4-1 : 과제 피드백 및 3주차 내용 토론

들어가기 전에

이번 시간에는…

과제 피드백 & QnA

스프링 핵심 개념(IoC, DI, AOP)
Spring vs Spring Boot
Spring Controller, Service, Dao Layer
Spring 트랜잭션

과제 피드백

ERD 및 REST API 설계(협업)
스프링부트 템플릿을 활용한 REST API 개발

토론 키워드

스프링의 핵심 기능

스프링 특징

스프링 대표적인 특징

스프링은 객체지향 원칙을 강제한다.

1. 스프링 컨테이너
2. 제어의 역전(IoC)
3. 의존성 주입(DI)
4. 관점 지향 프로그래밍(AOP)

스프링 특징

스프링 대표적인 특징

스프링은 객체지향 원칙을 강제한다.

1. 스프링 컨테이너
2. 제어의 역전(IoC)
3. 의존성 주입(DI)
4. 관점 지향 프로그래밍(AOP)

스프링 특징

객체지향원칙

"역할과 구현을 분리한다.
우리는 구현체가 아닌 역할을 중심으로 설계해야 한다."

스프링 특징

객체지향원칙 – 인터페이스와 클래스

서비스 설계 시 인터페이스(기본 틀 설계도)를 기준으로 역할을 부여
역할의 세부적인 구현체는 클래스로 구현

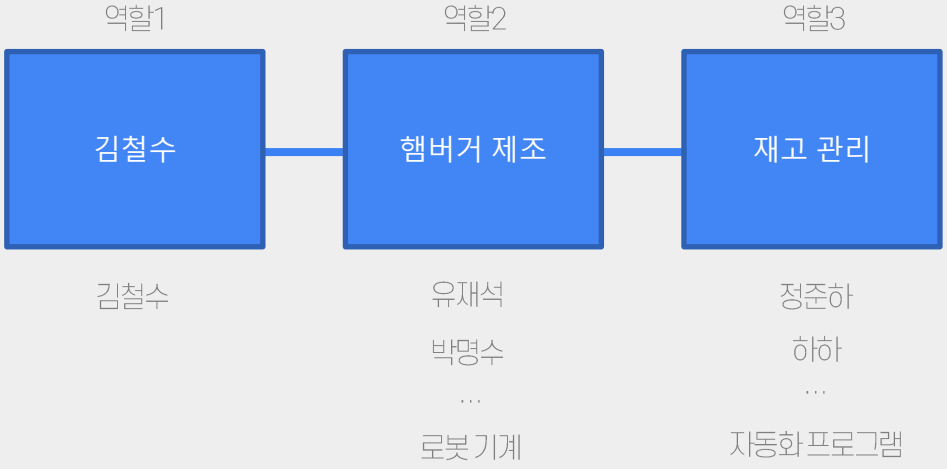
Ex) 패스트푸드점 아르바이트 역할

스프링 특징

객체지향원칙 – 인터페이스와 클래스

서비스 설계 시 인터페이스(기본 틀 설계도)를 기준으로 역할을 부여
역할의 세부적인 구현체는 클래스로 구현

Ex) 패스트푸드점 아르바이트 역할



스프링 특징

객체지향원칙 – 인터페이스와 클래스

서비스 설계 시 인터페이스(기본 틀 설계도)를 기준으로 역할을 부여
역할의 세부적인 구현체는 클래스로 구현

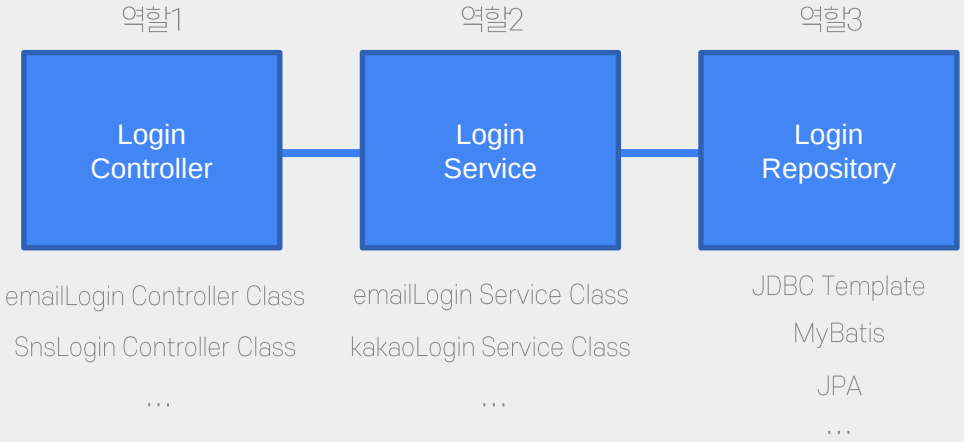
Ex) 패스트푸드점 아르바이트 역할

스프링 특징

객체지향원칙 – 인터페이스와 클래스

서비스 설계 시 인터페이스(기본 틀 설계도)를 기준으로 역할을 부여
역할의 세부적인 구현체는 클래스로 구현

Ex) 스프링에서 로그인 기능을 구현한다고 했을 때

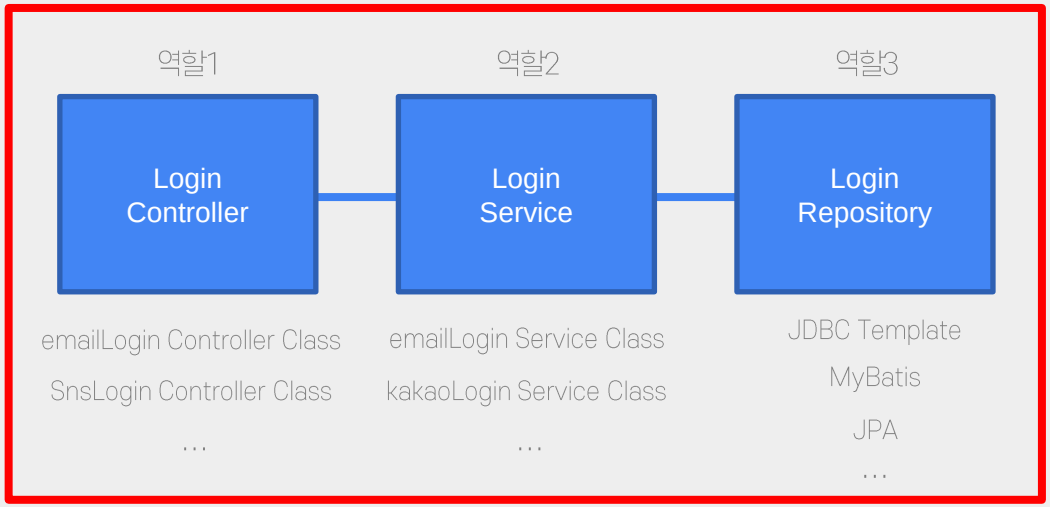


스프링 특징

객체지향원칙 – 인터페이스와 클래스

서비스 설계 시 인터페이스(기본 틀 설계도)를 기준으로 역할을 부여
역할의 세부적인 구현체는 클래스로 구현

Ex) 스프링에서 로그인 기능을 구현한다고 했을 때



각 역할에 맞게 구현체를 배정해주는 구성 역할이 있으면 좋겠다.

스프링 특징

스프링 컨테이너

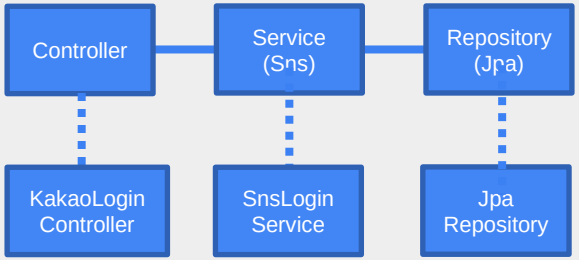
스프링은 내부적으로 스프링 컨테이너가 있다
스프링 컨테이너는 빈(Bean)이라는 단위로 역할을 구성, 관리해준다.
스프링 컨테이너는 구현 객체를 생성하고 연결하는 책임을 갖는다.

역할 구성, 관리

스프링 빈 저장소

빈 이름	빈 객체
LoginController	KakaoLoginController
LoginService	SnsLoginService
LoginRepository	JpaRepository

역할관계 설정



스프링 특징

제어의 역전(IoC)

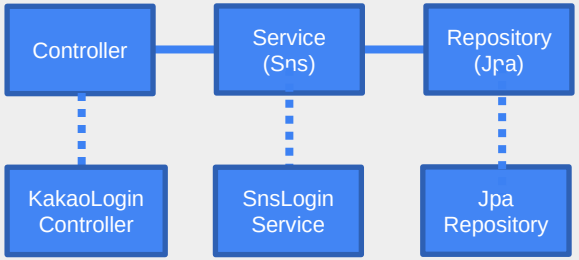
객체에 대한 제어권이 바뀐 것을 의미
스프링에서는 자바 객체를 개발자가 관리하는 것이 아닌 스프링이 제공하는 컨테이너가 관리
개발자는 필요한 부분만 개발하고, 호출은 프레임워크 내부에서 결정

역할 구성, 관리

스프링 빈 저장소

빈 이름	빈 객체
LoginController	KakaoLoginController
LoginService	SnsLoginService
LoginRepository	JpaRepository

역할관계 설정



스프링 특징

의존성 주입(DI)

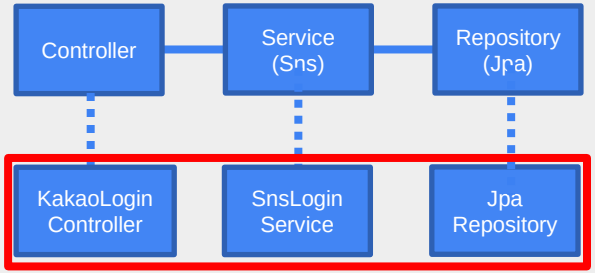
클래스 사이의 의존관계를 빈 설정 정보를 바탕으로 컨테이너가 자동으로 연결
연결된 역할에 맞게 각 역할에 맞는 구현체(클래스)를 필요한 시점에 할당해준다
제어의 역전(IoC)의 한 종류로 볼 수 있다.

역할 구성, 관리

스프링 빈 저장소

빈 이름	빈 객체
LoginController	KakaoLoginController
LoginService	SnsLoginService
LoginRepository	JpaRepository

역할관계 설정



스프링 특징

의존성 주입(DI) 예제

클래스 사이의 의존관계를 빈 설정 정보를 바탕으로 컨테이너가 자동으로 연결
연결된 역할에 맞게 각 역할에 맞는 구현체(클래스)를 필요한 시점에 할당해준다

의존성 주입(DI) X

```
public class MemberService {

    private MemberRepository memberRepository;

    public MemberService() {
        this.memberRepository = new JdbcMemberRepository(); // 1
        this.memberRepository = new MyBatisMemberRepository(); // 2
    }
}
```

새로운 구현체가 생겨날 때 마다 지속적으로 바꿔줘야한다.

의존성 주입(DI) O

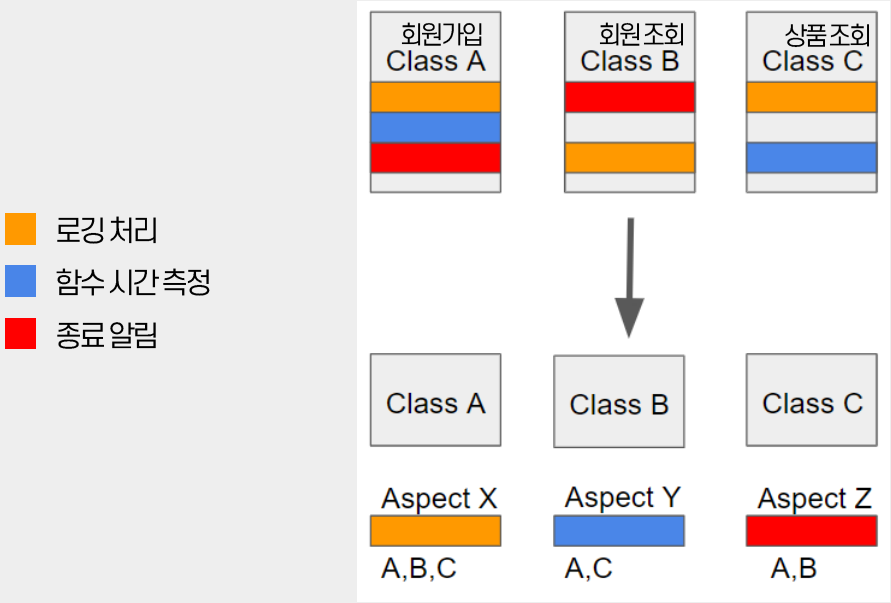
```
@Service
public class MemberService {

    private MemberRepository memberRepository;

    @Autowired
    public MemberService(MemberRepository memberRepository) {
        this.memberRepository = memberRepository;
    }
}
```

빈 설정 정보에 맞게 필요한 구현체를 외부에서 주입해준다.
어노테이션을 통해 구현

스프링 특징

관점 지향 프로그래밍(AOP)

핵심 로직과 부가 로직을 나누어 각각 모듈화 하여 관리
코드에서 반복적으로 사용되는 부가 로직을 분리, 모듈화 하여 재사용 하는 것이 목적



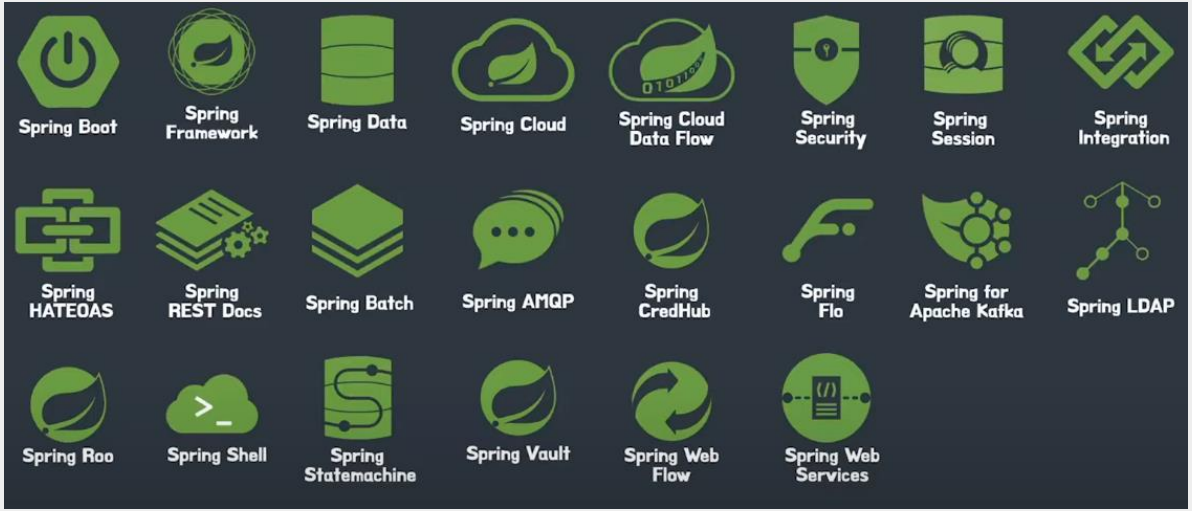
토론 키워드

Spring vs Spring Boot

스프링 생태계

스프링 프로젝트

스프링은 스프링 프레임워크를 포함하여 스프링 부트, 스프링 데이터 등
여러 프로젝트들의 모음을 의미한다.



스프링 생태계

스프링 부트

스프링 부트는 스프링 애플리케이션을 좀 더 쉽게 만들 수 있게 해준다
스프링의 초기 설정이 많아짐에 따라 스프링 부트가 탄생

스프링 프레임워크를 쉽게 사용하게 해주는 도구. 별개로 사용하는 것이 아님.

1. 의존성 관리
자주 사용하는 모듈 간의 의존성과 버전 조합을 자동으로 제공

2. 자동 설정
라이브러리 추가 등 설정 시 필요한 환경 설정 등을 간단하게 제공

3. 내장 WAS
Tomcat 등 WAS가 내장되어 따로 설치, 설정할 필요가 없다.

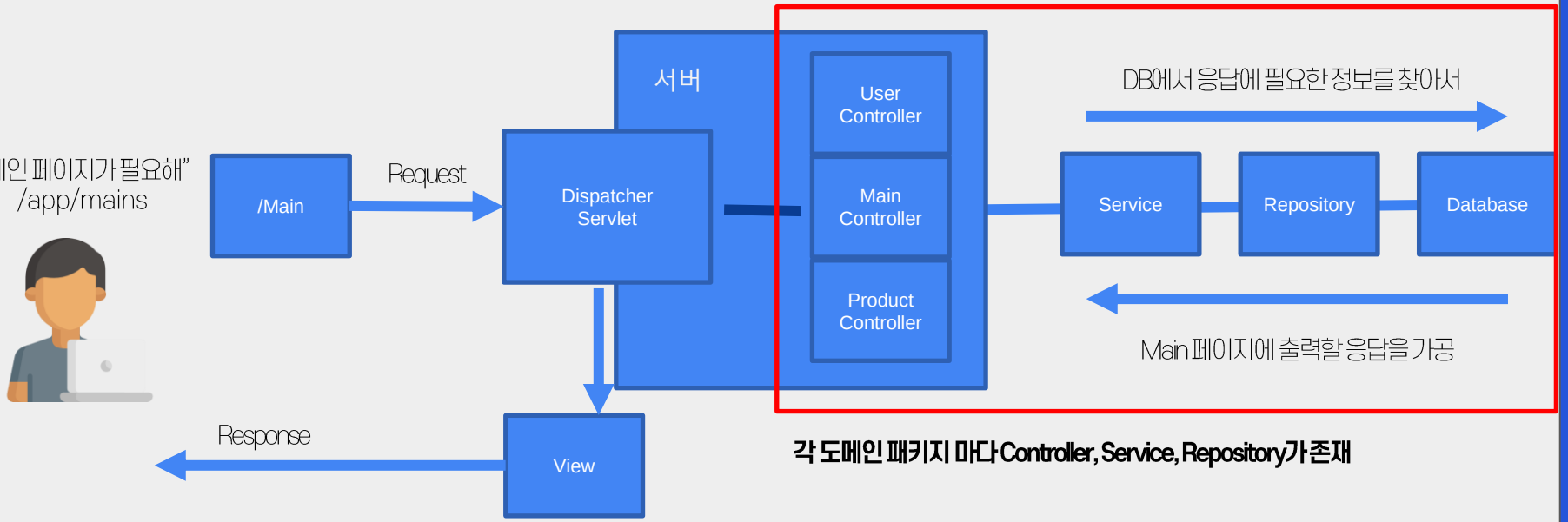
4. 모니터링 기능
스프링 부트 Actuator 모듈은 애플리케이션 관리 및 모니터링 지원

토론 키워드

Controller, Service, Dao

WAS 개발 큰 그림

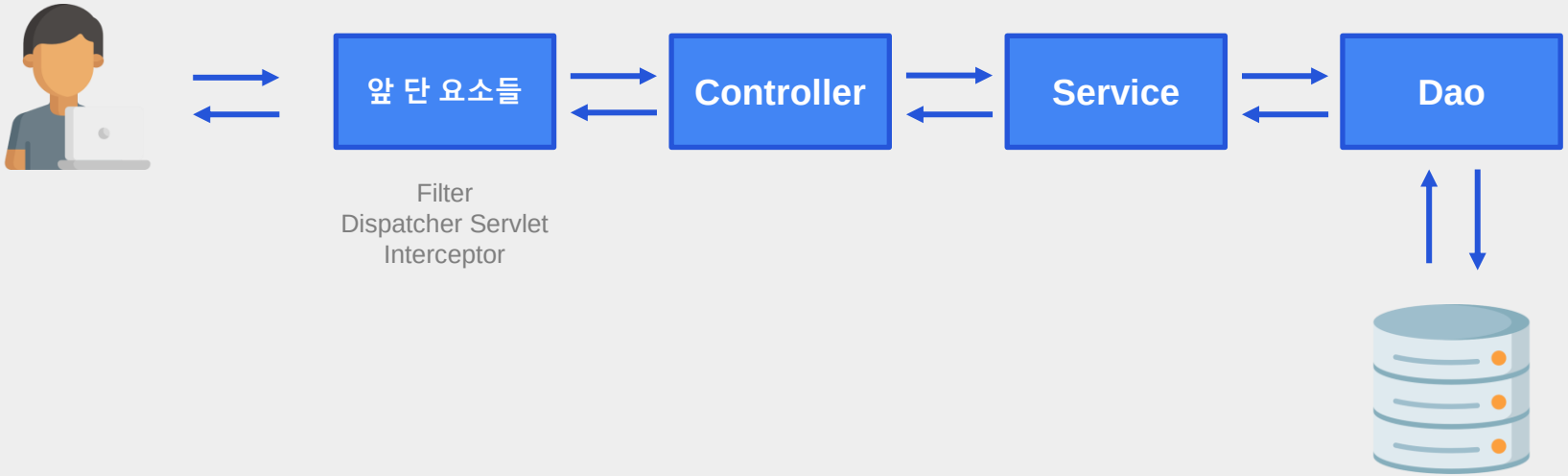
클라이언트가 특정 API를 요청하면,
해당 API에 맞는 응답을 처리해주는 Controller, Service, Repository(DAO)를 개발



request 처리

request 처리 프로세스

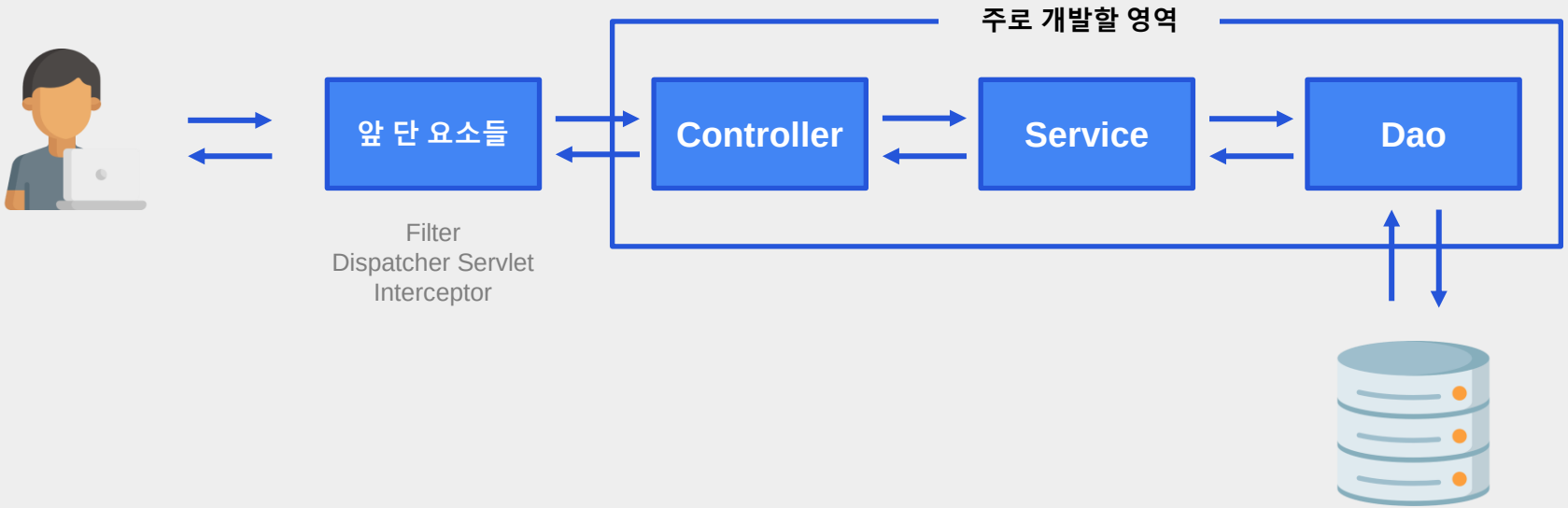
java 패키지 내에 각 도메인 패키지를 구현
각 도메인 패키지는 크게 model, controller, service, dao로 구성



request 처리

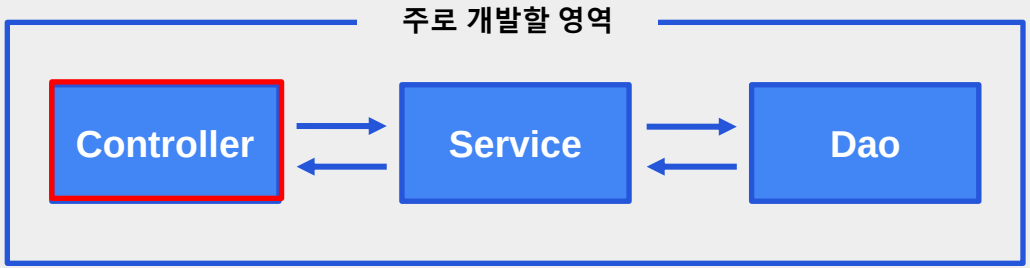
request 처리 프로세스

java 패키지 내에 각 도메인 패키지를 구현
각 도메인 패키지는 크게 model, controller, service, dao로 구성



Controller

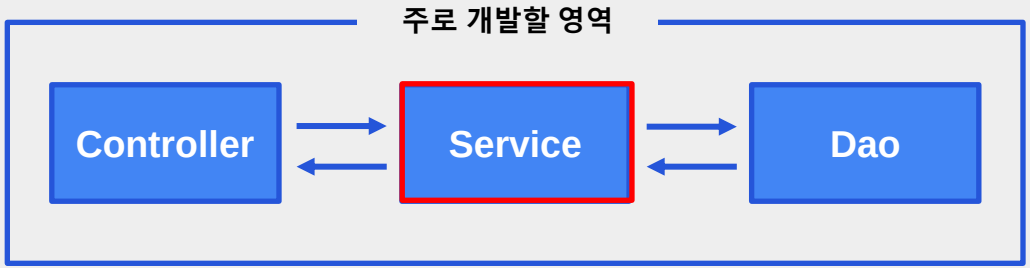
Controller



- Controller 관련 어노테이션
- Controller 메소드 구성(CRUD Mapping, 요청 & 응답 객체 등)
- 형식적 예외처리
- BaseResponse(HTTP 상태 코드)

Service

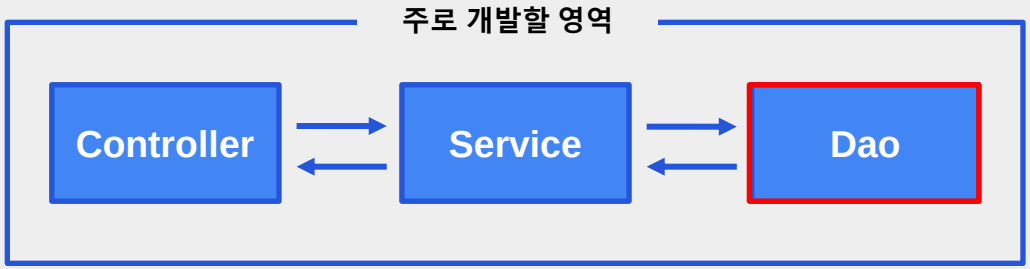
Service



- Service 관련 어노테이션 설명
- 논리적 예외처리

Dao

Dao



- Dao 관련 어노테이션 설명
  - JDBC Template

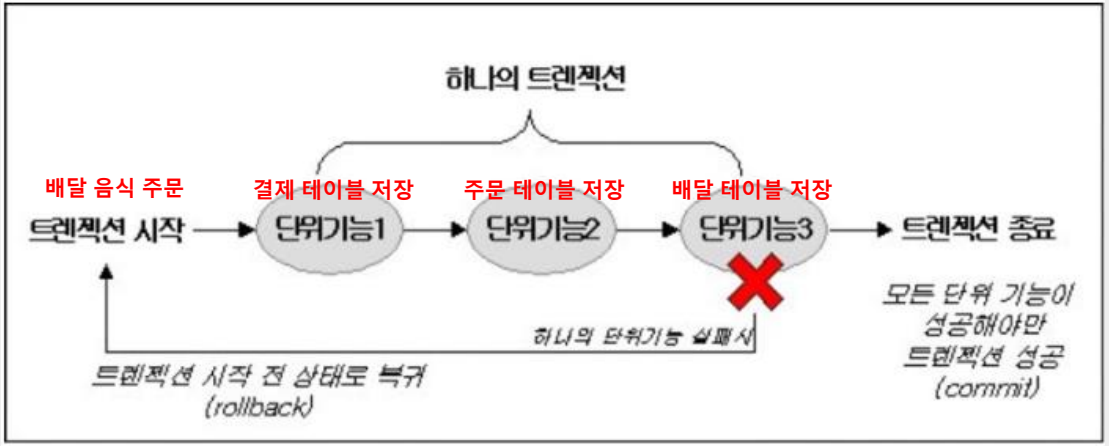
토론 키워드

Spring 트랜잭션

트랜잭션

트랜잭션

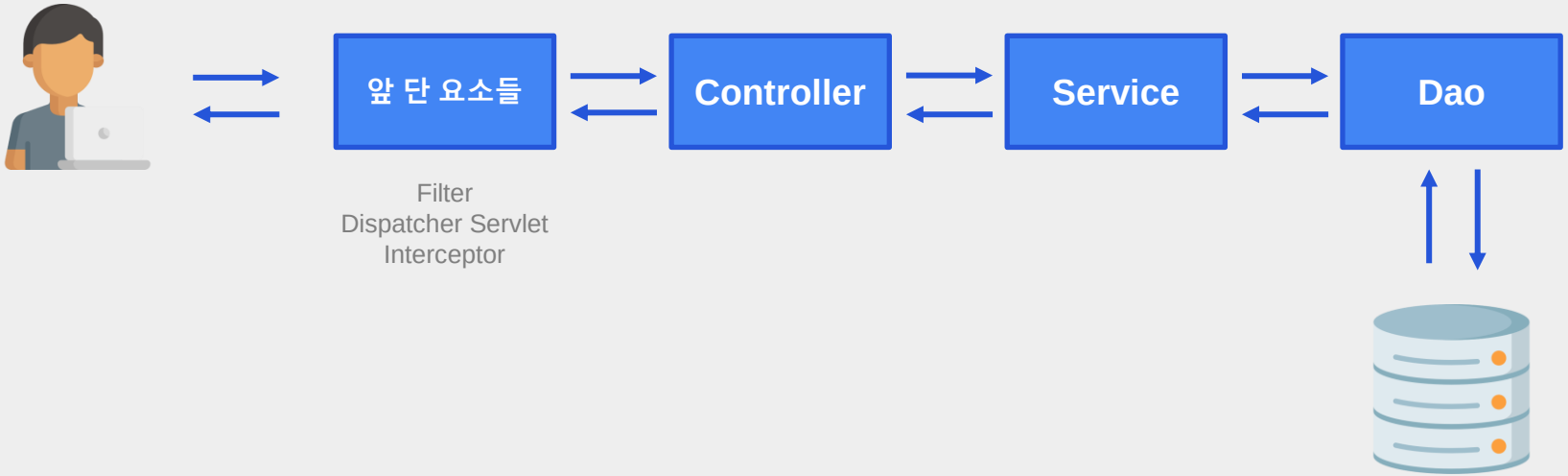
데이터베이스의 상태를 변환시키는 **하나의 논리적 기능**을 수행하기 위한 **작업의 단위**
트랜잭션은 **독립적**으로 이루어진다.



트랜잭션

Spring 트랜잭션

어디를 기준으로 트랜잭션 범위를 나누면 좋을까?



토론 키워드

Spring 예외처리

Bean Validation

Bean Validation

형식적인 예외처리를 쉽게 적용할 수 있게 표준화 시킨 기술
검증 어노테이션과 여러 인터페이스의 모음
일반적으로 하이버네이트 Validator 구현체를 사용한다.

형식적 예외처리(타입 예외 등)를 보다 쉽게 구현할 수 있다.

Bean Validation

Bean Validation 활용 예시

Bean Validation은 다음과 같이 활용할 수 있다.

DTO 코드

```java
@Data
public class Item {

    private Long id;

    @NotBlank
    private String itemName;

    @NotNull
    @Range(min = 1000, max = 1000000)
    private Integer price;

    @NotNull
    @Max(9999)
    private Integer quantity;
```

Controller 코드

```java
@PostMapping("/add")
public String addItemV6(@Validated @ModelAttribute Item item,
                BindingResult bindingResult, RedirectAttributes redirectAttributes, Model model)
```

Bean Validation

검증 어노테이션

Bean Validation은 기본적으로 다음과 같은 검증 어노테이션을 제공한다.

```
@Null  // null만 허용합니다.
@NotNull  // null을 허용하지 않습니다. "", " "는 허용합니다.
@NotEmpty  // null, ""을 허용하지 않습니다. " "는 허용합니다.
@NotBlank  // null, "", " " 모두 허용하지 않습니다.

@Email  // 이메일 형식을 검사합니다. 다만 ""의 경우를 통과 시킵니다
@Pattern(regexp = )  // 정규식을 검사할 때 사용됩니다.
@Size(min=, max=)  // 길이를 제한할 때 사용됩니다.

@Max(value = )  // value 이하의 값을 받을 때 사용됩니다.
@Min(value = )  // value 이상의 값을 받을 때 사용됩니다.

@Positive  // 값을 양수로 제한합니다.
@PositiveOrZero  // 값을 양수와 0만 가능하도록 제한합니다.

@Negative  // 값을 음수로 제한합니다.
@NegativeOrZero  // 값을 음수와 0만 가능하도록 제한합니다.

@Future  // 현재보다 미래
@Past  // 현재보다 과거

@AssertFalse  // false 여부, null은 체크하지 않습니다.
@AssertTrue  // true 여부, null은 체크하지 않습니다.
```

Bean Validation 실습

Bean Validation 실습

1. build.gradle에 validation 의존성 추가

```
implementation ('org.springframework.boot:spring-boot-starter-validation')
```

2. validation을 적용할 객체의 멤버변수에 어노테이션을 작성

3. REST API 매개변수 중 validation이 적용된 객체 @Validated 어노테이션 추가

4. 개별적인 예외처리가 필요한 경우 BindingResult을 활용

@ExceptionHandler

@ExceptionHandler

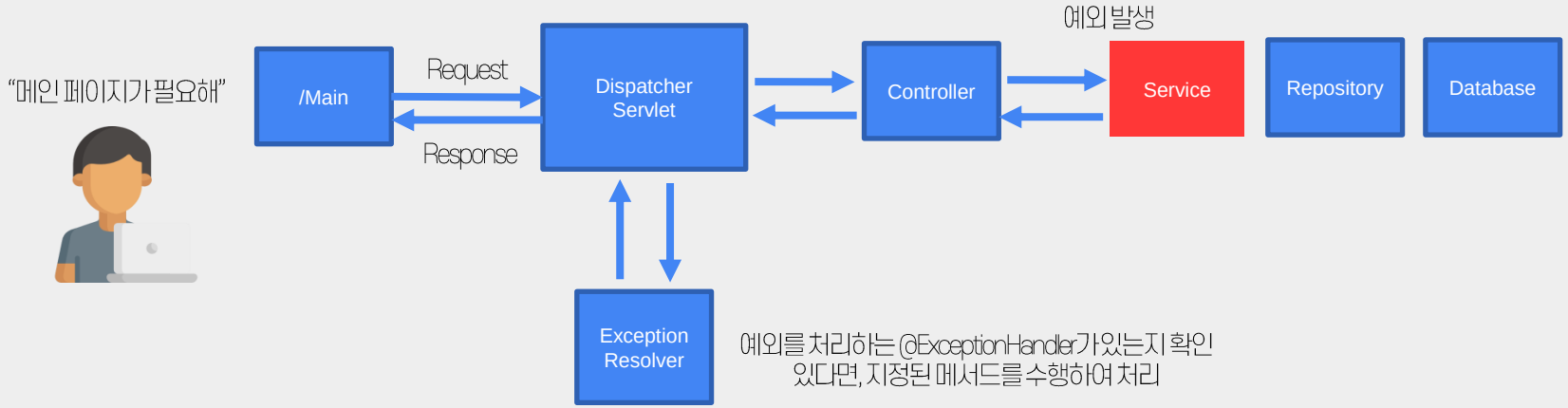
Spring에서는 @ExceptionHandler 어노테이션을 활용하여 손쉽게 예외처리를 할 수 있다.
잡아서 처리하고 싶은 예외를 어노테이션에 지정하고, 메서드에 관련 예외처리를 구현하는 형식

- @ExceptionHandler는 마치 스프링 컨트롤러처럼 다양한 파라미터와 응답을 지정할 수 있다.
- 부모 예외를 지정할 경우 자식 예외까지 함께 처리된다.
- @ResonseStatus 어노테이션을 활용하여 response status를 설정할 수 있다.

```
@ResponseStatus(HttpStatus.BAd_REQUEST) // response status 설정 가능
@ExceptionHandler(IllegalArgumentException.class)
public ErrorResult illegalExHandle(IllegalArgumentException e) {
  log.error("[exceptionHandle] ex", e);
  return new ErrorResult("BAD", e.getMessage());
}
```

@ExceptionHandler

@ExceptionHandler 예외처리 동작 원리



1. 예외 발생 시, 예외가 컨트롤러 밖으로 던져진다.
2. Spring 내부의 ExceptionResolver가 작동. 발생한 예외를 처리하는 @ExceptionHandler가 있는지 확인한다.
3. @ExceptionHanlder가 지정된 메서드를 수행한다.

@ControllerAdvice

@ControllerAdvice

@ControllerAdvice를 활용하여
흩어져 있는 에러를 한 곳에서 처리할 수 있다

@ExceptionHandler를 각 Class에서 분리하여
한 곳에서 처리할 수 있도록 제공

```java
@RestControllerAdvice
public class ExControllerAdvice {

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(Exception1.class)
    public ErrorResult exceptionHandle1(Exception1 e) {
        // exception1 예외처리 로직
        return new ErrorResult("BAD", e.getMessage());
    }

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(Exception2.class)
    public ErrorResult exceptionHandle1(Exception2 e) {
        // exception2 예외처리 로직
        return new ErrorResult("BAD", e.getMessage());
    }

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(Exception3.class)
    public ErrorResult exceptionHandle3(Exception1 e) {
        // exception3 예외처리 로직
        return new ErrorResult("BAD", e.getMessage());
    }
}
```

@ControllerAdvice

@ControllerAdvice 범위 지정

특정 범위를 지정하여 에러 처리를 할 수도 있다
대상을 지정하지 않으면 모든 컨트롤러에 적용(글로벌 적용)

```java
// Target all Controllers annotated with @RestController
@RestControllerAdvice(annotations = RestController.class)
public class ExampleAdvice1 {}

// Target all Controllers within specific packages
@RestControllerAdvice("org.example.controllers")
public class ExampleAdvice2 {}

// Target all Controllers assignable to specific classes
@RestControllerAdvice(assignableTypes = {ControllerInterface.class, AbstractController.class})
public class ExampleAdvice3 {}
```

Hello World!