

개발 경험 프로젝트 두 번째

—

Maven

빌드란?

Model View Controller
(MVC)

Java Server Pages
(JSP)

Tomcat

Web Application Server
(WAS)

Web Tools Platform
(WTP)

Create Read(or Retrieve) Update Delete
(CRUD)

학습 목표

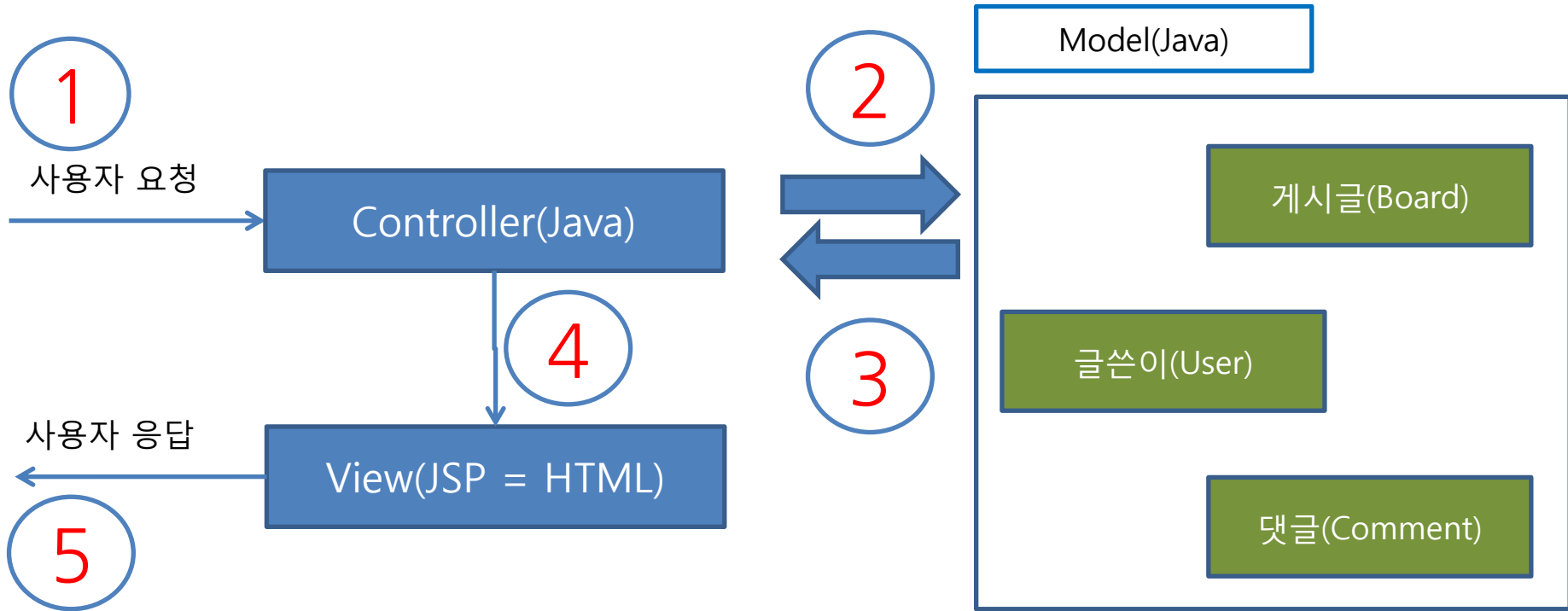
데이터베이스에 데이터 추가 및 조회

웹 서버에 파일 첨부

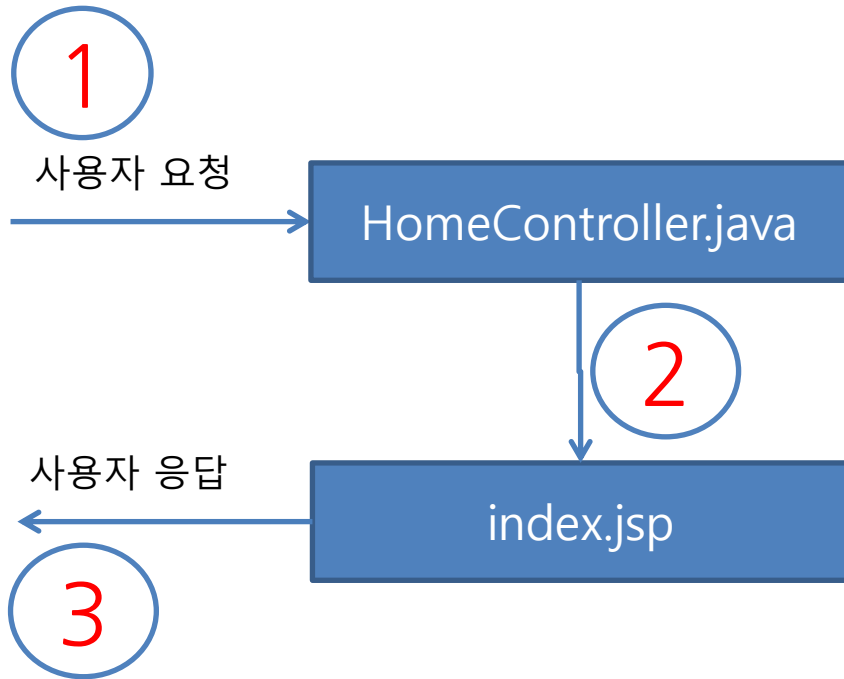
이전 시간 복습 및 과제 내용

—

MVC



Controller + View 구조



<http://localhost:8080>

Controller

```
@Controller  
public class HomeController {  
    @RequestMapping("/")  
    public String index() {  
        return "index";  
    }  
}
```

View

```
// HomeController.java
```

```
package org.nhnnext.web;
```

```
@Controller
```

```
public class HomeController {
```

```
    @RequestMapping("/")
```

```
    public String index() {
```

```
        return "index";
```

```
    }
```

```
}
```



```
// webapp/WEB-INF/web-servlet.xml
```

```
<bean id="viewResolver"
```

```
    class="org.springframework.web.servlet.  
    view.InternalResourceViewResolver"
```

```
    p:prefix="/"
```

```
    p:suffix=".jsp" />
```

프로젝트의 웹 자원 디렉토리(webapp) + prefix 경로 + Controller 반환 경로 + suffix 경로
(next-board/webapp/index.jsp 파일을 View 파일로 찾는다.)

Controller

```
@Controller
public class BoardController {
    @RequestMapping("/board/form")
    public String form() {
        return "form";
    }

    @RequestMapping(value="/board", method=RequestMethod.POST)
    public String create(String title, String contents) {
        System.out.println("title : " + title + " contents : " + contents);
        return "redirect:/";
    }
}
```

HTTP method

- 웹 서버에 요청을 보내는 방법
- HTTP method에는 get, post, put, delete 등 8가지의 요청 방식이 있음
- 브라우저는 get과 post 요청 방식만 지원

GET vs POST

- GET은 서버에서 자원을 조회하는 경우 사용한다.
- POST는 서버에서 자원을 추가하거나 상태를 변경하는 경우 사용한다.

BoardController 클래스를 보니 중복이 있다. 중복 제거할 수 없을까?

BoardController 클래스를 보니 중복이 있다. 중복 제거할 수 없을까?

```
@Controller
public class BoardController {
    @RequestMapping("/board/form")
    public String form() {
        return "form";
    }

    @RequestMapping(value="/board", method=RequestMethod.POST)
    public String create(String title, String contents) {
        System.out.println("title : " + title + " contents : " + contents);
        return "redirect:/";
    }
}
```

클래스 단위로 URL 매핑을 설정할 수 있다.

```
@Controller
@RequestMapping("/board")
public class BoardController {
    @RequestMapping("/form")
    public String form() {
        return "form";
    }

    @RequestMapping(value="", method=RequestMethod.POST)
    public String create(String title, String contents) {
        System.out.println("title : " + title + " contents : " + contents);
        return "redirect:/";
    }
}
```

BoardController.create()의 반환 값에 "redirect:/"는 뭐지?
(forward 방식과 redirect 방식의 차이점은?)

BoardController.create()의 반환 값에 "redirect:/"는 뭐지?
(forward 방식과 redirect 방식의 차이점은?)

forward

```
@Controller
public class BoardController {
    [...]

    @RequestMapping(value="/board", method=RequestMethod.POST)
    public String create(String title, String contents) {
        System.out.println("title : " + title + " contents : " + contents);
        return "index";
    }
}
```


BoardController.create()의 반환 값에 "redirect:/"는 뭐지?
(forward 방식과 redirect 방식의 차이점은?)

redirect

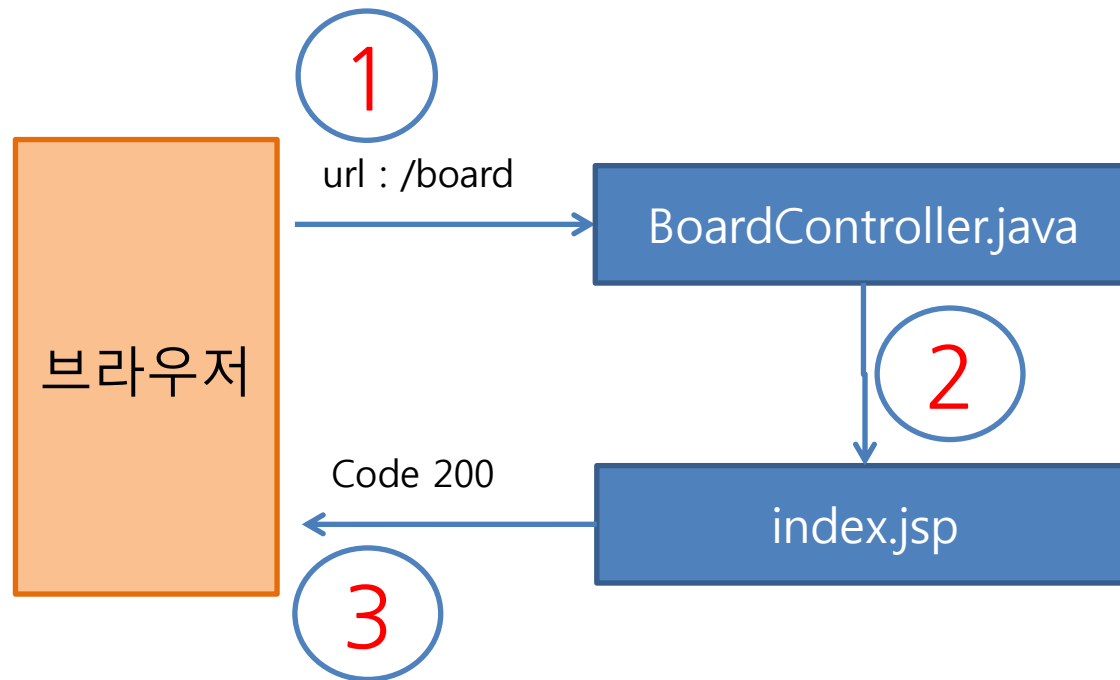
```
@Controller
public class BoardController {
    [...]

    @RequestMapping(value="/board", method=RequestMethod.POST)
    public String create(String title, String contents) {
        System.out.println("title : " + title + " contents : " + contents);
        return "redirect:/" ;
    }
}
```

forward

```
@Controller
public class BoardController {
    [...]

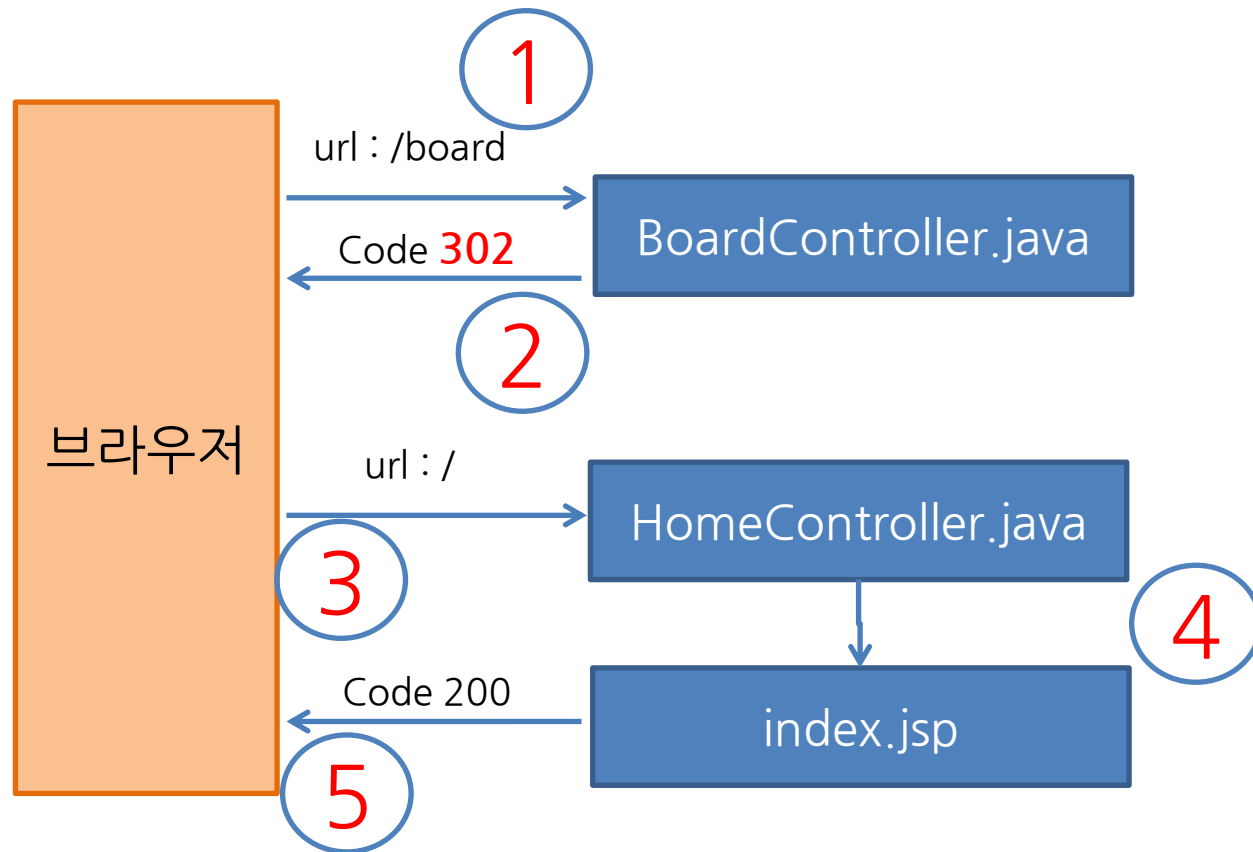
    @RequestMapping(value="/board", method=RequestMethod.POST)
    public String create(String title, String contents) {
        System.out.println("title : " + title + " contents : " + contents);
        return "index";
    }
}
```



redirect

```
@Controller
public class BoardController {
    [...]

    @RequestMapping(value="/board", method=RequestMethod.POST)
    public String create(String title, String contents) {
        System.out.println("title : " + title + " contents : " + contents);
        return "redirect:/";
    }
}
```

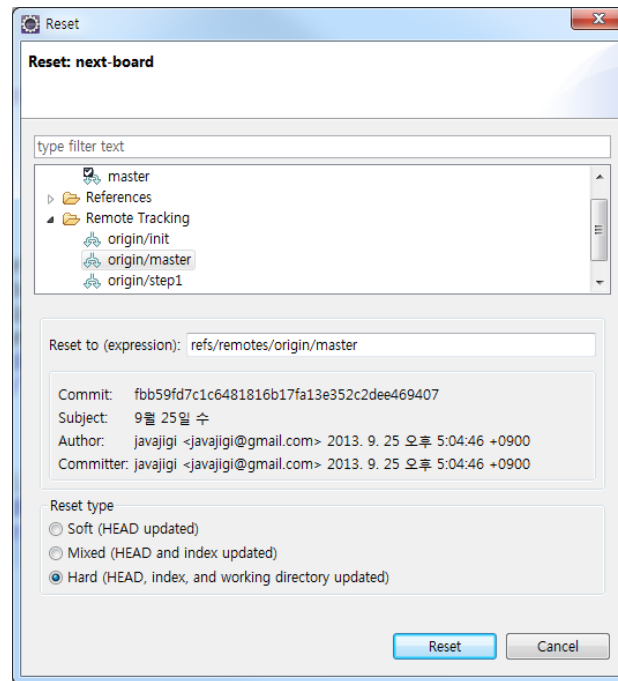


게시 글쓰기 - 데이터 매핑 개선

-

수업 진행을 위한 프로젝트 초기화

1. next-board 프로젝트 오른쪽 클릭 > Team > Commit
2. next-board 프로젝트 오른쪽 클릭 > Team > Pull
3. next-board 프로젝트 오른쪽 클릭 > Team > Reset... > Remote Tracking의 origin/master를 선택 > Reset type을 Hard 선택 후 Reset한다.



```
@Controller
public class BoardController {
    [...]

    @RequestMapping(value="/board", method=RequestMethod.POST)
    public String create(String title, String contents) {
        System.out.println("title : " + title + " contents : " + contents);
        return "redirect:/";
    }
}
```

위 Controller는 클라이언트에서 전달하는 데이터가 두 개(title, contents) 밖에 없다.
전달하는 데이터 수가 많아질 경우 어떻게 구현하는 것이 좋을까?

클라이언트와 서버 데이터 매핑

실습

```
public class Board {
    private String title;

    private String contents;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContents() {
        return contents;
    }

    public void setContents(String contents) {
        this.contents = contents;
    }

    @Override
    public String toString() {
        return "Board [title=" + title + ", contents=" + contents + "]\n";
    }
}
```

여러 개의 데이터를 하나의 클래스에 저장해
처리하는 것이 일반적인 개발 방식이다.

클래스에 데이터를 저장하고, 조회할 때
사용하는 메서드를 **setter(저장)**, **getter(조회)**
메서드라 한다.

setter, getter 메서드 생성 규칙은 다음과 같다.

- setter : set + 필드명(단, 첫글자는 대문자)
- getter : get + 필드명(단, 첫글자는 대문자)

```
public class Board {  
    private String title;  
  
    private String contents;  
  
    public String getTitle() {  
        return title;  
    }  
  
    public void setTitle(String title) {  
        this.title = title;  
    }  
  
    public String getContents() {  
        return contents;  
    }  
  
    public void setContents(String contents) {  
        this.contents = contents;  
    }  
  
    [...]   
}
```


클라이언트와 서버 데이터 매핑

실습

```
<form action="/board" method="post">
  제목 : <input type="text" name="title" size=40> <br />
  <textarea name="contents" rows="10" cols="50"></textarea><br />
  <input type="submit" value="보내기">
</form>
```

```
public class Board {
  private String title;

  private String contents;

  [...]

  public void setTitle(String title) {
    this.title = title;
  }

  public void setContents(String contents) {
    this.contents = contents;
  }
}
```

클라이언트에서 전달되는 데이터는 setter 메소드를 통해 전달된다.

예를 들어 클라이언트에서 전달하는 변수명이 title이면 setTitle() 메소드를 통해 데이터가 전달된다.

Controller

실습

```
@Controller
@RequestMapping("/board")
public class BoardController {
    [...]

    @RequestMapping(value="", method=RequestMethod.POST)
    public String create(Board board) {
        System.out.println("Board : " + board);
        return "redirect:/";
    }
}
```

게시 글쓰기 - DB 저장

—

BoardController.create()에서 System.out.println() 사용해 출력만 하고 있다.
다음으로 무슨 작업을 해야 할까?

데이터베이스 시작

1. 터미널을 연 후 next-board/src/main/db 디렉토리 이동한다.
2. `> chmod 755 startup.sh` 를 실행해 startup.sh 실행 가능한 파일로 변경한다.
3. `./startup.sh` 파일을 실행한다.
4. Saved Settings를 Generic H2(Embedded)에서 Generic H2(Server)로 변경한다.
5. 브라우저가 열리면 JDBC URL을 `jdbc:h2:tcp://localhost/~next-board` 로 변경한 후 connect 버튼을 클릭해 접근한다.

데이터베이스 설정 변경

1. next-board/src/main/resources 디렉토리의 applicationContext-next.xml 파일을 연다.
2. 주석 처리되어 있는 부분의 주석을 해제한다.
3. 데이터베이스 연결을 위한 접속 정보는 next-board/src/main/resources 디렉토리의 application-properties.xml 파일에서 관리하고 있다.

다른 데이터베이스로 변경하고자 한다면(선택사항)

1. pom.xml 파일에 데이터베이스 접속을 위한 라이브러리를 추가한다. 예를 들어 Maria db는 maria 접속 라이브러리가 별도로 존재한다.
2. next-board/src/main/resources 디렉토리의 application-properties.xml 파일에서 데이터베이스 접속 정보를 변경한다. “jdbc maria url example” 또는 “jdbc mysql url example” 같은 키워드로 검색하면 정보를 얻을 수 있다.
3. next-board/src/main/resources/META-INF 디렉토리의 persistence.xml의 hibernate.dialect 속성을 해당 데이터베이스 설정으로 변경한다. “mariadb hibernate dialect” 같은 형태로 키워드로 검색하면 정보를 얻을 수 있다.

서버 데이터와 DB 테이블 매핑

실습

```
package org.nhnnext.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

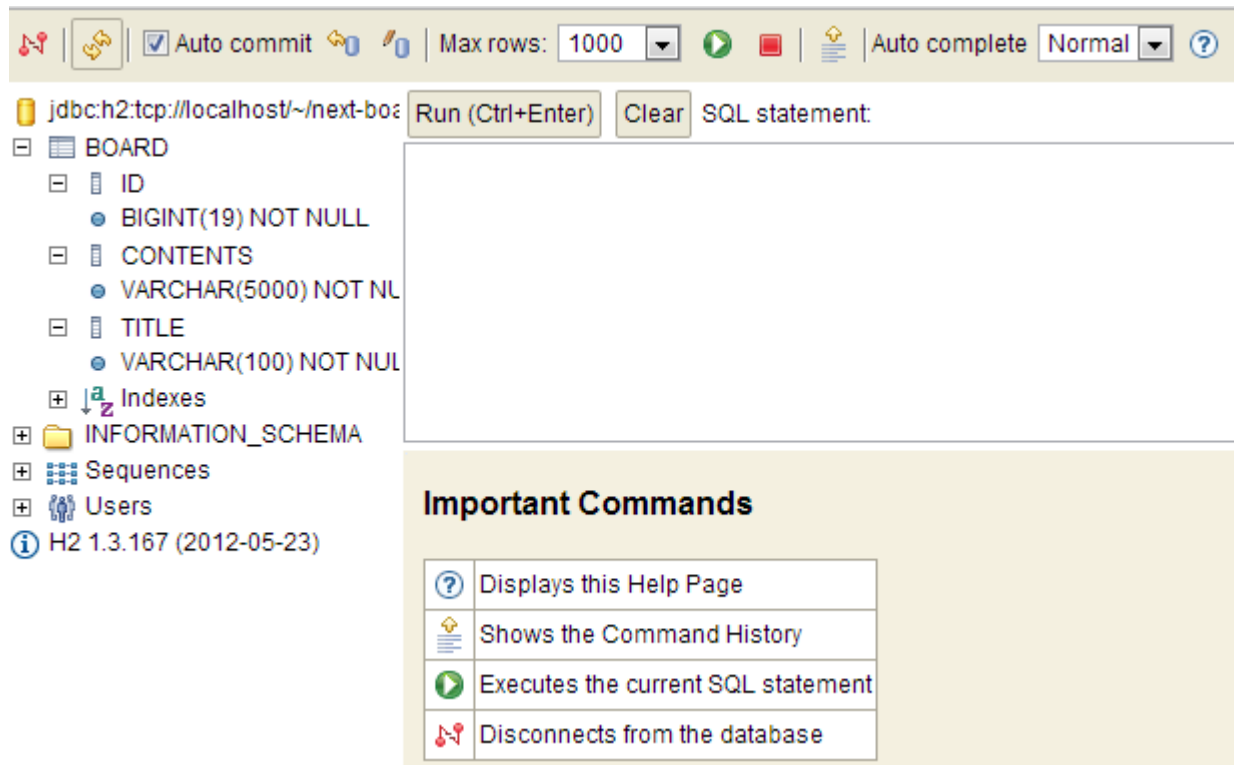
    @Column(length=100, nullable = false)
    private String title;

    @Column(length=5000, nullable = false)
    private String contents;

    [...]
}
```


데이터베이스 확인

1. 앞과 같이 서버 데이터와 데이터베이스 테이블을 매핑한다.
2. WTP 서버를 재시작한다.
3. 데이터베이스에 연결되어 있는 브라우저를 새로고침 한다.
4. 테이블이 생성되어 있는지 확인한다.



Repository

실습

```
package org.nhnnext.repository;

import org.nhnnext.model.Board;
import org.springframework.data.repository.CrudRepository;

public interface BoardRepository extends CrudRepository<Board, Long>{

}
```


Controller

실습

```
@Controller
public class BoardController {
    @Autowired
    private BoardRepository boardRepository;

    [...]

    @RequestMapping(value="/board", method=RequestMethod.POST)
    public String create(Board board) {
        boardRepository.save(board);
        return "redirect:/";
    }
}
```



자바 객체와 테이블 매핑

@Entity

```
@Entity  
public class Board {  
  
}
```

@Entity Annotation은 자바 객체와 데이터베이스 테이블을 매핑한다.

@Id 매핑 및 자동 증가

```
@Entity
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
}
```

@Id Annotation은 데이터베이스의 primary key와 매핑한다.

@GeneratedValue Annotation은 ID 생성 규칙에 대한 설정이다.

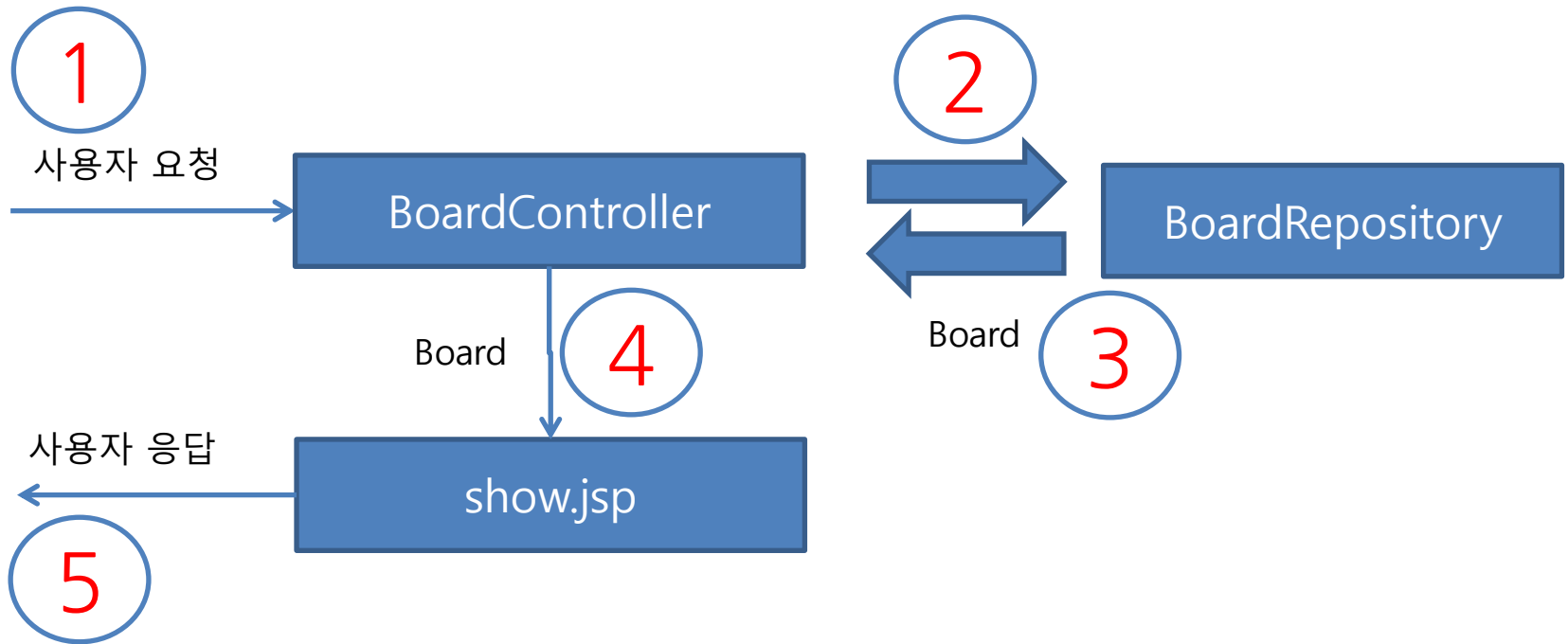
@Column

```
@Entity
public class Board {
    @Column(name = "title", length=100, nullable = false)
    private String title;
}
```

@Column Annotation은 자바의 클래스 필드와 테이블 칼럼을 매핑한다.

게시 글 상세보기

—



BoardController - forward

실습

```
import org.springframework.ui.Model;

@Controller
@RequestMapping("/board")
public class BoardController {
    @Autowired
    private BoardRepository boardRepository;

    [...]

    @RequestMapping(value="", method=RequestMethod.POST)
    public String create(Board board, Model model) {
        Board savedBoard = boardRepository.save(board);
        model.addAttribute("board", savedBoard);
        return "show";
    }
}
```

Model 클래스는 jsp에 출력할 데이터를 전달하는 역할을 한다.
jsp에 데이터를 전달할 때는 Key/Value 구조로 전달한다.

show.jsp

실습

```
[...]  
<body>  
<h1>${board.title}</h1>  
<div>  
  ${board.contents}  
</div>  
</body>  
</html>
```

Controller에서 Model 클래스에 저장할 때의 Key 값으로 접근한다.

`${board.title}`은 Board 클래스의 `getTitle()` 메서드를 호출한다.

BoardController - redirect

실습

```
@Controller
@RequestMapping("/board")
public class BoardController {
    @Autowired
    private BoardRepository boardRepository;

    [...]

    @RequestMapping(value="", method=RequestMethod.POST)
    public String create(Board board) {
        Board savedBoard = boardRepository.save(board);
        return "redirect:/board/" + savedBoard.getId();
    }
}
```

forward 방식으로 구현할 경우 새로고침할 때마다 데이터가 중복으로 저장된다. redirect 방식으로 위와 같이 구현해야 한다.

BoardController - show

미션

```
@Controller
@RequestMapping("/board")
public class BoardController {
    @Autowired
    private BoardRepository boardRepository;

    [...]

    @RequestMapping("/{id}")
    public String show(@PathVariable Long id) {
        // TODO DB에서 id에 해당하는 Board 데이터를 조회해야 한다.
        // TODO 조회한 Board 데이터를 Model에 저장해야 한다.
        return "show";
    }
}
```

TODO에 해당하는 기능을 구현한다.

게시 글쓰기 - 파일 첨부 UI

-

File을 서버에 업로드 할 때 무엇이 다를까?

File은 문자열을 전송하는 것과 다르다.

File은 **binary data**이다.

따라서 form 전송시에 다른 형태의 데이터임을 표시해줘야 한다.

아래와 같이 form에 encoding type 표시해주는 속성을 이용한다.

enctype = “multipart/form-data”

▼ Request Headers [view source](#)

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4
Cache-Control: max-age=0
Connection: keep-alive
Content-Length: 12682
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryWjak8o
Cookie: JSESSIONID=FF7E0D620610097B65FFF081E0D198E4
Host: localhost:8080
```

File을 전송하기 위한 input type

```
<input type="file" name="file" />
```


기존 title과 contents에 file이 추가된 form 전체 코드

```
<form action="UploadFile.jsp" method="post" enctype="multipart/form-data">

<input type="text" name="title" size=30> <br>
<textarea cols="70" rows="3" name="contents"></textarea><br>
<input type="file" name="file" size="50" /> <br>
<input type="submit" value="Upload File" />

</form>
```

서버로 전송되는 것은?

Chrome dev tool로 확인해보자.

```
<form action="UploadFile.jsp" method="post" enctype="multipart/form-data">

<input type="text"  name="title" size=30> <br>
<textarea cols="70" rows="3" name="contents"></textarea><br>
<input type="file" name="file" size="50" /> <br>
<input type="submit" value="Upload File" />

</form>
```

게시 글쓰기 - 파일 첨부 서버

-

파일 첨부 설정

pom.xml 파일

```
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>1.4</version>
</dependency>
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.2.1</version>
</dependency>
```

의존성을 추가하면 프로젝트를 다시 빌드해 추가한 라이브러리가 클래스 패스에
추가되도록 해야한다. How?

파일 첨부 설정

실습

webapp/WEB-INF/next-servlet.xml 파일

```
<bean id="multipartResolver"  
class="org.springframework.web.multipart.commons.CommonsMultipartResolver"  
    p:maxUploadSize="2000000" />
```

파일 첨부 설정

실습

`org.nhnnnext.support.FileUploader.java`

```
private static final String ATTACHMENT_ROOT_DIR = "{your_workspace}/next-board/webapp/images";
```

이미지 파일을 업로드할 디렉토리를 수정한다. 현재 next-board 프로젝트가 위치하고 있는 디렉토리 경로로 수정한다.

파일 첨부 구현

```
@Controller
@RequestMapping("/board")
public class BoardController {
    @Autowired
    private BoardRepository boardRepository;

    [...]

    @RequestMapping(value="", method=RequestMethod.POST)
    public String create(Board board, MultipartFile attachment) {
        // TODO FileUploader API를 활용해 업로드한 파일을 복사한다.
        // TODO 첨부한 이미지 정보를 데이터베이스에 추가한다.
        Board savedBoard = boardRepository.save(board);
        return "redirect:/board/" + savedBoard.getId();
    }
}
```

파일 첨부 구현

// TODO 첨부한 이미지 정보를 데이터베이스에 추가한다.(힌트)

- MultipartFile API나 FileUploader API에서 업로드한 파일 이름 정보를 구한다.
- Board 클래스에 파일 이름을 저장할 수 있는 필드를 추가한다.
- 필드에 대한 setter, getter 메서드를 생성한다.
- setter 메서드를 활용해 Board에 파일 이름을 저장한다.

이미지 보여주기

실습

// TODO 첨부한 이미지를 상세 보기 화면(show.jsp)에서 볼 수 있도록 구현한다.

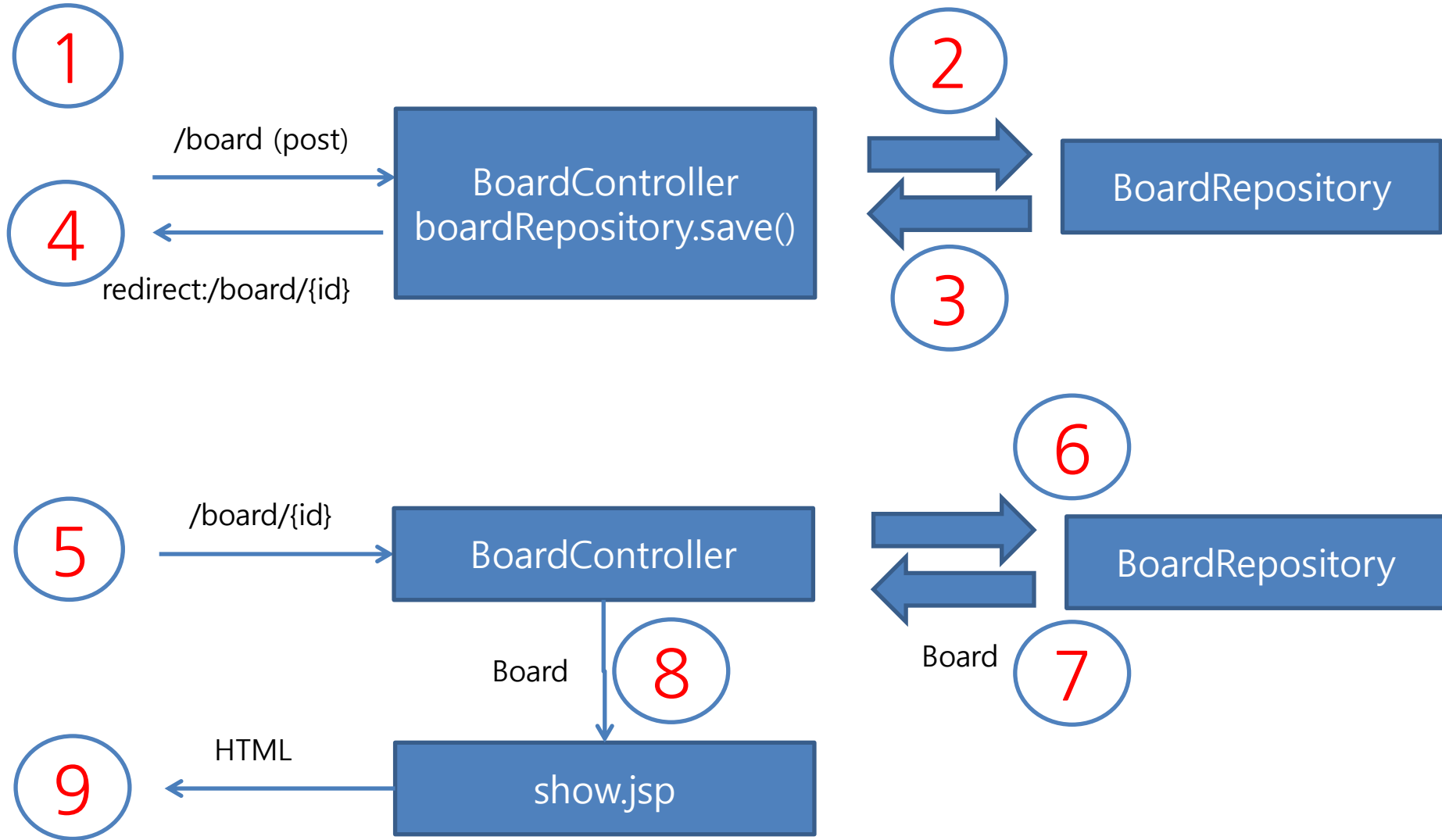
이미지 보여주기

실습

// TODO 첨부한 이미지를 상세 보기 화면(show.jsp)에서 볼 수 있도록 구현한다.(힌트)

- Board 클래스에서 첨부한 이미지 정보를 구할 수 있다.
- 첨부한 이미지는 /images/{이미지 파일 이름}으로 접근할 수 있다.
- 정확히 구현했음에도 불구하고 이미지가 보이지 않을 경우 next-board 프로젝트를 refresh한다.(프로젝트 오른쪽 클릭 refresh 또는 F5)
- HTML의 img 태그를 활용해서 이미지를 화면에 보여준다.

게시판 글쓰기와 내용 보기



실습 시간 미션 - 10월 2일

—

설정 변경

- next-board에 추가되어 있는 org.nhnnnext.support.FileUploader 소스 코드를 자신의 프로젝트 복사한다.
- 수업 내용 참고해서 파일 업로드를 위한 설정을 추가한다.
- src/main/resources/application-properties.xml 파일에서 데이터베이스 경로를 next-board에서 다른 이름으로 변경한다.

1 단계

미션 1. 사진 웹 서비스에 이미지를 첨부한다.

미션 2. 입력한 데이터와 이미지 정보를 데이터베이스에 추가한다.

미션 3. 데이터베이스에 추가한 데이터를 조회할 수 있는 페이지를 구현한다.

2단계

미션 1. 상세 보기 페이지에서 데이터를 수정할 수 있는 수정 페이지를 구현하다.

미션 2. 수정 페이지에서 데이터를 수정하면 데이터베이스에 변경된 데이터가 반영되어야 한다.

힌트 : 데이터 수정은 데이터 생성과 같이 `save()` 메소드를 활용해 가능하다.

3단계

미션 1. 생성한 데이터를 삭제한다.

힌트 : Repository의 delete() 메서드를 활용해 데이터를 삭제할 수 있다.

과제 - 10월 15일까지

-

1 단계

미션 1. 사진 웹 서비스의 목록 페이지를 구현한다.

힌트 : 목록은 Repository의 findAll() 메소드를 활용한다.

jsp의 jstl이라는 태그를 활용해 구현한다.

<c:forEach ...> 태그 활용해 구현한다.