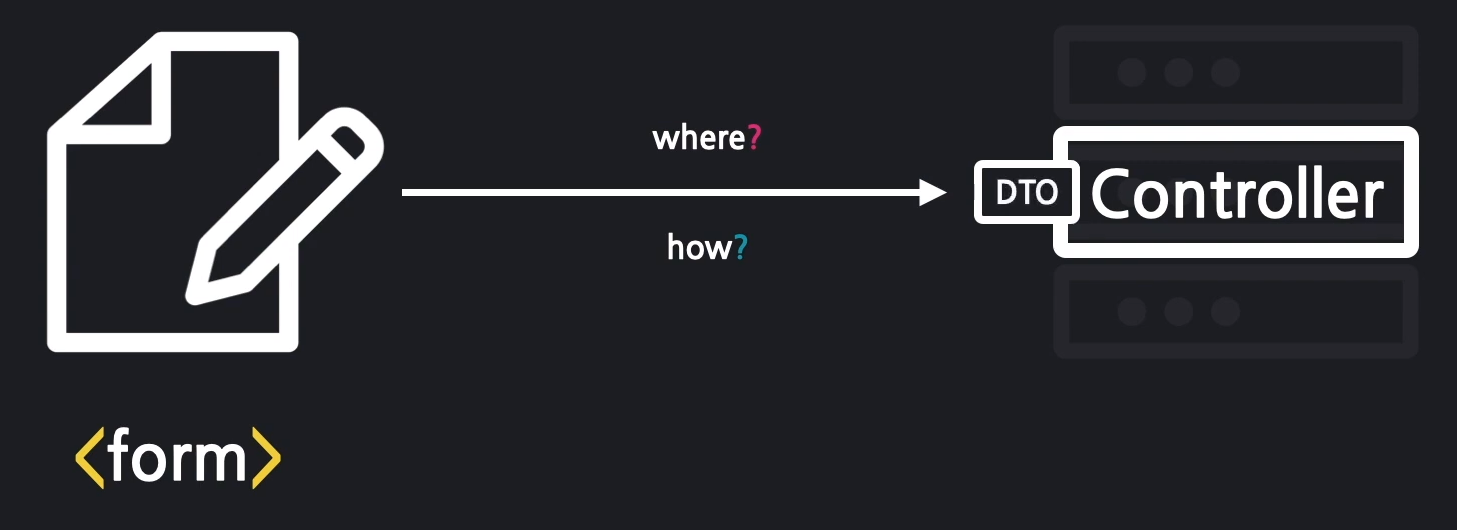
폼(form) 데이터 주고 받기

  
  
폼 데이터를 컨트롤러에서 받을때 DTO 객체를 이용해서 받는다.

**입력 폼 만들기**

**new.mustache**

부트스트랩사용

{{>layouts/header}}

<form action="" class="container">

<div class="mb-3">

<label class="form-label">제목</label>

<input type="text" class="form-control">

</div>

<div class="mb-3">

<label class="form-label">내용</label>

<textarea class="form-control" rows="3"></textarea>

</div>

<button type="submit" class="btn btn-primary">submit</button>

</form>

{{>layouts/footer}}

**컨트롤러 만들기**

**ArticleController.java**

package com.example.firstproject.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class ArticleController {

@GetMapping("articles/new")

public String newArticleForm() {

return "articles/new";

}

}

**폼 데이터 전송**

폼 데이터 전송을 위해 form태그에  
action과 method를 추가하기.

{{>layouts/header}}

<form action="/articles/creats" class="container" method="post">

<div class="mb-3">

<label class="form-label">제목</label>

<input type="text" class="form-control">

</div>

<div class="mb-3">

<label class="form-label">내용</label>

<textarea class="form-control" rows="3"></textarea>

</div>

<button type="submit" class="btn btn-primary">submit</button>

</form>

{{>layouts/footer}}

**폼 데이터 받기**

폼 데이터 받기위해 컨트롤러에 메소드 추가하기

@PostMapping("/articles/create")

public String createArticle() {

return "";

}

**DTO 작성하기**

**ArticleForm.java**

form에서 넘어오는 title,content 변수 선언 후  
생성자와 잘 넘어오는 지 확인 하기위해 toString() 메서드 선언.

package com.example.firstproject.dto;

public class ArticleForm {

private String title;

private String content;

public ArticleForm(String title, String content) {

this.title = title;

this.content = content;

}

@Override

public String toString() {

return "ArticleForm{" +

"title='" + title + '\'' +

", content='" + content + '\'' +

'}';

}

}

**데이터 받아오기**

컨트롤러에서 DTO 데이터를 받아오기 위해서는 파라미터로 DTO를 받아줘야한다.  
파라미터에 ArticleForm추가  
확인을 위해서 System.out.println(form.toString());

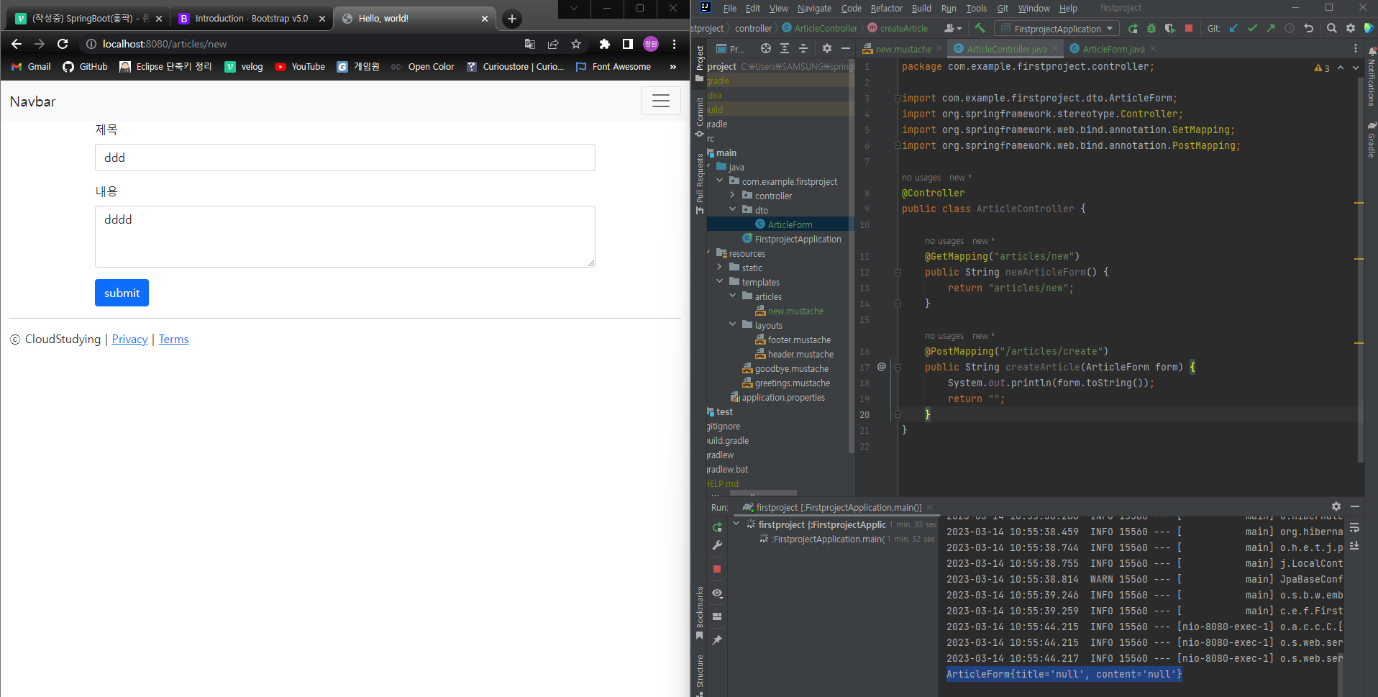
@PostMapping("/articles/create")

public String createArticle(ArticleForm form) {

System.out.println(form.toString());

return "";

}



파라미터 추가 후 데이터 전송해보면 null이 찍힌다.  
이유는 입력값 이름을 추가해 주어야한다.

**입력값 이름주기**

form에 DTO 변수와 이름이 같에 name 속성 추가하기

<form action="/articles/create" class="container" method="post">

<div class="mb-3">

<label class="form-label">제목</label>

<input type="text" class="form-control" name="title">//name추가

</div>

<div class="mb-3">

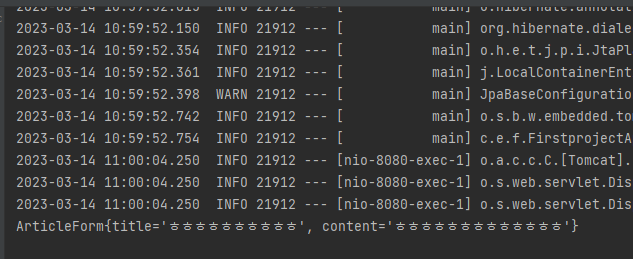
<label class="form-label">내용</label>

<textarea class="form-control" rows="3" name="content">//name추가</textarea>

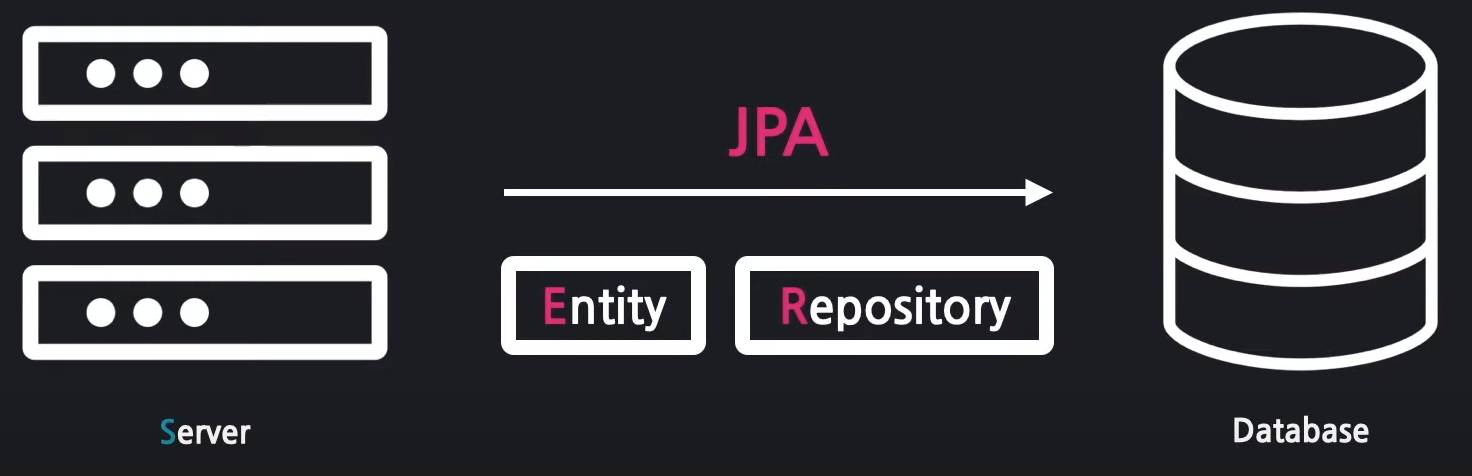
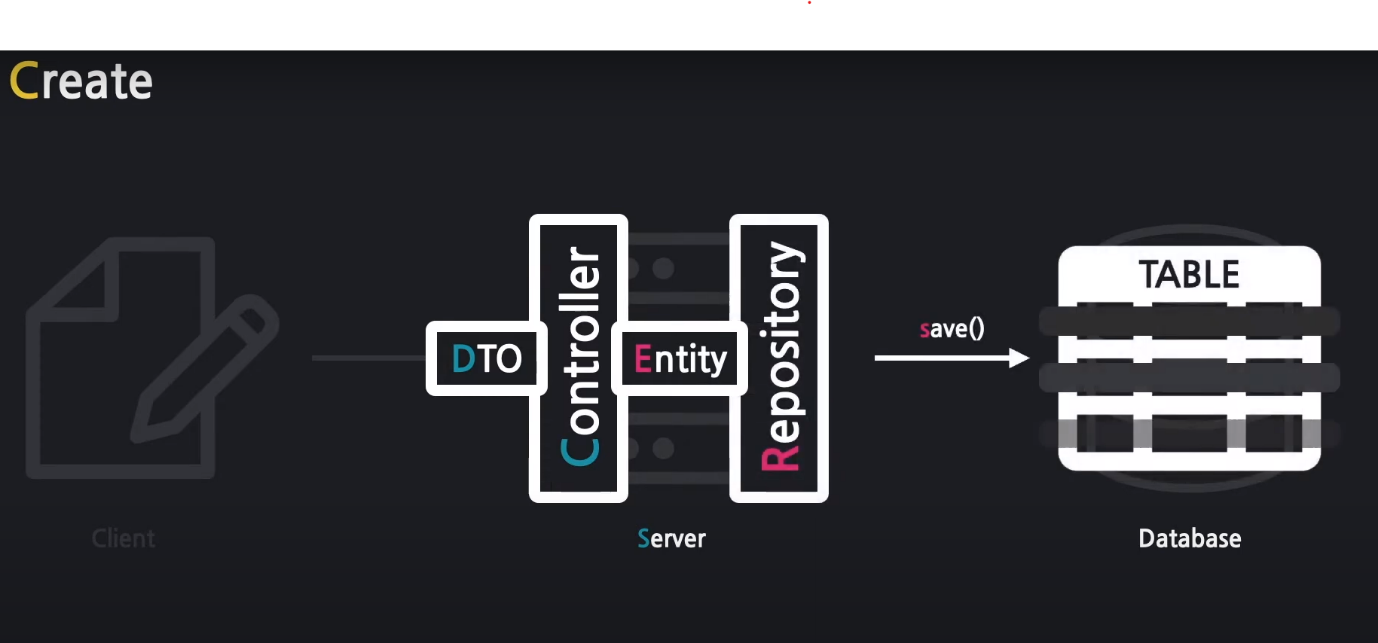
</div>

<button type="submit" class="btn btn-primary">submit</button>

</form>

속성 추가 하면 값 잘 가지고 온다.  


**데이터 생성 with JPA**

Form 데이터를 받아와서 DB에 저장해보자.  
  
DB는 JAVA를 모르기때문에 JPA를 이용한다.  


1. DTO -> Entity 변환
2. Repository 에게 Entity를 DB안에 저장하게 함

@PostMapping("/articles/create")

public String createArticle(ArticleForm form) {

System.out.println(form.toString());

// 1. DTO -> Entity 변환

Article article = form.toEntity();

// 2. Repository 에게 Entity를 DB안에 저장하게 함

return "";

}

**Entity 생성(Article.java)**

DTO와 같게 변수 선언 후 어노테이션 추가하기.

package com.example.firstproject.entity;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

@Entity // 1.DB가 해당 객체를 인식 가능하게 됨.

public class Article {

@Id // 3.Entity 대표값 추가

@GeneratedValue // 1,2,3... 자동 생성 어노테이션

private Long id;

@Column // 2.Column 어노테이션 추가

private String title;

@Column

private String content;

public Article(Long id, String title, String content) {

this.id = id;

this.title = title;

this.content = content;

}

@Override

public String toString() {

return "Article{" +

"id=" + id +

", title='" + title + '\'' +

", content='" + content + '\'' +

'}';

}

}

**변환 메소드 추가(toEntity())**

ArticleForm.java에 toEntity() 추가.  
toEntity()는 Article을 반환하기 때문에 새 Article객체를 생성해서 전달한다.  
파라미터로 Article객체의 변수들을 전달한다.  
id는 자동생성이기때문에 null

public Article toEntity() {

return new Article(null, title, content);

}

**2. Repository 에게 Entity를 DB안에 저장하게 함**

private ArticleRepository articleRepository; // ArticleRepository 선언

...

@PostMapping("/articles/create")

public String createArticle(ArticleForm form) {

System.out.println(form.toString());

// 1. DTO -> Entity 변환

Article article = form.toEntity();

// 2. Repository 에게 Entity를 DB안에 저장하게 함

Article saved = articleRepository.save(article);

return "";

}

**레파지토리 생성**

JPA의 CrudRepository를 상속받아서 사용한다.  
다른 구현없이 CRUD만 구현할때 사용.  
CrudRepository< > 안에는 Entity, Entity의 id의 타입을 넣어준다.

CrudRepository<Article, Long>

package com.example.firstproject.repository;

import com.example.firstproject.entity.Article;

import org.springframework.data.repository.CrudRepository;

public interface ArticleRepository extends CrudRepository<Article, Long> {

}

@Entity

public class Article {

@Id

@GeneratedValue

private Long id;

...

**객체 주입하기(DI)**

private ArticleRepository articleRepository = new ArticleRepository() { }.. 할 필요없이  
@Autowired 어노테이션을 붙이면 스프링 부트가 미리 생성해놓은 객체를 가져다가 자동 연결해준다.

@Controller

public class ArticleController {

@Autowired // 스프링 부트가 미리 생성해놓은 객체를 가져다가 자동 연결!

private ArticleRepository articleRepository;

...

✨ sysout 결과

ArticleForm{title='dlddld', content='용요용'} //DTO

Article{id=null, title='dlddld', content='용요용'} // DTO -> Entity

Article{id=1, title='dlddld', content='용요용'} // saved

My sql에 연결

Build.gradel

runtimeOnly 'com.mysql:mysql-connector-j'

추가하고 h2 유는 주석처리 + 코끼리(오른쪽 상단) 누르기

dependencies **{** implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
 implementation 'org.springframework.boot:spring-boot-starter-mustache'  
 implementation 'org.springframework.boot:spring-boot-starter-web'  
// runtimeOnly 'com.h2database:h2'  
 runtimeOnly 'com.mysql:mysql-connector-j'  
 testImplementation 'org.springframework.boot:spring-boot-starter-test'  
 testRuntimeOnly 'org.junit.platform:junit-platform-launcher'  
**}**

**application.properties 에 아래 내용 삽입**

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/firstProject\_db?serverTimezone=UTC&characterEncoding=UTF-8

spring.datasource.username=root

spring.datasource.password=1234

#true 설정 시 jpa 쿼리문 확인 가능

spring.jpa.show-sql=true

#DDL 정의시 DB의 고유 기능을 사용 가능

spring.jpa.hibernate.ddl-auto=create

# JPA의 구현체인 Hibernate가 동작하면서 발생한 SQL의 가독성을 높여줌

spring.jpa.properties.hibernate.format\_sql=true

mysql – workbench

왼쪽 스키마

