysdate FROM dual
  </select>

</mapper>
```

4.3 log4jdbc-log4j2 설정

- SQL로그 확인하는 좋은 방법

  1. 라이브러리 추가

```
        <dependency>
            <groupId>org.bgee.log4jdbc-log4j2</groupId>
            <artifactId>log4jdbc-log4j2-jdbc4.1</artifactId>
            <version>1.16</version>
        </dependency>
```

2. properties 파일 추가

   파일명 : log4jdbc.log4j2.properties

```
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDele
gator
```

3. root-context.xml

   DriverClass와 JdbcURL 변경

```
<property name="driverClassName"
value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
<property name="jdbcUrl"
value="jdbc:log4jdbc:mysql://localhost:3306/bizspring"></property>
```

- 로그의 레벨 설정

  다음 코드를 src/main/resources 가 아닌 src/**test**/resources에 추가

```
    <logger name="jdbc.audit">
        <level value="warn" />
    </logger>

    <logger name="jdbc.resultset">
        <level value="warn" />
    </logger>
    <logger name="jdbc.connection">
        <level value="warn" />
    </logger>
```

Part 2 스프링 MVC 설정

5장 스프링 MVC의 기본 구조

5.1 스프링 MVC 프로젝트의 내부 구조

패키지 경로 org.zerock.controller

수정사항

1. 스프링5.0.7 과 자바1.8버전
2. Lombok / 테스트
3. 서블릿 3.1.0이상의 버전 (자바설정을 하지 않는다면 필요없음)
4. Maven 컴파일 옵션 1.8

스프링과 자바버전 수정

```
<properties>
        <java-version>1.8</java-version>
        <org.springframework-version>5.0.7.RELEASE</org.springframework-version>
```

Lombok 과 테스트에 필요한 의존성 설치

```
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${org.springframework-version}</version>
        </dependency>

        <!--etc  -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.0</version>
            <scope>provided</scope>
        </dependency>
```

Java설정을 이용하기 위해선 서블릿 3.1.0이상의 버전을 사용하도록 수정

```
<!-- Servlet -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
```

Maven 컴파일 옵션 1.8버전으로 변경

```
           <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <compilerArgument>-Xlint:all</compilerArgument>
                    <showWarnings>true</showWarnings>
                    <showDeprecation>true</showDeprecation>
                </configuration>
            </plugin>
```

(자바설정)

root-context.xml, servlet-context.xml, web.xml 이 없다는 설정. `Business Component`가 있음.

```
  <!-- 추가함 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.2.0</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
```

WebConfig.java (web.xml 대체) `톰캣구동`과 관련

```
package org.zerock.config;
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServle
tInitializer;
public class WebConfig extends
AbstractAnnotationConfigDispatcherServletInitializer{
    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return new Class[]{RootConfig.class};
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        // TODO Auto-generated method stub
        return new Class[]{ServletConfig.class};
    }
    @Override
    protected String[] getServletMappings() {
        // TODO Auto-generated method stub
        return new String[]{"/"};
    }

}
```

ServletConfig.java (servlet-context.xml 대체) `presentation - layer`임.

```
@EnableWebMvc
@ComponentScan(basePackages = { "org.zerock.controller", "org.zerock.exception"
})
public class ServletConfig implements WebMvcConfigurer {
    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {
        InternalResourceViewResolver bean = new InternalResourceViewResolver();
        bean.setViewClass(JstlView.class);
        bean.setPrefix("/WEB-INF/views/");
        bean.setSuffix(".jsp");
        registry.viewResolver(bean);
    }
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/resources/**").addResourceLocations("/resources/")
;
    }
```

```
    @Bean(name = "multipartResolver")
    public CommonsMultipartResolver getResolver() throws IOException {
        CommonsMultipartResolver resolver = new CommonsMultipartResolver();
        // 10MB
        resolver.setMaxUploadSize(1024 * 1024 * 10);
        // 2MB
        resolver.setMaxUploadSizePerFile(1024 * 1024 * 2);
        // 1MB
        resolver.setMaxInMemorySize(1024 * 1024);
        // temp upload
        resolver.setUploadTempDir(new FileSystemResource("C:\\upload\\tmp"));
        resolver.setDefaultEncoding("UTF-8");
        return resolver;
    }
}
```

RootConfig.java (root-context.xml 대체)

```
@Configuration
public class RootConfig {

}
```

5.2 예제 프로젝트의 로딩 구조

1. 서블릿컨테이너(Tomcat)은 `web.xml`을 로딩하여 구동된다.
2. ContextLoaderListener를 통해 `root-context`를 로딩하여 `스프링컨테이너ROOT`를 구동
3. 클라이언트에서 `localhost:8080/프로젝트/서블릿맵핑명` ----(요청)---> Tomcat
4. 요청을 받은 그제서야 서블릿컨테이너안에 DispatcherServlet객체 생성,
5. DispatcherServlet생성 직후, init( )실행으로 `servlet-context.xml`을 로딩하여 `스프링컨테이너` 구동

5.3 스프링 MVC의 기본 사상

Servlet/JSP의 HttpServletRequest / HttpServeltResponse 타입의 객체를 다루는 일은 매우 번거롭다.

SpringMVC구조를 통해 이 두 객체를 직접 다룰 일이 없어져 개발이 편해졌다.

5.4 모델2와 스프링 MVC

6장 스프링 MVC의 Controller

6.1 @Controller, @RequestMapping

- **@Controller**
  - servlet-contex.xml 에서 scan대상이 되면 @Contrllor가 있는 클래스에 대해 Spring컨테이너가 Bean객체로 생성해둠.

- **@RequestMapping**
  - Class단 / 메소드단 으로 설정가능함
- 참고
  - @Log 는 `java.util.Logging`을 이용
  - @Log4j 는 `Log4j 라이브러리`를 이용함.

6.2 @RequestMapping의 변화

- 스프링 4.3버전부터 @ReuquestMapping(value=" ", method={ }) 방식을 지양
- @GetMapping / @PostMapping 추가

6.3 Controller의 파라미터 수집

- HttpServletRequest의 필드를 분석하여 VO객체의 같은 이름을 가진 필드에 값을 대입하여 VO객체를 생성한다. 이후, Controoler 메소드에 파라미터로 넣어줌.
- getter / setter는 필수(@Data로 대체가능)
- 요청파라미터는 기본적으로 String 이다.
- Stirng값이 19, 200등 숫자형 문자열일 경우, VO객체의 필드에 맞게 int로 변환해준다.

1. Primitive, String 자료형을 이용할 때

- @RequestParam("name") 을 파라미터 옆에 작성하면, 요청파라미터의 key값을 name으로 변경해서 서버로 요청할 수 있다.

2. ArrayList, 배열 [ ] 처리
   - /projName/servletMapping?ids=a&ids=b&ids=c&ids=d 를 처리하는 방법
   - 파라미터 타입 ArrayList ids 또는 String [ ] ids 로 처리한다.
3. 객체리스트 처리
   - 기본 DTO의 리스트를 갖는 새 DTO를 설계한다
   - ```
     @Data//getter, setter, equals, toStirng
     public class SampleDTOList {
         private List<SampleDTO> list;
     }
     ```

- 위 2, 3번을 실습할 경우 Tomcat 버전에 따라 url의 특수문자 `[` 와 `]` 를 허용하지 않는 경우가 있다. 이때, `[`를 %5B로 `]`를 %5D로 변경하도록 한다.

4. @InitBinder
5. @DateTimeFormat

6.4 Model이라는 데이터 전달자

JavaBeans규칙

1. 생성자 없거나 , 빈생성자
2. getter/setter가 있다

- JavaBeans규칙에 들어맞는 DTO객체를 Command객체라고 한다. Controller이후 화면에서도 Command객체에 접근할 수 있음.
- 반면, primitive 형의 데이터 int, double 등은 @ModelAttribue("name")을 붙여야 View에서도 쓸 수가 있음.
     ```

- RedirectAttributes
  - 화면에 한 번만 사용하고 다음에는 사용되지 않는 데이터를 전달하기 위해서 사용합니다.

6.5 Controller의 리턴 타입

1. void 타입

   localhost:8080/sample/ex05 로 요청이 들어왔는데 Controller의 메소드의 리턴타입이 void라면, 결과는 ViewResolver를 통해 **/WEB-INF/views/sample/ex05.jsp**가 리턴된다.

2. String타입

   기본 : forward

   응용: redirect

3. 객체타입(JSON)

   JSON데이터를 리턴하기 위해서 라이브러리 필요함

```
    <!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
databind -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.9.4</version>
        </dependency>
```

   Controller의 메소드단에 @ResponseBody나 클래스단에 @Restfull 지정해줘야함

4. ResponseEntity타입

   HTTP 프로토콜의 헤더를 다루고 싶은 경우 (3번기능 포함)

   - 리턴타입은 `ResponseEntity<String>`
   - JSON, header, StateCode 를 동시에 보내는 방법

```
    @GetMapping("/ex07")
    public ResponseEntity<String> ex07() {
        String msg = "{\"name\": \"홍길동\"}";
        HttpHeaders header = new HttpHeaders();
        header.add("Content-Type", "application/json;charset=UTF-8");
        return new ResponseEntity<>(msg, header, HttpStatus.OK);
    }
```

5. 파일 업로드 처리 [Controller단에서 처리하는 것]

   - Servlet 3.0 이전
     - **commons**의 파일업로드 또는 **cos.jar** 를 이용해 업로드처리함
   - Servlet 3.0(Tomcat7)이후
     - 기본적으로 업로드되는 파일을 처리할 수 있는 기능이 추가됨
     - 더이상 추가적인 라이브러리 필요없음
     - Spring Legacy Project로 생성된 프로젝트는 Servlet2.5 기준이므로 사용불가능함.
   - 교재는 3.0이전 방법으로 진행함

```
<beans:bean id="multipartResolver"
class="org.springframework.web.mutipart.commons.CommonsMultipartResolver">
    <beans:property name="defaultEncoding" value="utf-8"></beans:property>
    <beans:property name="maxUploadSize" value="104857560"></beans:property>
    <beans:property name="maxUploadSizePerFile" value="2097152">
</beans:property>
    <beans:property name="uploadTempDir" value="file:/C:/upload/tmp">
</beans:property>
    <beans:property name="maxInMemorySize" value="10485756"></beans:property>
</beans:bean>
```

- maxUploadSize : 한 번에 Request로 전달될 수 있는 최대의 크기
- maxUploadSizePerFile : 파일 하나의 최대크기
- maxInMemorySize: 메모리상에서 유지하는 최대크기
- maxInMemorySize 이상의 데이터는 uploadTempDir에 임시파일의 형태로 보관된다.
- 절대경로로 표현하는 방법 **file:/** 로 시작
- defaultencoding : 파일이름이 한글일 경우 깨지는것을 방지해줌

6.6 Controller의 Exception 처리

- ControllerAdivce
- 404에러페이지
    - 500에러는 @ExceptionHandler를 이용해서 처러됨
    - 따라서, 404에러 처리해줄 일만 남았다.
    - appServlet 에 다음 코드를 추가

```
        <init-param>
            <param-name>throwExceptionIfNoHandlerFound</param-name>
            <param-value>true</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
```

Part 3 기본적인 웹 게시물 관리

7장 스프링 MVC 프로젝트의 기본 구성

7.1 각 영역의 Naming Convention(명명규칙)

7.2 프로젝트를 위한 요구 사항

7.3 예제 프로젝트 구성

7.4 데이터베이스 관련 설정 및 테스트

8장 영속/비즈니스 계층의 CRUD 구현

https://gmlwjd9405.github.io/2018/12/25/difference-dao-dto-entity.html

8.1 영속 계층의 구현 준비

두 가지 방법중 하나를 선택

Mapper Interface

- @SELECT( ) 등 어노테이션
- Mapper XML 파일

8.2 영속 영역의 CRUD 구현

방법1)

`useGeneratedKeys="true" keyProperty="bno"` 속성 두개로 Oracle의 nextVal()을 대체한다.

```
    <insert id="insert" useGeneratedKeys="true" keyProperty="bno">
        insert into tbl_board (bno,title,content,writer)
        values (#{bno}, #{title}, #{content}, #{writer})
    </insert>
```

방법2)

`<selectKey keyProperty="bno" order="BEFORE">` 로 선행작업을 진행시킨다.

```
<insert id="insertSelectKey">
        <selectKey keyProperty="bno" order="BEFORE" resultType="long">
            select max(bno)+1 from tbl_board
        </selectKey>
        insert into tbl_board (bno,title,content, writer)
        values (#{bno}, #{title}, #{content}, #{writer})
    </insert>
```

9장 비즈니스 계층

9.1 비지니스 계층의 설정

9.2 비즈니스 계층의 구현과 테스트

10장 프레젠테이션(웹) 계층의 CRUD 구현

10.1 Controller의 작성

10.2 BoardController의 작성

11장 화면 처리

11.1 목록 페이지 작업과 includes

11.2 목록 화면 처리

11.3 등록 입력 페이지와 등록 처리

11.4 조회 페이지와 이동

p.257 뒤로가기 방지방법

- history.state 값을 이용

```
            function checkModal(result) {

                        if (result === '' || history.state) {
                            return;
                        }
                        //새글 등록되고 수행할 작업
                        if (parseInt(result) > 0) {
                            $(".modal-body").html(
                                    "게시글 " + parseInt(result)
                                            + " 번이 등록되었습니다.");
                        }

                        $("#myModal").modal("show");
                    }
```

- 새 게시글 번호를 받아왔거나, history.state가 null이면 위 주석아래 부분을 실행
  - 새 게시글 번호가 없고, history.state가 not null이면 위 주석 아래부분 무시

11.5 게시물의 수정/삭제 처리

"/modify "

- `/get` 이 제공하는 정보와 같다.
- 따라서, 같은 Controller를 공유한다.
- 단, get.jsp는 readOnly 속성값을 갖는 input 태그들이 많다.
- 단, modify.jsp 의 날짜포멧은 get.jsp 와 달리 `yyyy/mm/dd` 의 포멧을 취한다

`<form>`태그 활용

- `<form>` 태그와 javascript를 활용하여 `<buttom>` 태그의 onClick Event를 제어한다.
- button 의 location.href 속성값을 직접이용하면 url이 장황해질 수 있다.
- `<form>` 태그의 action 속성, method 속성을 Javascript로 제어하여 `<form>` 태그를 절약할 수 있다.

p.262 자바스크립트 처리

- e.preventDefault
- data-oper속성을 이용
- p.266 formObj.empty() 는 왜하는 거지?

12장 오라클 데이터베이스 페이징 처리

MySQL로 대체하여 실습진행.

12.1 order by의 문제

- 시스템에 부하를 주지 않기 위해 가능하면 정렬을 하지 말아야함

12.2 order by 보다는 인덱스

12.3 인덱스를 이용하는 정렬

12.4 ROWNUM과 인라인뷰

MySQL에는 ROWNUM보다 편리한 **limit** 가 있다.

예) `select *from tbl_board limit 5, 10` : 검색결과의 5번째부터 시작하여 10개의 데이터를 조회한다.

13장 MyBatis와 스프링에서 페이징 처리

Criteria 추가

```
public class Criteria {

  private int pageNum; // 페이징 처리에서 현 페이지의 위치
  private int amount;   // 한 페이지에서 보여줄 게시글의 갯수
}
```

13.1 MyBatis 처리와 테스트

13.2 BoardController와 BoardService 수정

14장 페이징 화면 처리

페이지: 게시판 하단에 있는 번호 보통 10단위로 1 ~ 10, 11 ~ 20, 21 ~ 30 .....으로 위치함

1. DataBase로 부터 추출해오는 값
    - totoal (전체 게시글 수)
2. 사용자 입력으로 부터 얻어오는 값
    - Criteria(pageNum, amount)

14.1 페이징 처리할 때 필요한 정보들

1. pageNum (사용자 input)
2. total (DB 조회)
3. amount (상수, 일반적으로 10)

endPage (pageNum 를 통해 계산)

startPage(pageNum 를 통해 계산)

realEnd (total, amount 를 통해 계산)

14.2 페이징 처리를 위한 클래스 설계

```
@GetMapping("/list")
    public void list (Criteria cri, Model model) {

        //Criteria의 필드에 PageNum과 amount가 맵핑됨.
        model.addAttribute("list",service.getList(cri));
        model.addAttribute("pageMaker",new PageDTO(cri, 123));//123이 DB에서 받아
와야 할 total을 대신함
        System.out.println(new PageDTO(cri, 123));
    }
```

- Input
  - 사용자 측 (Criteria VO를 커맨드객체로 이용함)
    - PageNum : 게시판에 있는 여러 게시물들 中 사용자가 보고 있는 위치
    - amount : 여러 게시물을 최대 몇개 단위로 볼 것인지 정하는 값
  - DB 측
    - total : 게시물 총 갯수를 query를 통해 결과값 받아옴
- output
  - Input작업을 통해 PageDTO객체 생성
  - CriteriaVO에서 계산된 값
    - 게시물
      - start
        - 현재 페이지 PageNum 에서 첫 게시물 번호
        - start = (pageNum-1) * amount
      - end (계산할 필요없음)
        - MySQL query의 limit #{start} , #{amount} 로 가능함
          - #{start} + 1 부터 amount 갯수만큼 조회함.
  - PageDTO에서 계산된 값
    - 페이징
      - endPage
        - 하단 페이지 번호칸의 마지막 번호
        - 현재 pageNum이 결정함
        - endPage = (int) (Math.ceil(cri.getPageNum() / 10.0)) * 10
      - startPage
        - endPage - (pageLength - 1)
        - pageLength
          - 하단 페이지 번호칸들의 길이
      - prev, next
        - prev
          - startPage가 결정
          - `this.prev = this.startPage > 1;`

- next
  - endPage 와 realEnd 가 결정
  - ```
    this.next = this.endPage < realEnd;
    ```

- 마지막 페이지 단까지 온 상황
  - endPage값을 더 세밀하게 조정해줘야 함(값을 더 내림)

```
    int realEnd = (int) (Math.ceil((total * 1.0) /
cri.getAmount()));
    if (realEnd <= this.endPage) {
      this.endPage = realEnd;
    }
```

14.3 JSP에서 페이지 번호 출력

p.309 "태그의 href 속성값으로 페이지 번호를 가지도록 수정하는 것은 직접 링크를 처리하는 방식이며, 검색조건이 붙고난 후의 처리가 복잡하게 됨"

p.314도 마찬가지임

--> 게시글목록이 있는 /board/list 에서 `게시글 번호` 와 `페이징 번호`에 있는 값을 제어할 때,

태그와 hidden 속성, JavaScript를 이용해서 URL을 간단하게 만들어라

이유. 검색조건이 URL에 달리게 될 때 , 복잡함을 가중시킴

14.4 조회 페이지로 이동ㅍ페이지 이동을 위해 Criteria는 왠만하면 있음

수정, 삭제가 이뤄졌으면 더 이상의 데이터는

get.jsp -> list.jsp 로 이동시

get.jsp -> modify 로 이동시

- Criteria 객를 두어 돌아갈 정보를 저장해

14.5 수정과 삭제 처리

리다이렉트를 통해 더 이상의 request를 사용하지 않고 새 요청을 브라우저에게 요청함

15장 검색 처리

15.1 검색 기능과 SQL

15.2 MyBatis의 동적 SQL

- `<if>`
```

- `<choose>`
- `<trim prefix="a", prefixOverrides="b", suffix="c", suffixOverrides="d">`
  - trim태그 하위의 query 결과문을 살펴봤을 때,
  - 추가
    - prefix="a" : 앞에 a가 없으면 붙여라
    - suffix="c" : 뒤에 c가 없으면 붙여라
  - 제거
    - priefixOverrides="b" : 앞에 b가 있으면 지워라
    - suffixOverrides="d" : 뒤에 d가 있으면 지워라
  - `<if>` 태그가 여러개 있을 때는 각 태그마다 trim 이 매번 작동함
- `<where>` : query문 where 키워드 추가
- `<foreach>`
  - `<foreach index="key" item="val" collection="map">`
  - collection 자리
    - List일 때)
      - item 속성만
    - Map일 때)
      - idx, item 속성 모두가능

15.3 검색 조건 처리를 위한 Criteria의 변화

- Criteria
  - 기존
    - pageNum
    - amount
  - 추가
    - type
    - amount
    - String[] getTypeArr() : type(eg. "T", "W","C","TW","TC","TWC")를 각 문자로 분리시키는 메소드
  - ```
    <foreach item='type' collection="typeArr">
                    <trim prefix="OR">
                        .......
                    </trim>
                </foreach>
    ```
    - Mapper interface에 getListwithPaging(Criteria cri)메소드에 파라미터 cri가 있다.
    - **cri의 getTypeArr 메소드의 실행으로 반환값을 자동으로 받음**
  - SQL 모듈화
    - `<sql id="criteria">` 와 `<include refid="criteria">` 를 통해 sql문을 모듈화할 수 있다
    - 이렇게 모듈화된 sql쿼리문은 where 절에 편리하게 반복사용됨.
    ```

15.4 화면에서 검색 조건 처리

- pageMaker(= pageDTO)
  - Cri
    - pageNum
    - amount
    - type
    - keyword
  - startPage
  - endPage
  - prev
  - next
  - total
- pageMaker는 충분히 많은 데이터를 가지고 있어, 페이징에 관한 정보를 JSP에게 전달해준다.
- ```
  <form id='searchForm' action="/board/list" method='get'>
      <select name='type'>
      <option value=""
      <c:out value="${pageMaker.cri.type == null?'selected':''}"/>>--</option>
          .....
  ```
- cri의 정보를 통해 '검색조건' 과 '키워드'를 JSP의 화면이동간에 유지할 수 있다.
- 일부, 게시글 수정/삭제 후 redirect가 발생할 때는 cri 정보를 다른 jsp화면에 보낼 수가 없는데, 이를 해결할 방법은

  RedirectAttributes rttr 에 Cri의 필드를 적재시키는 것이다.

  ```
  @PostMapping("/modify")
      public String modify(BoardVO board, @ModelAttribute("cri") Criteria cri,
  RedirectAttributes rttr) {
          log.info("modify:" + board);

          if (service.modify(board)) {
              rttr.addFlashAttribute("result", "success");
          }

          rttr.addAttribute("pageNum", cri.getPageNum());
          rttr.addAttribute("amount", cri.getAmount());
          rttr.addAttribute("type", cri.getType());
          rttr.addAttribute("keyword", cri.getKeyword());

          return "redirect:/board/list";
      }
  ```

  - 매번 Cri 파라미터를 유지하는 것이 번거롭다면 , Criteria 클래스에 getListLink() 를 추가한다.

    ```
    UriComponentsBuilder builder = UriComponentsBuilder.fromPath("")
        .queryParam("pageNum", this.pageNum)
        .queryParam("amount", this.getAmount());
    return builder.toUrlString();
    ```
    ```

Part 4 REST 방식과 Ajax를 이용하는 댓글 처리

16장 REST 방식으로 전환

16.1 @RestController

의존성 설치

이유: Java객체 ----> JSON객체로 변환

```
<dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.9.6</version>
        </dependency>

        <!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-
dataformat-xml -->
        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-xml</artifactId>
            <version>2.9.6</version>
        </dependency>
```

이유: Java객체 ---> JSONString

```
<dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.2</version>
        </dependency>
```

16.2 @RestController의 반환 타입

```
public class SampleController {

    @GetMapping(value = "/getText", produces = "text/plain; charset=UTF-8")
    public String getText() {

        log.info("MIME TYPE: " + MediaType.TEXT_PLAIN_VALUE);

        return "안녕하세요";

    }
```

- produces 속성
  - MediaType의 상수를 사용할 수 있음.
  - 생략가능
    - 기본 xml 리턴
    - url주소 맨 뒤에 .json을 붙이면 JSON객체 리턴
- ResponseEntity< > 리턴타입
  - Controller안 메소드 리턴타입의 한 종류
  - 구성요소
    - 결과데이터
    - StateCode
  - ```
    @GetMapping(value = "/check", params = { "height", "weight" })
        public ResponseEntity<SampleVO> check(Double height, Double weight)
    {

            SampleVO vo = new SampleVO(000, "" + height, "" + weight);

            ResponseEntity<SampleVO> result = null;

            if (height < 150) {
                result =
    ResponseEntity.status(HttpStatus.BAD_GATEWAY).body(vo);
            } else {
                result = ResponseEntity.status(HttpStatus.OK).body(vo);
            }

            return result;
        }
    ```

16.3 @RestController에서 파라미터

서버는 브라우저의 데이터를 어디에서 찾을까?

- 방법1 (@PathVariable)
  - URL 의 query문 이전 경로에서
  - /sample/{sno}/page/{pno}
  - Controller안 메소드의 파라미터에 @PathVariable("sno") String sno 로 사용됨
  - query를 쓰지 않는 현명한 방법
- 방법2 (@RequestBody)
    ```

- @ResponseBody 아님, 헤깔리지 말것
- request의 body에 있는 파라미터를 JSON으로 변환함.
- 즉, POST방식
- 게시글을 생성할 때(RESTful 규칙에 맞게)

16.4 REST 테스트

```
@Log4j
public class SampleControllerTests {

  @Setter(onMethod_ = { @Autowired })
  private WebApplicationContext ctx;

  private MockMvc mockMvc;

  @Before
  public void setup() {
    this.mockMvc = MockMvcBuilders.webAppContextSetup(ctx).build();
  }

  @Test
  public void testConvert() throws Exception {

    Ticket ticket = new Ticket();
    ticket.setTno(123);
    ticket.setOwner("Admin");
    ticket.setGrade("AAA");

    String jsonStr = new Gson().toJson(ticket);

    log.info(jsonStr);

    mockMvc.perform(post("/sample/ticket")
        .contentType(MediaType.APPLICATION_JSON)
        .content(jsonStr))
        .andExpect(status().is(200));
  }

}
```

16.5 다양한 전송 방식

작업	전송방식	URI
Create	POST	/member/new
Read	GET	/member/{id}
Update	PUT	/member/{id} + body
Delete	DELETE	/member/{id}

17장 Ajax 댓글 처리

17.1 프로젝트의 구성

- log4jdbc.log4j2.properties파일 추가
  - 로그를 남겨 서버에서 진행되는 일들을 가시화한다.

17.2 댓글 처리를 위한 영속 영역

ReplyVO 클래스 추가

- rno
- bno
- reply
- replyer
- replyDate
- updateDate

두개 이상의 파라미터를 Mapper가 사용할 상황

- 이유
  - bno : 게시물의 번호
  - Cri : 그 게시물에 관련된 모든 댓글목록및 페이징처리

```
    public List<ReplyVO> getListWithPaging(@Param("cri") Criteria cri,
@Param("bno") Long bno);
```

@Param("a") Type x 해석: 인터페이스에서 x로 받은 데이터를 Mapper.xml 에서 a로 사용하겠다.

17.3 서비스 영역과 Controller 처리

REST방식으로 동작하는 URL을 설계할 때는 PK를 기준으로 작성하는 것이 좋습니다.

@RequestBody를 하는 이유 : String이 전송될 때는 어노테이션없이도 메소드의 파라미터로 오는 vo 객체의 필드에 자동 바인딩이 되나, Json String이 전송될 때는 @RequestBody어노테이션을 꼭 붙여야한다.

예제 코드

```
@RequestMapping(method = { RequestMethod.PUT,
            RequestMethod.PATCH }, value = "/{rno}", consumes =
"application/json", produces = {
                    MediaType.TEXT_PLAIN_VALUE })
    public ResponseEntity<String> modify(
              @RequestBody ReplyVO vo, //ReplyVO에 rno 가 있으나, request body에서
만 맵핑하기 때문에
              @PathVariable("rno") Long rno) {
```

```
        vo.setRno(rno);

        log.info("rno: " + rno);
        log.info("modify: " + vo);

        return service.modify(vo) == 1
                ? new ResponseEntity<>("success", HttpStatus.OK)
                : new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);

    }
```

ReplyVO에 rno 가 있으나, request body에서만 맵핑하기 때문에 vo 객체의 rno는 null임.

따라서, @RequestBody를 항상 Post와 같이 사용하는건 아니다. 서버가 JSONString을 전송받을 때 사용

17.4 JavaScript 준비

즉시실행함수를 통해 변수의 은닉화, 모듈화를 구현한다.

```
var replyService = (function() {

    function add(reply, callback, error) {
        console.log("add reply..............");

        $.ajax({
            type : 'post',
            url : '/replies/new',
            data : JSON.stringify(reply),
            contentType : "application/json; charset=utf-8",
            success : function(result, status, xhr) {
                if (callback) {
                    callback(result);
                }
            },
            error : function(xhr, status, er) {
                if (error) {
                    error(er);
                }
            }
        })
    }
    return {
        add : add,

    };
}
```

17.5 이벤트 처리와 HTML 처리

- get.jsp (코드량이 상당함)
    - $(document).ready(function(){}) // showList() 호출함

    - showList(page) // 화면처리를 담당
    - 이벤트
        - addReplyBtn
        - modalRegisterBtn
        - .chat
        - replyPageFooter
- reply.js (get.jsp에서 사용됨)
    - add
    - getList
    - update
    - get
    - displayTime

17.6 댓글의 페이징 처리

ReplyVO에 대한 페이징은 boardVO에 대한 페이징처리와 동일하다

- ReplyPageDTO 클래스
    - replyCnt // 댓글의 전체 수
    - List `<ReplyVO>` // 전체 댓글정보
- pageNum 은 사용자의 입력값에 따라 유동적

17.7 댓글 페이지의 화면 처리

get.jsp // get.jsp하단의 이벤트 함수에서 많이 재사용됨.

```
function showList(page){ // 정의

      console.log("show list " + page);

    replyService.getList({bno:bnoValue,page: page|| 1 }, function(replyCnt,
list) {

    console.log("replyCnt: "+ replyCnt );
    console.log("list: " + list);
    console.log(list);
    //page 값은 -1 아니면 else
    if(page == -1){
      pageNum = Math.ceil(replyCnt/10.0); //realEnd 값임
      showList(pageNum); //재귀함수.
      return;
    }
     var str="";
     if(list == null || list.length == 0){
       return;
     }
     for (var i = 0, len = list.length || 0; i < len; i++) {
       str +="<li class='left clearfix' data-rno='"+list[i].rno+"'>";
       str +="  <div><div class='header'><strong class='primary-font'>["
           +list[i].rno+"] "+list[i].replyer+"</strong>";
       str +="    <small class='pull-right text-muted'>"
           +replyService.displayTime(list[i].replyDate)+"</small></div>";
       str +="    <p>"+list[i].reply+"</p></div></li>";
```

```
      }
      replyUL.html(str);
      showReplyPage(replyCnt);
    });//end function
  }//end showList
```

Part 5 AOP와 트랜잭션

18장 AOP라는 패러다임

18.1 AOP 용어들

- Target : 순수 비즈니스 로직
- Proxy : 비즈니스 로직을 감싸고 있고, Target을 호출
- JoinPoint : 순수 비즈니스 로직이 가진 모든 메소드
- Point Cut : 관심있는 메소드
- Aspect : pointcut + 관심사(Advice)

18.2 AOP 실습

LogAdivce.java

```
@Aspect
@Log4j
@Component
public class LogAdvice {

  @Before( "execution(* org.zerock.service.SampleService*.*(..))")
  public void logBefore() {

    log.info("========================");
  }
}
```

root-context.xml

```
<context:component-scan
        base-package="org.zerock.service"></context:component-scan>
    <context:component-scan
        base-package="org.zerock.aop"></context:component-scan>

    <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
```

19장 스프링에서 트랜잭션 관리

19.1 데이터베이스 설계와 트랜잭션

19.2 트랜잭션 설정 실습

20장 댓글과 댓글 수에 대한 처리

20.1 프로젝트수정

BoardVO 수정

- bno
- title
- content
- writer
- regdate
- updateDate
- replyCnt (추가) -- Board 영역에 Reply 필드가 있음. Transaction처리를 해야하는 이유

list.jsp 화면에 Board와 Reply 를 동시에 표현해야함.

```
public class ReplyServiceImpl implements ReplyService {

    @Setter(onMethod_ = @Autowired)
    private ReplyMapper mapper;

    @Setter(onMethod_ = @Autowired)
    private BoardMapper boardMapper;

    @Transactional
    @Override
    public int register(ReplyVO vo) {

        log.info("register......" + vo);

        boardMapper.updateReplyCnt(vo.getBno(), 1);

        return mapper.insert(vo);

    }

    @Transactional
    @Override
    public int remove(Long rno) {

        log.info("remove...." + rno);

        ReplyVO vo = mapper.read(rno);

        boardMapper.updateReplyCnt(vo.getBno(), -1);
        return mapper.delete(rno);

    }

}
```

Part 6 파일 업로드 처리

21장 파일 업로드 방식

21.1 스프링의 첨부파일을 위한 설정

web.xml 변경(첨부파일 처리에 대한 톰캣WAS자체 설정)

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    id="WebApp_ID" version="3.1">
```

추가

```
<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>

        <multipart-config>
            <location>C:\\upload\\temp</location> //업로드 경로
            <max-file-size>20971520</max-file-size> <!--1MB * 20 -->
            <max-request-size>41943040</max-request-size><!-- 40MB -->
            <file-size-threshold>20971520</file-size-threshold> <!-- 20MB -->
        </multipart-config>


    </servlet>
```

- 특정 사이즈의 메모리 사용(file-size-threshold)
- 업로드되는 파일을 저장할 공간(location)
- 업로드 되는 파일의 최대크기(max-file-size)
- 한번에 올릴 수 있는 최대 크기(max-request-size)

servlet-context.xml에 multipartResolver Bean 등록

```
    <context:component-scan
        base-package="org.zerock.controller" />

    <beans:bean id="multipartResolver"

class="org.springframework.web.multipart.support.StandardServletMultipartResolve
r">

    </beans:bean>
```

21.2 방식의 파일업로드

multiple을 명시한다(input태그에)

```
<form action="uploadFormAction" method="post" enctype="multipart/form-data">

<input type='file' name='uploadFile' multiple>

<button>Submit</button>

</form>
```

```
    @PostMapping("/uploadFormAction")
    public void uploadFormPost(MultipartFile[] uploadFile, Model model) {

        String uploadFolder = "C:\\upload";

        for (MultipartFile multipartFile : uploadFile) {

            log.info("-------------------------------------");
            log.info("Upload File Name: " +
multipartFile.getOriginalFilename());
            log.info("Upload File Size: " + multipartFile.getSize());
            uploadFileName = multipartFile.getOriginalFilename();//IE읽 경우 전체
경로를 리턴해옴.
            // IE 일 경우 subString작업을 추가로 요함
           // uploadFileName =
uploadFileName.substring(uploadFileName.lastIndexOf("\\") + 1);

            File saveFile = new File(uploadFolder, uploadFileName);

            try {
                multipartFile.transferTo(saveFile);//업로드 진행
            } catch (Exception e) {
                log.error(e.getMessage());
            } // end catch
        } // end for
    }
```

21.3 Ajax를 이용하는 파일 업로드

Q. p.502 상단 inputFIle[0] 에서 idx 0의 의미는??

`브라우저`에서Ajax로 FormData를 전송

```
<div class='uploadDiv'>
    <input type='file' name='uploadFile' multiple>
</div>

<div class='uploadResult'>
    <ul>

    </ul>
</div>


<button id='uploadBtn'>Upload</button>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"
    integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
    crossorigin="anonymous"></script>

<script>
    $(document).ready(function(){
         var regex = new RegExp("(.*?)\.(exe|sh|zip|alz)$");
        var maxSize = 5242880; //5MB
        //boolean을 리턴하는 사전처리함수 (파일사이즈, 확장자검사해줌)
        function checkExtension(fileName, fileSize) {
            if (fileSize >= maxSize) {
                alert("파일 사이즈 초과");
                return false;
            }
            if (regex.test(fileName)) {
                alert("해당 종류의 파일은 업로드할 수 없습니다.");
                return false;
            }
            return true;
        }
         $("#uploadBtn").on("click", function(e){
                var formData = new FormData();
                var inputFile = $("input[name='uploadFile']");
                var files = inputFile[0].files;
                console.log(files);

               //add filedata to formdata
                for(var i = 0; i < files.length; i++){
                    //사전검사
                    if (!checkExtension(files[i].name, files[i].size)) {
                       return false;
                   }
                    //검사통과하면 formData갖추기
                    formData.append("uploadFile", files[i]);
                }

                $.ajax({
```

```
                //받아올 데이터는 아직없음(dataType이 공란), 추후 섬네일생성후 섬네일정
보를 받아올예정
                        url: '/uploadAjaxAction',
                        processData: false,
                        contentType: false,
                        data: formData,
                        type: 'POST',
                        success: function(result){
                        alert("Uploaded");
                        }
                    }); //$.ajax
            });
        });
    </script>
```

서버에서 처리

```
<Form>방식과 거의일
@PostMapping("/uploadFormAction")
    public void uploadFormPost(MultipartFile[] uploadFile, Model model) {
        String uploadFolder = "C:\\upload";
        for (MultipartFile multipartFile : uploadFile) {
            log.info("-------------------------------------");
            log.info("Upload File Name: " +
multipartFile.getOriginalFilename());
            log.info("Upload File Size: " + multipartFile.getSize());
            String uploadFileName = multipartFile.getOriginalFilename();//IE읽 경
우 전체경로를 리턴해옴.            // IE 일 경우 subString작업을 추가로 요함
          // uploadFileName =
uploadFileName.substring(uploadFileName.lastIndexOf("\\") + 1);
            File saveFile = new File(uploadFolder, uploadFileName);
            try {
                multipartFile.transferTo(saveFile);//업로드 진행
            } catch (Exception e) {
                log.error(e.getMessage());
            } // end catch
        } // end for
    }
```

22장 파일 업로드 상세 처리

22.1 파일의 확장자나 크기의 사전 처리

방법1 : 날짜폴더

날짜로 폴더를 추가 생성하기 때문에, uploadFolder에서 uploadPath로 날짜폴더가 함께 있는 경로로 업로드경로설정

```
<Form>방식과 거의일
@PostMapping("/uploadFormAction")
    public void uploadFormPost(MultipartFile[] uploadFile, Model model) {
```

```
        String uploadFolder = "C:\\upload";

        // // make folder --------
        File uploadPath = new File(uploadFolder, getFolder());//uploadPath
         log.info("upload path: " + uploadPath);

         if (uploadPath.exists() == false) {
             // make "yyyy/MM/dd" folder
             uploadPath.mkdirs();
         }

        for (MultipartFile multipartFile : uploadFile) {
            log.info("-------------------------------------");
            log.info("Upload File Name: " +
multipartFile.getOriginalFilename());
            log.info("Upload File Size: " + multipartFile.getSize());
            String uploadFileName = multipartFile.getOriginalFilename();//IE읽 경
우 전체경로를 리턴해옴.
            // IE 일 경우 subString작업을 추가로 요함
           // uploadFileName =
uploadFileName.substring(uploadFileName.lastIndexOf("\\") + 1);
            //교체 File saveFile = new File(uploadFolder, uploadFileName);
             File saveFile = new File(uploadPath, uploadFileName);
            try {
                multipartFile.transferTo(saveFile);//업로드 진행
            } catch (Exception e) {
                log.error(e.getMessage());
            } // end catch
        } // end for
    }

    private String getFolder() {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        Date date = new Date();
        String str = sdf.format(date);
        return str.replace("-", File.separator);//  yyyy/MM/dd로 변환해줌
    }
```

방법2 : 날짜폴더 + UUID

```
//UUID 코드 추가
            UUID uuid = UUID.randomUUID();
           uploadFileName = uuid.toString() + "_" + uploadFileName;
```

```
    @PostMapping("/uploadAjaxAction")
    public void uploadAjaxPost(MultipartFile[] uploadFile) {
        String uploadFolder = "C:\\upload";
        // make folder --------
        File uploadPath = new File(uploadFolder, getFolder());
        log.info("upload path: " + uploadPath);
        if (uploadPath.exists() == false) {
            uploadPath.mkdirs();
        }
        // make yyyy/MM/dd folder
```

```
        for (MultipartFile multipartFile : uploadFile) {
            log.info("-------------------------------------");
            log.info("Upload File Name: " +
multipartFile.getOriginalFilename());
            log.info("Upload File Size: " + multipartFile.getSize());
            String uploadFileName = multipartFile.getOriginalFilename();
            // IE has file path
            uploadFileName =
uploadFileName.substring(uploadFileName.lastIndexOf("\\") + 1);
            log.info("only file name: " + uploadFileName);

            //UUID 코드 추가
            UUID uuid = UUID.randomUUID();
           uploadFileName = uuid.toString() + "_" + uploadFileName;

            File saveFile = new File(uploadPath, uploadFileName);
            try {
                multipartFile.transferTo(saveFile);
            } catch (Exception e) {
                log.error(e.getMessage());
            } // end catch

        } // end for

    }
```

##

22.2 '섬네일 이미지 생성' 및 '업로드된 파일의 데이터 반환'

UUID적용된 원본을 이용

```
<!-- https://mvnrepository.com/artifact/net.coobird/thumbnailator -->
<dependency>
    <groupId>net.coobird</groupId>
    <artifactId>thumbnailator</artifactId>
    <version>0.4.8</version>
</dependency>
```

클라이언트단

```
$.ajax({
             url: '/uploadAjaxAction',
             processData: false,
             contentType: false,
             data: formData,
             type: 'POST',
             success: function(result){
             alert("Uploaded");
             }
});
```

서버단 코드

아래 Controller에서 섬네일이미지의 정보를 반환할 VO를 설계

```
package org.zerock.domain;
import lombok.Data;
@Data
public class AttachFileDTO {
    private String fileName;
    private String uploadPath;
    private String uuid;
    private boolean image;
}
```

```
@PostMapping(value = "/uploadAjaxAction", produces =
MediaType.APPLICATION_JSON_UTF8_VALUE)
    @ResponseBody
    public ResponseEntity<List<AttachFileDTO>> uploadAjaxPost(MultipartFile[]
uploadFile) {
        List<AttachFileDTO> list = new ArrayList<>();
        String uploadFolder = "C:\\upload";
        String uploadFolderPath = getFolder();
        // make folder --------
        File uploadPath = new File(uploadFolder, uploadFolderPath);
        if (uploadPath.exists() == false) {
            uploadPath.mkdirs();
        }
        // make yyyy/MM/dd folder
        for (MultipartFile multipartFile : uploadFile) {
            AttachFileDTO attachDTO = new AttachFileDTO();
            String uploadFileName = multipartFile.getOriginalFilename();
            // IE has file path
            uploadFileName =
uploadFileName.substring(uploadFileName.lastIndexOf("\\") + 1);
            log.info("only file name: " + uploadFileName);
            attachDTO.setFileName(uploadFileName);

             // UUID 적용
            UUID uuid = UUID.randomUUID();
            uploadFileName = uuid.toString() + "_" + uploadFileName;
            try {
                File saveFile = new File(uploadPath, uploadFileName);
                //파일 업로드
                 multipartFile.transferTo(saveFile);
                attachDTO.setUuid(uuid.toString());
                attachDTO.setUploadPath(uploadFolderPath);

                // check image type file
                if (checkImageType(saveFile)) { //checkImageType 메소드는 아래에
있음
                    attachDTO.setImage(true);
                    FileOutputStream thumbnail
                        = new FileOutputStream(new File(uploadPath, "s_" +
uploadFileName));
                  //섬네일 생성 (UUID적용된 원본을 이용)
```

```
                    Thumbnailator.createThumbnail(multipartFile.getInputStream(),
thumbnail, 100, 100);
                    thumbnail.close();
                }

                // add to List
                list.add(attachDTO);
            } catch (Exception e) {
                e.printStackTrace();
            }
        } // end for
        return new ResponseEntity<>(list, HttpStatus.OK);
    }

    private boolean checkImageType(File file) {
        try {
            String contentType = Files.probeContentType(file.toPath());
            return contentType.startsWith("image");
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return false;
    }
```

23장 브라우저에서 섬네일 처리

23.1 < input type='file' >의 초기화

type=file인 input 태그는 readOnly라 따로 수정이 불가능함. 따라서 clone을 이용

var cloneObj = $(".uploadDiv").clone();

```
var cloneObj = $(".uploadDiv").clone();
$("#uploadBtn").on("click", function(e) {
            var formData = new FormData();

            ......

            $.ajax({
                url : '/uploadAjaxAction',
                processData : false,
                contentType : false,
                data : formData,
                type : 'POST',
                dataType : 'json',
                success : function(result) {
                    console.log(result);

                    $(".uploadDiv").html(cloneObj.html());

                }
```

```
            }); //$.ajax
}
```

23.2 업로드된 이미지 처리

업로드된 결과를 JSON으로 받아왔기 때문에 이를 이용해서 화면에 적절한 섬세닝을 보여주는 것으로 피드백을 줄 수 있다.

1. 파일이름 출력

controller로 부터 JSON데이터를 결과로써 받는다.

`controller`

```
    @GetMapping("/display")
    @ResponseBody
    public ResponseEntity<byte[]> getFile(String fileName) {
        log.info("fileName: " + fileName);
        File file = new File("c:\\upload\\" + fileName);
        log.info("file: " + file);
        ResponseEntity<byte[]> result = null;

        try {
            HttpHeaders header = new HttpHeaders();
            //probeContentType()을 이용해서 확장자에 따른 MIME 타입을 지정해서 브라우
저로 알려줄 준비를 한다
            header.add("Content-Type", Files.probeContentType(file.toPath()));
            result = new ResponseEntity<>(FileCopyUtils.copyToByteArray(file),
header,HttpStatus.OK);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return result;
    }
```

`uploadAjax.jsp`

```
<div class='uploadDiv'>
        <input type='file' name='uploadFile' multiple>
    </div>

    <div class='uploadResult'>
        <ul>

        </ul>
    </div>
```

Ajax처리의 success 에서 이뤄질 작업을 showUploaderFile 함수에 정의해 둔다.

이때, 받아온 JSON데이터안의 필드를 통해 img태그의 src url을 획득하고, 요청을 다시 보낼 준비를 한다.

`uploadAjax.jsp`

```
<li><img src='/display?fileName="+fileCallPath+"'><li>
```

`uploadAjax.jsp`

ul 태그 선택

```
var uploadResult = $(".uploadResult ul");
function showUploadedFile(uploadResultArr) {
            var str = "";
            $(uploadResultArr).each(
                    function(i, obj) {
                        if (!obj.image) {
                          //이미지가 아닌 일반파일이라면  정적리소스인 default 이미지
파일을 출력한다.
                            str += "<li><img src='/resources/img/attach.png'>"
                                    + obj.fileName + "</li>";
                        } else {
                           //이미지 첨부파일
                         //파일이름//str += "<li>" + obj.fileName + "</li>";


                            var fileCallPath
                            = encodeURIComponent( obj.uploadPath+
"/s_"+obj.uuid+"_"+obj.fileName);
                            str += "<li><img src='/display?
fileName="+fileCallPath+"'><li>";//URL에 이미지파일 이름의 공백문자나 한글이름등이 문
제가 될 수 있다 URL호출에 적합한 문자열로 인코딩해야함.
                        }
                    });
            uploadResult.append(str);
        }
```

Ajax의 success에서 showUploadedFile을 호출한다.

`uploadAjax.jsp`

```
                success : function(result) {
                    console.log(result);

                      //호출
                    showUploadedFile(result);
                    $(".uploadDiv").html(cloneObj.html());
                }
```

//업로드 끝

24장 첨부파일의 다운로드 혹은 원본 보여주기

실습의 편의를 위해 C:/upload/ 폴더에 여러 영문파일을 다운로드할 예정.

24.1 첨부파일의 다운로드

```
@GetMapping(value = "/download", produces =
MediaType.APPLICATION_OCTET_STREAM_VALUE)
    @ResponseBody
    public ResponseEntity<Resource> downloadFile(@RequestHeader("User-Agent")
String userAgent,             String fileName) {
        //리턴타입으로 byte[]를 사용할 수 있으나, 로직이 복잡하다.
        //MIME 타입은 다운로드를 할 수 있는 application/octet-stream 으로 지정한다.
        Resource resource = new FileSystemResource("c:\\upload\\" + fileName);
        if (resource.exists() == false) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        String resourceName = resource.getFilename();
        // UUID제거후 원본파일이름
        String resourceOriginalName =
resourceName.substring(resourceName.indexOf("_") + 1);
        HttpHeaders headers = new HttpHeaders();
        try {
            boolean checkIE = (userAgent.indexOf("MSIE") > -1 ||
userAgent.indexOf("Trident") > -1);
            String downloadName = null;
            // URL 인코딩 작업 ???
            if (checkIE) {
               downloadName = URLEncoder.encode(resourceOriginalName,
"UTF8").replaceAll("\\+", " ");
            } else {
              downloadName = new String(resourceOriginalName.getBytes("UTF-8"),
"ISO-8859-1");
            }
            //Content-Disposion 덕분에 한글이름이 깨지는 것을 방지할 수 있음.
            headers.add("Content-Disposition", "attachment; filename=" +
downloadName);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return new ResponseEntity<Resource>(resource, headers, HttpStatus.OK);
    }
```

24.2 원본 이미지 보여주기

원본이미지를 보여주는 자바스크립트 함수 showImage는 $(document).ready() 바깥쪽에 정의한다.

이유: DOM에서(태그에서 자바스크립트 함수를 직접호출 할 수 있게끔)

- uploadPath : 2020\01\28 등으로 역슬래시임.

```
function showImage(fileCallPath){
    //alert(fileCallPath);
    $(".bigPictureWrapper").css("display","flex").show();
    $(".bigPicture")
      .html("<img src='/display?fileName="+encodeURI(ileCallPath)+"'>")
      .animate({width:'100%', height: '100%'}, 1000);
    }
    $(".bigPictureWrapper").on("click", function(e){
```

```
      $(".bigPicture").animate({width:'0%', height: '0%'}, 1000);
      setTimeout(() => {
        $(this).hide();
    }, 1000);
});

 function showUploadedFile(uploadResultArr){
   var str = "";
   $(uploadResultArr).each(function(i, obj){
     if(!obj.image){
         //쌈네일 이미지를 <a>태그로 둘러, 클릭시 일반파일 다운로드가 진행되도록
         //링크url은 파일이름이 한글일 경우 문제가 발생함. 따라서, 인코딩을 진행한다.
       var fileCallPath =  encodeURIComponent( obj.uploadPath+"\\"+ obj.uuid
+"_"+obj.fileName);
       var fileLink = fileCallPath.replace(new RegExp(/\\/g),"/");

         str += "<li><a href='/download?fileName="+fileLink+"'><img
src='/resources/img/attach.png'>"+obj.fileName+"</a>"+
           "<span data-file=\'"+fileLink+"\' data-type='file'> x </span>"+
           "<div></li>"
     }else{
       //썸네일 path (<img>에 사용됨)  ??아래는 역슬래시처리해주고 여기는 왜 안해줘??
       var fileCallPath =  encodeURIComponent( obj.uploadPath+ "\\s_"+obj.uuid
+"_"+obj.fileName);
       //원본(uuid적용된)path  (<a>에 사용됨, download진행)
       var originPath = obj.uploadPath+ "\\"+obj.uuid +"_"+obj.fileName;
       originPath = originPath.replace(new RegExp(/\\/g),"/");
         //썸내일 이미지를 <a>태그로 둘러, 클릭시 이미지파일 원본이 다운로드가 진행되도록
       str += "<li><a href=\"javascript:showImage(\'"+originPath+"\')\"><img
src='/display?fileName="+fileCallPath+"'></a><li>";
     }
   });

   uploadResult.append(str);
 }
```

24.3 첨부파일 삭제

```
// X표시를 눌렀을 때. 이벤트발생
// 이벤트 위임방식 , <span>태그는 파일업로드가 진행된 이후에 생성될 태그이기 때문에
// uploadResult에 이벤트가 걸리는게 아니고 span에 걸림
// 아래 this 는 span을 뜻함
$(".uploadResult").on("click","span", function(e){
      var targetFile = $(this).data("file");
      var type = $(this).data("type");
      console.log(targetFile);
      $.ajax({
        url: '/deleteFile',
        data: {fileName: targetFile, type:type},
        dataType:'text',
        type: 'POST',
          success: function(result){
             alert(result);
          }
      }); //$.ajax
```

```
    });

// 썸네일 옆에 X 표시 생성
function showUploadedFile(uploadResultArr){
   var str = "";
   $(uploadResultArr).each(function(i, obj){
     if(!obj.image){
       var fileCallPath =  encodeURIComponent( obj.uploadPath+"/"+ obj.uuid
+"_"+obj.fileName);
       var fileLink = fileCallPath.replace(new RegExp(/\\/g),"/");
       str += "<li><div><a href='/download?fileName="+fileCallPath+"'>"+
           "<img src='/resources/img/attach.png'>"+obj.fileName+"</a>"+
           "<span data-file=\'"+fileCallPath+"\' data-type='file'> x </span>"+
           "<div></li>"
     }else{
       var fileCallPath =  encodeURIComponent( obj.uploadPath+ "/s_"+obj.uuid
+"_"+obj.fileName);
       var originPath = obj.uploadPath+ "\\"+obj.uuid +"_"+obj.fileName;
       originPath = originPath.replace(new RegExp(/\\/g),"/");
       str += "<li><a href=\"javascript:showImage(\'"+originPath+"\')\">"+
              "<img src='display?fileName="+fileCallPath+"'></a>"+
              "<span data-file=\'"+fileCallPath+"\' data-type='image'> x
</span>"+
              "<li>";
     }
   });
   uploadResult.append(str);
 }
```

p.549 코드 다시 보기

25장 프로젝트의 첨부파일 - 등록

25.1 첨부파일 정보를 위한 준비

BoardAttachVO 설계하기

- uuid -pk
- uploadPath
- fileName
- fileType
- **bno (어떤 게시물 소속인지 게시물PK를 갖을 수 있도록 FK를 지정한다.)**

BoardVO 설계

- bno
- title
- content
- writer
- regdate
- updateDate
- replyCnt

- `List<BoardAttachVO>`

BoardAttachMapper

- insert
- delete
- **findByBno // bno를 통해 첨부파일에 대한 모든정보를 가져오는 함수**

25.2 등록을 위한 화면 처리(register.jsp)

`<input type = 'file' >` 에 파일을 첨부할 때, Javascript에서 change 이벤트로 감지한다

change이벤트 ---> Ajax 파일첨부 진행 ---> success에서 showUploadResult( ) 메소드 실행

showUplaodResult ( ) : 코드양이 많다. 모두 dom 조작을 위한 코드

25.3 BoardController, BoardService의 처리

- BoardService에서 Mapper를 실행할 때
  - Transaction 처리 중요, 두 번의 질의진행
    - boardMapper
    - attachMapper
  - 따라서, ServiceImpl에 @Transactional 를 명시한다.

26장 게시물의 조회와 첨부파일

26.1 BoardService와 BoardController 수정

- (서버측) 첨부파일목록을 가져올 메소드 정의
  - attachMapper.findByBno(bno);

26.2 BoardController의 변경과 화면 처리

- (클라이언트) Ajax로 요청 - get.jsp 에서 진행됨.

  $(document).ready()

```
$.getJSON("/board/getAttachList", {bno: bno}, function(arr){

    console.log(arr);

    var str = "";

    $(arr).each(function(i, attach){

      //image type
      if(attach.fileType){
```

```
            var fileCallPath =  encodeURIComponent( attach.uploadPath+
"/s_"+attach.uuid +"_"+attach.fileName);

            str += "<li data-path='"+attach.uploadPath+"' data-
uuid='"+attach.uuid+"' data-filename='"+attach.fileName+"' data-
type='"+attach.fileType+"' ><div>";
            str += "<img src='/display?fileName="+fileCallPath+"'>";
            str += "</div>";
            str +"</li>";
          }else{

            str += "<li data-path='"+attach.uploadPath+"' data-
uuid='"+attach.uuid+"' data-filename='"+attach.fileName+"' data-
type='"+attach.fileType+"' ><div>";
            str += "<span> "+ attach.fileName+"</span><br/>";
            str += "<img src='/resources/img/attach.png'></a>";
            str += "</div>";
            str +"</li>";
          }
        });

        $(".uploadResult ul").html(str);


      });//end getjson
```

get.jsp 다운로드 이벤트

```
$(".uploadResult").on("click","li", function(e){

    console.log("view image");

    var liObj = $(this);

    var path = encodeURIComponent(liObj.data("path")+"/" +
liObj.data("uuid")+"_" + liObj.data("filename"));

    if(liObj.data("type")){
      showImage(path.replace(new RegExp(/\\/g),"/"));
    }else {
      //download
      self.location ="/download?fileName="+path
    }


  });
```

27장 게시물의 삭제와 첨부파일

27.1 첨부파일 삭제 처리

```
1. DB 에서 첨부파일의 경로를 삭제
```

2. 첨부파일 삭제

섬네일 삭제 처리가 없다. 좀더 세분화한다면

1. 해당 게시물의 모든 첨부파일 정보준비
2. DB상에서 첨부파일의 경로 삭제
3. 첨부파일 원본 삭제
4. 썸네일 삭제

28장 게시물의 수정과 첨부파일

28.1 화면에서 첨부파일 수정

- 수정화면에서 X 표시를 누른다고해서 첨부파일 삭제를 진행하면 안된다.
- 화면상에서만 그림에서 안보이게 만든다.
- 최종 수정이 이뤄졌을 때, 첨부파일 삭제를 진행한다.

28.2 서버측 게시물 수정과 첨부파일

- 해당 게시물과 관련된 모든 데이터를 삭제 후,
- 데이터 삽입

```
@Transactional
    @Override
    public boolean modify(BoardVO board) {

        log.info("modify......" + board);

        attachMapper.deleteAll(board.getBno());

        boolean modifyResult = mapper.update(board) == 1;

        if (modifyResult && board.getAttachList().size() > 0) {

            board.getAttachList().forEach(attach -> {

                attach.setBno(board.getBno());
                attachMapper.insert(attach);
            });
        }

        return modifyResult;
    }
```