

Spring Framework

웹 어플리케이션 개발

강경미(carami@nate.com) & 김성박(urstory@gmail.com)

김성박

- NHN 엔터테인먼트 Dooray개발실 수석
- (주)써니베일 대표이사
- T3Q(주) 기술이사
- 삼성 SDS 멀티캠퍼스 전임강사

무엇이 개발자를 괴롭히는가?

- 수천라인의 코드
- 고구마 줄기처럼 파도파도 끝이 없는 코드들
- 같은 기능의 개발
- 유지보수

개발자의 고민

- 좀 더 나은 코드
- 좀 더 나은 성능



로드 존슨

항상 프레임워크 기반으로 접근하라

2003년 *Expert One-on-One J2EE Design and Development* 출판
J2EE 설계와 개발의 모든 영역에 대한 개발 전략을 다룬 책
이 책에 포함된 3만 라인의 샘플 애플리케이션 예제에 포함된 프레임워크가
스프링 프레임워크의 기원

엔터프라이즈 어플리케이션을 만들기 위한 기반이 되는 코드 -
스프링의 기원

Spring Core

Spring 등장배경



Rod Johnson

EJB 개구려서 내가 이번에
‘EJB’ 안쓰고 개발하기 책 냈으니
용아들 한 번만 봐주삼~



Yann Caroff

○○~
EJB가 개겨울 같았으니까
이름을
‘봄’으로 하는게 어때?



Juergen Hoeller

오! 대박~ 이생퀴
잘 만들었는데?
이거 오픈소스로
함 만들어보까?

Spring Framework란?

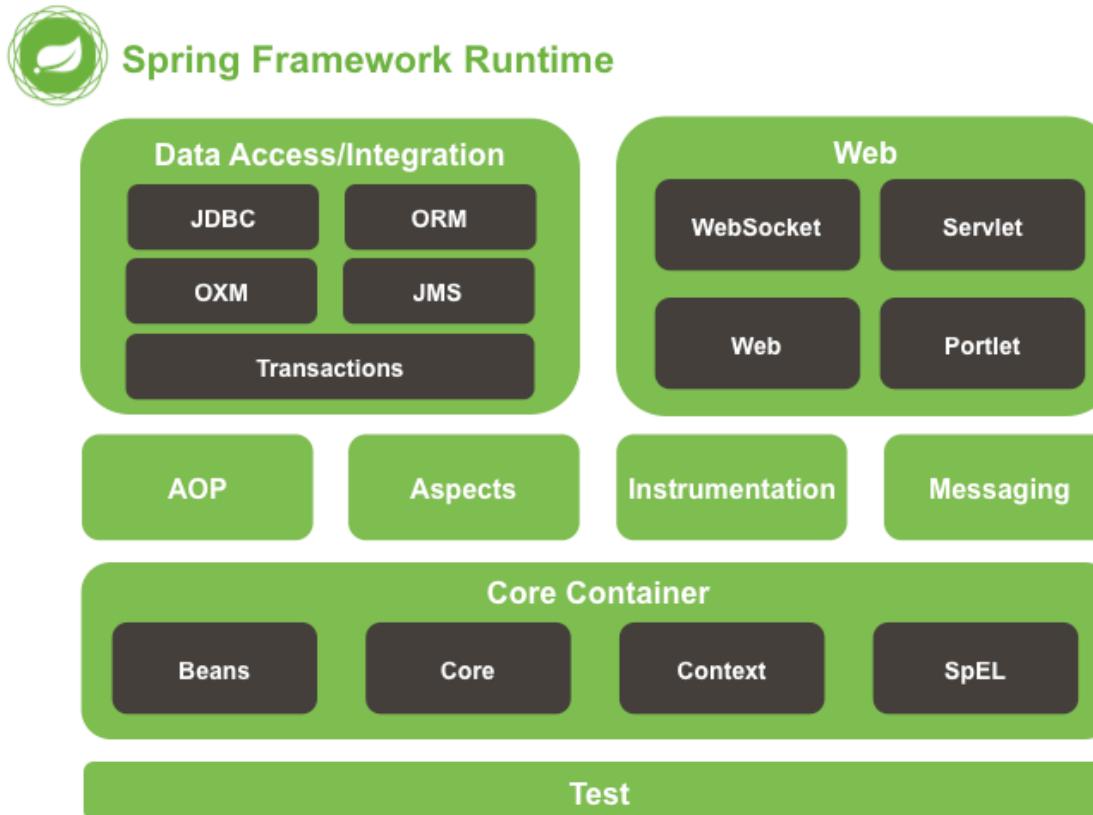
- 엔터프라이즈급 어플리케이션을 구축할 수 있는 가벼운 솔루션이자, 원스-스탑-숍(One-Stop-Shop)
- 원하는 부분만 가져다 사용할 수 있도록 모듈화가 잘 되어 있다.
- POJO를 이용한 가볍고 non-invasive(비 침투적) 개발
- IoC 컨테이너이다.
- DI와 인터페이스 지향을 통한 느슨한 결합도
- 선언적으로 트랜잭션을 관리할 수 있다.
- 완전한 기능을 갖춘 MVC Framework를 제공한다.
- AOP와 공통 규약을 통한 선언적 프로그래밍
- 템플릿을 통한 상투적인 코드 축소
- 스프링은 도메인 논리 코드와 쉽게 분리될 수 있는 구조를 가지고 있다.

* 원-스탑-숍 (One-Stop-Shop) : 모든 과정을 한꺼번에 해결하는 상점.

Release History

- 1.0 : 2004년 4월
- 2.0 : 2006년 6월
- 2.5 : 2007년 11월
- 3.0 : 2011년 12월
- 3.1.4 : 2013년 1월
- 4.0.1 : 2014년 1월
- 5.0 : 2017년 9월
- 2003년 6월 아파치 2.0 라이센스로 공개

프레임 워크 모듈



- 스프링 프레임워크는 약 20개의 모듈로 구성되어 있다.
- 필요한 모듈만 가져다 사용할 수 있다.

AOP 와 인스트루멘테이션 (Instrumentation)

- **spring-AOP** : AOP 얼라이언스(Aliance)와 호환되는 방법으로 AOP를 지원한다.
- **spring-aspects** : AspectJ와의 통합을 제공한다.
- **spring-instrument** : 인스트루멘테이션을 지원하는 클래스와 특정 WAS에서 사용하는 클래스로더 구현체를 제공한다. 참고로 BCI(Byte Code Instrumentations)은 런타임이나 로드(Load) 때 클래스의 바이트 코드에 변경을 가하는 방법을 말합니다.

메시징(Messaging)

- spring-messaging : 스프링 프레임워크 4는 메시지 기반 어플리케이션을 작성할 수 있는 Message, MessageChannel, MessageHandler 등을 제공한다. 또한, 해당 모듈에는 메소드에 메시지를 맵핑하기 위한 어노테이션도 포함되어 있으며, Spring MVC 어노테이션과 유사하다.

데이터 엑세스(Data Access) / 통합 (Integration)

- 데이터 엑세스/통합 계층은 JDBC, ORM, OXM, JMS 및 트랜잭션 모듈로 구성되어 있다.
- **spring-jdbc** : 자바 JDBC프로그래밍을 쉽게 할 수 있도록 기능을 제공한다.
- **spring-tx** : 선언적 트랜잭션 관리를 할 수 있는 기능을 제공한다.
- **spring-orm** : JPA, JDO및 Hibernate를 포함한 ORM API를 위한 통합 레이어를 제공한다.
- **spring-oxm** : JAXB, Castor, XMLBeans, JiBX 및 XStream과 같은 Object/XML 맵핑을 지원한다.
- **spring-jms** : 메시지 생성(producing) 및 사용(consuming)을 위한 기능을 제공. Spring Framework 4.1 부터 **spring-messaging**모듈과의 통합을 제공한다.

웹 (Web)

- 웹 계층은 spring-web, spring-webmvc, spring-websocket, spring-webmvc-portlet 모듈로 구성된다.
- **spring-web** : 멀티 파트 파일 업로드, 서블릿 리스너 등 웹 지향 통합 기능을 제공한다. HTTP클라이언트와 Spring의 원격 지원을 위한 웹 관련 부분을 제공한다.
- **spring-webmvc** : Web-Servlet모듈이라고도 불리며, Spring MVC및 REST 웹 서비스 구현을 포함한다.
- **spring-websocket** : 웹 소켓을 지원한다.
- **spring-webmvc-portlet** : 포틀릿 환경에서 사용할 MVC구현을 제공한다.

Spring Framework sub-projects

- Spring IO Platform – 스프링 프레임워크 의존성 관리
- Spring Boot
- Spring Framework
- Spring Cloud Data Flow – Big Data
- Spring Cloud – 클라우드 기반
- Spring Data – JPA, MongoDB, Redis, Elasticsearch ...
- Spring Integration – Enterprise Integration Pattern
- Spring Batch
- Spring Security
- Spring HATEOAS
- Spring Social
- Spring AMQP
- Spring Mobile

컨테이너(Container)란?

- 컨테이너는 인스턴스의 생명주기를 관리한다.
- 생성된 인스턴스들에게 추가적인 기능을 제공한다.

IoC란?

- IoC란 Inversion of Control의 약어이다. inversion은 사전적 의미로는 '도치, 역전'이다. 보통 IoC를 제어의 역전이라고 번역한다.
- 개발자는 프로그램의 흐름을 제어하는 코드를 작성한다. 그런데, 이 흐름의 제어를 개발자가 하는 것이 아니라 다른 프로그램이 그 흐름을 제어하는 것을 IoC라고 말한다.

POJO

- POJO = Plain Old Java Object
- <https://www.martinfowler.com/bliki/POJO.html>

DI란?

- DI는 Dependency Injection의 약자로, 의존성 주입이란 뜻을 가지고 있다.
- DI는 클래스 사이의 의존 관계를 빈(Bean) 설정 정보를 바탕으로 컨테이너가 자동으로 연결해주는 것을 말한다.

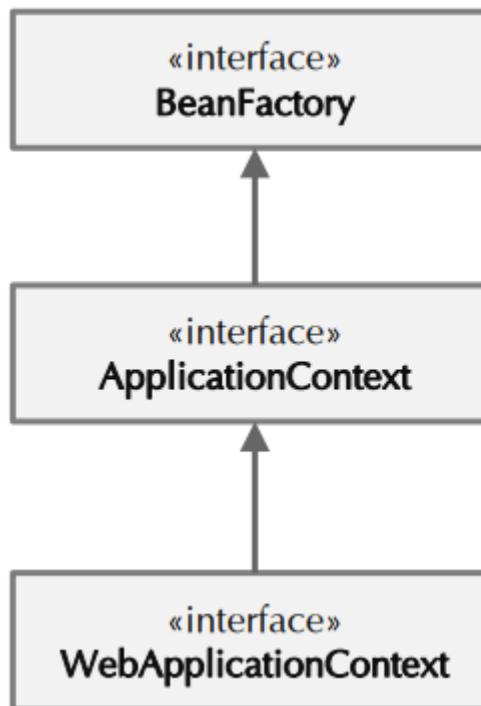
DI 예

- DI가 적용 안 된 예.
- 개발자가 직접 인스턴스를 생성한다.
- Spring에서 DI가 적용된 예.
- 엔진 type의 v5변수에 아직 인스턴스가 할당되지 않았다.
- 컨테이너가 v5변수에 인스턴스를 할당해주게 된다.

```
class 엔진 {  
}  
  
class 자동차{  
    엔진 v5 = new 엔진()  
}  
}
```

```
@Component  
  
class 엔진{  
}  
  
@Component  
  
class 자동차{  
    @Autowired  
    엔진 v5;  
}  
}
```

IoC 컨테이너



- 빈(bean) 객체에 대한 생성과 제공을 담당
 - 단일 유형의 객체를 생성하는 것이 아니라, 여러 유형의 빈(bean)을 생성, 제공
 - 객체 간의 연관 관계를 설정, 클라이언트의 요청 시 빈을 생성
 - 빈의 라이프 사이클을 관리
-
- BeanFactory가 제공하는 모든 기능 제공
 - 엔터프라이즈 애플리케이션을 개발하는데 필요한 여러 기능을 추가함
 - I18N, 리소스 로딩, 이벤트 발생 및 통지
 - 컨테이너 생성 시 모든 빈 정보를 메모리에 로딩함
-
- 웹 환경에서 사용할 때 필요한 기능이 추가된 애플리케이션 컨텍스트
 - 가장 많이 사용하며, 특히 XmlWebApplicationContext를 가장 많이 사용

ApplicationContext

- AnnotationConfigApplicationContext – 하나 이상의 Java Config 클래스에서 스프링 애플리케이션 컨텍스트를 로딩
- AnnotationConfigWebApplicationContext – 하나 이상의 Java Config 클래스에서 웹 애플리케이션 컨텍스트를 로딩
- ClassPathXmlApplicationContext – 클래스패스에 위치한 xml파일에서 컨텍스트를 로딩
- FileSystemXmlApplicationContext – 파일 시스템에서 지정된 xml파일에서 컨텍스트를 로딩
- XmlWebApplicationContext – 웹 애플리케이션에 포함된 xml파일에서 컨텍스트를 로딩

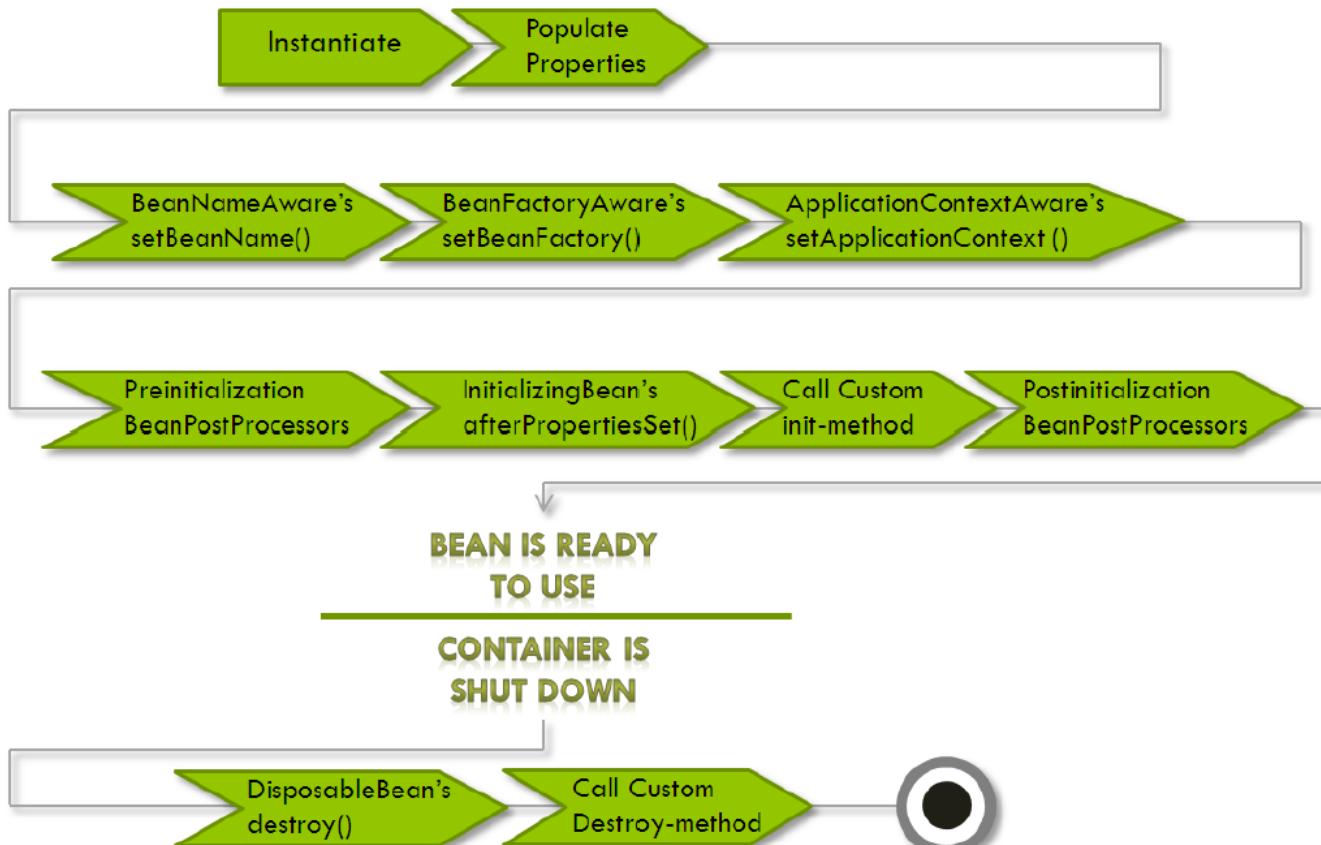
사용법

```
ApplicationContext context = new  
FileSystemXmlApplicationContext("/spring/context.xml");
```

```
ApplicationContext context = new  
ClassPathXmlApplicationContext("context.xml");
```

```
ApplicationContext context = new  
AnnotationConfigApplicationContext(Config.class);
```

Bean의 일생



1. 스프링이 빈을 인스턴스화 한다.
2. 스프링이 값과 빈의 레퍼런스를 빈의 프로퍼티로 주입한다.
3. 빈이 BeanNameAware를 구현하면 스프링이 빈의 ID를 setBeanName() 메소드에 넘긴다.
4. 빈이 BeanFactoryAware를 구현하면 setBeanFactory() 메소드를 호출하여 빈 팩토리 전체를 넘긴다.
5. 빈이 ApplicationContextAware를 구현하면 스프링이 setApplicationContext() 메소드를 호출하고 둘러싼 애플리케이션 컨텍스트에 대한 참조를 넘긴다.
6. 빈이 Bean PostProcessor 인터페이스를 구현하면 스프링은 postProcessBeforeInitialization() 메소드를 호출한다.
7. 빈이 InitializingBean 인터페이스를 구현하면 스프링은 afterPropertiesSet() 메소드를 호출한다. 마찬가지로 빈이 init-method와 함께 선언됐으면 지정한 초기화 메소드가 호출된다.
8. 빈이 BeanPostProcessor를 구현하면 스프링은 postProcessAfterInitialization() 메소드를 호출한다.
9. 빈은 애플리케이션에서 사용할 준비가 된 것이며, 애플리케이션 컨텍스트가 소멸될 때까지 애플리케이션 컨텍스트에 남아 있게 된다.
10. 빈이 DisposableBean 인터페이스를 구현하면 스프링은 destroy() 메소드를 호출한다. 마찬가지로 빈이 destroy-method와 함께 선언됐으면 지정된 메소드가 호출된다.

생명주기 관련 애노테이션

- `@PostConstruct` - JSR-250 스펙으로 JSR-250을 구현하고 있는 다른 프레임워크에서도 사용 가능하다. 인스턴스 생성 후에 호출된다.
- `@Bean(initMethod)` - `@Bean(initMethod="init")` 과 같은 형태로 사용함으로써 초기화 메소드를 사용할 수 있다. 아래는 Java Config에서의 사용 예이다.

```
@Bean(initMethod="init")  
  
public MyBean mybean() {  
    return new MyBean();  
}
```

- `@PreDestroy` - JSR-250 스펙에서 정의되어 있으며 종료될 때 사용할 메소드 위에 사용하면 된다.
- `@Bean(destroyMethod)` - `@Bean(destroyMethod="destroy")` 와 같은 형태로 사용함으로써 종료될 때 호출되도록 할 수 있다.

스프링 버전별 새로운 기능

- <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/new-in-3.0.html>
- <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/new-in-3.1.html>
- <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/new-in-3.2.html>

스프링 버전별 새로운 기능

스프링 프레임워크 4.0의 새로운 기능 및 향상된 기능

1. 향상된 시작경험
2. Deprecated된 패키지 및 메서드 제거
3. Java 8 지원
4. Java EE 6, 7 지원
5. Groovy 빈 정의 DSL
6. 코어 컨테이너 개선
7. 전반적인 웹 개선
8. 웹소켓, SockJS, STOMP 메시징
9. 테스트 개선

스프링 버전별 새로운 기능

스프링 프레임워크 4.1의 새로운 기능 및 향상된 기능

1. JMS 개선
2. 캐싱 개선
3. 웹 개선
4. 웹소켓 메시징 개선
5. 테스트 개선

스프링 버전별 새로운 기능

스프링 프레임워크 4.2의 새로운 기능 및 향상된 기능

1. 코어 컨테이너 개선
2. 데이터 액세스 개선
3. JMS 개선
4. 웹 개선
5. 웹소켓 메시징 개선
6. 테스트 개선

스프링 버전별 새로운 기능

스프링 프레임워크 4.3의 새로운 기능 및 향상된 기능

1. 코어 컨테이너 개선
2. 데이터 엑세스 개선
3. 캐싱 개선
4. JMS 개선
5. 웹 개선
6. 웹소켓 메시징 개선
7. 테스트 개선
8. 새로운 라이브러리 및 서버 세대 지원

스프링 버전별 새로운 기능

1. JDK 8+9와 Java EE 7 베이스라인
2. 패키지, 클래스, 메서드 제거
3. 코어 컨테이너 개선
4. 일반적인 웹 개선
5. 리액티브(Reactive) 프로그래밍 모델
6. 테스트 개선

Maven

Maven

- Maven은 지금까지 애플리케이션을 개발하기 위한 반복적으로 진행해 왔던 작업들을 지원하기 위하여 등장한 도구이다.
- Maven을 사용하면 빌드(Build), 패키징, 문서화, 테스트와 테스트 리포팅, git, 의존성관리, svn등과 같은 형상 관리서버와 연동(SCMs), 배포 등의 작업을 손쉽게 할 수 있다.
- Maven을 이해하려면 CoC(Convention over Configuration)라는 단어를 먼저 이해해야 한다.CoC란 일종의 관습을 말하는데, 예를 들자면 프로그램의 소스파일은 어떤 위치에 있어야 하고, 소스가 컴파일된 파일들은 어떤 위치에 있어야 하고 등을 미리 정해놨다는 것이다.이 말은 관습에 이미 익숙한 사용자는 쉽게 Maven을 사용할 수 있는데, 관습에 익숙하지 않은 사용자는 이러한 제약사항에 대해서 심한 거부감을 느낄 수 있다.Maven을 사용한다는 것은 어쩌면 이러한 관습 즉 CoC에 대해서 알아나가는 것이라고도 말할 수 있다.

Maven을 사용할 경우 얻게 되는 이점

- Maven을 사용할 경우, 굉장히 편리한 점들이 많다.
- 많은 사람들이 손꼽는 장점중에는 편리한 의존성 라이브러리 관리가 있다. 앞에서 JSTL을 학습할 때, 몇가지 파일을 다운로드 하여 /WEB-INF/lib폴더에 복사하여 사용했었다.
- 관련된 라이브러리가 많아질 수록 이러한 방식은 상당히 불편해진다. Maven을 사용하면 설정파일에 몇줄 적어줌으로써 직접 다운로드 받거나 하는 것을 하지 않아도 라이브러리를 사용할 수 있다.
- 프로젝트에 참여하는 개발자가 많아지게 되면, 프로젝트를 빌드하는 방법에 대하여 가이드하는 것도 쉬운일이 아니다. Maven을 사용하게 되면 Maven에 설정한 대로 모든 개발자가 일관된 방식으로 빌드를 수행할 수 있게 된다.
- Maven은 또한 다양한 플러그인을 제공해줘서, 굉장히 많은 일들을 자동화 시킬 수 있다.

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>kr.co.sunnyvale</groupId>
  <artifactId>examples</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>mysample</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

project : pom.xml 파일의 최상위 루트 엘리먼트(Root Element)이다.

modelVersion : POM model의 버전이다.

groupId : 프로젝트를 생성하는 조직의 고유 아이디를 결정한다. 일반적으로 도메인 이름을 거꾸로 적는다.

artifactId : 해당 프로젝트에 의하여 생성되는 artifact의 고유 아이디를 결정한다. Maven을 이용하여 pom.xml을 빌드할 경우 다음과 같은 규칙으로 artifact가 생성된다. artifactid-version.packaging. 위 예의 경우 빌드할 경우 mysamexamples-1.0-SNAPSHOT.jar 파일이 생성된다.

packaging : 해당 프로젝트를 어떤 형태로 packaging 할 것인지 결정한다. jar, war, ear 등이 해당된다.

version : 프로젝트의 현재 버전. 추후 살펴보겠지만 프로젝트가 개발 중일 때는 SNAPSHOT을 절미사로 사용한다. Maven의 버전 관리 기능은 라이브러리 관리를 편하게 한다.

name : 프로젝트의 이름이다.

url : 프로젝트 사이트가 있다면 사이트 URL을 등록하는 것이 가능하다.

Maven 을 이용할 경우 얻게 되는 큰 이점 중의 하나는 Dependency Management 기능이다. 위 pom.xml 파일에서 <dependencies/> 엘리먼트가 Dependency Management 기능의 핵심이라고 할 수 있다. 해당 엘리먼트 안에 필요한 라이브러리를 지정하게 된다.

Spring 실습

Spring 설정

- xml 파일을 이용한 명시적 설정
- Java Config를 이용한 명시적 설정
- Component scan을 이용해서 빈을 찾아 자동으로 DI하기

@ComponentScan

- package가 지정되지 않으면 Java Config파일이 있는 패키지부터 하위 패키지까지 검색을 한다.
- @Component , @Controller, @Service, @Repository 애노테이션이 붙은 객체를 찾고, 자동으로 DI한다.
- xml파일로 설정할 때는 <context:component-scan>을 사용한다.

Spring JDBC

Spring에서 제공하는 IoC/DI 컨테이너

- BeanFactory : IoC/DI에 대한 기본 기능을 가지고 있다.
- ApplicationContext : BeanFactory의 모든 기능을 포함하며, 일반적으로 BeanFactory보다 추천된다. 트랜잭션처리, AOP등에 대한 처리를 할 수 있다. BeanPostProcessor, BeanFactoryPostProcessor등을 자동으로 등록하고, 국제화 처리, 어플리케이션 이벤트 등을 처리할 수 있다.

Spring JDBC

- JDBC 프로그래밍을 보면 반복되는 개발 요소가 있다.
- 이러한 반복적인 요소는 개발자를 지루하게 만든다.
- 개발하기 지루한 JDBC의 모든 저수준 세부사항을 스프링 프레임워크가 처리해준다.
- 개발자는 필요한 부분만 개발하면 된다.

Spring JDBC – 개발자가 해야 할 일은?

동작	스프링	어플리케이션 개발자
연결 파라미터 정의.		O
연결 오픈.	O	
SQL 문 지정.		O
파라미터 선언과 파라미터 값 제공		O
스테이트먼트(statement) 준비와 실행.	O	
(존재한다면) 결과를 반복하는 루프 설정	O	
각 이터레이션에 대한 작업 수행.		O
모든 예외 처리.	O	
트랜잭션 제어.	O	
연결, 스테이트먼트(statement), 리절트셋(resultset) 닫기.	O	

Spring JDBC 패키지

- org.springframework.jdbc.core
- org.springframework.jdbc.datasource
- org.springframework.jdbc.object
- org.springframework.jdbc.support

JdbcTemplate

- org.springframework.jdbc.core에서 가장 중요한 클래스
- 리소스 생성, 해지를 처리해서 연결을 닫는 것을 잊어 발생하는 문제등을 피할 수 있도록 한다.
- 스테이먼트(Statement)의 생성과 실행을 처리한다.
- SQL조회, 업데이트, 저장 프로시저 호출, ResultSet 반복호출 등을 실행한다.
- JDBC예외가 발생할 경우 org.springframework.dao패키지에 정의되어 있는 일반적인 예외로 변환시킨다.

JdbcTemplate select 예제 1

- 열의 수 구하기

```
int rowCount = this.jdbcTemplate.queryForInt("select count(*)  
from t_actor");
```

JdbcTemplate select 예제 2

- 변수 바인딩 사용하기

```
int countOfActorsNamedJoe =  
    this.jdbcTemplate.queryForInt("select count(*) from t_actor  
    where first_name = ?", "Joe");
```

JdbcTemplate select 예제 3

- String값으로 결과 받기

```
String lastName = this.jdbcTemplate.queryForObject("select  
last_name from t_actor where id = ?", new Object[] {1212L},  
String.class);
```

JdbcTemplate select 예제 4

- 한 건 조회하기

```
Actor actor = this.jdbcTemplate.queryForObject(  
    "select first_name, last_name from t_actor where id = ?",
    new Object[] {1212L},
    new RowMapper<Actor>() {
        public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {
            Actor actor = new Actor();
            actor.setFirstName(rs.getString("first_name"));
            actor.setLastName(rs.getString("last_name"));
            return actor;
        }
    });
}
```

JdbcTemplate select 예제 5

- 여러 건 조회하기

```
List<Actor> actors = this.jdbcTemplate.query(  
    "select first_name, last_name from t_actor",  
    new RowMapper<Actor>() {  
        public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {  
            Actor actor = new Actor();  
            actor.setFirstName(rs.getString("first_name"));  
            actor.setLastName(rs.getString("last_name"));  
            return actor;  
        }  
    });
```

JdbcTemplate select 예제 6

- 중복 코드 제거 (1건 구하기와 여러건 구하기가 같은 코드에 있을 경우)

```
public List<Actor> findAllActors() {  
  
    return this.jdbcTemplate.query( "select first_name, last_name from t_actor", new ActorMapper());  
  
}  
  
private static final class ActorMapper implements RowMapper<Actor> {  
  
    public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {  
  
        Actor actor = new Actor();  
  
        actor.setFirstName(rs.getString("first_name"));  
  
        actor.setLastName(rs.getString("last_name"));  
  
        return actor;  
  
    }  
}
```

JdbcTemplate insert 예제

- INSERT 하기

```
this.jdbcTemplate.update("insert into t_actor (first_name,  
last_name) values (?, ?)", "Leonor", "Watling");
```

JdbcTemplate update 예제

- UPDATE 하기

```
this.jdbcTemplate.update("update t_actor set = ? where id = ?",
"Banjo", 5276L);
```

JdbcTemplate delete 예제

- DELETE 하기

```
this.jdbcTemplate.update("delete from actor where id = ?",
Long.valueOf(actorId));
```

JdbcTemplate 외의 접근방법

- NamedParameterJdbcTemplate
- SimpleJdbcTemplate
- SimpleJdbcInsert

JPA

참고 -

<http://projects.spring.io/spring-data/>

http://www.querydsl.com/static/querydsl/4.0.1/reference/ko-KR/html_single/

자바 ORM 표준 JPA 프로그래밍 - 저자 : 김영한

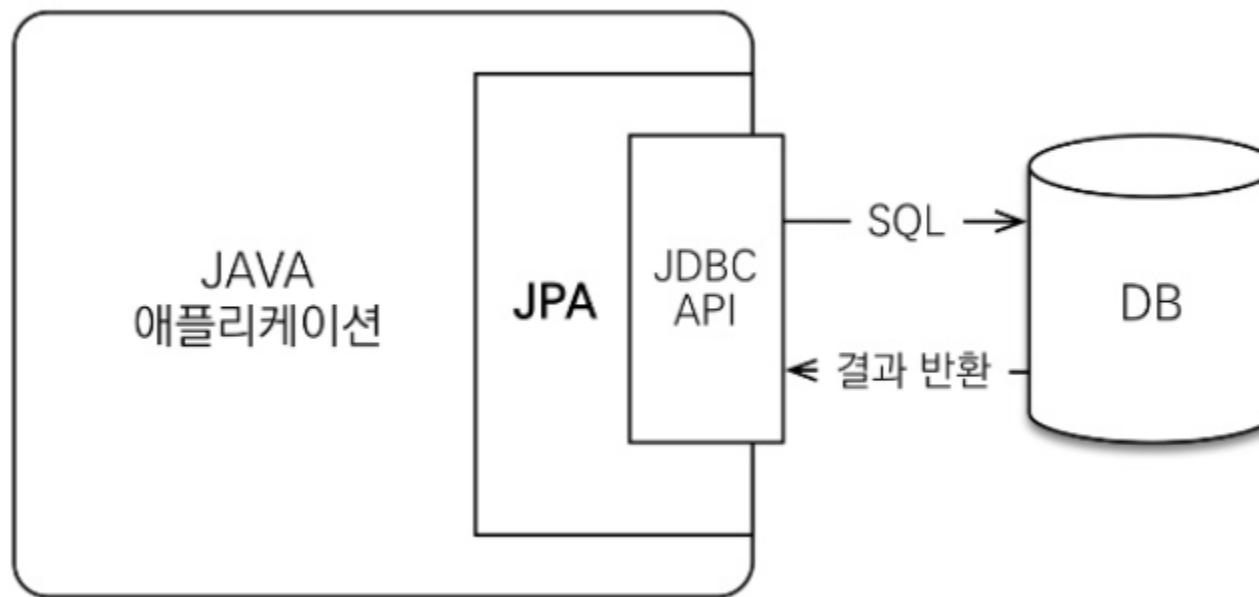
JPA?

- Java Persistence API
- 자바 진영의 ORM 기술 표준

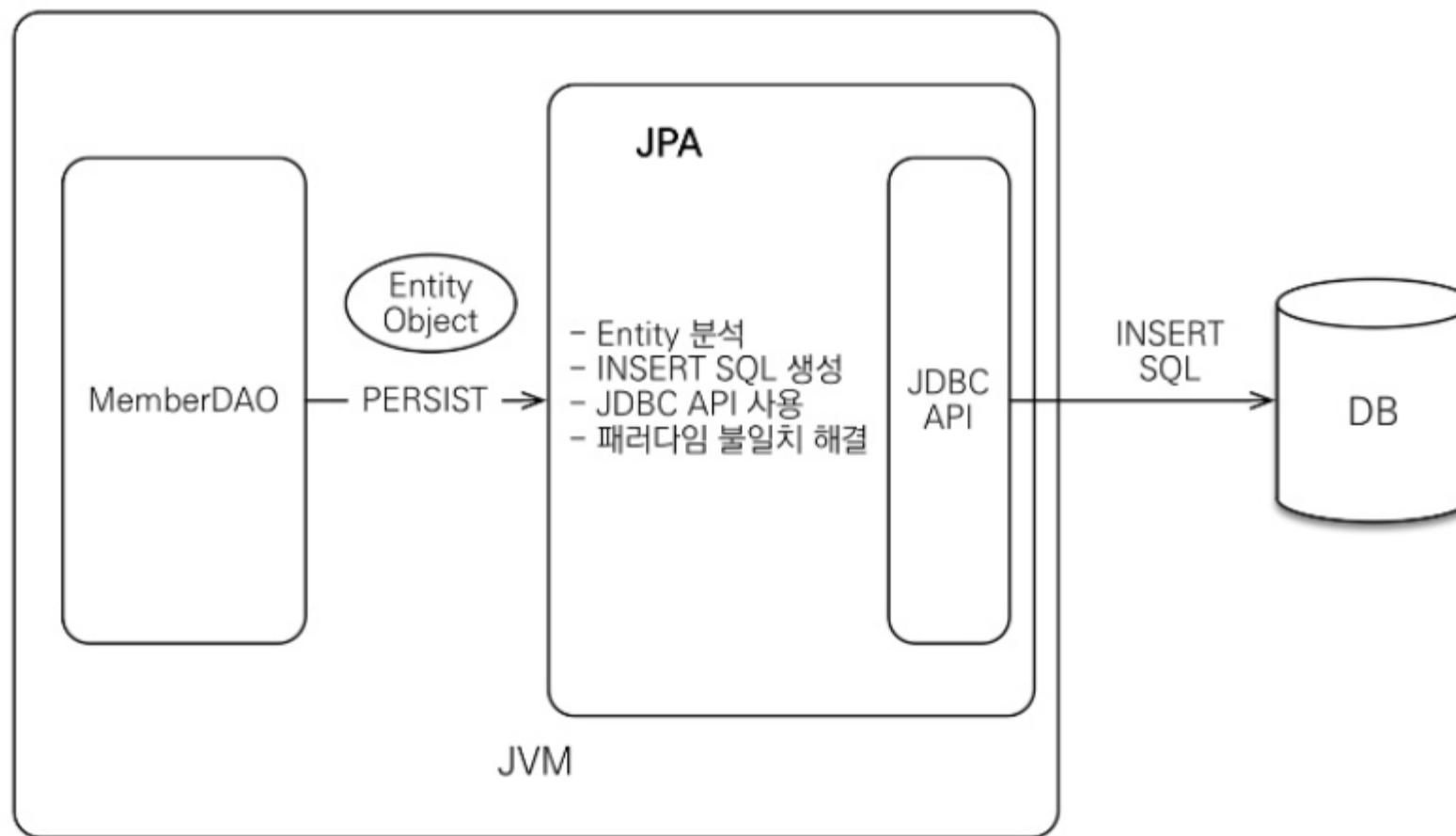
ORM?

- Object-relational mapping
- 객체 관계 매핑
- 객체는 객체대로 설계
- 관계형 데이터베이스는 관계형 데이터베이스대로 설계
- ORM 프레임워크가 중간에서 매핑
- 대중적인 언어에는 대부분 ORM 기술이 존재

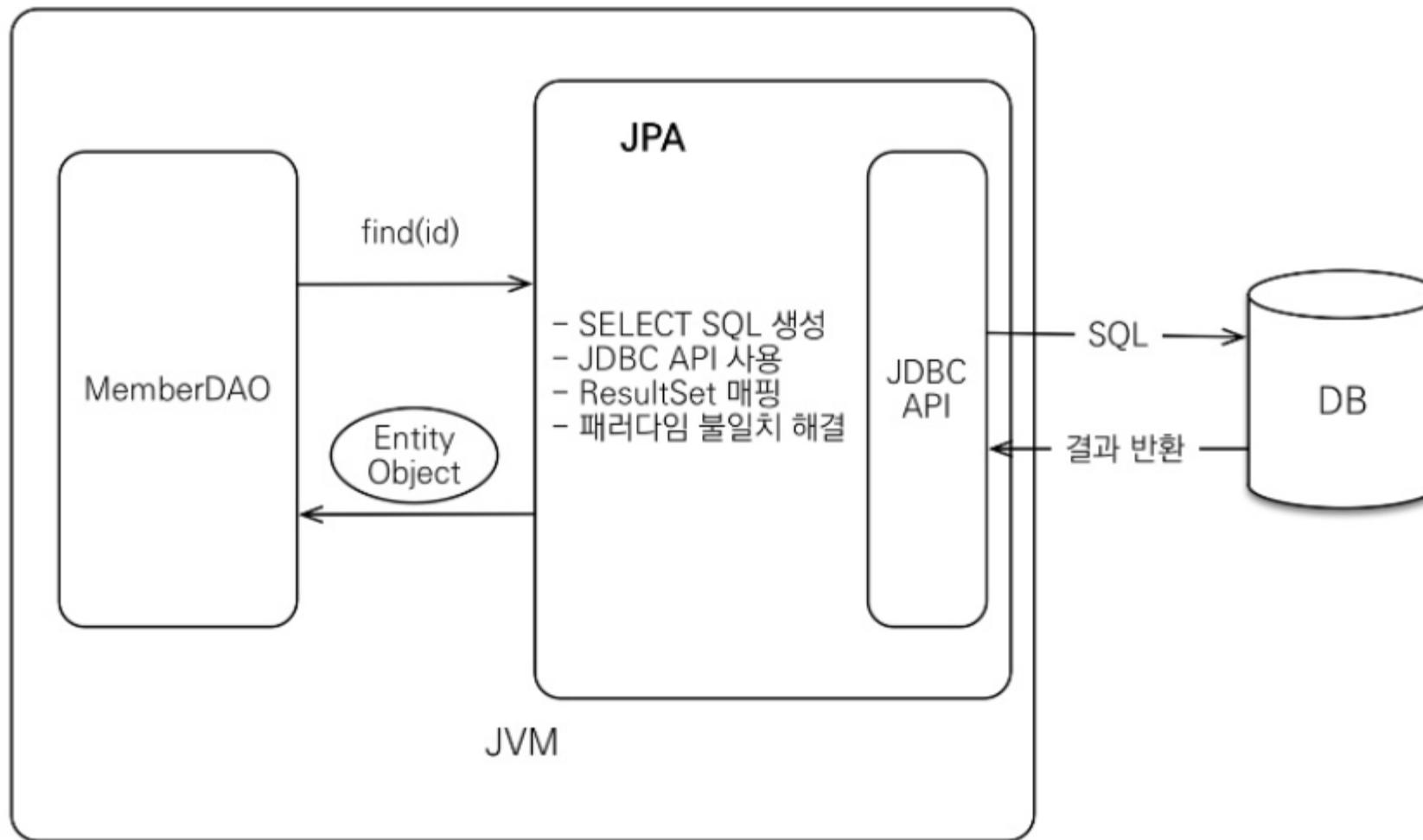
JPA는 애플리케이션과 JDBC사이에서 동작



JPA 동작 - 저장



JPA동작 - 조회



JPA 소개

하이버네이트(오픈 소스)

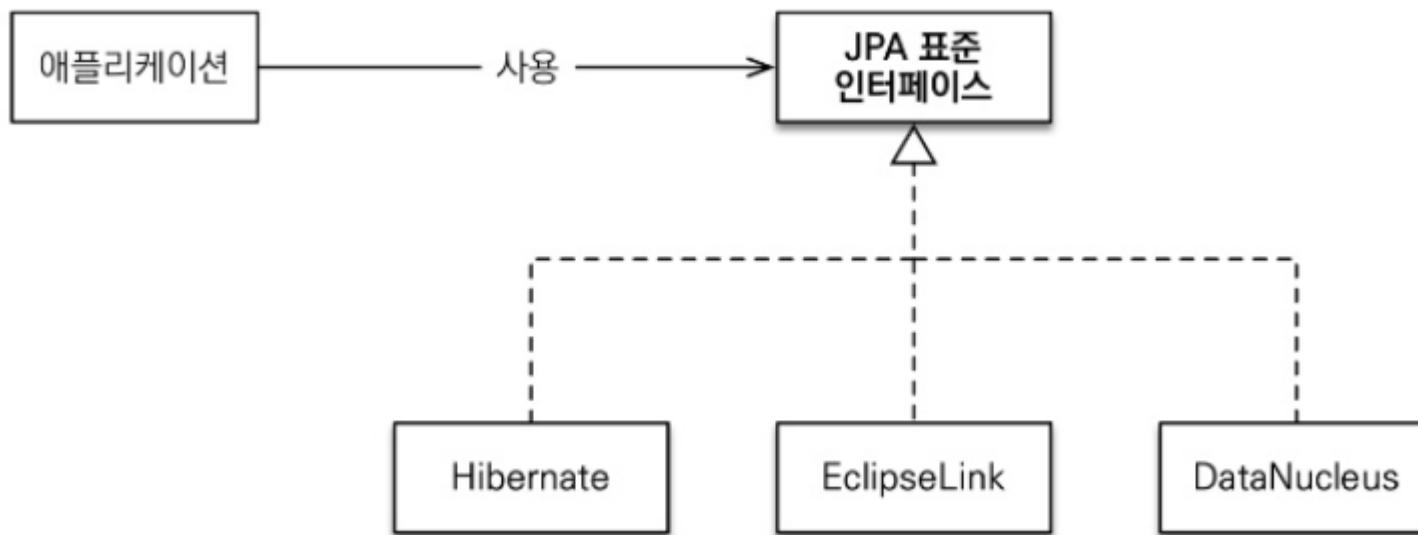


EJB - 엔티티 빙(자바 표준)

JPA(자바 표준)

JPA는 표준 명세

- JPA는 인터페이스의 모음
- JPA 2.1 표준 명세를 구현한 3가지 구현체
- 하이버네이트, EclipseLink, DataNucleus



JPA를 사용하는 이유

- SQL 중심적인 개발에서 객체 중심으로 개발
- 생산성
- 유지보수
- 패러다임의 불일치 해결
- 성능
- 데이터 접근 추상화와 벤더 독립성
- 표준

생산성 – JPA와 CRUD

- jpa.persist(member); //저장
- Member member = jpa.find(memberId); // 조회
- member.setName("변경할 이름"); // 수정
- jpa.remove(member); // 삭제

유지보수 - 필드 변경시 모든 SQL 수정

```
public class Member {
```

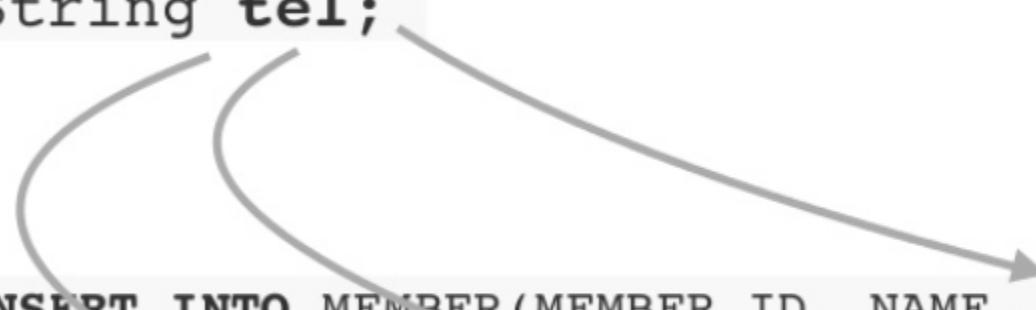
```
    private String memberId;  
    private String name;  
    private String tel;
```

```
    ...  
}
```

```
INSERT INTO MEMBER(MEMBER_ID, NAME, TEL) VALUES ...
```

```
SELECT MEMBER_ID, NAME, TEL FROM MEMBER M
```

```
UPDATE MEMBER SET ... TEL=?
```



유지보수 - 필드만 추가하면 됨, SQL은 JPA가 알아서 처리

```
public class Member {  
  
    private String memberId;  
    private String name;  
    private String tel;  
    ...  
}
```

~~INSERT INTO MEMBER(MEMBER_ID, NAME, TEL) VALUES ...~~

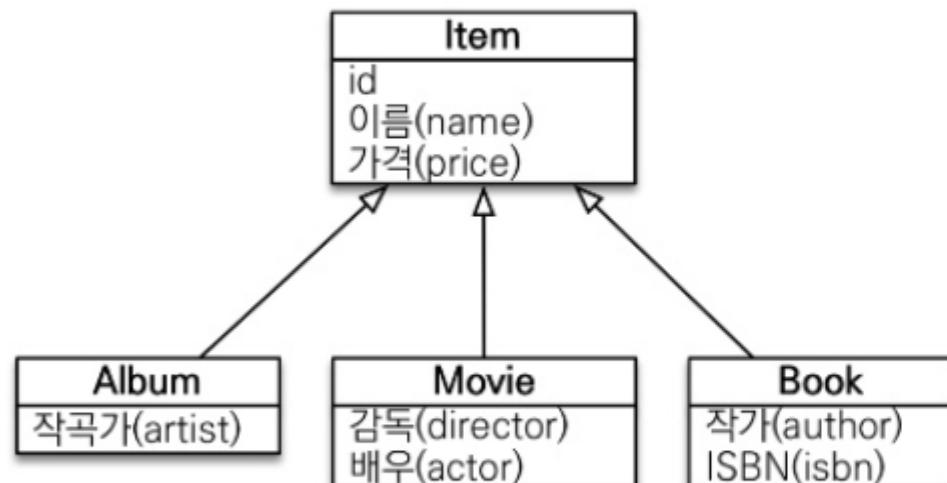
~~SELECT MEMBER_ID, NAME, TEL FROM MEMBER M~~

~~UPDATE MEMBER SET ... TEL=?~~

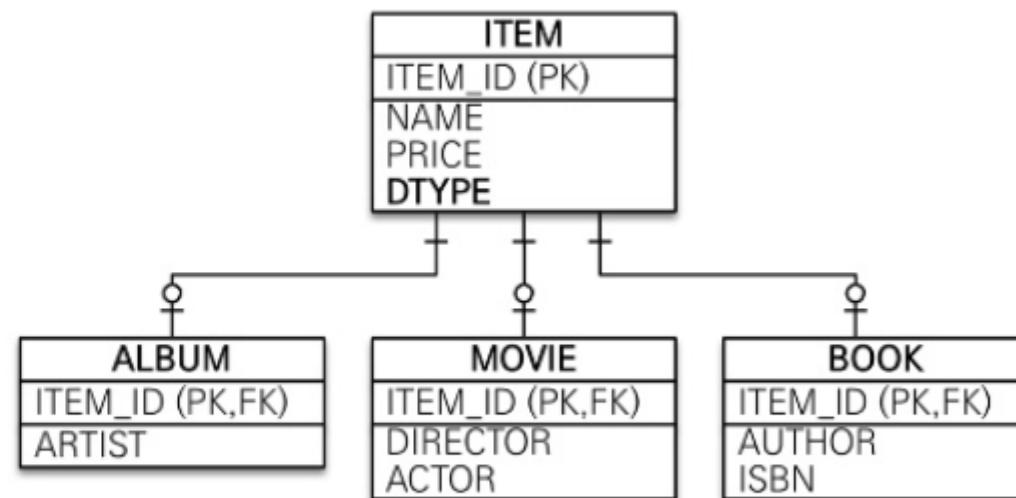
JPA와 패러다임의 불일치 해결

- JPA와 상속
- JPA와 연관관계
- JPA와 객체 그래프 탐색
- JPA와 비교하기

JPA와 상속



[객체 상속 관계]



[Table 슈퍼타입 서브타입 관계]

JPA와 상속

```
jpa.persist(album);
```

개발자가 할일

```
INSERT INTO ITEM ...
```

```
INSERT INTO ALBUM ...
```

나머진 JPA가 처리

JPA와 상속

```
Album album = jpa.find(Album.class, albumId);
```

개발자가 할일

```
SELECT I.* , A.*  
FROM ITEM I  
JOIN ALBUM A ON I.ITEM_ID = A.ITEM_ID
```

나머진 JPA가 처리

JPA와 연관관계, 객체 그래프 탐색

```
member.setTeam(team);  
jpa.persist(member);
```

연관관계 저장

```
Member member = jpa.find(Member.class, memberId);  
Team team = member.getTeam();
```

객체 그래프 탐색

신뢰할 수 있는 엔티티 계층

```
class MemberService {  
    ...  
    public void process() {  
        Member member = memberDAO.find(memberId);  
        member.getTeam(); //자유로운 객체 그래프 탐색  
        member.getOrder().getDelivery();  
    }  
}
```

JPA와 비교하기

```
String memberId = "100";
Member member1 = jpa.find(Member.class, memberId);
Member member2 = jpa.find(Member.class, memberId);

member1 == member2; //같다.
```

동일한 트랜잭션에서 조회한 엔티티는 같음을 보장

Spring Data JPA

- JPA + Spring

Query Mehtod

- find...By : findBoardByTitle
- read...By : readBoardByTitle
- query...By : queryBoardByTitle
- get...By : getBoardByTitle
- count...By : countBoardByTitle

Query Method – find by 특정 칼럼 처리

- Collection findBy + 속성이름(속성타입)

Query Method – like 구문처리

- 단순 like : Like
- 키워드 + '%' : StartingWith
- '%' + 키워드 " : EndingWith
- '%' + 키워드 + '%' : Cotaining

Query Method – and 혹은 or 조건 쳐

- 2개 이상 속성 검색할 경우 And or Or를 사용
ex) findByTitleContainingOrContentContaining

Query Method - 부등호 처리

- >와 < 부등호 처리는 GreaterThan 과 LessThan을 이용
 - public Collection<Board>
findByTitleContainingAndBnoGreaterThan(String keyword, Long num);

Query Method – order by 처리

- OrderBy + 속성 + Asc or Desc를 이용

```
public Collection<Board> findByBnoGreaterThanOrOrderByBnoDesc(Long  
bno);
```

Query Method – 페이징 처리와 정렬

- 모든 쿼리 메소드의 마지막 파라미터로 페이지 처리를 할 수 있는 Pageable 인터페이스와 정렬을 처리하는 Sort 인터페이스를 사용할 수 있다.
- org.springframework.data.domain.Pageable 인터페이스 구현한 PageRequest 클래스.
- boot 2.0에서 new PageRequest()는 deprecated 되어서 PageRequest.of()를 사용.
- bno > 0 order by bno desc 조건

```
public List<Board> findByBnoGreaterThanOrOrderByBnoDesc(Long bno, Pageable paging);
```
- 파라미터에 Pageable 적용되고, 리턴타입이 Collection<> 대신 Slice<>, Page<>, List<>로 이용.

Query Method – 페이지 생성자 설정

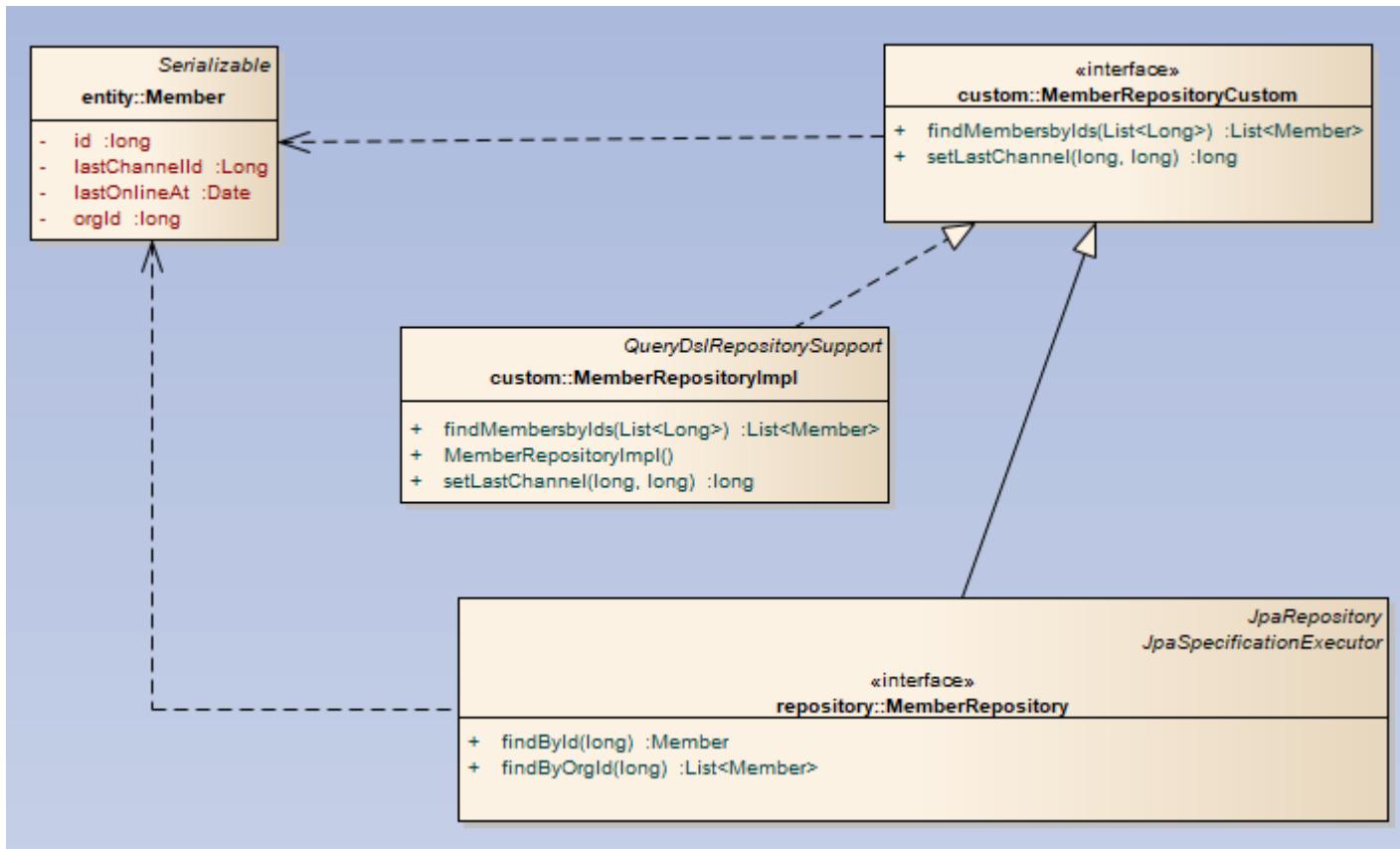
- final Pageable paging = new PageRequest(0, 10); // 페이지 번호(0부터 시작), 페이지당 데이터수

Query Method - 페이지 생성자에 정렬 객체 전달 가능

```
PageRequest(int page, int size, SortDirection direction, String...  
props); // 페이지 번호, 페이지당 데이터 수, 정렬 방향, 속성(칼럼) 들
```

```
new PageRequest(0, 10, Sort.Direction.ASC, "bno");
```

Spring Data + QueryDSL



JPA JPQL

Spring MVC

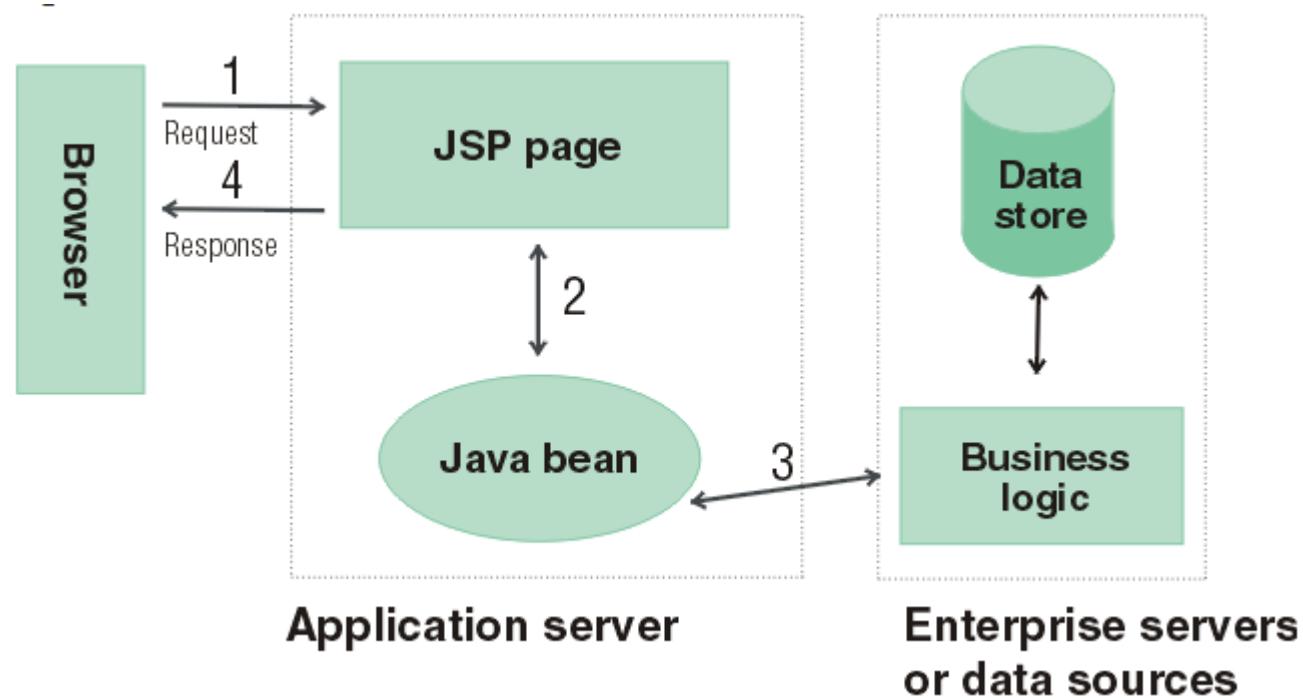
MVC?

- MVC는 Model–View–Controller의 약자이다.
- 원래는 제록스 연구소에서 일을 하던 트뤼그베 린즈커그가 처음으로 소개한 개념으로, 데스크톱 어플리케이션용으로 고안되었다.

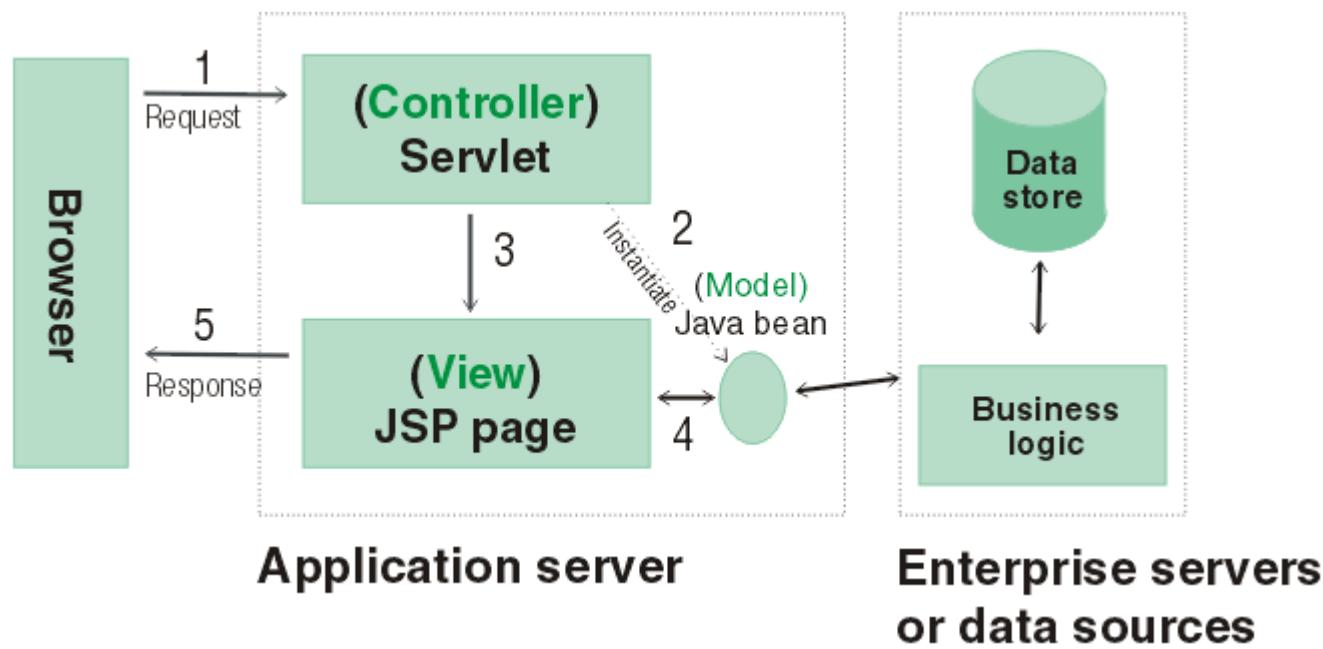
MVC?

- Model : 모델은 뷰가 렌더링하는데 필요한 데이터이다. 예를 들어 사용자가 요청한 상품 목록이나, 주문 내역이 이에 해당한다.
- View : 웹 애플리케이션에서 뷰(View)는 실제로 보이는 부분이며, 모델을 사용해 렌더링을 한다. 뷰는 JSP, JSF, PDF, XML등으로 결과를 표현한다.
- Controller : 컨트롤러는 사용자의 액션에 응답하는 컴포넌트다. 컨트롤러는 모델을 업데이트하고, 다른 액션을 수행한다.

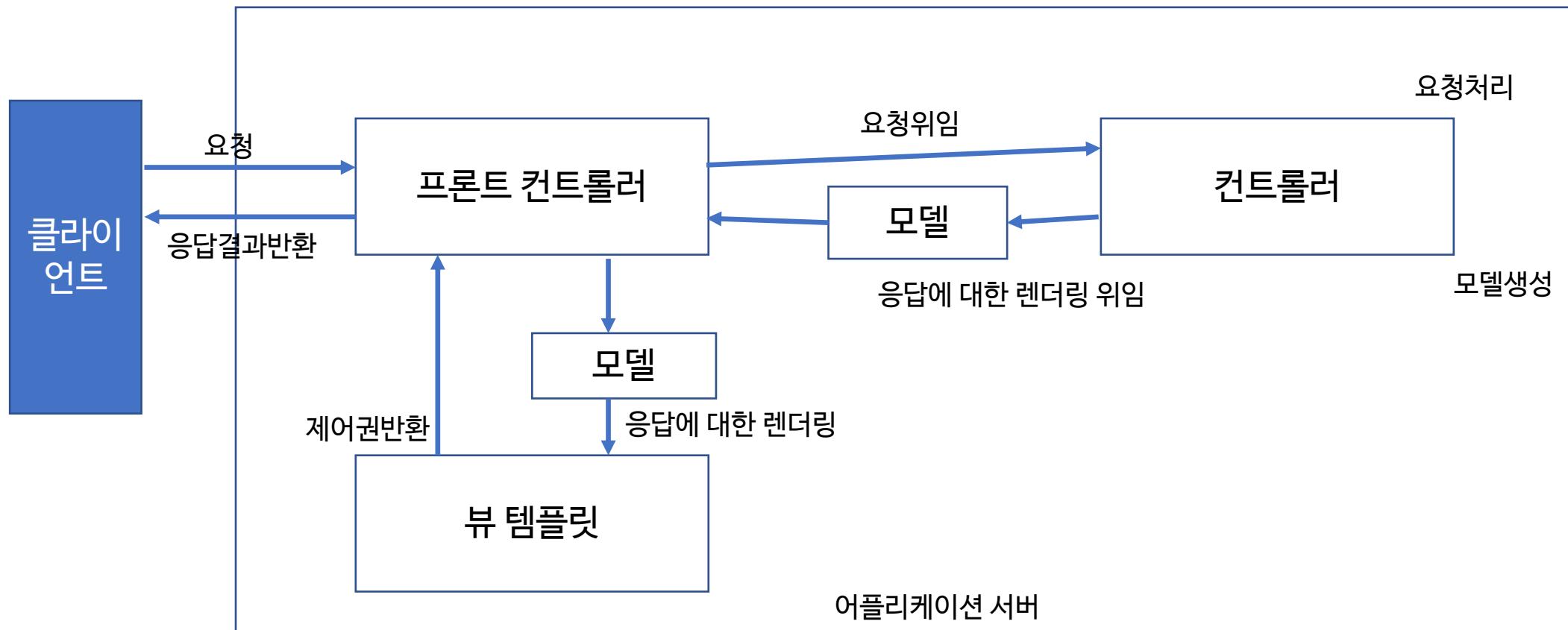
MVC Model 1 아키텍처



MVC Model 2 아키텍처

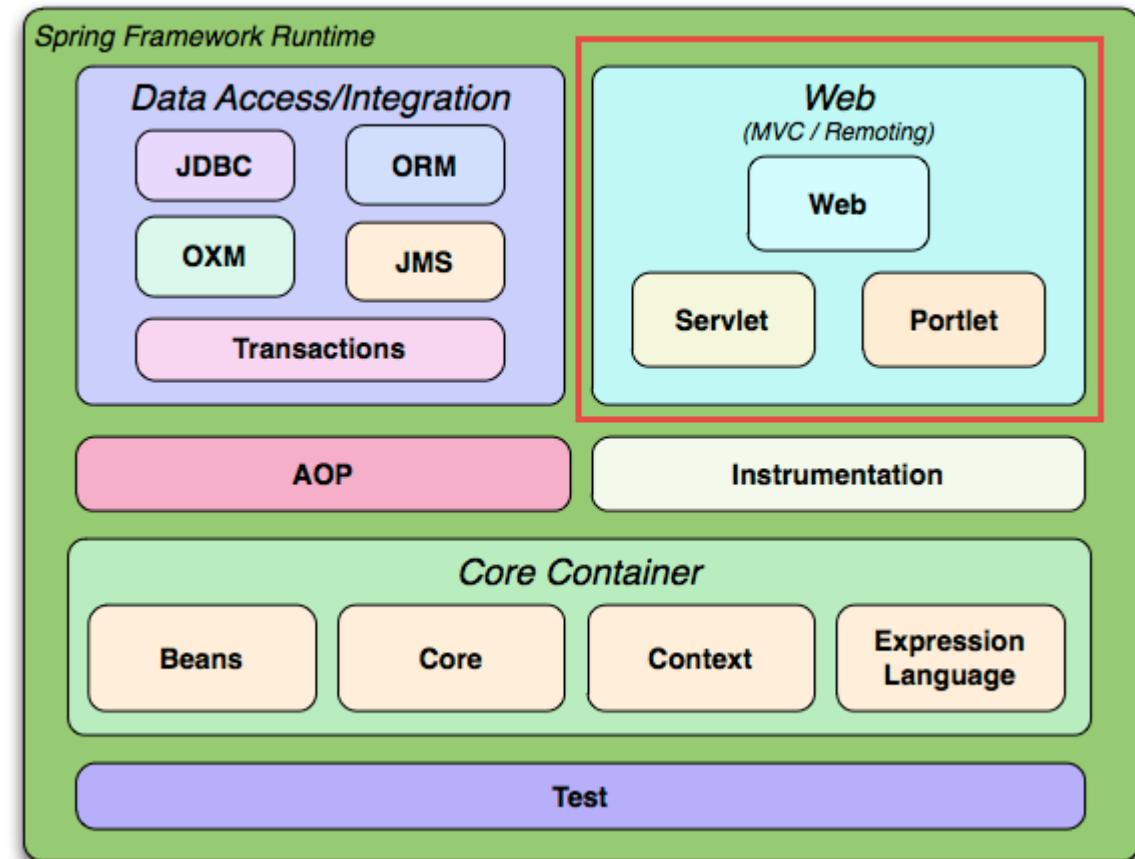


MVC Model2 발전형태



Spring Web Module

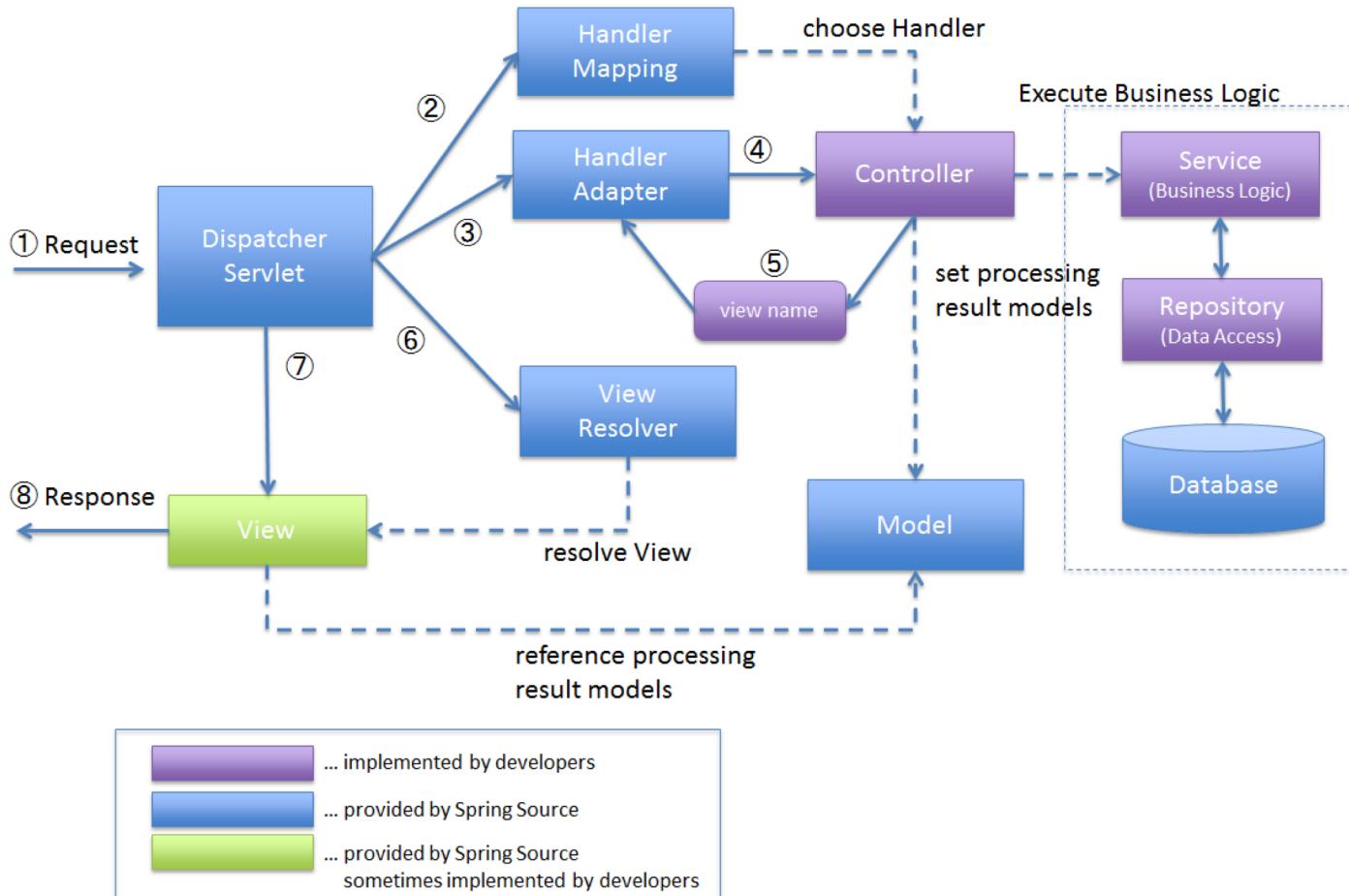
- Model2 MVC 패턴을 지원하는 Spring Module



Spring MVC

구성요소와 동작 이해

Spring MVC 기본 동작 흐름



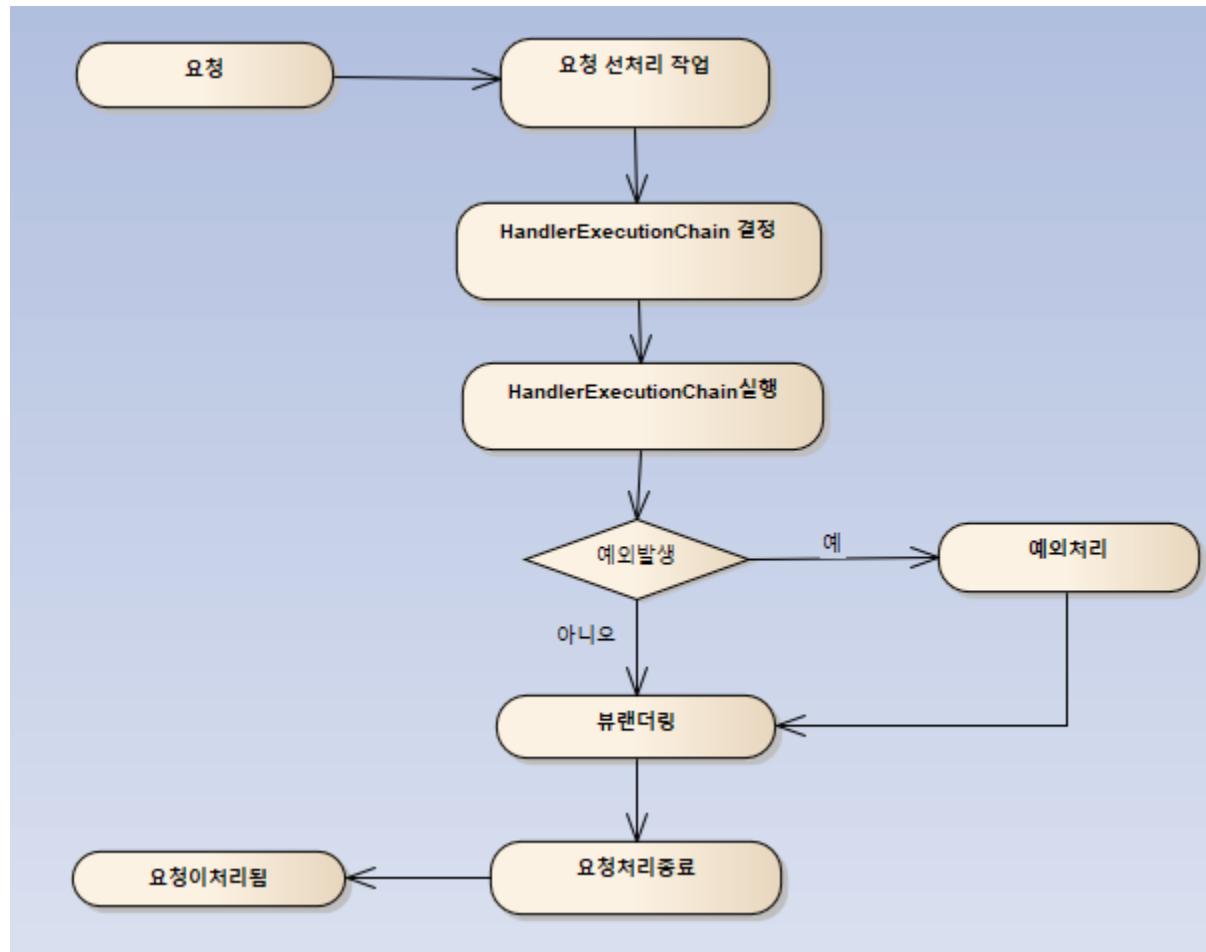
요청 처리를 위해 사용되는 컴포넌트

- DispatcherServlet
 - HandlerMapping
 - HandlerAdapter
 - MultipartResolver
 - LocaleResolver
 - ThemeResolver
 - HandlerExceptionResolver
 - RequestToViewNameTranslator
 - ViewResolver
 - FlashMapManager

DispatcherServlet

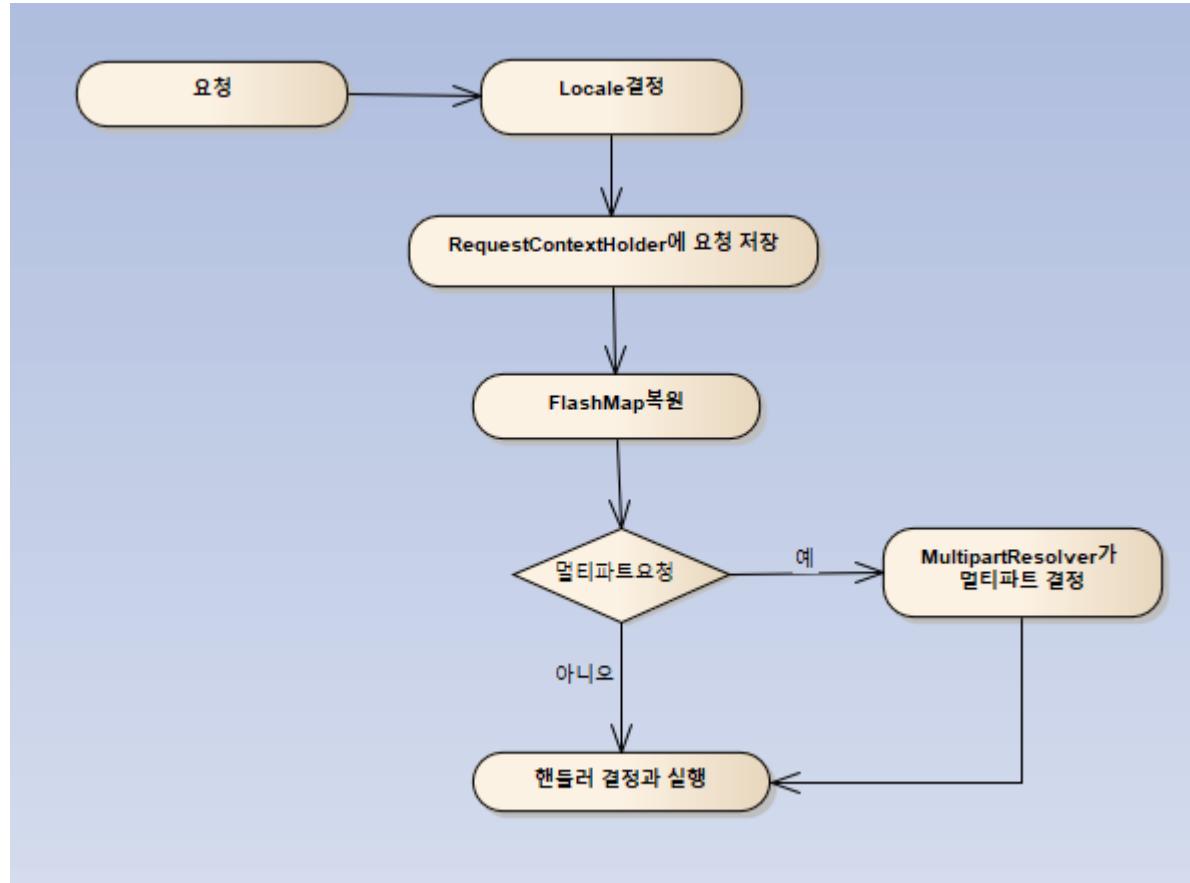
- 프론트 컨트롤러 (Front Controller)
- 클라이언트의 모든 요청을 받은 후 이를 처리할 핸들러에게 넘기고 핸들러가 처리한 결과를 받아 사용자에게 응답 결과를 보여준다.
- DispatcherServlet은 여러 컴포넌트를 이용해 작업을 처리한다.

DispatcherServlet 내부 동작흐름



DispatcherServlet 내부 동작흐름 상세 - 요청 선처리 작업

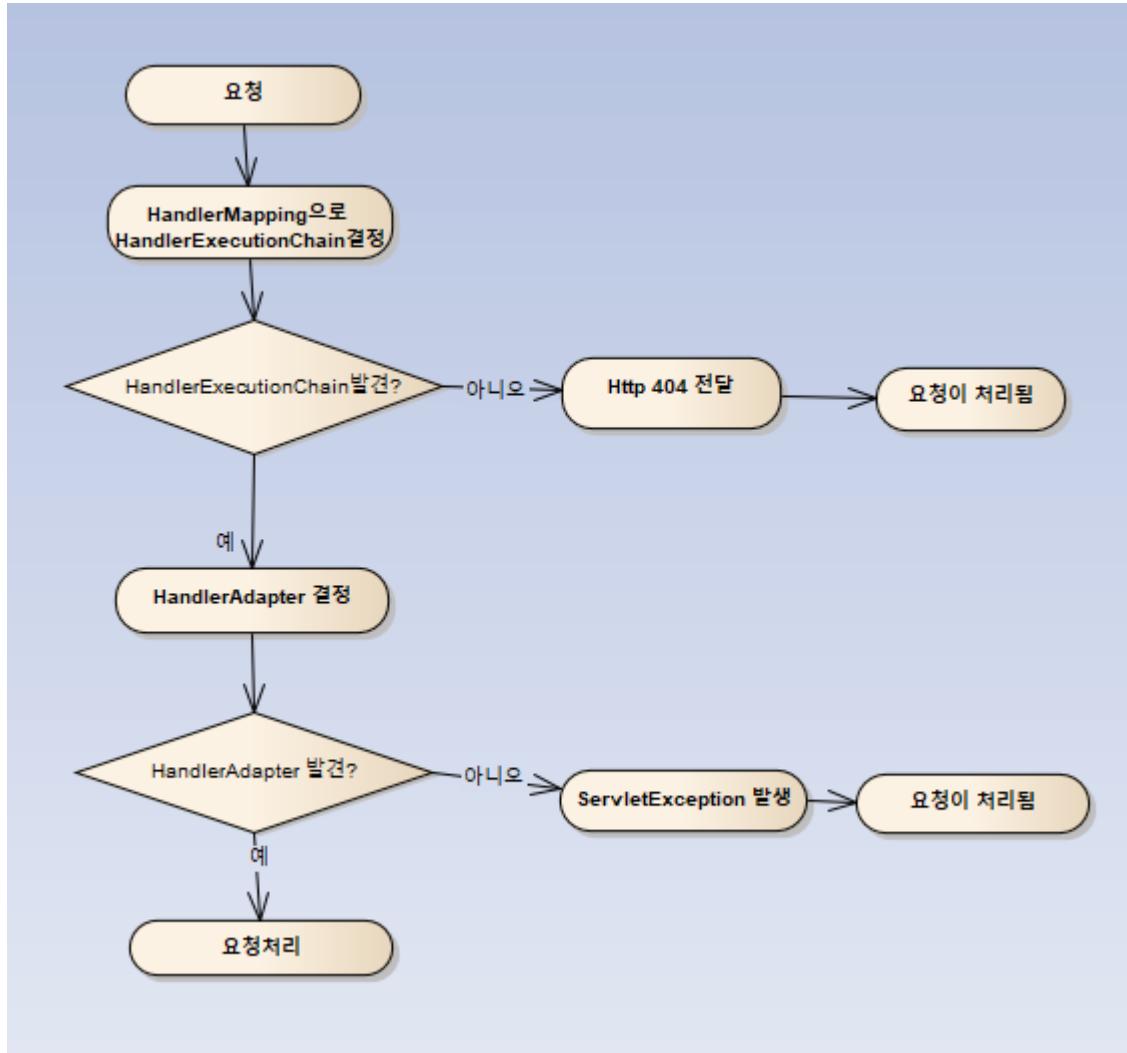
ThreadLocal



요청 선처리 작업시 사용된 컴포넌트

- org.springframework.web.servlet.LocaleResolver
 - 지역 정보를 결정해주는 전략 오브젝트이다. 디폴트인 AcceptHeaderLocaleResolver는 HTTP 헤더의 정보를 보고 지역정보를 설정해준다.
- org.springframework.web.servlet.FlashMapManager
 - FlashMap객체를 조회(retrieve) & 저장을 위한 인터페이스
 - RedirectAttributes의 addFlashAttribute메소드를 이용해서 저장한다.
 - 리다이렉트 후 조회를 하면 바로 정보는 삭제된다.
- org.springframework.web.context.request.RequestContextHolder
 - 일반 빈에서 HttpServletRequest, HttpServletResponse, HttpSession 등을 사용할 수 있도록 한다.
 - 해당 객체를 일반 빈에서 사용하게 되면, Web에 종속적이 될 수 있다.
- org.springframework.web.multipart.MultipartResolver
 - 멀티파트 파일 업로드를 처리하는 전략

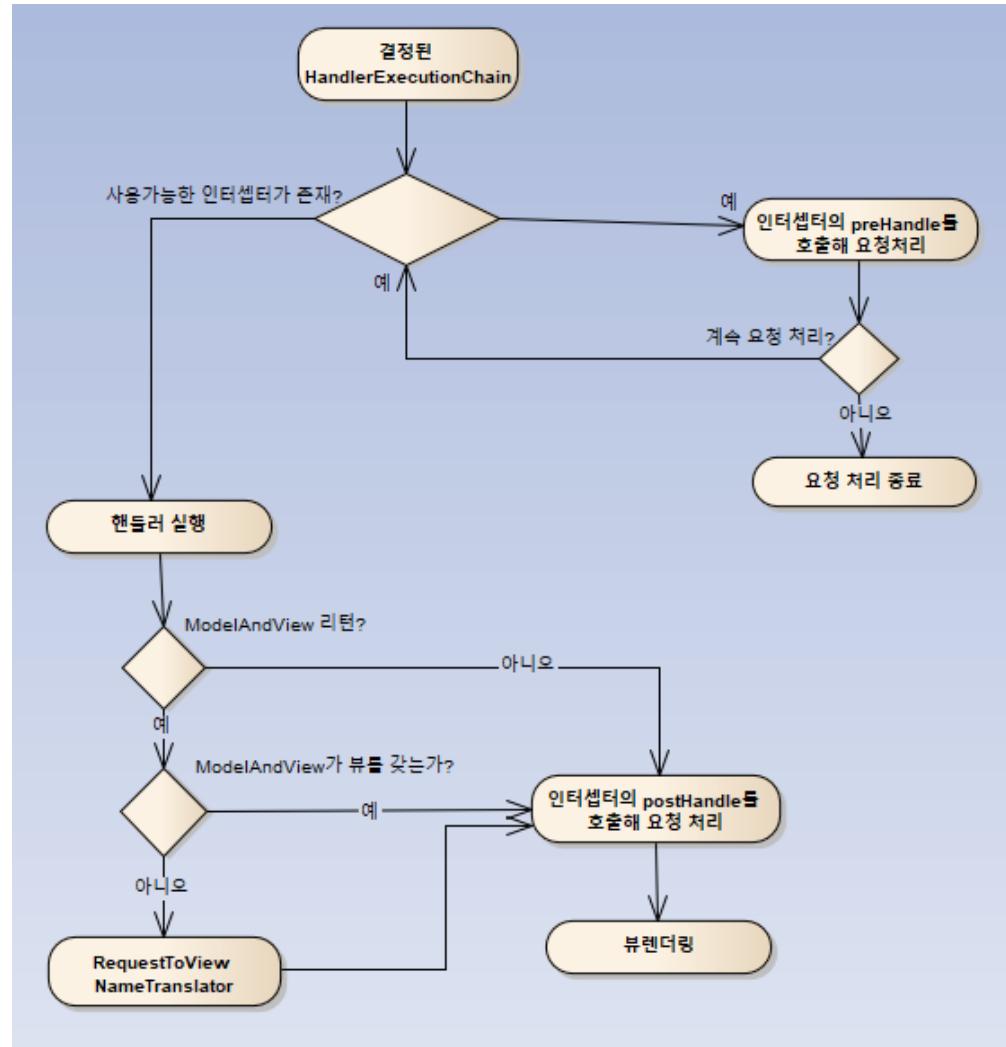
DispatcherServlet 내부 동작흐름 상세 - 요청 전달



요청 전달시 사용된 컴포넌트

- org.springframework.web.servlet.HandlerMapping
 - HandlerMapping 구현체는 어떤 핸들러가 요청을 처리할지에 대한 정보를 알고 있다.
 - 디폴트로 설정되는 있는 핸들러매핑은 BeanNameHandlerMapping과 DefaultAnnotationHandlerMapping 2가지가 설정되어 있다.
- org.springframework.web.servlet.HandlerExecutionChain
 - HandlerExecutionChain 구현체는 실제로 호출된 핸들러에 대한 참조를 가지고 있다. 즉, 무엇이 실행되어야 될지 알고 있는 객체라고 말할 수 있으며, 핸들러 실행전과 실행후에 수행될 HandlerInterceptor도 참조하고 있다.
- org.springframework.web.servlet.HandlerAdapter
 - 실제 핸들러를 실행하는 역할을 담당한다.
 - 핸들러 어댑터는 선택된 핸들러를 실행하는 방법과 응답을 ModelAndView로 변화하는 방법에 대해 알고 있다.
 - 디폴트로 설정되어 있는 핸들러어댑터는 HttpRequestHandlerAdapter, SimpleControllerHandlerAdapter, AnnotationMethodHandlerAdapter 3가지이다.
@RequestMapping과 @Controller 애노테이션을 통해 정의되는 컨트롤러의 경우 DefaultAnnotationHandlerMapping에 의해 핸들러가 결정되고, 그에 대응되는 AnnotationMethodHandlerAdapter에 의해 호출이 일어난다.

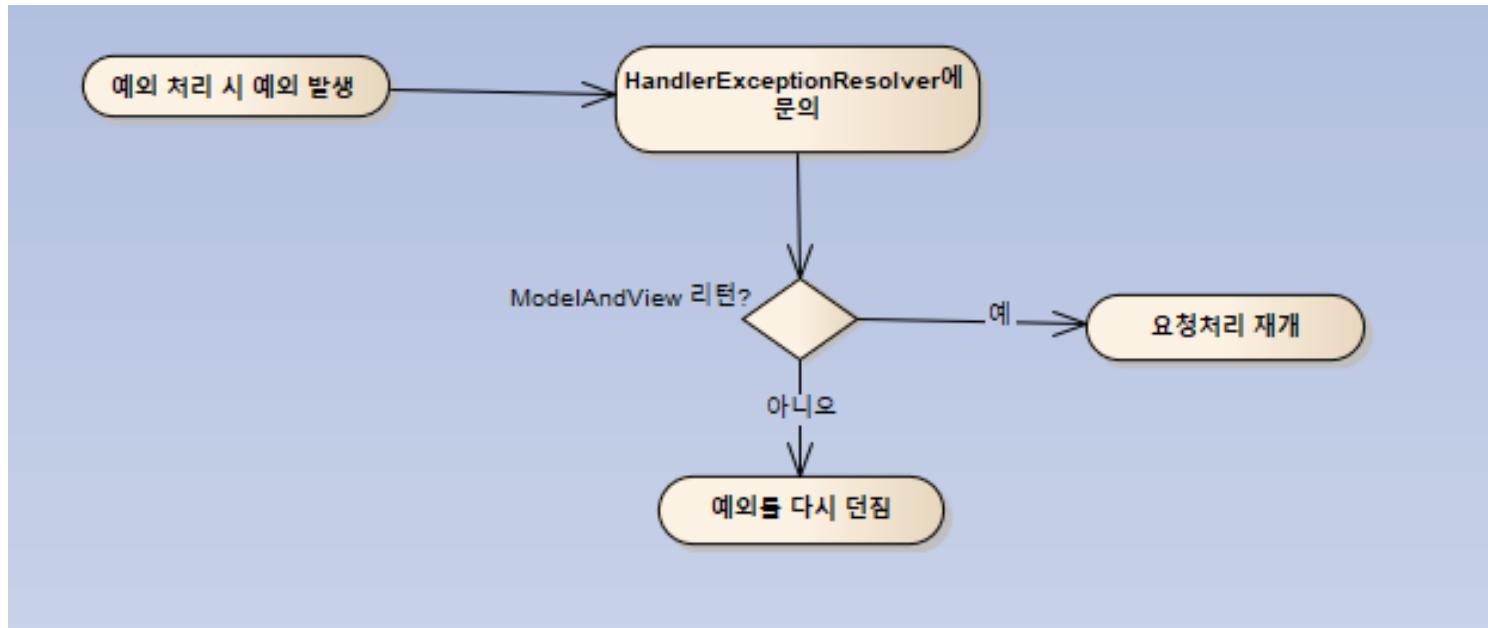
DispatcherServlet 내부 동작흐름 상세 - 요청 처리



요청 처리시 사용된 컴포넌트

- **org.springframework.web.servlet.ModelAndView**
 - ModelAndView는 Controller의 처리 결과를 보여줄 view와 view에서 사용할 값을 전달하는 클래스이다.
- **org.springframework.web.servlet.RequestToViewNameTranslator**
 - 컨트롤러에서 뷰 이름이나 뷰 오브젝트를 제공해주지 않았을 경우 URL과 같은 요청정보를 참고해서 자동으로 뷰 이름을 생성해주는 전략 오브젝트이다. 디폴트는 DefaultRequestToViewNameTranslator이다.

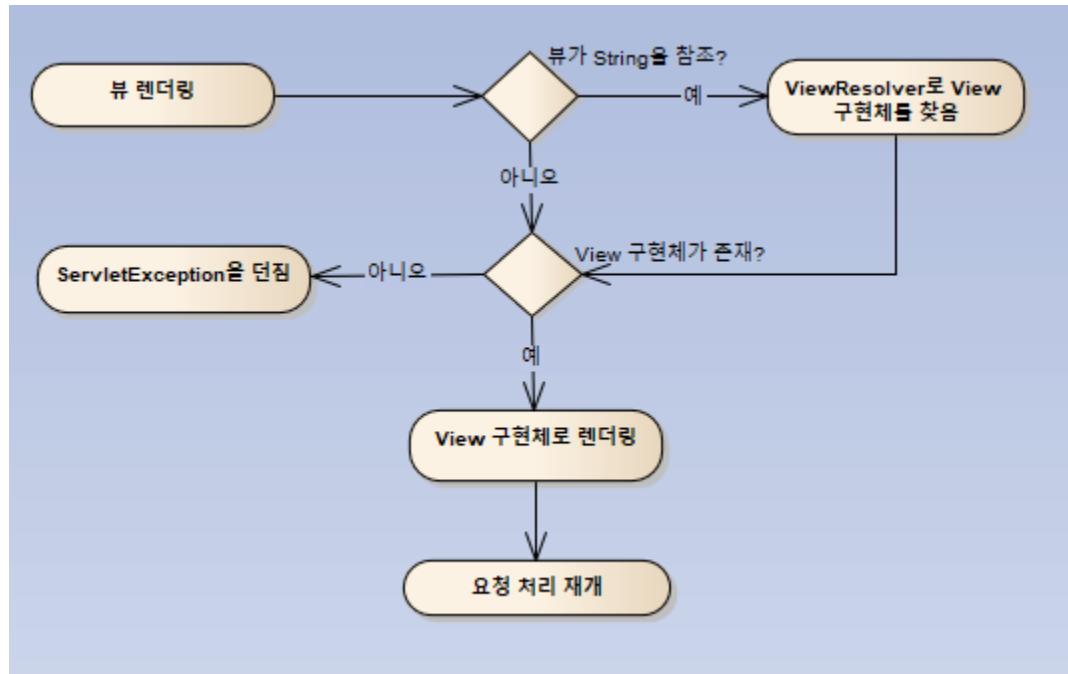
DispatcherServlet 내부 동작흐름 상세 – 예외처리



예외 처리시 사용된 컴포넌트

- **org.springframework.web.servlet.HandlerExceptionResolver**
 - 기본적으로 DispatcherServlet이 DefaultHandlerExceptionResolver를 등록한다.
 - HandlerExceptionResolver는 예외가 던져졌을 때 어떤 핸들러를 실행할 것인지에 대한 정보를 제공한다.

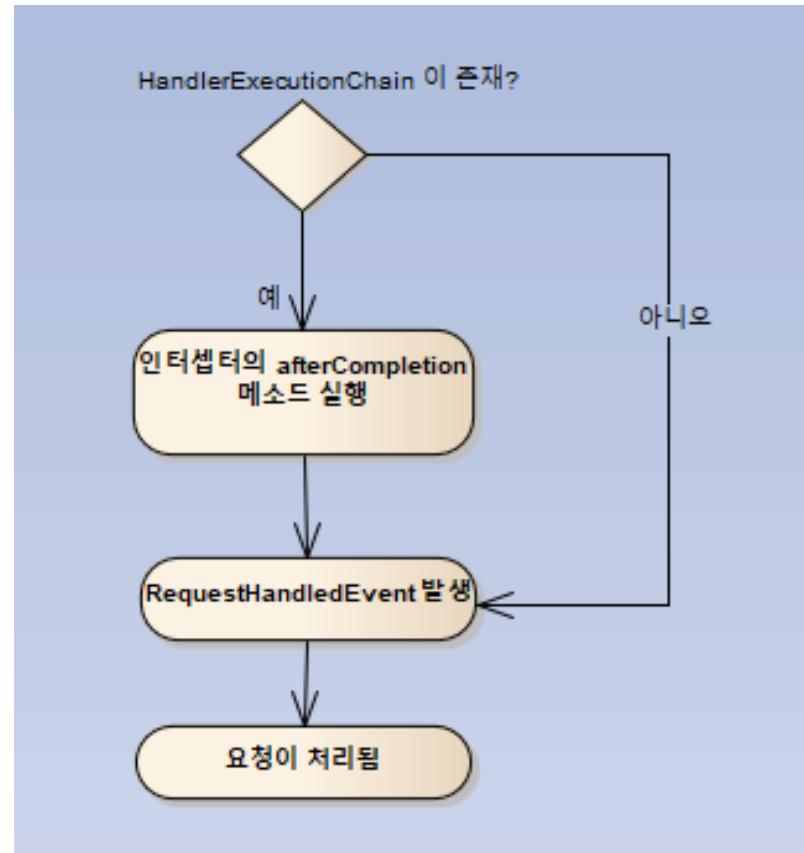
DispatcherServlet 내부 동작흐름 상세 - 뷰 렌더링 과정



뷰 렌더링 과정시 사용된 컴포넌트

- org.springframework.web.servlet.ViewResolver
 - 컨트롤러가 리턴한 뷰 이름을 참고해서 적절한 뷰 오브젝트를 찾아주는 로직을 가진 전략 오프젝트이다. 뷰의 종류에 따라 적절한 뷰 리졸버를 추가로 설정해줄 수 있다.

DispatcherServlet 내부 동작흐름 상세 - 요청 처리 종료



DispatcherServlet을 FrontController로 설정하기

- web.xml 파일에 설정
- javax.servlet.ServletContainerInitializer 사용
 - 서블릿 3.0 스펙 이상에서 web.xml파일을 대신해서 사용할 수 있다.
- org.springframework.web.WebApplicationInitializer 인터페이스를 구현해서 사용

web.xml파일에서 DispatcherServlet 설정하기 1/2

- xml spring 설정 읽어들이도록 DispatcherServlet설정

```
<?xml version="1.0" encoding = "UTF-8"?>
<web-app>

    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:WebMVCConfig.xml</param-value>
        </init-param>
    </servlet>
</web-app>
```

web.xml파일에서 DispatcherServlet 설정하기

- Java config spring 설정 읽어들이도록 DispatcherServlet설정

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="2.5" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
metadata-complete="true">

    <display-name>Spring JavaConfig Sample</display-name>

    <servlet>
        <servlet-name>mvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextClass</param-name>
            <param-value>org.springframework.web.context.support.AnnotationConfigWebApplicationContext</param-value>
        </init-param>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>kr.or.connect.webmvc.config.WebMvcContextConfiguration</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>mvc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

WebApplicationInitializer를 구현해서 설정하기 1/2

- Spring MVC는 ServletContainerInitializer를 구현하고 있는 SpringServletContainerInitializer를 제공한다.
- SpringServletContainerInitializer는 WebApplicationInitializer 구현체를 찾아 인스턴스를 만들고 해당 인스턴스의 onStartup메소드를 호출하여 초기화 한다.

WebApplicationInitializer를 구현해서 설정하기 2/2

```
public class WebApplicationInitializer implements WebApplicationInitializer {

    private static final String DISPATCHER_SERVLET_NAME = "dispatcher";

    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        registerDispatcherServlet(servletContext);
    }

    private void registerDispatcherServlet(ServletContext servletContext) {
        AnnotationConfigWebApplicationContext dispatcherContext =
            createContext(WebMvcContextConfiguration.class);
        ServletRegistration.Dynamic dispatcher;
        dispatcher = servletContext.addServlet(DISPATCHER_SERVLET_NAME,
            new DispatcherServlet(dispatcherContext));
        dispatcher.setLoadOnStartup(1);
        dispatcher.addMapping("/");
    }

    private AnnotationConfigWebApplicationContext createContext(final Class<?>... annotatedClasses) {
        AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplicationContext();
        context.register(annotatedClasses);
        return context;
    }
}
```

Spring MVC 설정

- kr.or.connect.webmvc.config.WebMvcContextConfiguration

```
@Configuration  
@EnableWebMvc  
@ComponentScan(basePackages = { "kr.or.connect.webmvc.controller" })  
public class WebMvcContextConfiguration extends WebMvcConfigurerAdapter {  
  
    ....  
  
}
```

@Configuration

- org.springframework.context.annotation 의 Configuration 애노테이션과 Bean 애노테이션 코드를 이용하여 스프링 컨테이너에 새로운 빈 객체를 제공할 수 있다.

@EnableWebMvc

- DispatcherServlet의 RequestMappingHandlerMapping, RequestMappingHandlerAdapter, ExceptionHandlerExceptionResolver, MessageConverter 등 Web에 필요한 빈들을 대부분 자동으로 설정 해준다.
- xml로 설정의 <mvc:annotation-driven/> 와 동일하다.
- 기본 설정 이외의 설정이 필요하다면 WebMvcConfigurerAdapter 를 상속받도록 Java config class를 작성한 후, 필요한 메소드를 오버라이딩 하도록 한다.

@EnableWebMVC

```
....  
@Import(DelegatingWebMvcConfiguration.class)  
public @interface EnableWebMvc {  
}
```

```
@Configuration  
public class DelegatingWebMvcConfiguration extends WebMvcConfigurationSupport {  
    ....  
    @Autowired(required = false)  
    public void setConfigurers(List<WebMvcConfigurer> configurers) {  
        if (!CollectionUtils.isEmpty(configurers)) {  
            this.configurers.addWebMvcConfigurers(configurers);  
        }  
    }  
}
```

WebMvcConfigurationSupport

- <https://github.com/spring-projects/spring-framework/blob/master/spring-webmvc/src/main/java/org/springframework/web/servlet/config/annotation/WebMvcConfigurationSupport.java>

@ComponentScan

- ComponentScan 애노테이션을 이용하면 Controller, Service, Repository, Component 애노테이션이 붙은 클래스를 찾아 스프링 컨테이너가 관리하게 된다.
- DefaultAnnotationHandlerMapping과 RequestMappingHandlerMapping 구현체는 다른 핸들러 매핑보다 훨씬 더 정교한 작업을 수행한다. 이 두 개의 구현체는 애노테이션을 사용해 매핑 관계를 찾는 매우 강력한 기능을 가지고 있다. 이들 구현체는 스프링 컨테이너 즉 애플리케이션 컨텍스트에 있는 요청 처리 빈에서 RequestMapping 애노테이션을 클래스나 메소드에서 찾아 HandlerMapping 객체를 생성하게 된다.
 - HandlerMapping은 서버로 들어온 요청을 어느 핸들러로 전달할지 결정하는 역할을 수행한다.
- DefaultAnnotationHandlerMapping은 DispatcherServlet이 기본으로 등록하는 기본 핸들러 맵핑 객체이고, RequestMappingHandlerMapping은 더 강력하고 유연하지만 사용하려면 명시적으로 설정해야 한다.

WebMvcConfigurerAdapter

- org.springframework.web.servlet.config.annotation.

WebMvcConfigurerAdapter

- @EnableWebMvc 를 이용하면 기본적인 설정이 모두 자동으로 되지만, 기본 설정 이외의 설정이 필요할 경우 해당 클래스를 상속 받은 후, 메소드를 오버라이딩 하여 구현한다.

Controller(Handler) 클래스 작성하기

- @Controller 애노테이션을 클래스 위에 붙인다.
- 맵핑을 위해 @RequestMapping 애노테이션을 클래스나 메소드에서 사용한다.

@RequestMapping

- Http 요청과 이를 다루기 위한 Controller 의 메소드를 연결하는 어노테이션
- Http Method 와 연결하는 방법
 - `@RequestMapping("/users", method=RequestMethod.POST)`
 - From Spring 4.3 version
 - `@GetMapping`
 - `@PostMapping`
 - `@PutMapping`
 - `@DeleteMapping`
 - `@PatchMapping`
- Http 특정 헤더와 연결하는 방법
 - `@RequestMapping(method = RequestMethod.GET, headers = "content-type=application/json")`
- Http Parameter 와 연결하는 방법
 - `@RequestMapping(method = RequestMethod.GET, params = "type=raw")`
- Content-Type Header 와 연결하는 방법
 - `@RequestMapping(method = RequestMethod.GET, consumes = "application/json")`
- Accept Header 와 연결하는 방법
 - `@RequestMapping(method = RequestMethod.GET, produces = "application/json")`

Spring MVC가 지원하는 Controller메소드 인수 타입

- javax.servlet.ServletRequest
- javax.servlet.http.HttpServletRequest
- org.springframework.web.multipart.MultipartRequest
- org.springframework.web.multipart.MultipartHttpServletRequest
- javax.servlet.ServletResponse
- javax.servlet.http.HttpServletResponse
- javax.servlet.http.HttpSession
- org.springframework.web.context.request.WebRequest

Spring MVC가 지원하는 Controller메소드 인수 타입

- org.springframework.web.context.request.NativeWebRequest
- java.util.Locale
- java.io.InputStream
- java.io.Reader
- java.io.OutputStream
- java.io.Writer
- javax.security.Principal
- java.util.Map
- org.springframework.ui.Model
- org.springframework.ui.ModelMap

Spring MVC가 지원하는 Controller메소드 인수 타입

- org.springframework.web.multipart.MultipartFile
- javax.servlet.http.Part
- org.springframework.web.servlet.mvc.support.RedirectAttributes
- org.springframework.validation.Errors
- org.springframework.validation.BindingResult
- org.springframework.web.bind.support.SessionStatus
- org.springframework.web.util.UriComponentsBuilder
- org.springframework.http.HttpEntity<?>
- Command 또는 Form 객체

Srping MVC가 지원하는 메소드 인수 애노테이션

- `@RequestParam`
- `@RequestHeader`
- `@RequestBody`
- `@RequestPart`
- `@ModelAttribute`
- `@PathVariable`
- `@CookieValue`

@RequestParam

- Mapping 된 메소드의 Argument에 붙일 수 있는 어노테이션
- @RequestParam의 name에는 http parameter의 name과 맵핑
- @RequestParam의 required는 필수인지 아닌지 판단.

@PathVariable

- @RequestMapping 의 path 에 변수명을 입력받기 위한 place holder 가 필요함
- place holder 의 이름과 PathVariable 의 name 값과 같으면 mapping 됨.
- required 속성은 default true 임.

@RequestHeader

- 요청정보의 헤더 정보를 읽어들일 때 사용
- @RequestHeader(name="헤더명") String 변수명

Spring MVC가 지원하는 메소드 리턴 값

- org.springframework.web.servlet.ModelAndView
- org.springframework.ui.Model
- java.util.Map
- org.springframework.ui.ModelMap
- org.springframework.web.servlet.View
- java.lang.String
- java.lang.Void
- org.springframework.http.HttpEntity<?>
- org.springframework.http.ResponseEntity<?>
- 기타 리턴 타입

웹 페이지는 중복 개발되는 요소가 존재한다.

NAVER 지식iN

이용안내 보기 로그인

홈 답변하기 Q&A 오픈사전 사람들 베스트 프로필 지식iN전당 오늘의질문 교육기부 장학기부 파트너센터 질문하기

오늘의질문 자식iN의선택 봉어빵 vs 잉어빵

혼자이고 싶다는 생각이 드는 순간은?

전문가들에게 질문해 보세요! 의사 한의사 변호사 노무사 수의사 약사 세무사

지식iN 통계 19시간 기준 답변자 질문자 QA를 본 사람

2월 7일 컴퓨터통신 분야에서는 4천명의 지식iN들이 답변하셨습니다.

1 2 19% 81%

많이 본 Q&A 07월 18시간 기준

1 강수지 첫번째남편 2 나 자신에게 상을 하나 줄 수 있다면? 3 노후 경유차 운행제한 질문

2018 GLOBAL V LIVE TOP10 MONSTA X 2/8 오후 10시 V LIVE

NAVER 지식iN

이용안내 보기 로그인

홈 답변하기 Q&A 오픈사전 사람들 베스트 프로필 지식iN전당 오늘의질문 교육기부 장학기부 파트너센터 질문하기

Q&A > 생활 > 자동차 > 자동차 구조, 역사

Q 노후 경유차 운행제한 질문

비공개 | 질문 25건 | 질문마감률 100% | 질문체택률 95.4% | 2016.08.20. 20:50 | 조회수 36,718

제 차는 2005년 3월에 구매한 쏘렌토(2005) 경유차입니다. 차량등록증을 보니 2560kg이라고 되어있네요. 노후 경유차 기사를 접해보면 2005년 이전 2.5톤 경유자는 2017~2020년 서울, 인천, 경기에서 탈수 없다고 하던데요. 그럼 제 차의 경우 2021년부터 운행을 못한다는 얘기인가요? 조심히 타서 앞으로 10년은 더 탈수 있을거 같은데.... 혹시 매연저감장치 설치시 계속탈수는 있는지 알고싶습니다.

나도 궁금해요 1

① 질문자 체택된 경우, 추가 답변 등록이 불가합니다.

많이 본 Q&A 07월 18시간 기준

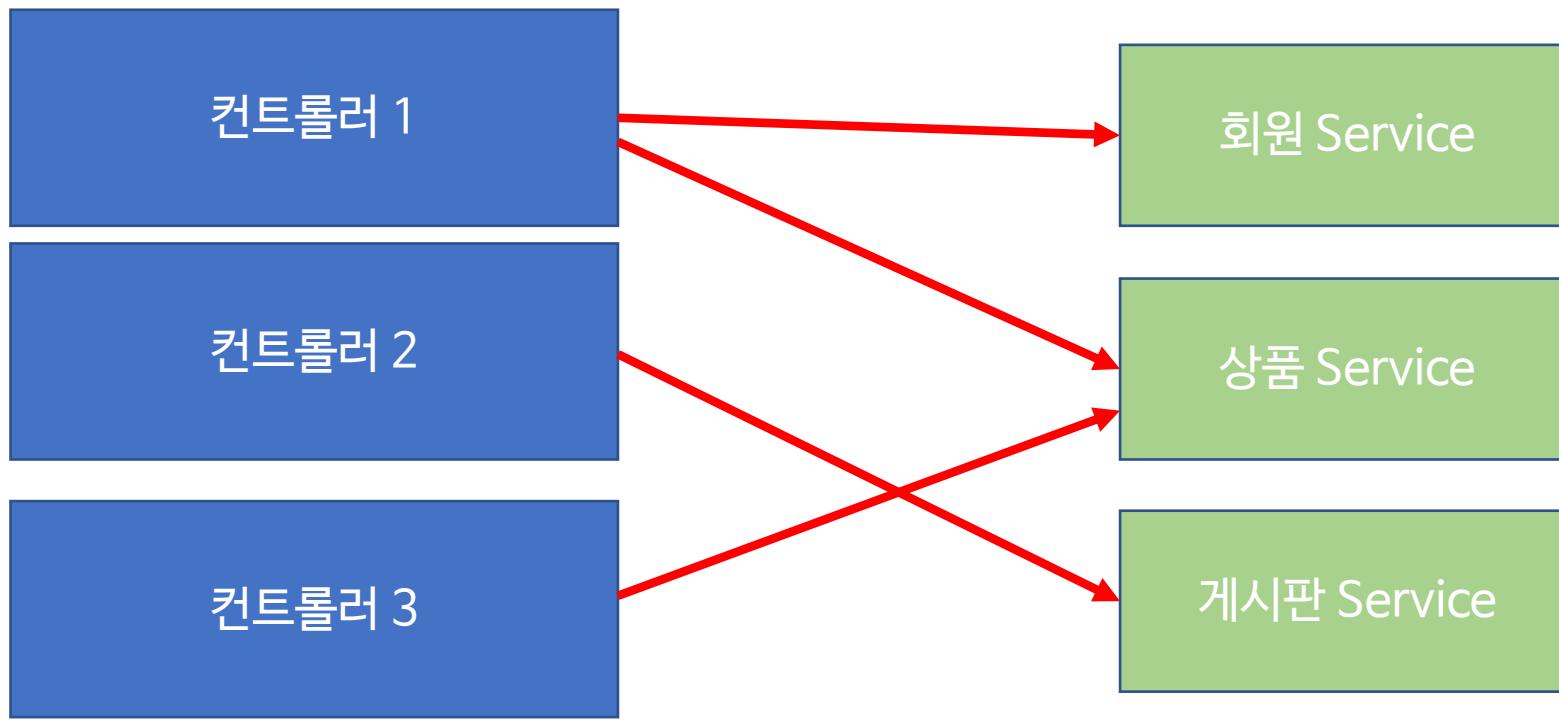
1 강수지 첫번째남편 2013년형 제네시스 뒷자석 전동시트 작... 3 bldc 모터도 부하가 걸리나요?? 4 사계 에어서스 자립 간선

Controller에서 중복되는 부분을 처리하려면?

- 별도의 객체로 분리한다.
- 별도의 메소드로 분리한다.
- 예를 들어 쇼핑몰에서 게시판에서도 회원 정보를 보여주고, 상품 목록 보기에서도 회원 정보를 보여줘야 한다면 회원 정보를 읽어오는 코드는 어떻게 해야 할까?

컨트롤러와 서비스

- 비지니스 메소드를 별도의 Service 객체에서 구현하도록 하고 컨트롤러는 Service 객체를 사용하도록 한다.



서비스(Service) 객체란?

- 비지니스 로직(Business logic)을 수행하는 메소드를 가지고 있는 객체를 서비스 객체라고 한다.
- 보통 하나의 비지니스 로직은 하나의 트랜잭션으로 동작한다.

트랜잭션(Transaction)이란?

- 트랜잭션은 하나의 논리적인 작업을 의미한다.
- 트랜잭션의 특징은 크게 4가지로 구분된다.
 - 원자성 (Atomicity)
 - 일관성 (Consistency)
 - 독립성 (Isolation)
 - 지속성 (Durability)

원자성 (Atomicity)

- 전체가 성공하거나 전체가 실패하는 것을 의미한다.
- 예를 들어 "출금"이라는 기능의 흐름이 다음과 같다고 생각해 보자.
 1. 잔액이 얼마인지 조회한다.
 2. 출금하려는 금액이 잔액보다 작은지 검사한다.
 3. 출금하려는 금액이 잔액보다 작다면 ($\text{잔액} - \text{출금액}$)으로 수정한다.
 4. 언제, 어디서 출금했는지 정보를 기록한다.
 5. 사용자에게 출금한다.
- 위의 작업이 4번에서 오류가 발생했다면 어떻게 될까? 4번에서 오류가 발생했다면, 앞의 작업들을 모두 원래대로 복원을 시켜야 한다. 이를 rollback이라고 한다. 5번까지 모두 성공했을 때만 정보를 모두 반영해야 한다. 이를 commit한다고 한다. 이렇게 rollback하거나 commit을 하게 되면 하나의 트랜잭션 처리가 완료된다.

일관성 (Consistency)

- 일관성은 트랜잭션의 작업 처리 결과가 항상 일관성이 있어야 한다는 것이다. 트랜잭션이 진행되는 동안에 데이터가 변경 되더라도 업데이트된 데이터로 트랜잭션이 진행되는것이 아니라,처음에 트랜잭션을 진행 하기 위해 참조한 데이터로 진행된다. 이렇게 함으로써 각 사용자는 일관성 있는 데이터를 볼 수 있는 것이다.

독립성 (Isolation)

- 독립성은 둘 이상의 트랜잭션이 동시에 병행 실행되고 있을 경우에 어느 하나의 트랜잭션이라도 다른 트랜잭션의 연산을 끼어들 수 없다. 하나의 특정 트랜잭션이 완료될때까지, 다른 트랜잭션이 특정 트랜잭션의 결과를 참조할 수 없다.

지속성 (Durability)

- 지속성은 트랜잭션이 성공적으로 완료되었을 경우, 결과는 영구적으로 반영되어야 한다는 점이다.

JDBC 프로그래밍에서 트랜잭션 처리 방법

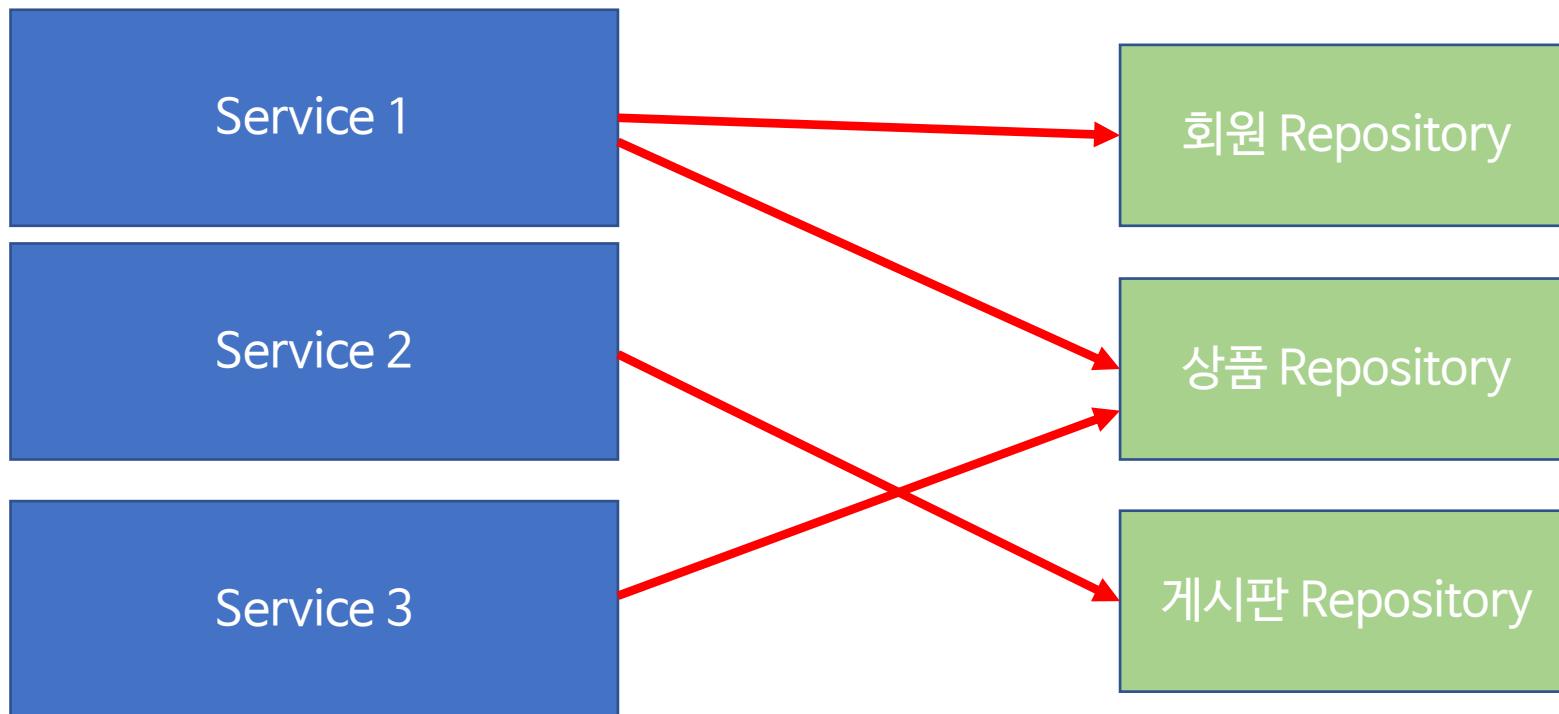
- DB에 연결된 후 Connection객체의 setAutoCommit메소드에 false를 파라미터로 지정한다.
- 입력,수정,삭제 SQL이 실행을 한 후 모두 성공했을 경우 Connection이 가지고 있는 commit()메소드를 호출한다.

@EnableTransactionManagement

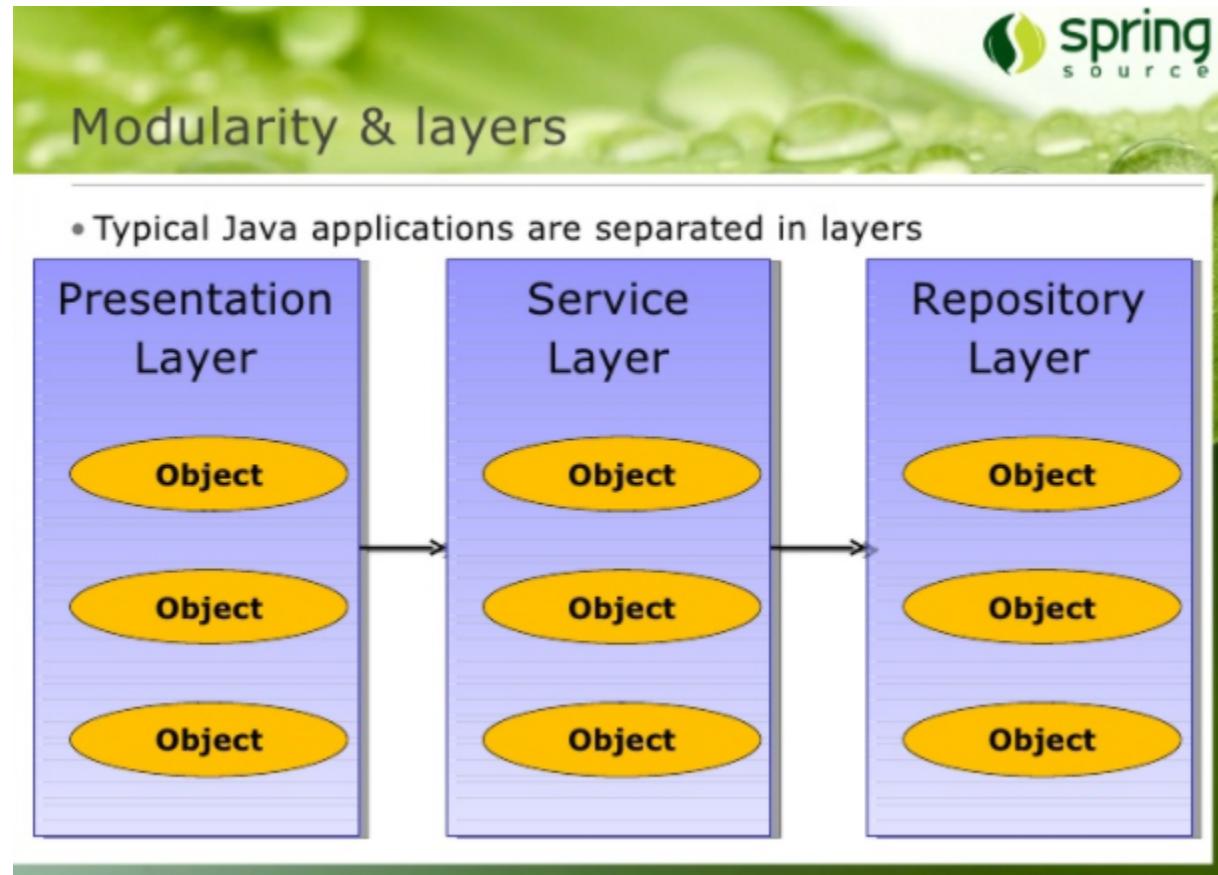
- Spring Java Config파일에서 트랜잭션을 활성화 할 때 사용하는 애노테이션
- Java Config를 사용하게 되면 PlatformTransactionManager 구현체를 모두 찾아서 그 중에 하나를 매팅해 사용한다.
- 특정 트랜잭션 메니저를 사용하고자 한다면 TransactionManagementConfigurer를 Java Config파일에서 구현하고 원하는 트랜잭션 메니저를 리턴하도록 한다.
- 아니면, 특정 트랜잭션 메니저 객체를 생성시 @Primary 애노테이션을 지정한다.

서비스 객체에서 중복으로 호출되는 코드의 처리

- 데이터 액세스 메소드를 별도의 Repository(Dao) 객체에서 구현하도록 하고 서비스는 Repository 객체를 사용하도록 한다.



레이어드 아키텍처



설정의 분리

- Spring 설정 파일을 프리젠테이션 레이어 쪽과 나머지를 분리할 수 있다.
- web.xml 파일에서 프리젠테이션 레이어에 대한 스프링 설정은 DispatcherServlet이 읽도록 하고, 그 외의 설정은 ContextLoaderListener를 통해서 읽도록 한다.
- DispatcherServlet을 경우에 따라서 2개 이상 설정 할 수 있는데 이 경우에는 각각의 DispatcherServlet의 ApplicationContext가 각각 독립적이기 때문에 각각의 설정 파일에서 생성한 빈을 서로 사용할 수 없다.
- 위의 경우와 같이 동시에 필요한 빈은 ContextLoaderListener를 사용함으로써 공통으로 사용하게 할 수 있다.
- ContextLoaderListener와 DispatcherServlet은 각각 ApplicationContext를 생성하는데, ContextLoaderListener가 생성하는 ApplicationContext가 root 컨텍스트가 되고 DispatcherServlet이 생성한 인스턴스는 root 컨텍스트를 부모로 하는 자식 컨텍스트가 된다. 참고로, 자식 컨텍스트들은 root 컨텍스트의 설정 빈을 사용할 수 있다.

@RestController

- Spring 4에서 Rest API 또는 Web API를 개발하기 위해 등장한 애노테이션
- 이전 버전의 @Controller와 @ResponseBody를 포함한다.

MessageConvertor

- 자바 객체와 HTTP 요청 / 응답 바디를 변환하는 역할
- @ResponseBody, @RequestBody
- @EnableWebMvc로 인한 기본 설정.
 - WebMvcConfigurationSupport를 사용하여 Spring MVC 구현을 하고 있음.
 - Default MessageConvertor를 제공하고 있음.
 - <https://github.com/spring-projects/spring-framework/blob/master/spring-webmvc/src/main/java/org/springframework/web/servlet/config/annotation/WebMvcConfigurationSupport.java>의 addDefaultHttpMessageConverters메소드 항목 참조

MessageConverter 종류

MessageConverter 종류	기능
ByteArrayHttpMessageConverter	converts byte arrays
StringHttpMessageConverter	converts Strings
ResourceHttpMessageConverter	converts org.springframework.core.io.Resource for any type of octet stream
SourceHttpMessageConverter	converts javax.xml.transform.Source
FormHttpMessageConverter	converts form data to/from a MultiValueMap<String, String>.
Jaxb2RootElementHttpMessageConverter	converts Java objects to/from XML (added only if JAXB2 is present on the classpath)
MappingJackson2HttpMessageConverter	converts JSON (added only if Jackson 2 is present on the classpath)
MappingJacksonHttpMessageConverter	converts JSON (added only if Jackson is present on the classpath)
AtomFeedHttpMessageConverter	converts Atom feeds (added only if Rome is present on the classpath)
RssChannelHttpMessageConverter	converts RSS feeds (added only if Rome is present on the classpath)

json 응답하기

- 컨트롤러의 메소드에서는 json으로 변환될 객체를 반환한다.
- jackson라이브러리를 추가할 경우 객체를 json으로 변환하는 메시지 컨버터가 사용되도록 @EnableWebMvc에서 기본으로 설정되어 있다.
- jackson라이브러리를 추가하지 않으면 json메시지로 변환할 수 없어 500오류가 발생한다.
- 사용자가 임의의 메시지 컨버터(MessageConverter)를 사용하도록 하려면 WebMvcConfigurerAdapter의 configureMessageConverters메소드를 오버라이딩하도록 한다.

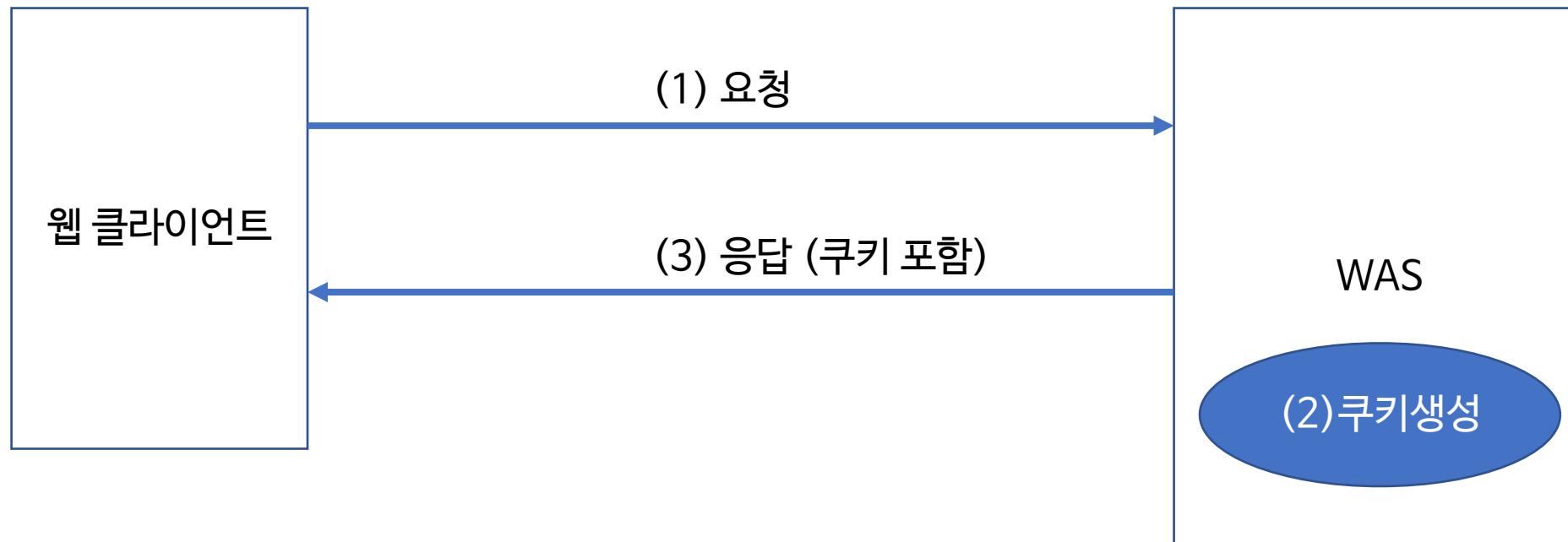
웹에서의 상태 유지 기술

- http프로토콜은 상태 유지가 안되는 프로토콜이다.
 - 이전에 무엇을 했고, 지금 무엇을 했는지에 대한 정보를 갖고 있지 않음
 - 웹 브라우저(클라이언트)의 요청에 대한 응답을 하고 나면 해당 클라이언트와의 연결을 지속하지 않음.
- 상태 유지를 위해 Cookie와 Session기술이 등장함.

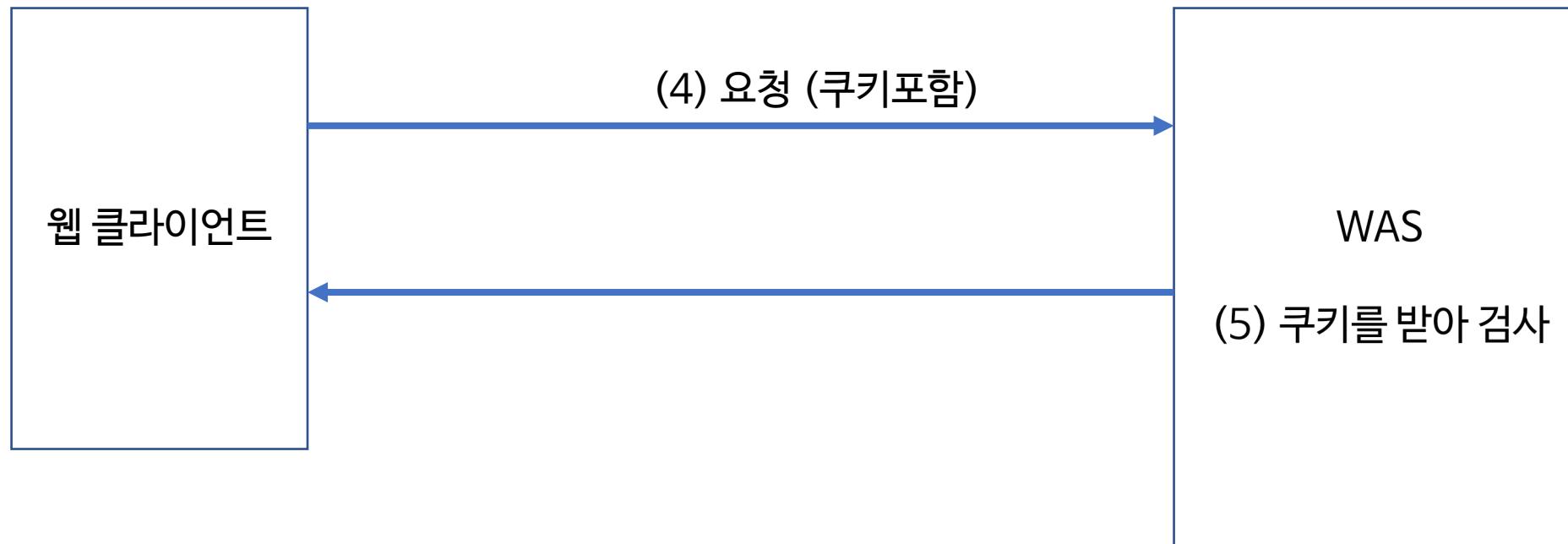
쿠키(Cookie)와 세션(Session)

- 쿠키
 - 사용자 컴퓨터에 저장
 - 저장된 정보를 다른 사람 또는 시스템이 볼 수 있는 단점
 - 유효시간이 지나면 사라짐
- 세션
 - 서버에 저장
 - 서버가 종료되거나 유효시간이 지나면 사라짐

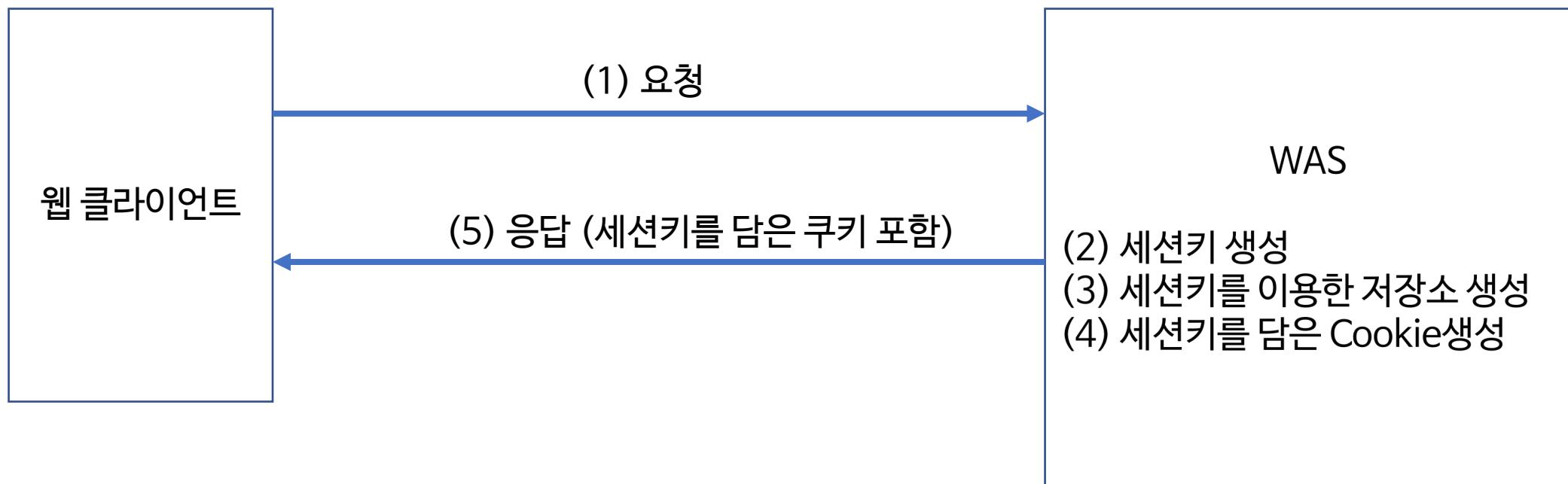
쿠키(Cookie) 동작 이해 1/2



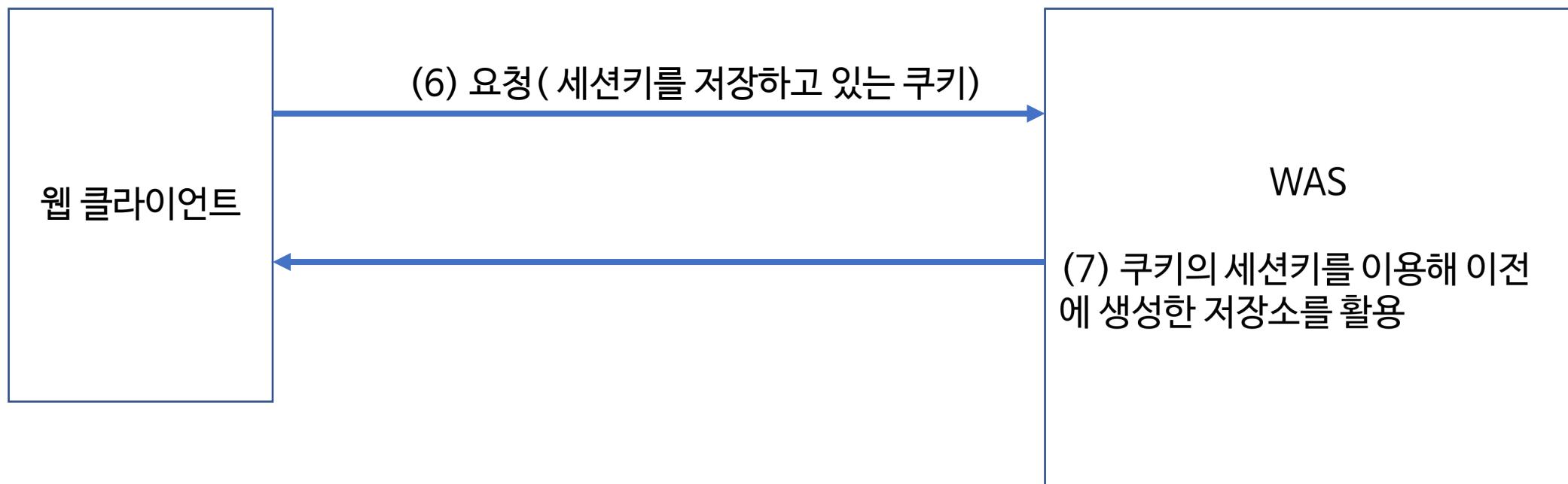
쿠키(Cookie) 동작 이해 2/2



세션의 동작 이해 1/2



세션의 동작 이해 2/2



쿠키 정의

- 정의
 - 클라이언트 단에 저장되는 작은 정보의 단위.
 - 클라이언트에서 생성하고 저장될 수 있고, 서버단에서 전송한 쿠키가 클라이언트에 저장될 수 있다.
- 이용 방법
 - 서버에서 클라이언트의 브라우저로 전송되어 사용자의 컴퓨터에 저장
 - 저장된 쿠키는 다시 해당하는 웹 페이지에 접속 할 때, 브라우저에서 서버로 쿠키를 전송
 - 쿠키는 이름(name)과 값(value)으로 구성된 자료를 저장
 - 이름과 값 외에도 주석(comment), 경로(path), 유효기간(maxage, expiry), 버전(version), 도메인(domain)과 같은 추가적인 정보를 저장

쿠키 정의

- 쿠키는 그 수와 크기에 제한
 - 하나의 쿠키는 4K Byte 크기로 제한
 - 브라우저는 각각의 웹사이트 당 20개의 쿠키를 허용
 - 모든 웹 사이트를 합쳐 최대 300개를 허용
 - 그러므로 클라이언트 당 쿠키의 최대 용량은 1.2M Byte

javax.servlet.http.Cookie

- 서버에서 쿠키 생성, Response의 addCookie메소드를 이용해 클라이언트에게 전송

```
Cookie cookie = new Cookie(이름, 값);
```

```
response.addCookie(cookie);
```

- 쿠키는 (이름, 값)의 쌍 정보를 입력하여 생성
- 쿠키의 이름은 알파벳과 숫자로만 구성되고, 쿠키 값은 공백, 괄호, 등호, 콤마, 콜론, 세미콜론 등은 포함 불가능

javax.servlet.http.Cookie

- 클라이언트가 보낸 쿠키 정보 읽기

```
Cookie[] cookies = request.getCookies();
```

- 쿠키 값이 없으면 null이 반환된다.
- Cookie가 가지고 있는 getName()과 getValue() 메소드를 이용해서 원하는 쿠키정보를 찾아 사용한다.

javax.servlet.http.Cookie

- 클라이언트에게 쿠키 삭제 요청
 - 쿠키를 삭제하는 명령은 없고, maxAge가 0인 같은 이름의 쿠키를 전송한다.

```
Cookie cookie = new Cookie("이름", null);
cookie.setMaxAge(0);
response.addCookie(cookie);
```

javax.servlet.http.Cookie

- 쿠키의 유효기간 설정
 - 메소드 `setMaxAge()`
 - 인자는 유효기간을 나타내는 초 단위의 정수형
 - 만일 유효기간을 0으로 지정하면 쿠키의 삭제
 - 음수를 지정하면 브라우저가 종료될 때 쿠키가 삭제
 - 유효기간을 10분으로 지정하려면
 - `cookie.setMaxAge(10 * 60); //초 단위 : 10분`
 - 1주일로 지정하려면 $(7*24*60*60)$ 로 설정한다.

javax.servlet.http.Cookie

반환형	메소드 이름	메소드 기능
int	getMaxAge()	쿠키의 최대지속 시간을 초단위로 지정 -1 일 경우 브라우저가 종료되면 쿠키를 만료
String	getName()	쿠키의 이름을 스트링으로 반환
String	getValue()	쿠키의 값을 스트링으로 반환
void	setValue(String newValue)	쿠키에 새로운 값을 설정할 때 사용

Spring MVC에서의 Cookie 사용

- @CookieValue 애노테이션 사용
 - 컨트롤러 메소드의 파라미터에서 CookieValue애노테이션을 사용함으로써 원하는 쿠키정보를 파라미터 변수에 담아 사용할 수 있다.

컨트롤러메소드(@CookieValue(value="쿠키이름", required=false, defaultValue="기본값") String 변수명)

세션 정의

- 정의
 - 클라이언트 별로 서버에 저장되는 정보
- 이용 방법
 - 웹 클라이언트가 서버측에 요청을 보내게 되면 서버는 클라이언트를 식별하는 session id를 생성
 - 서버는 session id를 이용해서 key와 value를 이용한 저장소인 HttpSession을 생성
 - 서버는 session id를 저장하고 있는 쿠키를 생성하여 클라이언트에 전송
 - 클라이언트는 서버측에 요청을 보낼때 session id를 가지고 있는 쿠키를 전송
 - 서버는 쿠키에 있는 session id를 이용해서 그 전 요청에서 생성한 HttpSession을 찾고 사용한다

세션 생성 및 얻기

```
HttpSession session = request.getSession();
```

```
HttpSession session = request.getSession(true);
```

- request의 getSession() 메소드는 서버에 생성된 세션이 있다면 세션을 반환하고 없다면 새롭게 세션을 생성하여 반환한다. 새롭게 생성된 세션인지는 HttpSession이 가지고 있는 isNew() 메소드를 통해 알 수 있다.

```
HttpSession session = request.getSession(false);
```

- request의 getSession() 메소드에 파라미터로 false를 전달하면, 이미 생성된 세션이 있다면 반환하고 없으면 null을 반환한다.

세션에 값 저장

- `setAttribute(String name, Object value)`
 - name과 value의 쌍으로 객체 Object를 저장하는 메소드
 - 세션이 유지되는 동안 저장할 자료를 저장

`session.setAttribute(이름, 값)`

세션에 값 조회

- `getAttribute(String name)` 메소드
 - 세션에 저장된 자료는 다시 `getAttribute(String name)` 메소드를 이용해 조회
 - 반환 값은 Object 유형이므로 저장된 객체로 자료유형 변환이 필요
 - 메소드 `setAttribute()`에 이용한 name인 “id”를 알고 있다면 바로 다음과 같이 바로 조회

```
String value = (String) session.getAttribute("id");
```

세션에 값 삭제

- `removeAttribute(String name)` 메소드
 - `name`값에 해당하는 세션 정보를 삭제한다.
- `invalidate()` 메소드
 - 모든 세션 정보를 삭제한다.

javax.servlet.http.HttpSession

반환형	메소드 이름	메소드 기능
long	getCreationTime()	를 세션이 생성된 시간까지 지난 시간을 계산하여 밀리세컨드로 반환
String	getId()	세션에 할당된 유일한 식별자(ID)를 String 타입으로 반환
int	getMaxInactiveInterval()	현재 생성된 세션을 유지하기 위해 설정된 최대 시간을 초의 정수 형으로 반환, 지정하지 않으면 기본 값은 1800초, 즉 30분이며, 기본 값도 서버에서 설정 가능

javax.servlet.http.HttpSession

반환형	메소드 이름	메소드 기능
Object	getAttribute(String name)	name이란 이름에 해당되는 속성값을 Object 타입으로 반환, 해당되는 이름이 없을 경우에는 null을 반환
Enumeration	getAttributeNames()	속성의 이름들을 Enumeration 타입으로 반환
void	invalidate()	현재 생성된 세션을 무효화 시킴
void	removeAttribute(String name)	name으로 지정한 속성의 값을 제거
void	setAttribute(String name, Object value)	name으로 지정한 이름에 value 값을 할당
void	setMaxInactiveInterval(int interval)	세션의 최대 유지시간을 초 단위로 설정
boolean	isNew()	세션이 새로이 만들어졌으면 true, 이미 만들어진 세션이면 false를 반환

javax.servlet.http.HttpSession

- 세션은 클라이언트가 서버에 접속하는 순간 생성
 - 특별히 지정하지 않으면 세션의 유지 시간은 기본 값으로 30분 설정
 - 세션의 유지 시간이란 서버에 접속한 후 서버에 요청을 하지 않는 최대 시간
 - 30분 이상 서버에 전혀 반응을 보이지 않으면 세션이 자동으로 끊어짐.
 - 이 세션 유지 시간은 web.xml 파일에서 설정 가능

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

@SessionAttributes & @ModelAttribute

- @SessionAttributes파라미터로 지정된 이름과 같은 이름이 @ModelAttribute에 지정되어 있을 경우 메소드가 반환되는 값은 세션에 저장된다.
- 아래의 예제는 세션에 값을 초기화하는 목적으로 사용되었다.

```
@SessionAttributes("user")  
  
public class LoginController {  
  
    @ModelAttribute("user")  
  
    public User setUpUserForm() {  
  
        return new User();  
  
    }  
  
}
```

@SessionAttributes & @ModelAttribute

- @SessionAttributes의 파라미터와 같은 이름이 @ModelAttribute에 있을 경우 세션에 있 객체를 가져온 후, 클라이언트로 전송받은 값을 설정한다.

```
@Controller
```

```
@SessionAttributes("user")
```

```
public class LoginController {
```

```
.....
```

```
    @PostMapping("/dologin")
```

```
    public String doLogin(@ModelAttribute("user") User user, Model model) {
```

```
.....
```

```
}
```

```
}
```

@SessionAttribute

- 메소드에 @SessionAttribute가 있을 경우 파라미터로 지정된 이름으로 등록된 세션 정보를 읽어와서 변수에 할당한다.

```
@GetMapping("/info")
public String userInfo(@SessionAttribute("user") User user) {
    //...
    //...
    return "user";
}
```

SessionStatus

- SessionStatus는 컨트롤러 메소드의 파라미터로 사용할 수 있는 스프링 내장 타입이다. 이 오브젝트를 이용하면 현재 컨트롤러의 @SessionAttributes에 의해 저장된 오브젝트를 제거할 수 있다.

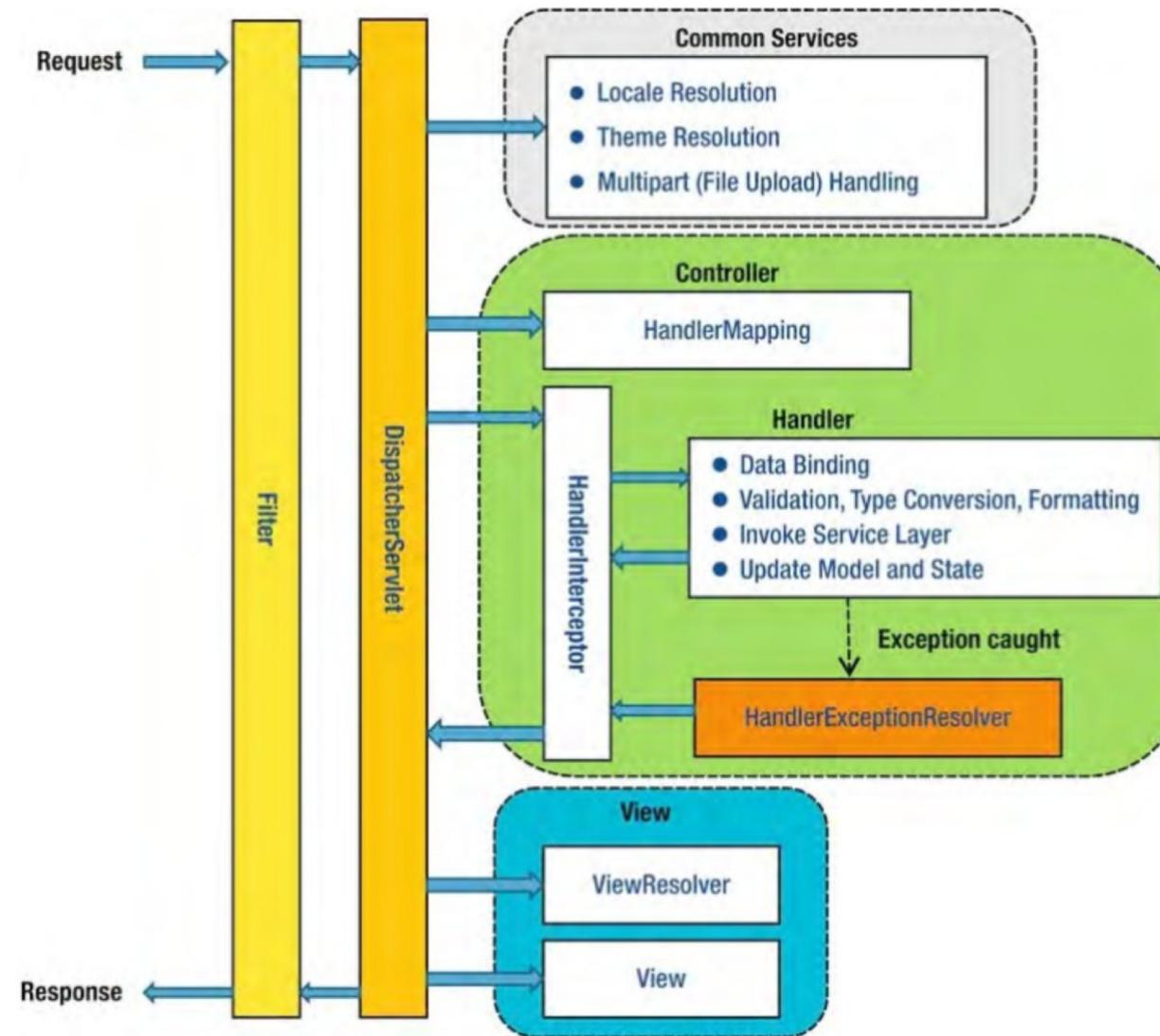
```
@Controller  
@SessionAttributes("user")  
public class UserController {  
    ....  
    @RequestMapping(value = "/user/add", method = RequestMethod.POST)  
    public String submit(@ModelAttribute("user") User user, SessionStatus sessionStatus) {  
        ....  
        sessionStatus.setComplete();  
        ....  
    }  
}
```

Spring MVC – form tag 라이브러리

- `modelAttribute`속성으로 지정된 이름의 객체를 세션에서 읽어와서 form태그로 설정된 태그에 값을 설정한다.

```
<form:form action="login" method="post" modelAttribute="user">  
Email : <form:input path="email" /><br>  
Password : <form:password path="password" /><br>  
<button type="submit">Login</button>  
</form:form>
```

인터셉터(Interceptor)?



인터셉터(Interceptor)?

- Interceptor는 Dispatcher servlet에서 Handler(Controller)로 요청을 보낼때, Handler에서 Dispatcher servlet으로 응답을 보낼때 동작한다.

인터셉터 작성법

- org.springframework.web.servlet.HandlerInterceptor 인터페이스 를 구현한다.
- org.springframework.web.servlet.handler.HandlerInterceptorAdapter 클래스를 상속받는다.
- Java Config를 사용한다면, WebMvcConfigurerAdapter가 가지고 있는 addInterceptors메소드를 오버라이딩하고 등록하는 과정을 거친다.
- xml 설정을 사용한다면, <mvc:interceptors>요소에 인터셉터를 등록한다.

아규먼트 리졸버란?

- 컨트롤러의 메소드의 인자로 사용자가 임의의 값을 전달하는 방법을 제공하고자 할 때 사용된다.
- 예를 들어, 세션에 저장돼 있는 값 중 특정 이름의 값을 메소드 인자로 전달한다.

아규먼트 리졸버 작성방법 1/2

- org.springframework.web.method.support.HandlerMethodArgumentResolver 를 구현한 클래스를 작성한다.
- supportsParameter메소드를 오버라이딩 한 후, 원하는 타입의 인자가 있는지 검사한 후 있을 경우 true가 리턴되도록 한다.
- resolveArgument메소드를 오버라이딩 한 후, 메소드의 인자로 전달할 값을 리턴한다.

아규먼트 리졸버 작성방법 2/2

- Java Config에 설정하는 방법
 - WebMvcConfigurerAdapter를 상속받은 Java Config파일에서 addArgumentResolvers 메소드를 오버라이딩 한 후 원하는 아규먼트 리졸버 클래스 객체를 등록한다.
- xml 파일에 설정하는 방법

```
<mvc:annotation-driven>
```

```
  <mvc:argument-resolvers>
```

```
    <bean class="아규먼트리졸버클래스"></bean>
```

```
  </mvc:argument-resolvers>
```

```
</mvc:annotation-driven>
```

Spring MVC의 기본 ArgumentResolver들

```
private List<HandlerMethodArgumentResolver> getDefaultArgumentResolvers() {
    List<HandlerMethodArgumentResolver> resolvers = new ArrayList<>();

    // Annotation-based argument resolution
    resolvers.add(new RequestParamMethodArgumentResolver(getBeanFactory(), false));
    resolvers.add(new RequestParamMapMethodArgumentResolver());
    resolvers.add(new PathVariableMethodArgumentResolver());
    resolvers.add(new PathVariableMapMethodArgumentResolver());
    resolvers.add(new MatrixVariableMethodArgumentResolver());
    resolvers.add(new MatrixVariableMapMethodArgumentResolver());
    resolvers.add(new ServletModelAttributeMethodProcessor(false));
    resolvers.add(new RequestResponseBodyMethodProcessor(getMessageConverters(), this.requestResponseBodyAdvice));
    resolvers.add(new RequestPartMethodArgumentResolver(getMessageConverters(), this.requestResponseBodyAdvice));
    resolvers.add(new RequestHeaderMethodArgumentResolver(getBeanFactory()));
    resolvers.add(new RequestHeaderMapMethodArgumentResolver());
    resolvers.add(new ServletCookieValueMethodArgumentResolver(getBeanFactory()));
    resolvers.add(new ExpressionValueMethodArgumentResolver(getBeanFactory()));
    resolvers.add(new SessionAttributeMethodArgumentResolver());
    resolvers.add(new RequestAttributeMethodArgumentResolver());

    // Type-based argument resolution
    resolvers.add(new ServletRequestMethodArgumentResolver());
    resolvers.add(new ServletResponseMethodArgumentResolver());
    resolvers.add(new HttpEntityMethodProcessor(getMessageConverters(), this.requestResponseBodyAdvice));
    resolvers.add(new RedirectAttributesMethodArgumentResolver());
    resolvers.add(new ModelMethodProcessor());
    resolvers.add(new MapMethodProcessor());
    resolvers.add(new ErrorsMethodArgumentResolver());
    resolvers.add(new SessionStatusMethodArgumentResolver());
    resolvers.add(new UriComponentsBuilderMethodArgumentResolver());

    // Custom arguments
    if (getCustomArgumentResolvers() != null) {
        resolvers.addAll(getCustomArgumentResolvers());
    }

    // Catch-all
    resolvers.add(new RequestParamMethodArgumentResolver(getBeanFactory(), true));
    resolvers.add(new ServletModelAttributeMethodProcessor(true));

    return resolvers;
}
```

/apps 폴더

tar xvfz apache-tomcat-8.5.24.tar.gz

apache-tomcat-8.5.24

ln - s apache-tomcat-8.5.24 apache-tomcat