

# 실전 프로젝트1





21

# **MyBatis & Spring interceptor**

# 공지

- 15주차 공지
  - 15주차 실습과제는 팀 프로젝트로 팀원과 협력하여 수행하고, 개인별로 보고서 제출!
  - 팀 프로젝트 보고서 제출 마감 : 12월 12일(일) 밤 12시
  - 반드시 개인별로 제출해야 함!
- 16주차 화요일
  - 비대면 수업으로 진행
  - 두번째 퀴즈 진행함!
- Hisnet 온라인 출석부 수시 체크
  - 수강과목 선택 > 강의정보 > 강의출석현황 메뉴에서 확인

# Database CRUD

- Java class를 사용한 JDBC 연결 (Basic API)

- Connection
- Statement
- ResultSet 클래스를 사용

```
public static Connection getConnection(){
    Connection con=null;
    try{
        Class.forName("com.mysql.cj.jdbc.Driver");
        con= DriverManager.getConnection("jdbc:mysql://[Server IP]/[DB Name]"
            ,"dbid","dbpwd");
    }catch(Exception e){
        System.out.println(e);
    }
    return con;
}
```

- Spring-jdbc library 사용

- xml 파일에 DB 연결정보 입력
- JdbcTemplate class 사용
- SQL 구문 변경 번거로움

```
<beans:bean id="ds" class="org.springframework.jdbc.datasource.DriverMan">
    <beans:property name="driverClassName" value="com.mysql.jdbc.Driver">
    <beans:property name="url" value="jdbc:mysql://DBServerIP/DBName"></beans:property>
    <beans:property name="username" value="DBId"></beans:property>
    <beans:property name="password" value="DBPwd"></beans:property>
</beans:bean>
<beans:bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
    <beans:property name="dataSource" ref="ds"></beans:property>
</beans:bean>
<beans:bean id="dao" class="com.mycompany.spring2.board.BoardDAO">
    <beans:property name="template" ref="jt"></beans:property>
</beans:bean>
```

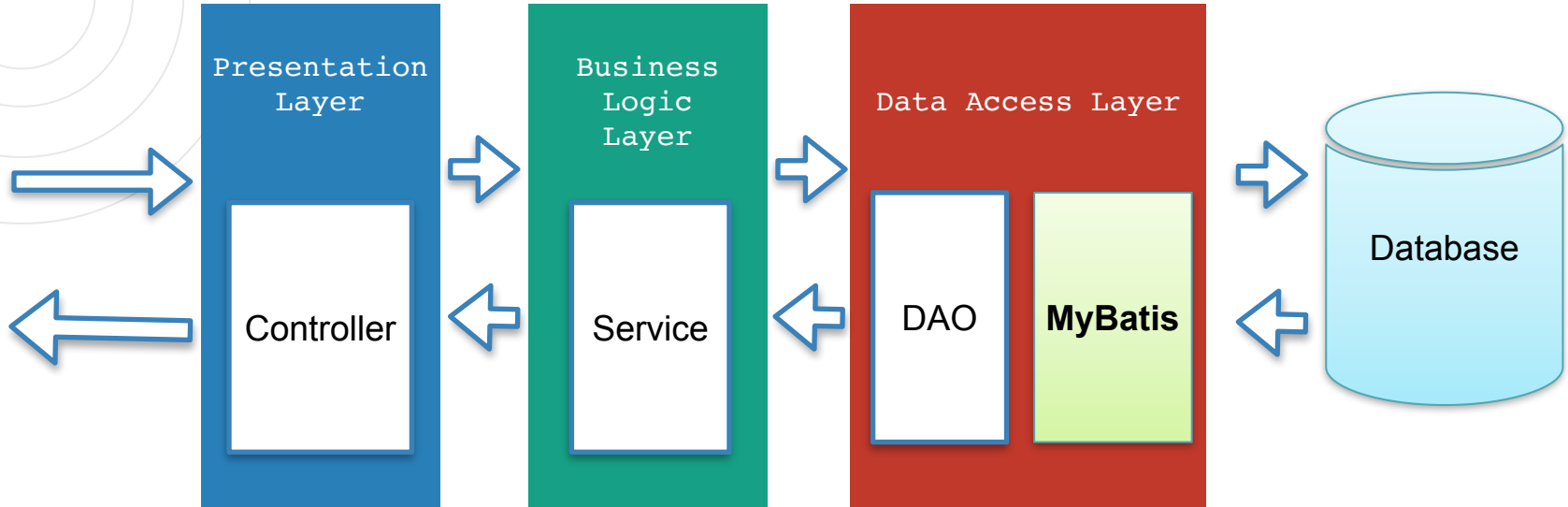
# MyBatis framework



# MyBatis

- RDB(Relational Database) 프로그래밍을 쉽게 해주는 Java Persistence framework
  - JDBC(Java Database Connectivity)를 편하게 사용할 수 있음
  - SQL문과 자바 객체 간의 Mapping 기능 제공
  - 간단한 코드로 DB 연동 처리 가능
  - SQL문은 자바코드에서 완전히 분리하여 XML 파일에 별도 관리
  - Spring과 연동하여 자동화 처리
  - 유지보수성 향상(동적 SQL 사용)
- 아파치 소프트웨어 재단에서 개발한 iBATIS
- 2010년에 구글코드로 이전하면서 MyBatis로 이름 변경

# Spring web project



# MyBatis-Spring 연동

- **SqlSessionFactoryBean** 클래스
  - 설정파일에 빈으로 등록하여 SqlSessionFactory 객체 생성
  - dataSource 를 사용하여 데이터베이스에 연결
  - MyBatis 설정파일 위치 설정
  - Mapper 파일 위치 설정
- **SqlSessionTemplate** 클래스
  - `public Object selectOne(String stmt, Object param)`
  - `public List selectList(String stmt, Object param)`
  - `public int insert(String stmt, Object param)`
  - `public int update(String stmt, Object param)`
  - `public int delete(String stmt, Object param)`

# MyBatis 구성

- **MyBatis configuration file(.xml) : MyBatis가 JDBC 사용을 위해 필요한 설정**
  - <typeAlias>
  - <environment> : DB 접속에 대한 설정
  - <mapper> : Mapper 파일 위치 설정
- **Mapper files(.xml) : SQL문 관련 설정**
  - SQL문, parameter, result, resultMap(ResultSet)
  - Insert, delete, update, select
  - <sql> 태그 사용
- **?? DAO에서 mapper에 있는 SQL문 실행**



# MyBatis 설정파일(xml)

- Connection pool 제공
- 다수 데이터베이스 연결 정보 설정 및 선택 가능
- VO(Value Object)에 alias 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <!-- Properties 파일 설정 -->
  <properties resource="db.properties" />
  <!-- Alias 설정 -->
  <typeAliases>
    <typeAlias alias="board" type="com.mycom.mapp.BoardVO" />
  </typeAliases>
  <!-- DataSource 설정 -->
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC" />
      <dataSource type="POOLED">
        <property name="driver" value="${jdbc.driverClassName}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
      </dataSource>
    </environment>
  </environments>
  <!-- Sql Mapper 설정 -->
  <mappers>
    <mapper resource="mappings/board-mapping.xml" />
  </mappers>
</configuration>
```

# MyBatis - Mapper 구조(xml)

```
<?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="BoardDAO">
  <insert id="insertBoard">
    insert into BOARD (title, writer, content)
    values (#{title} , #{writer} , #{content})
  </insert>
  <update id="updateBoard">
    update BOARD
    set title=#{title}, content=#{content} where seq=#{seq}
  </update>
  <delete id="deleteBoard">
    delete from BOARD where seq=#{seq}
  </delete>
  <select id="getBoard" resultType="board">
    select * from BOARD where seq=#{seq}
  </select>
  <select id="getBoardList" resultType="board">
    select * from BOARD order by seq desc
  </select>
</mapper>
```

# MyBatis – Mapper 사용

mapper(xml)

```
<?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="BoardDAO">
  <insert id="insertBoard">
    insert into BOARD (title, writer, content)
    values (#{title} , #{writer} , #{content})
  </insert>
  ...
</mapper>
```

DAO class

```
public void insertBoard(BoardVO vo) {
    System.out.println("insertBoard");
    mybatis.insert("BoardDAO.insertBoard", vo);
}
```

# Mapper xml : parameter, result type

- 여러 데이터 사용을 위해 VO 객체 사용
- 자바 클래스 전체 경로 포함
- (패키지+)클래스 이름이 긴 경우 <typeAlias>를 이용하여 별칭으로 사용

```
<!-- Alias 설정 -->
<typeAliases>
    <typeAlias alias="board" type="com.mycom.mapp.BoardVO" />
</typeAliases>
```

mybatis-config.xml

```
<select id="getBoard" resultType="board">
    select * from BOARD where seq=#{seq}
</select>
```

board-mapper.xml

# Mapper xml : CDATA Section

- SQL 문에 특수기호 <, > 등의 기호를 사용할 때 xml에 의해서 연산자로 처리되는 것을 막기 위함
- CDATA Section 내의 구분은 XML parser가 해석하지 않음

```
<select id="getBoard" resultType="board">  
    select * from BOARD where seq < #{seq}  
</select>
```

board-mapper.xml

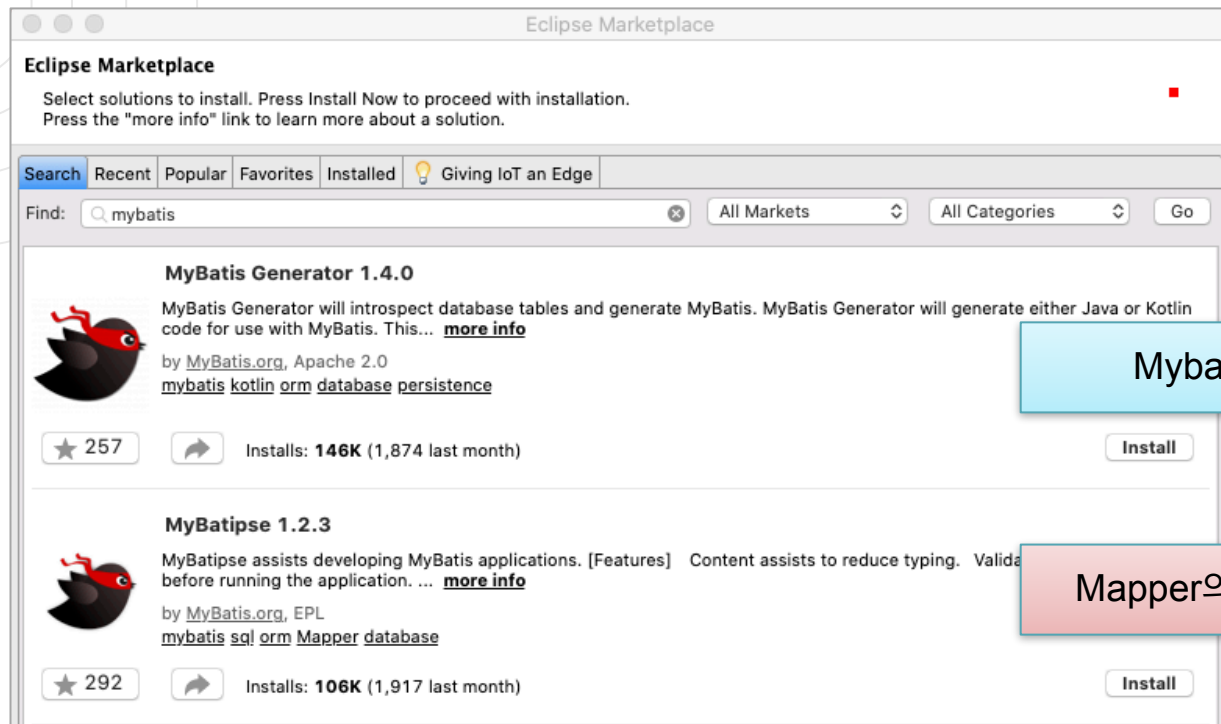
```
<select id="getBoard" resultType="board">  
<![CDATA[  
    SELECT * FROM BOARD  
    WHERE SEQ < #{seq}  
]]>  
</select>
```

# Spring CRUD Project(MyBatis)

1. 새 Spring MVC 프로젝트 생성
2. web.xml : UTF-8 encoding filter 추가
3. Library dependency 추가(MySQL Connector , Spring-jdbc, spring-test, MyBatis, MyBatis-Spring)
4. root-context.xml 파일에 data source, SqlSessionFactory, SqlSession bean 등록
5. mybatis-config.xml 파일 생성(MyBatis configuration file, XML)
6. board-mapper.xml 파일 생성(MyBatis mapper file, XML)
7. BoardVO class 생성(getters & setters)
8. BoardDAO class 생성(CRUD methods)
9. BoardService interface 생성
10. BoardServiceImpl class 생성
11. BoardController class 생성
12. JSP Pages 작성

# MyBatis plugin 설치

- sts4 : help > Eclipse Marketplace 에서 mybatis 검색하여 설치




Mybatis 설정파일 생성

Mapper의 XML 자동완성 지원

# MyBatis Library 추가

- junit
- Spring-jdbc
- Mysql-connector
- mybatis, mybatis-spring

Home » org.mybatis » mybatis » 3.5.3

 **MyBatis » 3.5.3**

The MyBatis SQL mapper framework makes it easier to use a relational database with object-oriented applications. MyBatis couples objects with stored procedures or SQL statements using a XML descriptor or annotations. Simplicity is the biggest advantage of the MyBatis data mapper over object relational mapping tools.

License	Apache 2.0
Categories	Object/Relational Mapping
HomePage	<a href="http://www.mybatis.org/mybatis-3">http://www.mybatis.org/mybatis-3</a>
Date	(Oct 20, 2019)
Files	<a href="#">jar (1.6 MB)</a> <a href="#">View All</a>
Repositories	Central
Used By	935 artifacts

```
<!-- Test -->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.7</version>
  <scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.20</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>${org.springframework-version}</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.5.6</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>2.0.6</version>
</dependency>
```



# Bean 생성: Data Source

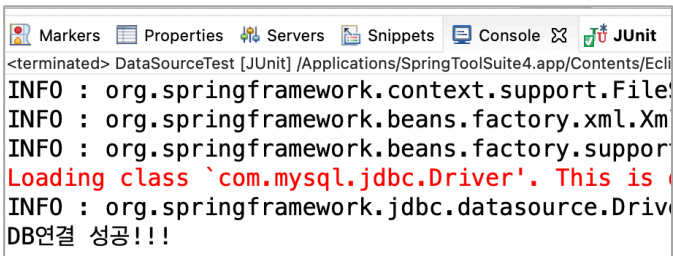
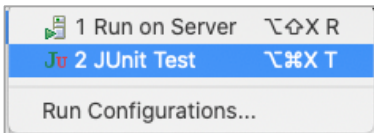
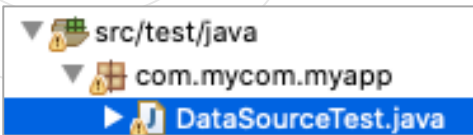
- src/main/webapp/WEB-INF/spring/root-context.xml
- 데이터베이스 연결 정보 설정

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

    <!-- Root Context: defines shared resources visible to all other web components -->
    <bean name="dataSource"
          class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/dbname" />
        <property name="username" value="userid" />
        <property name="password" value="pwd" />
    </bean>
</beans>
```

# Spring에서 JUnit 단위 테스트

- 자바 프로그래밍 유닛 테스트 프레임워크
- Data Source 단위테스트



```
import java.sql.Connection;
import java.sql.SQLException;
import javax.sql.DataSource;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class DataSourceTest {

    @Test
    public void testConnection() {
        ApplicationContext ctx =
            new FileSystemXmlApplicationContext(
                "file:src/main/webapp/WEB-INF/spring/root-context.xml");
        DataSource ds = (DataSource) ctx.getBean("dataSource");

        try {
            Connection con = ds.getConnection();
            System.out.println("DB연결 성공!!!");
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            System.out.println("DB연결 실패!!!");
        }
    }
}
```

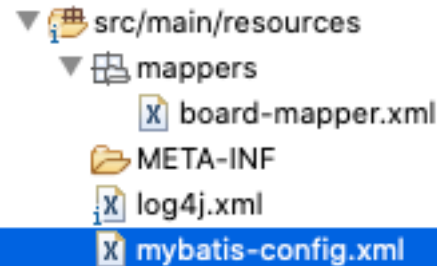
# MyBatis 연결

- src/main/webapp/WEB-INF/spring/root-context.xml
- SqlSessionFactory 객체 설정
  - Datasource, MyBatis 설정 파일, Mapper 파일 위치 설정

```
...  
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  
    <property name="dataSource" ref="dataSource"></property>  
    <property name="configLocation" value="classpath:mybatis-config.xml" />  
    <property name="mapperLocations" value="classpath:mappers/*-mapper.xml"></property>  
</bean>  
  
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate" destroy-method="clearCache">  
    <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory" />  
</bean>
```

# MyBatis configuration file

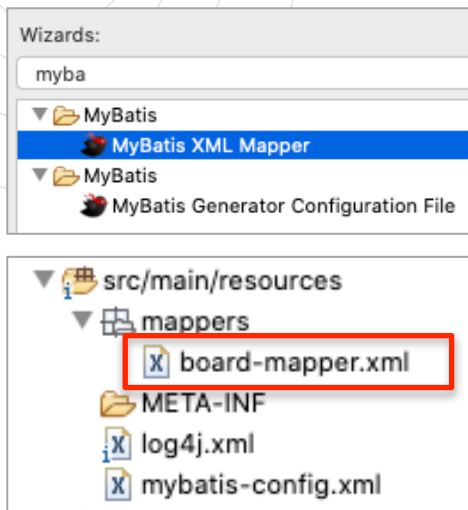
- src/main/resources 에 mybatis 설정파일 추가
- mybatis-config.xml



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <typeAliases>
    <typeAlias alias="board" type="com.mycom.myapp.board.BoardVO" />
  </typeAliases>
</configuration>
```

# Mapper xml 작성

- CRUD SQL작성



```
<?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="Board">
    <insert id="insertBoard">
        insert into BOARD (category, title, writer, content)
        values
        (#{category} , #{title} , #{writer} , #{content})
    </insert>
    <update id="updateBoard">
        update BOARD
        set title=#{title}, content=#{content}, writer=#{writer}, category=#{category}
        where seq=#{seq}
    </update>
    <delete id="deleteBoard">
        delete from BOARD where seq=#{seq}
    </delete>
    <select id="getBoard" resultType="board">
        select * from BOARD where seq=#{seq}
    </select>
    <select id="getBoardList" resultType="board">
        select * from BOARD order by seq desc
    </select>
</mapper>
```

# BoardDAO class

- BoardVO class 작성 후 BoardDAO 작성
- class명 위에 **@Repository** 추가 - DB나 파일과 같은 외부 I/O 작업 처리
  - Persistence layer
- SqlSession 멤버변수 선언 및 **@Autowired** 추가(DI)
  - root-context.xml 에 bean으로 등록
- 모든 SQL문 삭제

```
@Repository  
public class BoardDAO {
```

```
    @Autowired  
    SqlSession sqlSession;
```

```
    public int insertBoard(BoardVO vo) {  
        int result = sqlSession.insert("Board.insertBoard", vo);  
        return result;  
    }
```

```
<mapper namespace="Board">  
    <insert id="insertBoard">  
        insert into BOARD (category, title, writer, content)  
        values  
        (#{category} , #{title} , #{writer} , #{content})  
    </insert>
```

# BoardDAO class

```
public int insertBoard(BoardVO vo) {  
    int count = sqlSession.insert("Board.insertBoard", vo);  
    return count;  
}
```

```
public BoardVO getBoard(int seq) {  
    BoardVO one = sqlSession.selectOne("Board.getBoard", seq) ;  
    return one;  
}
```

```
public List<BoardVO> getBoardList(){  
    List<BoardVO> list = sqlSession.selectList("Board.getBoardList") ;  
    return list;  
}
```

# Library에 의한 BoardDAO class

```
Connection conn = null;
PreparedStatement stmt = null;
ResultSet rs = null;

private final String BOARD_INSERT = "insert into BOARD (title, writer, content, category) values (?, ?, ?, ?)";
private final String BOARD_UPDATE = "update BOARD set title=?, writer=?, content=?, category=? where id=?";
private final String BOARD_DELETE = "delete from BOARD where id=?";
private final String BOARD_GET = "select * from BOARD where id=?";
private final String BOARD_LIST = "select * from BOARD order by id desc";

public int insertBoard(BoardVO vo) {
    System.out.println("==> JDBC로 insertBoard() 가능 처리");
    try {
        conn = JDBCUtil.getConnection();
        stmt = conn.prepareStatement(BOARD_INSERT);
        stmt.setString(1, vo.getTitle());
        stmt.setString(2, vo.getWriter());
        stmt.setString(3, vo.getContent());
        stmt.setString(4, vo.getCategory());
        stmt.executeUpdate();
        return 1;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return 0;
}
```



Spring JDBC

```
@Repository
public class BoardDAO {

    @Autowired
    JdbcTemplate template;

    public int insertBoard(BoardVO vo) {
        String sql = "insert into BOARD (title, writer, content, category) values (?, ?, ?, ?)";
        return template.update(sql);
    }
}
```



MyBatis Spring

```
@Repository
public class BoardDAO {

    @Autowired
    SqlSession sqlSession;

    public int insertBoard(BoardVO vo) {
        int result = sqlSession.update("Board.insertBoard", vo);
        return result;
    }
}
```



# Spring CRUD Project(MyBatis)

- BoardService interface 생성
- BoardServiceImpl class 생성

```
public interface BoardService {  
    public int insertBoard(BoardVO vo);  
    public int deleteBoard(int id);  
    public int updateBoard(BoardVO vo);  
    public BoardVO getBoard(int seq);  
    public List<BoardVO> getBoardList();  
}
```

```
@Service  
public class BoardServiceImpl implements BoardService{  
  
    @Autowired  
    BoardDAO boardDAO;  
  
    @Override  
    public int insertBoard(BoardVO vo) {  
        return boardDAO.insertBoard(vo);  
    }  
    @Override  
    public BoardVO getBoard(int seq) {  
        return boardDAO.getBoard(seq);  
    }  
    @Override  
    public List<BoardVO> getBoardList(){  
        return boardDAO.getBoardList();  
    }  
}
```

# Spring CRUD Project(MyBatis)

- BoardController class 생성
- JSP Pages 작성(이전과 동일)
- Tomcat을 이용한 local 테스트
- 실제 서버에 웹 애플리케이션 서비스(Deploy)



Id	Category	Title	Writer	Content	Regdate	Edit	Delete
19	c1	T1	W1	Content1	Thu Nov 26 23:45:30 KST 2020	<a href="#">Edit</a>	<a href="#">Delete</a>
18	C2카테고리	T2	W2	Content2	Thu Nov 26 23:45:10 KST 2020	<a href="#">Edit</a>	<a href="#">Delete</a>
17	과일--	사과팔아요--	101호-	경북 부사 사과 팔니다!	Sat Nov 21 15:53:04 KST 2020	<a href="#">Edit</a>	<a href="#">Delete</a>
14	카테고리	제목	작성자	내용	Tue Nov 10 19:37:07 KST 2020	<a href="#">Edit</a>	<a href="#">Delete</a>

```
@Controller
public class BoardController {

    @Autowired
    BoardService boardService;

    @RequestMapping(value = "/board/list",
method = RequestMethod.GET)
    public String boardlist(Model model) {
        model.addAttribute("list",
boardService.getBoardList());
        return "posts";
    }

    @RequestMapping(value = "/board/add",
method = RequestMethod.GET)
    public String addPost() {
        return "addpostform";
    }

    ...
}
```

# 게시판에 로그인 체크 기능 추가

- 게시판은 인증된 회원만 사용하도록 기능 추가
- 게시판 URL에 접근했을 때, 로그인 되지 않은 경우 로그인 페이지로 이동



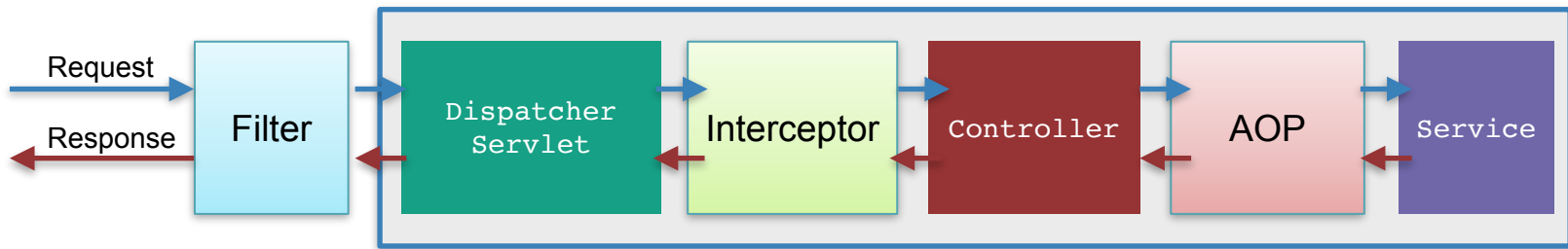
## 자유게시판

관리자 님 안녕하세요~ [logout](#)

Id	Category	Title	Writer	Content	Regdate	Edit	Delete
19	c1	T1	W1	Content1	Thu Nov 26 23:45:30 KST 2020	<a href="#">Edit</a>	<a href="#">Delete</a>
18	C2카테고리	T2	W2	Content2	Thu Nov 26 23:45:10 KST 2020	<a href="#">Edit</a>	<a href="#">Delete</a>
17	과일--	사과팔아요--	101호-	경북 부사 사과입니다!~	Sat Nov 21 15:53:04 KST 2020	<a href="#">Edit</a>	<a href="#">Delete</a>

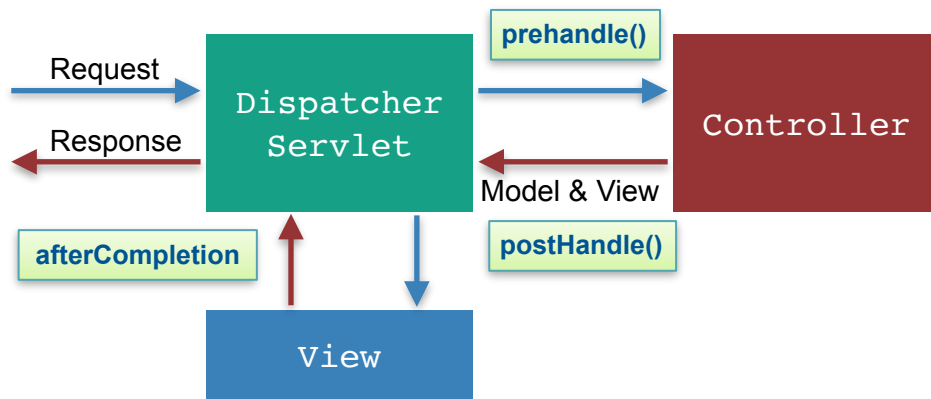
# Filter, Interceptor, AOP

- 프로그램 내에서 자주 사용되는 공통 기능을 따로 구현하여 처리하는 방법
- Filter : DispatcherServlet이 실행되기 전과 후 수행되는 기능을 처리
  - Encoding, XSS방어
- Interceptor : DispatcherServlet이 Controller를 호출하기 전과 후 수행되는 기능을 처리
  - http 프로토콜에 존재하는 정보를 활용가능(로그인 여부 확인)
- AOP(Aspect Object Programming)
  - Controller 처리 이후 비즈니스 로직에서 실행
  - 로깅, 트랜잭션, 에러 처리



# HandlerInterceptor

- 특정 URI 호출을 가로채는 역할
- 기존 컨트롤러 변경없이 사전이나 사후 제어가 가능
- 여러 URL에 적용하는 기능을 구현할 때 사용
- 로그인 여부 체크



# Interceptor 생성

- HandlerInterceptorAdapter를 상속받는 Interceptor 생성
  - `boolean preHandle(request, response, handler)`
    - 전처리기, 클라이언트에서 요청 후 Controller 호출 전에 실행
  - `void postHandle(request, response, handler)`
    - 후처리기, Controller 호출 후 실행됨
  - `void afterCompletion(request, response, handler, modelAndView)`
    - Controller 처리 및 화면처리 후 실행

# 로그인 체크 추가 실습

- Spring framework 4.2.4로 버전 변경(pom.xml)
- DB에 테이블 생성 및 계정 추가
  - `insert into member (userid, username, password) values ('admin', '관리자', '1234')`
- Mybatis Mapper XML file 생성
- UserVO class 생성
- UserDao class 생성
- UserServiceImpl class 생성 (UserService interface는 만들지 않음)
- LoginController class 생성
- login.jsp 페이지 생성
- LoginInterceptor class 생성
- Interceptor 설정(xml file) - 로그인 체크 페이지와 그렇지 않은 페이지로 구분됨



각 단계별 개발 후 반드시 실행하여 에러체크

# 로그인 체크 실습

- Spring framework 4.2.4로 버전 변경(pom.xml)
- DB에 테이블 생성 및 계정 추가
  - insert into member (userid, username, password) values ('admin', '관리자', '1234')

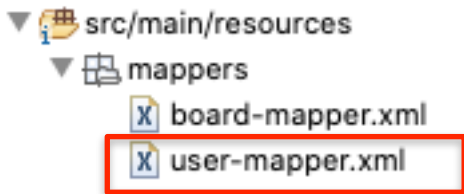
```
<properties>  
  <java-version>1.6</java-version>  
  <org.springframework-version>4.2.4.RELEASE</org.springframework-version>  
  <org.aspectj-version>1.6.10</org.aspectj-version>  
  <org.slf4j-version>1.6.6</org.slf4j-version>  
</properties>
```





# 로그인 체크 실습

- Mybatis Mapper XML file 생성
- 사용자가 입력한 id,pwd가 존재하는지 체크하는 SQL문 작성



```
<?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="User">

    <select id="getUser" resultType="user">
        select userid, username from member
        where userid=#{userid} and password=#{password}
    </select>

</mapper>
```

# 로그인 체크 실습

- UserVO class생성

```
public class UserVO {  
    private String userid;  
    private String password;  
    private String username;  
    public String getUserid() {  
        return userid;  
    }  
    public void setUserid(String userid) {  
        this.userid = userid;  
    }  
    public String getPassword() {  
        return password;  
    }  
    public void setPassword(String password) {  
        this.password = password;  
    }  
    public String getUsername() {  
        return username;  
    }  
    public void setUsername(String username) {  
        this.username = username;  
    }  
}
```

# 로그인 체크 실습

- UserDao class 생성
  - mapper에 작성해 놓은 SQL문 선택
- UserServiceImpl class 생성

user-mapper.xml

```
<mapper namespace="User">
  <select id="getUser" resultType="user">
    select userid, username from member
    where userid=#{userid} and password=#{password}
  </select>
```

@Repository

```
public class UserDao {
    @Autowired
    SqlSessionTemplate sqlSession;
    public UserVO getUser(UserVO vo) {
        return sqlSession.selectOne("User.getUser", vo);
    }
}
```

@Service

```
public class UserServiceImpl {
    @Autowired
    UserDao userDao;
    public UserVO getUser(UserVO vo) {
        return userDao.getUser(vo);
    }
}
```

# 로그인 체크 실습

- LoginController class 생성
- /login/login - login form page
- /login/loginOk
- /login/logout

/login/login

```
@Controller
@RequestMapping(value="/login")
public class LoginController {

    @Autowired
    UserServiceImpl service;

    @RequestMapping(value = "/login", method = RequestMethod.GET)
    public String login() {
        return "login";
    }
}
```

# 로그인 체크 실습

- LoginController class
  - /login/loginOk
  - /login/logout

```
@RequestMapping(value="/loginOk",method=RequestMethod.POST)
public String loginCheck(HttpSession session,UserVO vo){
    String returnUrl = "";
    if ( session.getAttribute("login") != null ){
        session.removeAttribute("login");
    }

    UserVO loginvo = service.getUser(vo);
    if ( loginvo != null ){ // 로그인 성공
        System.out.println("로그인 성공!");
        session.setAttribute("login", loginvo);
        returnUrl = "redirect:/board/list";
    }else { // 로그인 실패
        System.out.println("로그인 실패!");
        returnUrl = "redirect:/login/login";
    }
    return returnUrl;
}

// 로그아웃 하는 부분
@RequestMapping(value="/logout")
public String logout(HttpSession session) {
    session.invalidate();
    return "redirect:/login/login";
}
```

/login/loginOk

/login/logout

# 로그인 체크 실습

- /login/login
- login.jsp
- 로그인 테스트



User ID:

Password:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<style>
    img, label { display:inline-block;}
    label{ width:130px}
    button{ background-color:blue; color:white;font-size:15px}
</style>
</head>
<body>
<div style='width:100%;text-align:center;padding-top:100px'>
<img src='../img/snowman.jpg' height="250">
<form method="post" action="loginOk">
<div><label>User ID: </label><input type='text' name='userid' /></div>
<div><label>Password: </label>
    <input type='password' name='password' /></div>
<button type='submit'>login</button>
</form>
</div>
</body>
</html>
```

# 로그인 체크 실습

- LoginCheckInterceptor class 생성

```
public class LoginCheckInterceptor extends HandlerInterceptorAdapter {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
        throws Exception {
        HttpSession session = request.getSession();
        Object obj = session.getAttribute("login");
        if (obj == null) {
            response.sendRedirect(request.getContextPath() + "/login/login");
            return false;
        }
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
        ModelAndView modelAndView) throws Exception {
        super.postHandle(request, response, handler, modelAndView);
    }
}
```

# 로그인 체크 실습

- Interceptor 설정 : servlet-context.xml
- Spring framework 3.2 version부터 지원(exclude-mapping 기능)
- 테스트

```
<resources mapping="/resources/**" location="/resources/" />
```

```
<resources mapping="/img/**" location="/resources/img/" />
```

...

```
<beans:bean id="LoginCheckInterceptor" class="com.mycom.myapp.user.LoginCheckInterceptor">
```

```
</beans:bean>
```

```
<interceptors>
```

```
  <interceptor>
```

```
    <mapping path="/" />
```

```
    <exclude-mapping path="/resources/**" />
```

```
    <exclude-mapping path="/img/**" />
```

```
    <exclude-mapping path="/login/" />
```

```
    <beans:ref bean="LoginCheckInterceptor" />
```

```
  </interceptor>
```

```
</interceptors>
```

