



Index

- Introduction
- IOC/DI
- AOP
- Spring MVC
- MyBatis
- Spring Boot
- MSA - Spring Cloud

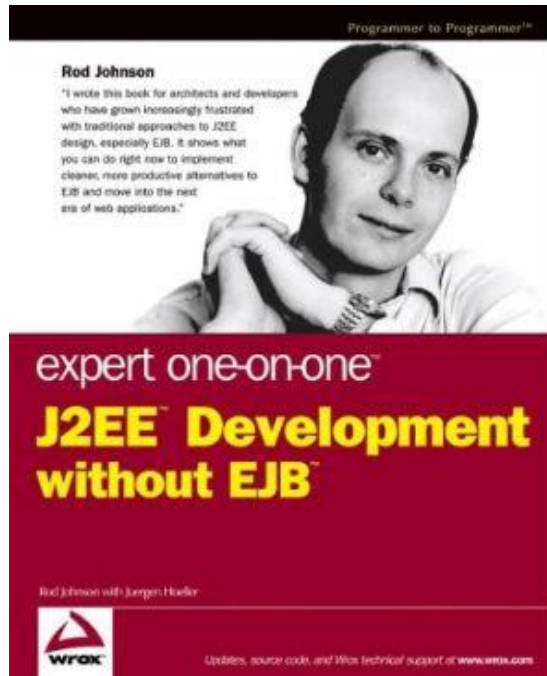
Introduction

Spring Framework Introduction

- 2004년 버전 1.0 출시 이후 2018년 5.x 까지 출시
- Spring : J2EE 겨울 뒤에 봄이 온다는 의미

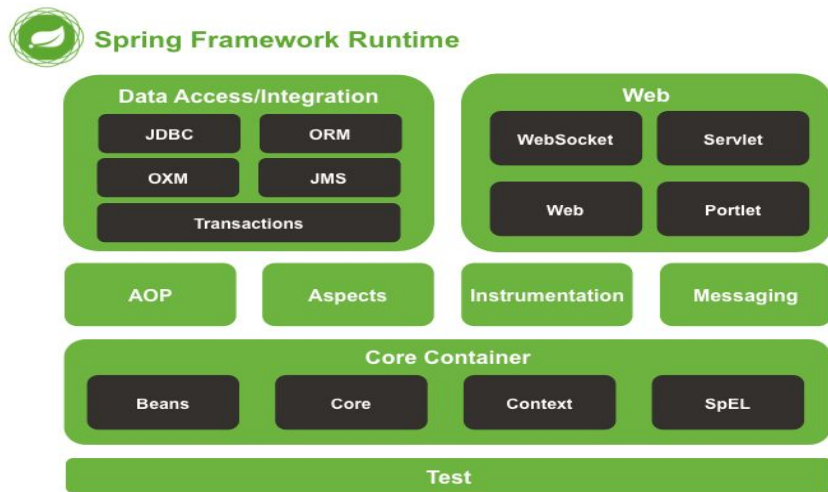
“스프링 프레임워크를 처음 개발할 때부터
변치않는 철학은 여러분의 개발환경이
오래된 인프라든 최신 인프라든 구애받지 않고
최신 프로그래밍 사상과 기법을 활용해
개발할 수 있게 만드는 것이었습니다”

- 유겐할러(개발리더)



Spring Framework Introduction

- 자바 엔터프라이즈 어플리케이션 개발을 위한 오픈소스 프레임워크
- 어플리케이션의 기반(**infrastructure**)을 제공하여 비즈니스 로직에 집중
- POJO (Plain Old Java Object) 기반 프레임워크
- Lightweight Container
- IOC / DI
- AOP
- PSA



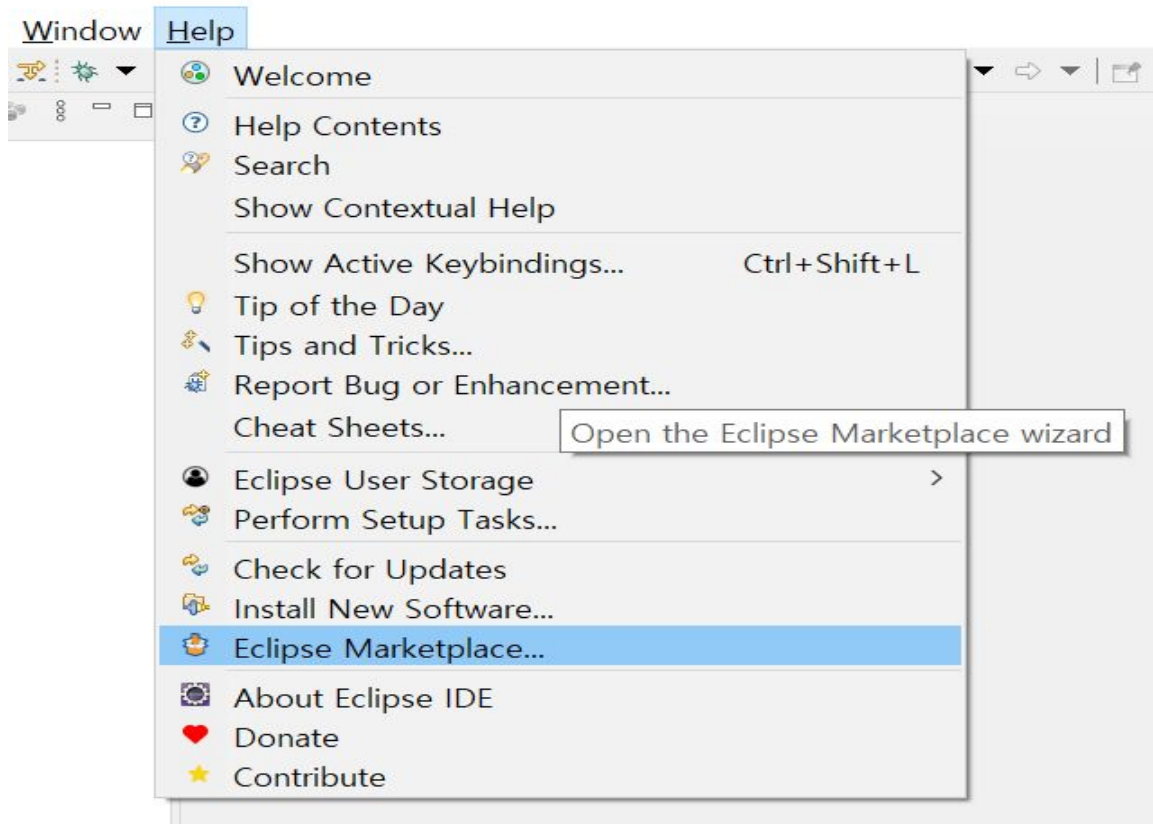
STS : Spring Tool Suite

스프링 툴 스위트(STS)는 스프링 기반 애플리케이션 개발을
위해 최적화된 이클립스 기반 통합 개발 환경을 제공

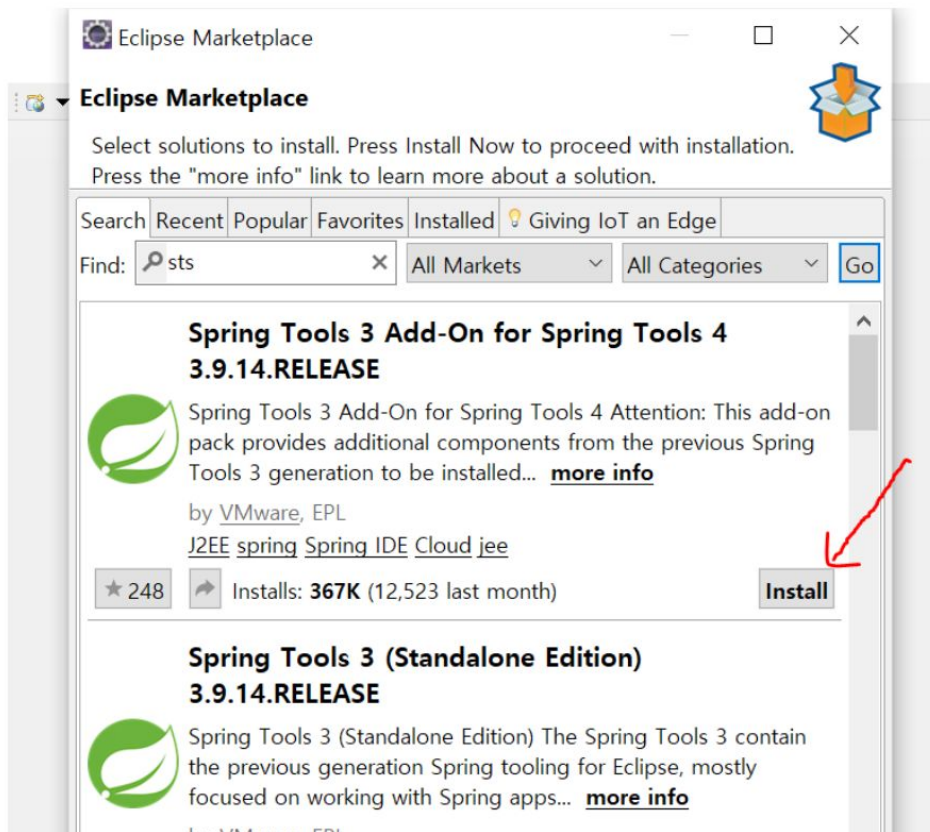
Maven , Git , AspectJ 등과 같은 툴이 기본적으로 내장



Eclipse에서 STS 설치 1



Eclipse에서 STS 설치 2



IOC / DI

IOC / DI

IOC (Inversion Of Control) : 제어의 역전

DI (Dependency Injection) : 의존관계 주입

“컴포넌트를 구성하는 인스턴스의 생성과 의존 관계 연결처리를

IOC 컨테이너에게 위임”

OO Design Principles

- *Loose coupling and high cohesion*

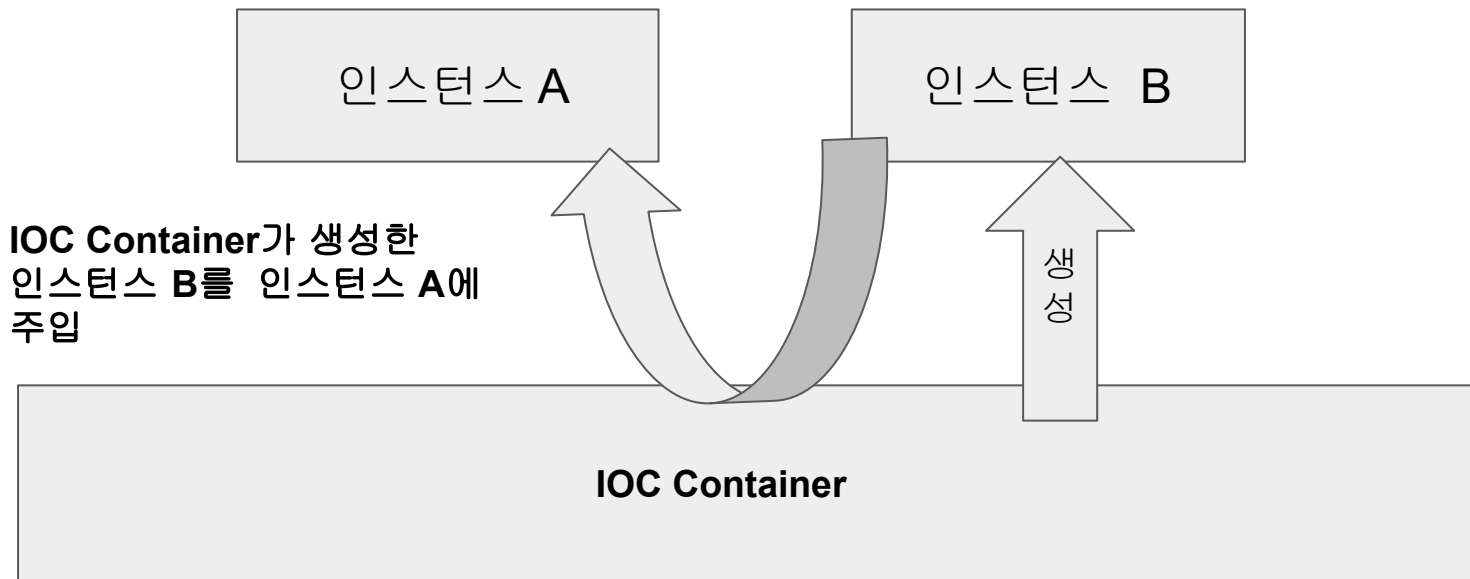
IOC / DI

기존 제어 방식 - 일반적인 의존 관계



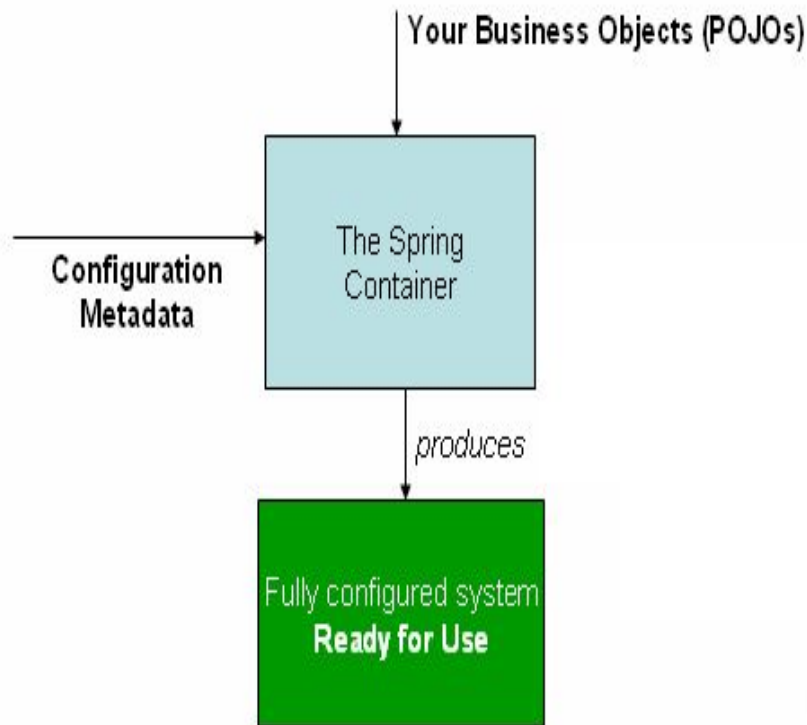
IOC / DI

DI 적용 제어 방식 : IOC 컨테이너 환경의 의존 관계

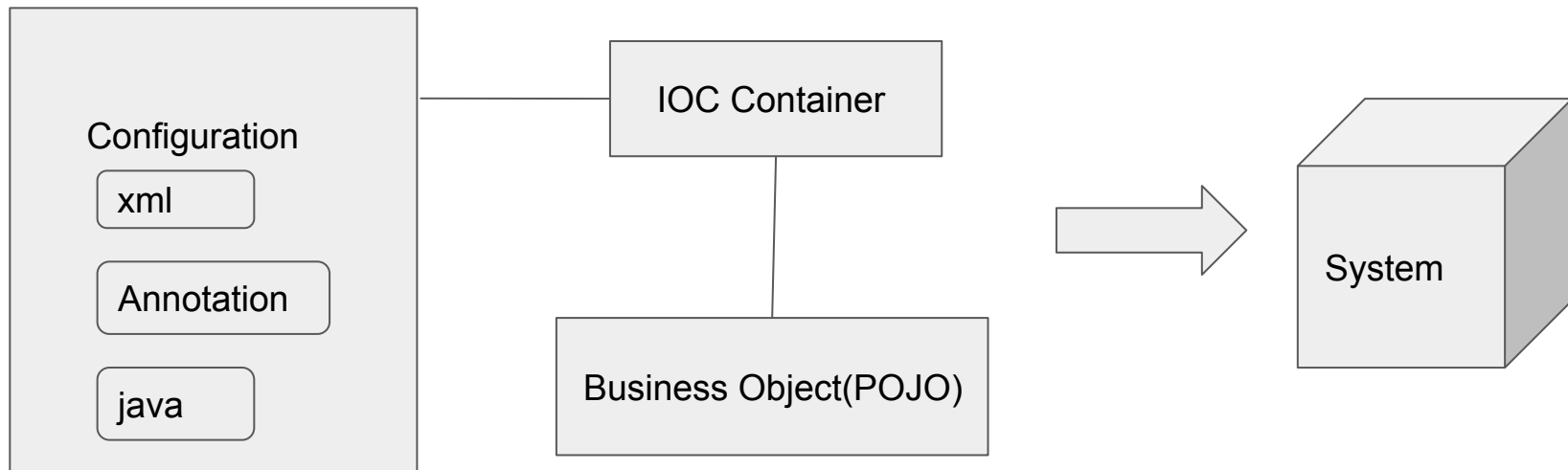


IOC / DI

- 인스턴스의 생명주기 제어
- Bean Scope 관리 (Singleton이 기본)
- 컴포넌트간의 결합도를 낮춤
- 단위 테스트 용이
- AOP 적용하여 공통 기능을 제공



IOC / DI



IOC / DI

Bean 설정 방법

XML 기반 설정 방식	<code><bean id="" class=""></code> 으로 bean 생성 <code><constructor-arg></code> or <code><property></code> 요소를 사용해 의존성주입
Annotation 기반 설정 방식	<code>@Component</code> 계열 어노테이션이 명시된 클래스를 <code>Component-Scan</code> 해서 IOC 컨테이너에 bean 으로 자동 등록
Java 기반 설정 방식	<code>@Configuration</code> 을 자바 클래스에 명시하여 설정을 하고 <code>@Bean</code> 을 사용해 bean 을 정의


XML + Annotation 또는 Java class + Annotation 의 조합으로 주로 설정

IOC / DI

XML 기반 설정 방식의 예 1 : constructor injection

```
<bean id="boardDAO" class="com.mydomain.model.BoardDAOImpl"></bean>  
<bean id="boardService" class="com.mydomain.model.BoardServiceImpl">  
    <constructor-arg ref="boardDAO"/>  
</bean>
```


```
public class BoardServiceImpl implements BoardService {  
    private BoardDAO boardDAO;  
    public BoardServiceImpl(BoardDAO boardDAO) {  
        this.boardDAO=boardDAO;  
    }  
    ...  
}
```



IOC / DI

XML 기반 설정 방식의 예 2 : setter injection

```
<bean id="bankDAO" class="model.KbBankDAOImpl"></bean>
<bean id="bankService" class="model.BankServiceImpl">
    <property name="bankDAO" ref="bankDAO" />
</bean>
```



```
public class BankServiceImpl implements BankService {
    private BankDAO bankDAO;
    @Override
    public void setBankDAO(BankDAO bankDAO) {
        this.bankDAO = bankDAO;
    }
}
```

IOC / DI

Annotation 기반 설정 방식의 예

```
<context:component-scan base-package="com.mydomain"/>
```

```
package com.mydomain.model;

@Service
public class BankServiceImpl implements BankService {
    @Autowired
    private BankDAO bankDAO;
    ...
}
```

IOC / DI

자바 기반 설정 방식의 예

```
@Configuration
public class SpringConfig {
    @Bean
    public MemberService memberService(){
        return new MemberServiceImpl();
    }
}
```

```
AnnotationConfigApplicationContext context
= new AnnotationConfigApplicationContext(SpringConfig.class);
MemberService memberService= context.getBean(MemberService.class);
```

IOC / DI

컴포넌트 스캔 : 지정한 특정 패키지 하위의 클래스 탐색한 후 IOC 컨테이너에 객체 생성 후 등록

XML 방식	<code><context:component-scan base-package="com.example.demo"/></code>
Java Annotation 방식	<code>@ComponentScan(basePackages="com.example.demo")</code>

IOC / DI

컴포넌트 스캔 대상 주요 어노테이션

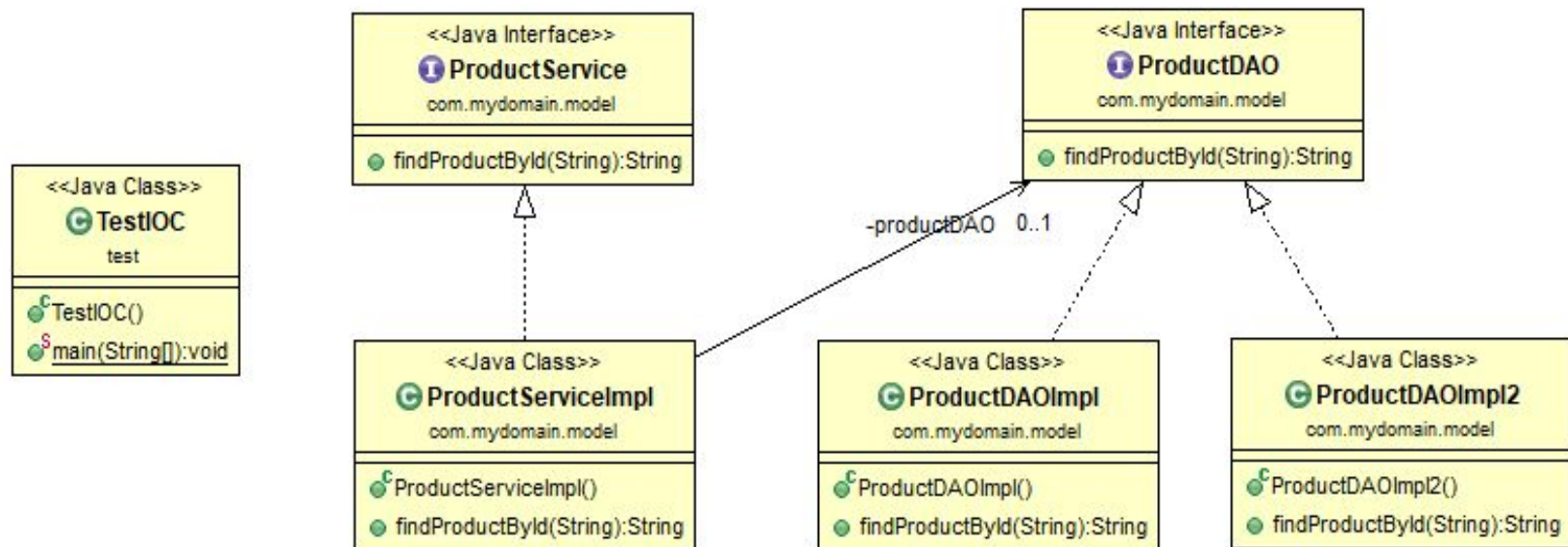
@Controller	MVC 의 Controller 역할을 하는 클래스에 명시하는 어노테이션
@Service	비즈니스 로직 처리하는 클래스에 명시하는 어노테이션 트랜잭션 처리는 @Service 명시 클래스에서 하도록 한다
@Repository	영속적 데이터 처리를 위한 클래스에 명시하는 어노테이션
@Component	모든 컴포넌트에 대한 제너릭 스테레오타입 위 세가지 경우에 해당하지 않는 기타 지원 클래스에 사용

IOC / DI

Dependency Injection 관련 주요 어노테이션

@Autowired	의존 대상 객체를 타입으로 검색해 주입 만약 동일한 타입의 객체가 여러개일 경우 NoUniqueBeanException 발생 추가적으로 @Qualifier("bean name") 이 필요
@Resource	@Resource : 의존 대상 객체를 타입으로 검색해 주입 @Resource(name="bean name") : 의존 대상 객체를 이름으로 검색해 주입

IOC / DI Exercise



AOP

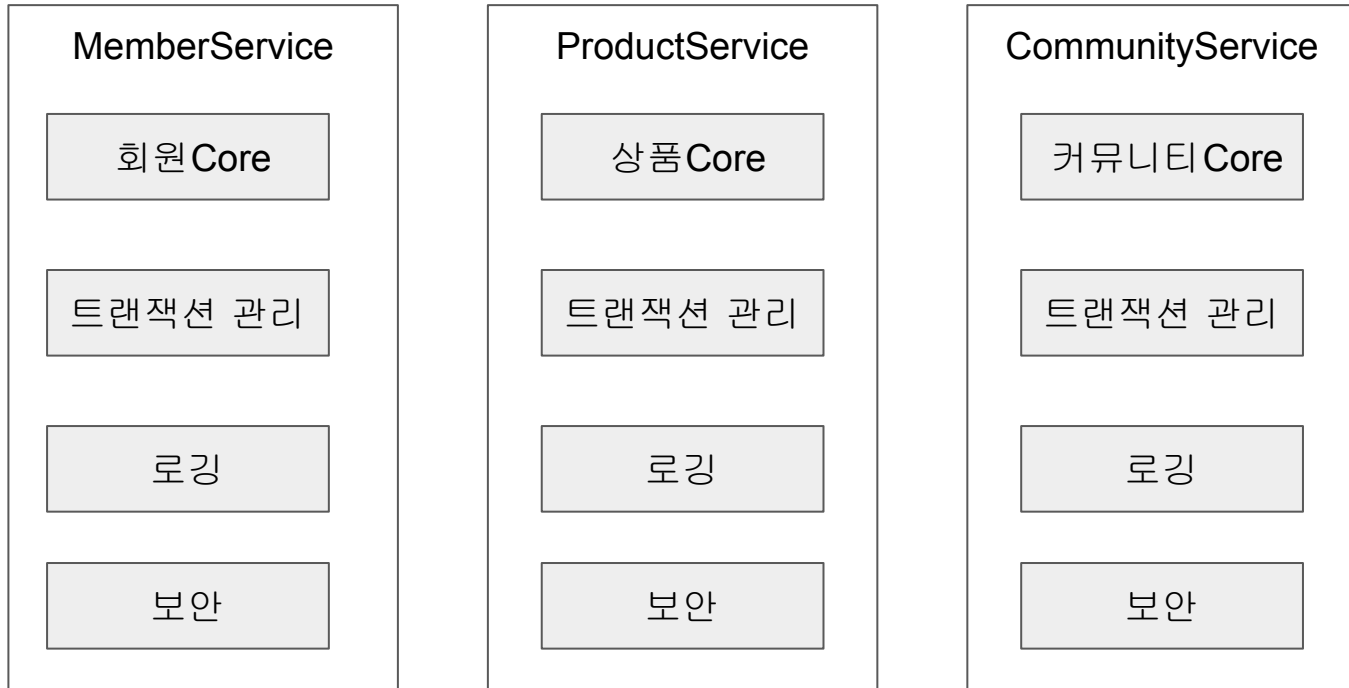
AOP 개요 1

- Aspect Oriented Programming , 관점 지향 프로그래밍
- 시스템을 핵심관심사(Core Concern) 와 횡단관심사(Cross-cutting Concern)로 구분하여 설계와 구현을 한다
- 핵심관심사(Core Concern)란 시스템의 목적에 해당하는 주요 로직 부분을 말한다
- 횡단관심사(Cross-cutting Concern)란 시스템의 여러 부분에 걸쳐 공통적이고 반복적으로 필요로 하는 처리내용을 말한다

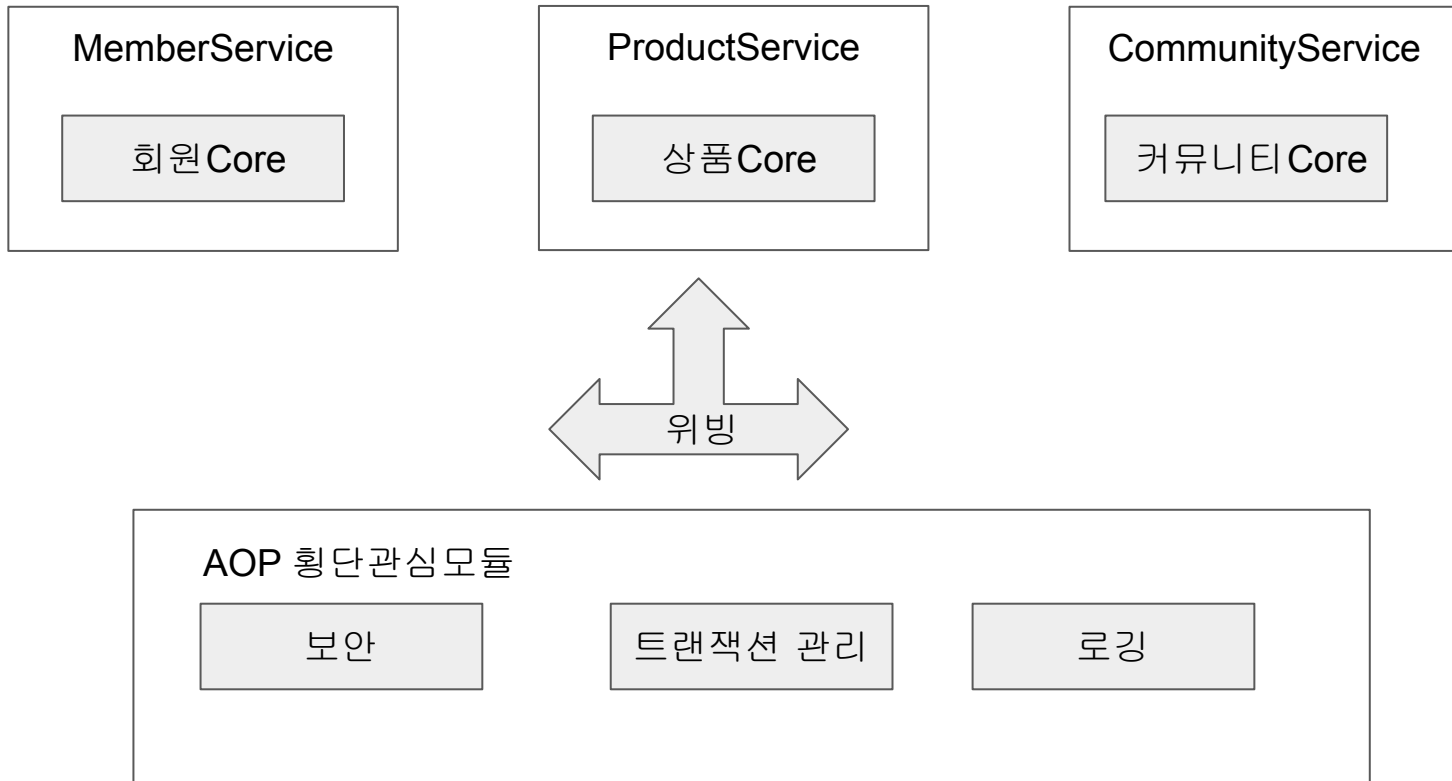
AOP 개요 2

- AOP는 시스템의 여러 영역에 걸쳐 공통적이고 반복적으로 적용된 횡단관심사(**Cross-Cutting Concern**)를 분리하여 별도의 모듈에서 설계, 구현, 운영하는 프로그래밍 기법이다
- 대표적인 횡단관심사는 로깅, 보안, 트랜잭션 관리, 예외 처리등이 있다
- AOP는 “OOP를 더욱 OOP 답게 한다”

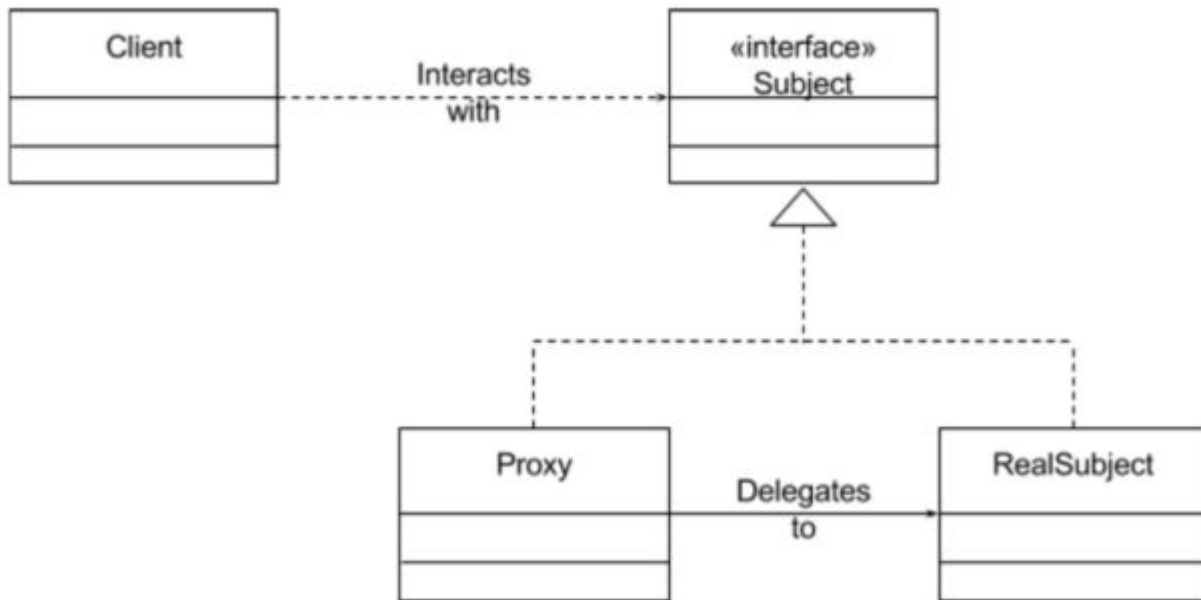
AOP 적용 전



AOP 적용 후

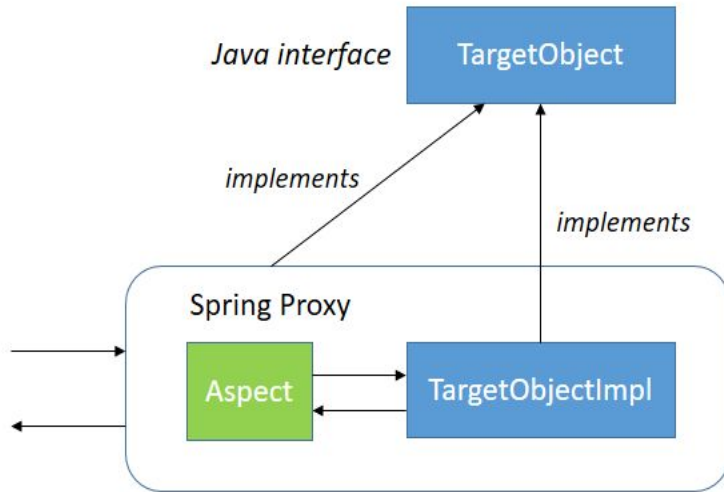


Proxy Design Pattern

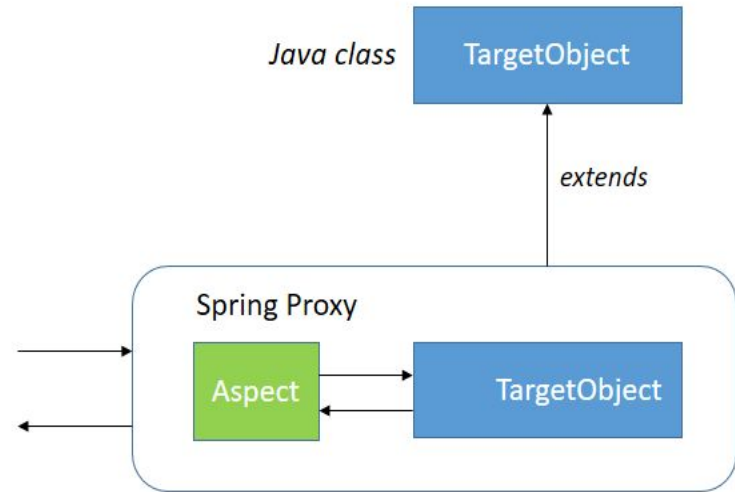


Spring AOP Process

JDK Proxy (interface based)



CGLib Proxy (class based)



AOP 주요 용어

Aspect	AOP의 단위가 되는 횡단관심사를 의미 ex) 로깅,트랜잭션관리 등
JoinPoint	횡단관심사가 실행될 지점
Advice	횡단관심사를 처리하는 부분, 특정 JoinPoint 에서 실행되는 코드로서 before, after returning , after throwing, after , around advice 가 있다
Pointcut	여러 JoinPoint 중 실제 어드바이스를 적용할 곳을 선별하기 위한 표현식
Weaving	애플리케이션의 적절한 지점에 aspect 를 적용하는 것을 말함
Target	Aspect 가 적용된 객체를 말한다

Spring AOP Advice 유형

Before	메서드 실행 전에 실행하는 Advice
After Returning	메서드 정상 실행 후 실행하는 Advice
After Throwing	메서드 실행시 예외 발생시 실행하는 Advice
After	메서드 정상 실행 또는 예외 발생 상관없이 실행하는 Advice
Around	위 네가지 Advice를 모두 포함 , 모든 시점에서 실행할 수 있는 Advice

Spring AOP 환경설정

→ Maven pom.xml

```
<dependency>  
<groupId>org.springframework</groupId>  
<artifactId>spring-context</artifactId>  
</dependency>
```

```
<dependency>  
<groupId>org.aspectj</groupId>  
<artifactId>aspectjweaver</artifactId>  
</dependency>
```

Spring AOP Pointcut AspectJ 표현식

→ AspectJ execution : 가장 많이 사용되는 지시자

```
execution(* com.exam.*Service.find*(..))
```

리턴타입

패키지

클래스

메서드

매개변수리스트

`com.exam` 하위의 `Service`로 마치는 클래스의 리턴타입 유형에 관계없이 `find` 로 시작하는 메서드명을 가진 매개변수 `0~*` 인 메서드를 **AOP** 적용대상으로 한다

Spring AOP Pointcut AspectJ 표현식 사용 예

→ `execution(String com.exam.service..*.search*(..))`

`com.exam.service` 패키지 또는 그 하위 패키지의 모든 클래스 중 `String` 리턴타입인 `search` 이름으로 시작하는 모든 메서드를 적용대상으로 한다

→ `within(com.exam.service.*)`

`com.exam.service` 패키지 하위의 모든 클래스의 모든 메서드를 대상으로 한다

→ `bean(*Service)`

IOC 컨테이너에 관리되는 `bean`이름이 `Service`로 끝나는 `bean`의 모든 메서드를 대상으로 한다

Spring AOP XML 설정 기반 구현

```
<bean id="aroundLoggingAspect " class="mydomain.common.AroundLoggingAspect "/>
<aop:config>
    <aop:aspect ref="aroundLoggingAspect">
        <aop:around method="aroundLogging"
            pointcut="execution(public * mydomain.model.*Service.*(..))"/>
    </aop:aspect>
</aop:config>
```

```
public class AroundLoggingAspect {

    public Object aroundLogging(ProceedingJoinPoint point) throws Throwable{
        // implements ..
    }
}
```

Spring AOP Annotation 기반 설정 구현

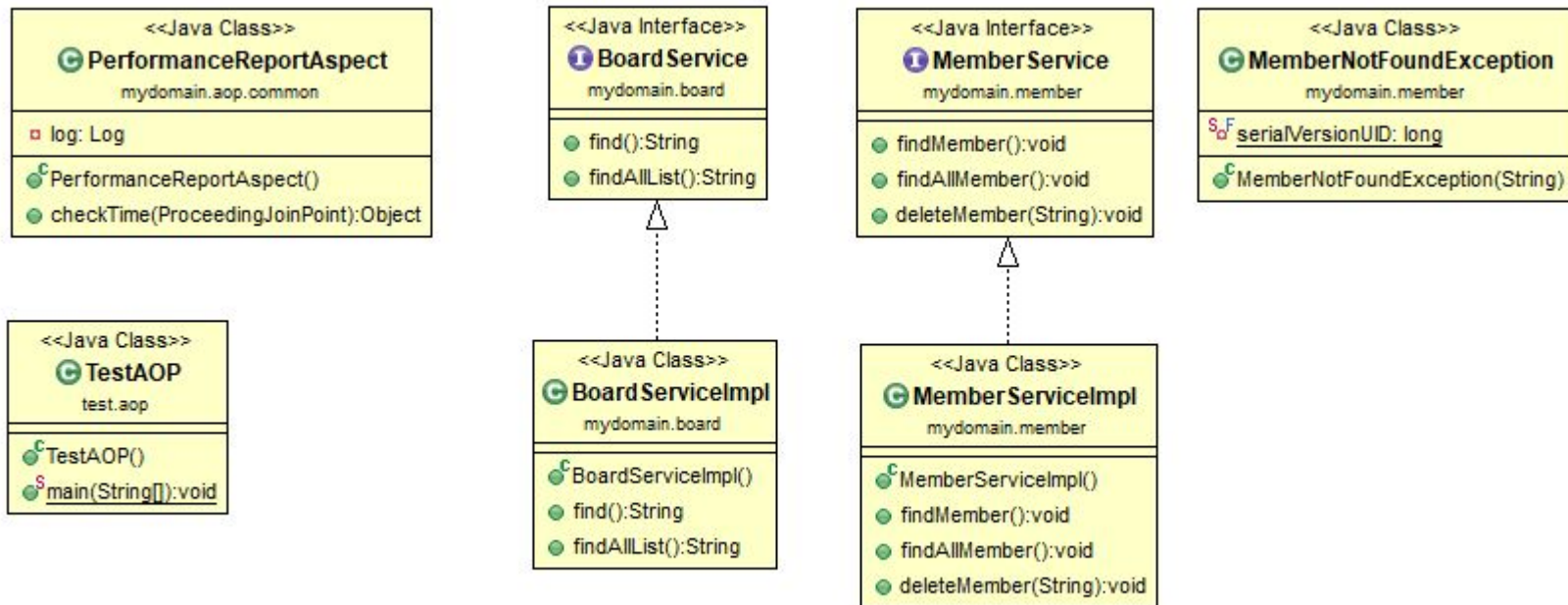
```
<context:component-scan base-package="mydomain"/>  
<aop:aspectj-autoproxy/>
```

```
@Component  
@Aspect  
public class PerformanceReportAspect {  
  
    @Around("execution(public * mydomain..*Service.find*(..))")  
    public Object checkTime(ProceedingJoinPoint point) throws Throwable {  
        // implementation ..  
    }  
}
```

Spring AOP Annotation

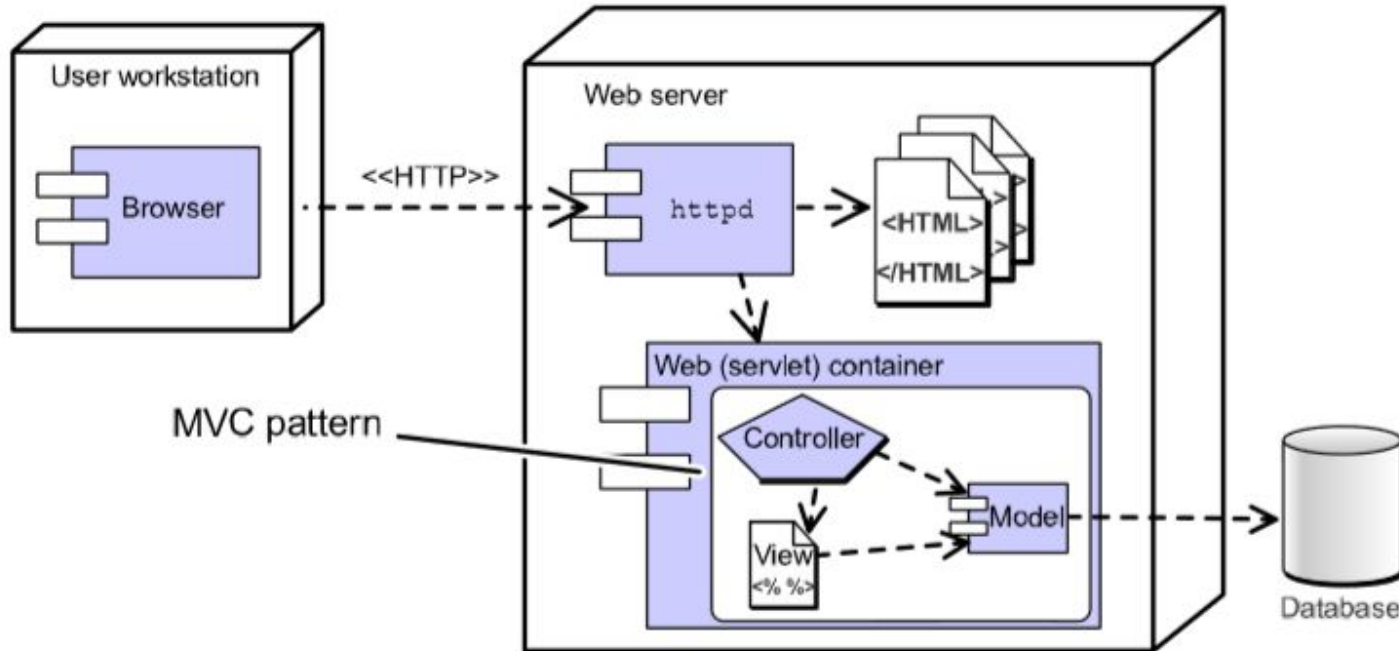
@Aspect	IOC 컨테이너에 AOP Aspect 컴포넌트임을 알려주는 어노테이션
@EnableAspectJAutoProxy	자바 기반 설정 방식에서 AOP 활성화 시키기 위한 어노테이션
@Before	메서드 실행 전에 실행하는 Advice 를 위한 어노테이션
@AfterReturning	메서드 정상 실행 후 실행하는 Advice 를 위한 어노테이션
@AfterThrowing	메서드 실행시 예외 발생시 실행하는 Advice 를 위한 어노테이션
@After	메서드 정상 실행 또는 예외 발생 상관없이 실행하는 Advice 를 위한 어노테이션
@Around	모든 Advice 시점에서 실행할 수 있는 Advice 를 위한 어노테이션
@Transactional	선언적 트랜잭션 제어를 지원하는 어노테이션 해당 메서드가 정상 종료하면 commit , 실패해서 예외 발생하면 rollback
@Secured	스프링시큐리티에서 인가 기능을 지원하는 어노테이션

Spring AOP Exercise



Spring MVC

Model2 Architecture MVC Pattern



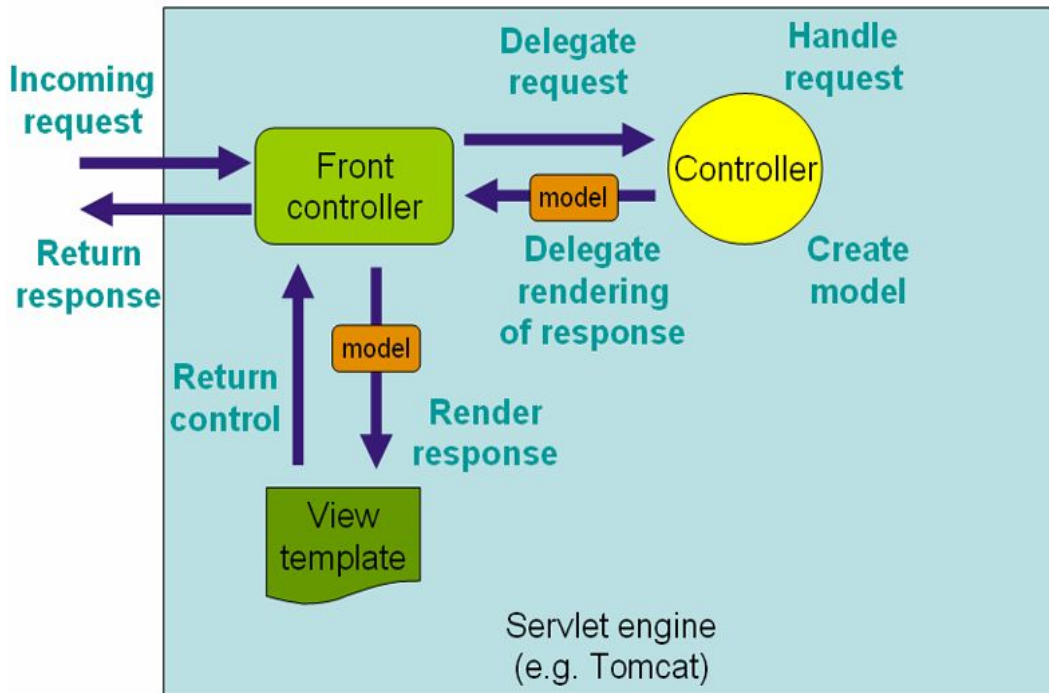
Model2 Architecture MVC Pattern

Model 2 Architecture 에서 MVC 요소별 역할

Model	어플리케이션의 Business Logic 과 Data Access Logic 을 담당
View	사용자 폼 데이터를 컨트롤러에 전달 클라이언트에 응답할 정보(화면 또는 데이터)를 생성해서 화면에 표현
Controller	클라이언트의 요청을 분석한 후 그에 의거하여 Model 과 View 와 연동한다 요청과 응답의 전반적인 처리 흐름을 제어한다

Model2 Architecture Front Controller Pattern

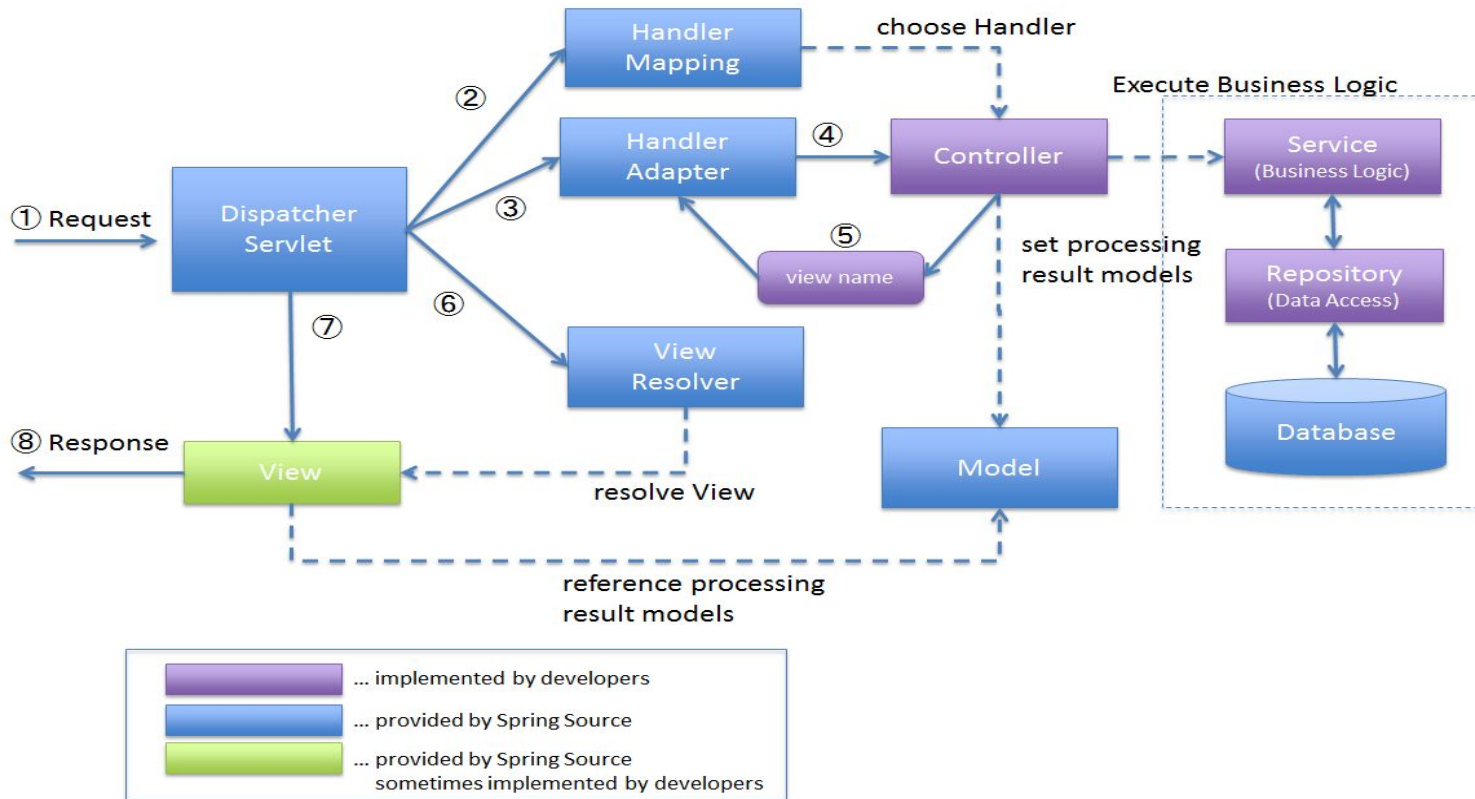
- 모든 클라이언트의 요청을 대표 창구인 FrontController (ex- DispatcherServlet)로 집중시키는 패턴
- 여러 컨트롤러들에 반복적으로 적용되는 Security, Tracking과 같은 공통로직을 한 곳에서 처리하고 제어
- URL 구성이 간편
- 구체적인 업무 로직은 담당 Handler Controller들이 수행한다



Spring MVC 특징

- POJO 기반 개발 , 즉 특정 기술에 대한 종속성 없이 개발
- @MVC 생산성 높은 웹 어플리케이션 개발을 위해 다양한 어노테이션을 지원
- Servlet API Abstraction
- 컨트롤러 메서드의 유연한 정의를 지원
- Spring IOC , AOP 적용

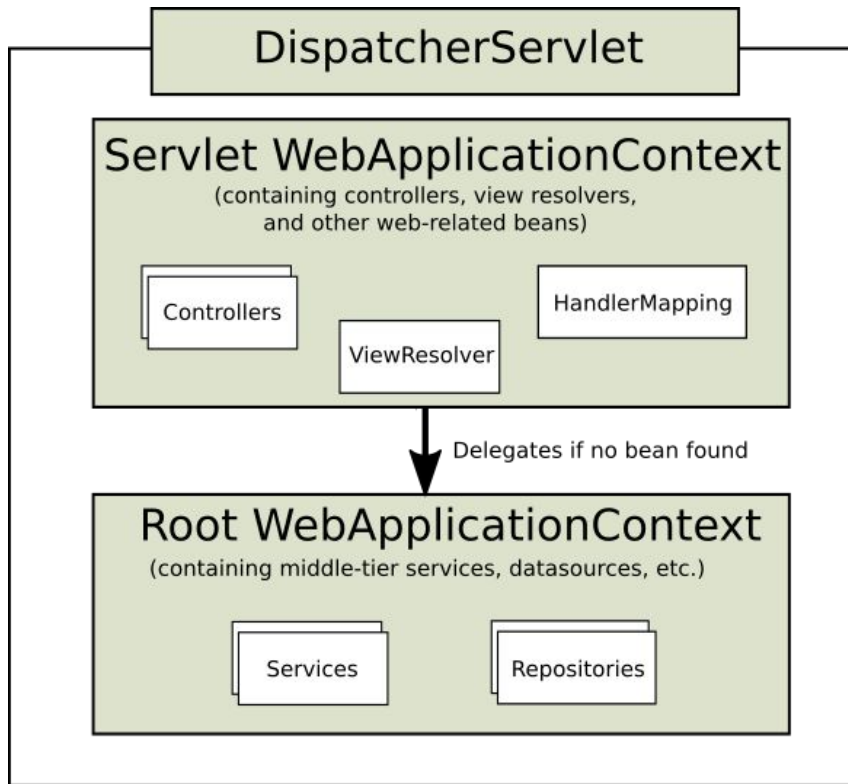
Spring MVC Processing Sequence



Spring MVC 구성 요소

DispatcherServlet	Front Controller 로 모든 클라이언트의 요청을 수신하고 처리를 제어
HandlerMapping	클라이언트의 요청을 처리할 담당 컨트롤러를 결정
HandlerAdapter	담당 컨트롤러 handler 메서드의 호출을 담당
Controller	클라이언트의 요청을 처리 , Model 계층과 연동 후 결과 정보를 ModelAndView 객체에 저장해서 반환
ViewResolver	클라이언트에게 응답하기 위한 View 를 선택하는 방식을 제공
View	클라이언트에게 응답하는 로직을 정의
ModelAndView	응답할 View 정보와 View 에 전달하기 위한 데이터를 가지고 있는 객체

Spring MVC Context Hierarchy



Spring MVC 환경 설정

→ Maven pom.xml

```
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-webmvc</artifactId>  
  <version>${org.springframework-version}</version>  
</dependency>
```


Spring MVC 환경 설정

→ DD : web.xml

```
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-*.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Spring MVC 환경 설정

→ Spring MVC configuration

```
<context:component-scan base-package="com.mydomain" />
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/views/"></property>
<property name="suffix" value=".jsp"></property>
</bean>
<mvc:annotation-driven />
```

Spring MVC Controller 구현 예

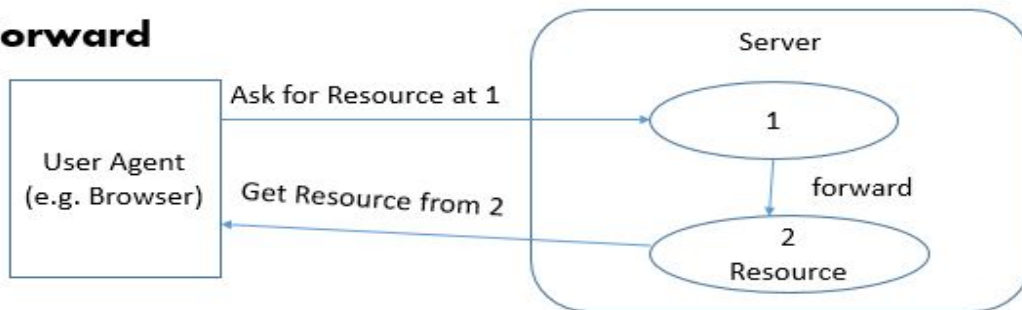
```
@Controller
public class MyAnnotationController {
    @RequestMapping("home")
    public ModelAndView home() {
        return new ModelAndView("result","info","응답데이터");
    }
    @RequestMapping("home2")
    public String home2(Model model) {
        model.addAttribute("info","응답데이터");
        return "result";
    }
    @PostMapping("register")
    public String registerPerson(PersonVO vo) {
        // DAO와 연동하여 db에 등록
        return "result";
    }
}
```

Spring MVC 주요 Annotation

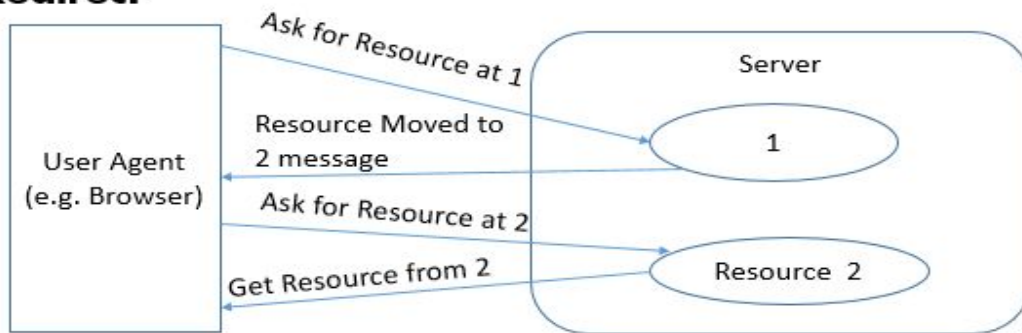
@RequestMapping	요청 경로 지정 , HttpRequest Method (GET,POST,PUT) 등을 지정할 수 있다
@PostMapping	요청 경로 지정 , HttpRequest Method Post 방식에만 응답한다 4.3 버전 이상에서 지원. 같은 원리로 @GetMapping , @DeleteMapping 등이 있다
@RequestParam	요청 파라미터 값을 가져오기 위한 어노테이션
@RequestBody	요청 Body 정보를 가져오기 위한 어노테이션
@PathVariable	특정 경로 변수 값을 취득한다
@ModelAttribute	Model에 저장하는 객체를 반환한다
@ResponseBody	응답 본문에 직렬화하는 객체를 반환. Ajax 응답시 사용한다

Spring MVC View 이동 지정방식 Forward vs Redirect

Forward



Redirect



Spring MVC Controller Redirect 예

```
@Controller
public class MyAnnotationController {
    @RequestMapping("/")
    public String home() {
        return "home"; // forward 방식으로 ViewResolver에 의해 /WEB-INF/views/home.jsp로 응답
    }
    @PostMapping("register")
    public String registerProduct(Product product, RedirectAttributes redirectAttributes) {
        redirectAttributes.addAttribute("id", product.getId());
        return "redirect:/product/register-result";
        //리다이렉트 될 URL이 /product/register-result?id=1 과 같은 형식으로 만들어져 응답
    }
}
```

Spring MVC



Asynchronous Javascript And Xml

비동기 웹어플리케이션 제작을 위한 웹개발기법

페이지를 로딩하는 것이 아니라
필요한 데이터만 전달받아 화면에 표현

Maven : pom.xml

```
<dependency>  
<groupId>com.fasterxml.jackson.core</groupId>  
<artifactId>jackson-databind</artifactId>  
<version>2.8.3</version>  
</dependency>
```

Spring MVC Ajax 연동 예

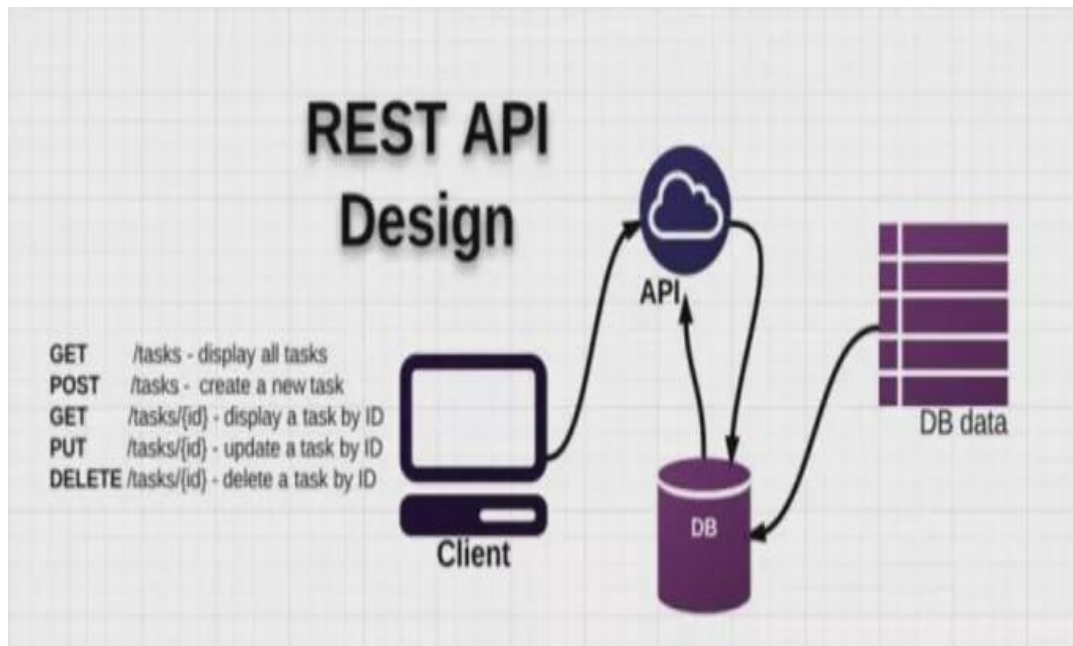
View

```
$("#testAjaxBtn1").click(function(){  
$.ajax({  
    type:"get",  
    url:"testAjax1.do",  
    dataType:"json",  
    success:function(car){  
        alert(car.model+" "+car.price);  
    }  
});  
});
```

Controller

```
@RequestMapping("testAjax1.do")  
@ResponseBody  
public CarVO testAjaxJSONObject() {  
    return new CarVO("sm5",500);  
}
```


Spring MVC - REST



출처 <https://medium.com/@rachna3singhal/restful-api-design-95b4a8630c26>

- REpresentational State Transfer
- 자원에 고유한 URI(주소)를 부여해 활용
- HTTP GET(조회) , POST(생성) , PUT(수정), DELETE(삭제)
- 다양한 클라이언트에게 서비스 API를 제공
- 서비스별 분리 , 통합에 대한 표준화된 방법을 제공
- SOA , MSA

Spring MVC REST

View

```
$.ajaxSetup({
    success:function(result){
        alert(result);
    },
    error: function (jqXHR) {
        alert(jqXHR.status+" "+jqXHR.responseText);
    }
});//ajaxSetup

$("#testGetBtn").click(function(){
$.ajax({
    type:"get",
    url:"products/"+$("#pid").val(),

    success:function(product){

    }
});//ajax
});//click
```

Controller

```
@RestController // @Controller + @ResponseBody
public class ProductController {

    @Resource
    private ProductMapper productMapper;

    @GetMapping("/products/{id}")
    public ResponseEntity findProductById(@PathVariable("id") String id) {
        ProductVO product=productMapper.findProductById(id);
        if (product == null) {
            return new ResponseEntity("not found!!", HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity(product, HttpStatus.OK);
    }

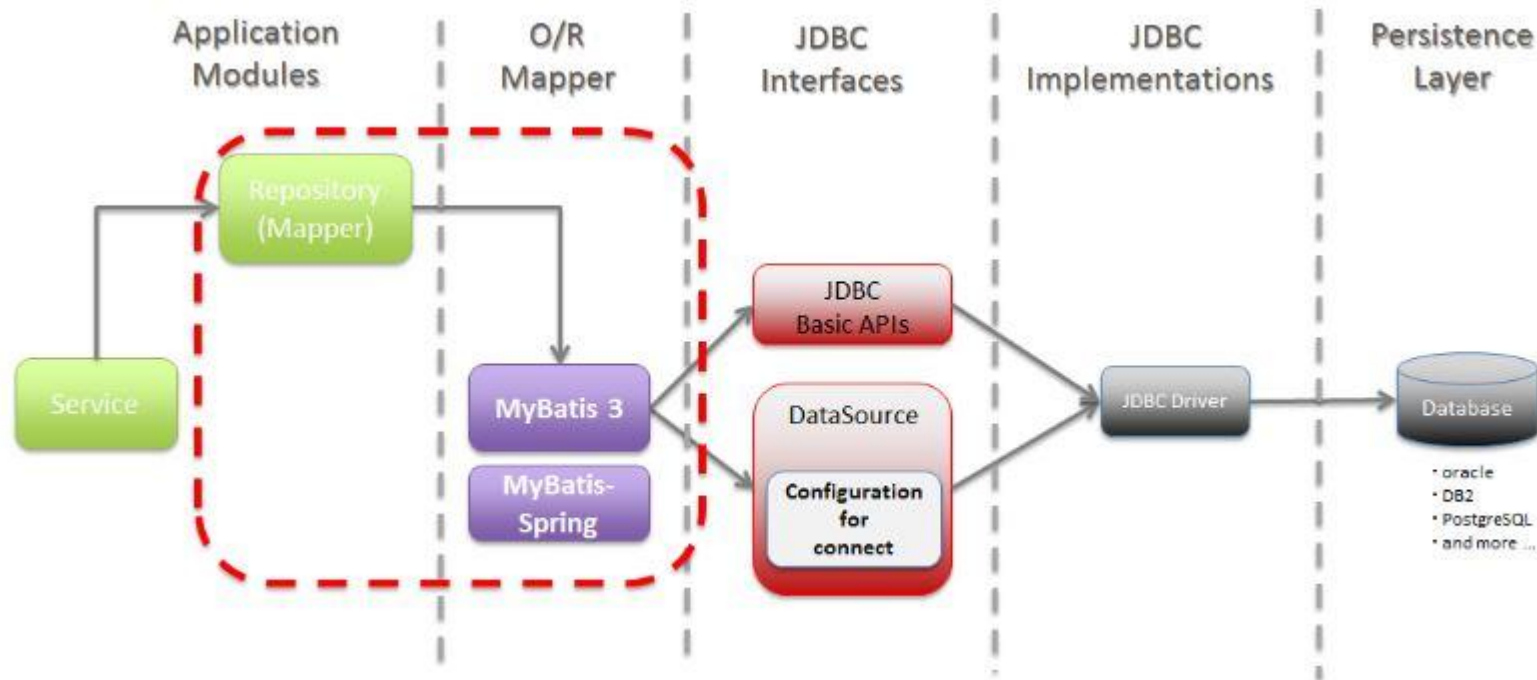
}
```



MyBatis

- Open Source Data Access Framework
- DB Connection , SQL , Code 분리
- 공통된 JDBC 로직을 Mybatis 가 처리
- 개발자는 비즈니스 로직에 집중

MyBatis Area



출처 <https://terasolunaorg.github.io>

MyBatis Spring 연동 환경설정

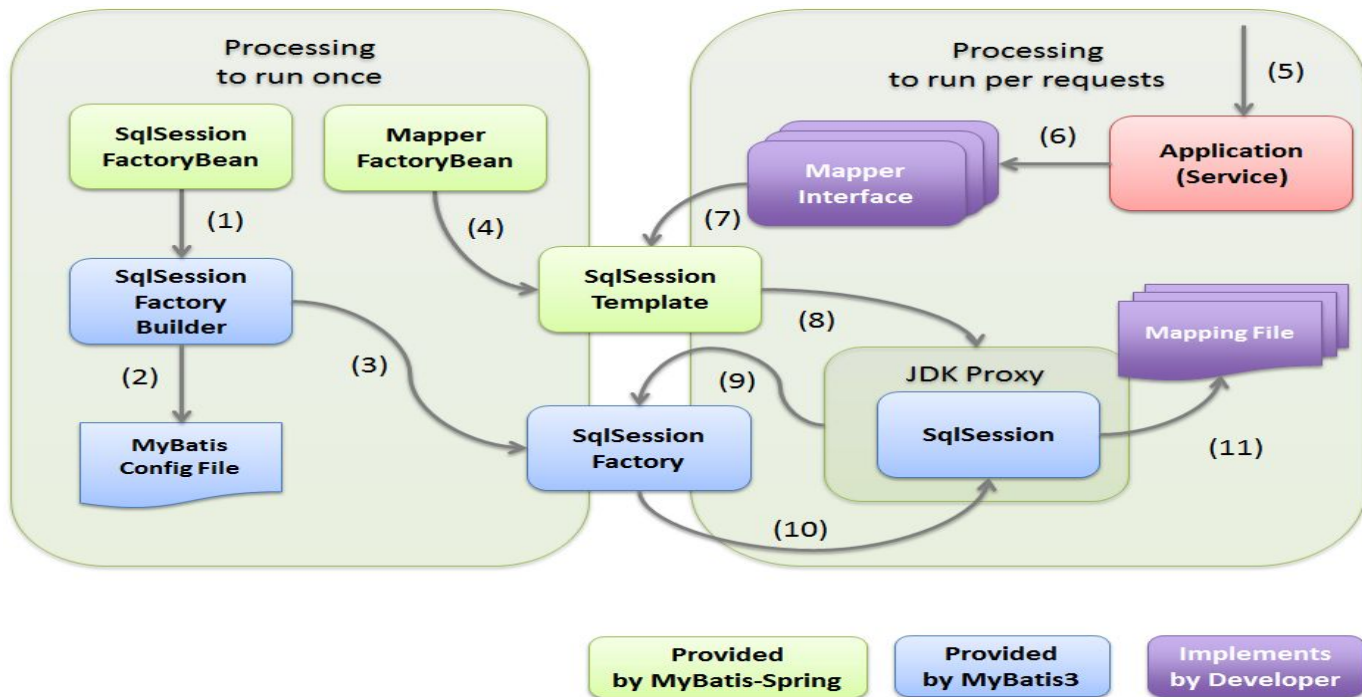
maven pom.xml

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>4.3.14.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.4.0</version>
</dependency>
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-dbcp2</artifactId>
  <version>2.1.1</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>1.3.0</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>4.3.14.RELEASE</version>
</dependency>
```

spring-mybatis configuration

```
<bean id="dbcp" class="org.apache.commons.dbcp2.BasicDataSource">
  <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
  <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:xe" />
  <property name="username" value="scott"/>
  <property name="password" value="tiger"/>
</bean>
<mybatis-spring:scan base-package="com.mydomain.model.mapper"/>
<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dbcp"/>
  <property name="typeAliasesPackage" value="com.mydomain.model"/>
  <property name="configuration">
    <bean class="org.apache.ibatis.session.Configuration">
      <property name="mapUnderscoreToCamelCase" value="true"/>
    </bean>
  </property>
</bean>
<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
  <constructor-arg ref="sqlSessionFactoryBean"/>
</bean>
```

MyBatis & Spring Framework Relationship



MyBatis Spring 구현 예

Mapper Interface

```
package com.mydomain.model.mapper;  
  
import org.apache.ibatis.annotations.Mapper;  
  
@Mapper  
public interface MemberMapper {  
    public MemberVO findMemberById(String id);  
  
    public int register(MemberVO vo);  
  
    public int getMemberTotalCount();  
}
```

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="com.mydomain.model.mapper.MemberMapper">  
  
    <select id="findMemberById" resultType="memberVO">  
        SELECT id,password,name,address FROM tx_member  
        WHERE id=#{value}  
    </select>  
  
    <insert id="register" parameterType="memberVO">  
        INSERT INTO tx_member(id,password,name,address)  
        VALUES(#{id},#{password},#{name},#{address})  
    </insert>  
  
    <select id="getMemberTotalCount" resultType="int">  
        SELECT count(*) FROM tx_member  
    </select>  
</mapper>
```

MyBatis Mapper Statement

```
<select id="findMemberByIdAndAddress" parameterType="memberVO" resultType="memberVO">
    SELECT id,password,name,email,address FROM tx_member
    WHERE id=#{id} and password=#{password}
</select>

<update id="updateMember" parameterType="map">
    UPDATE tx_member SET address=#{addr} WHERE name=#{name}
</update>
```

id	Mapper Sql 식별자 , @Mapper Interface의 메서드명과 일치
parameterType	인자로 전달되는 객체 또는 별칭 , Literal , Domain Object or Map 을 주로 사용한다
resultType	조회 후 하나의 row 에 해당하는 결과 데이터 타입을 명시
resultMap	ResultSet 과 resultType의 매핑을 외부에 설정 , 재사용성, Join 관련 다양한 설정 가능

MyBatis 주요 기본 TypeAlias

```
<sql id="selectMember">
    SELECT id,password,name,address FROM tx_member
</sql>
<select id="findMemberById" parameterType="string" resultType="map">
    <include refid="selectMember"></include>
    WHERE id=#{value}
</select>
```

primitive	Primitive Data Type
string	java.lang.String
map	java.util.Map
list	java.util.List

MyBatis Dynamic SQL

- 제어문 역할을 하는 태그를 이용해 SQL 문을 동적으로 사용하도록 지원
- 조건처리 <if> , <choose> <when><otherwise>
- 반복처리 <forEach>

```
<select id="getEmpListByDynamicSQL" resultType="empVO" parameterType="empVO">
  <include refid="selectEmp"/>
  <where>
    <if test="deptno!=0">
      deptno=#{deptno}
    </if>
    <if test="ename!=null and ename!='">
      and ename like '%' || #{ename} || '%'
    </if>
  </where>
</select>
```



- TDD (Test Driven Development)
- TDD 를 위한 Java Framework
- 높은 품질 , 결함률 감소
- 재설계 시간의 감소

Spring JUnit 연동

maven pom.xml

```
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.12</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-test</artifactId>
<version>${org.springframework-version}</version>
</dependency>
```

Test Class

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(
    locations={"file:src/main/webapp/WEB-INF/spring-*.xml"})

public class TestMemberUnit {
    @Resource
    private MemberDAO dao;
    @Test
    public void countTest() {
        int memberCount=dao.getTotalMemberCount();
        assertEquals(memberCount, is(0));
    }
}
```



Spring Boot

- Spring Boot 는 스프링의 하위 프로젝트
- “단독으로 신속하게 개발 가능한 스프링 프로젝트”
- 프로젝트 차원에서 웹컨테이너를 내장
- **stand-alone** 한 프로젝트를 지원하여 빠른 개발
- 스프링 관련 복잡한 설정을 자동화하고
간편화하여 효율적인 개발을 지원
(COC:Convention Over Configuration)

Spring Boot 주요 구성요소

Application.java	스프링 부트를 실행시키기 위한 클래스
@SpringBootApplication	@Configuration , @EnableAutoConfiguration , @ComponetScan 의 3가지 어노테이션이 합쳐진 어노테이션
@Configuration	Spring Container에 해당 클래스가 Bean 구성 class임을 알려준다
@ComponetScan	@Component 계열과 @Configuration 명시된 class를 스캔해서 빈으로 등록
@EnableAutoConfiguration	Spring Application Context 생성될 때 자동으로 설정하는 역할

Spring initializr 를 이용한 프로젝트 생성

<https://start.spring.io/>



Project

☒ Maven Project

☐ Gradle Project

Language

☒ Java ☐ Kotlin

☐ Groovy

Spring Boot

☐ 2.3.0 RC1 ☐ 2.3.0 (SNAPSHOT)

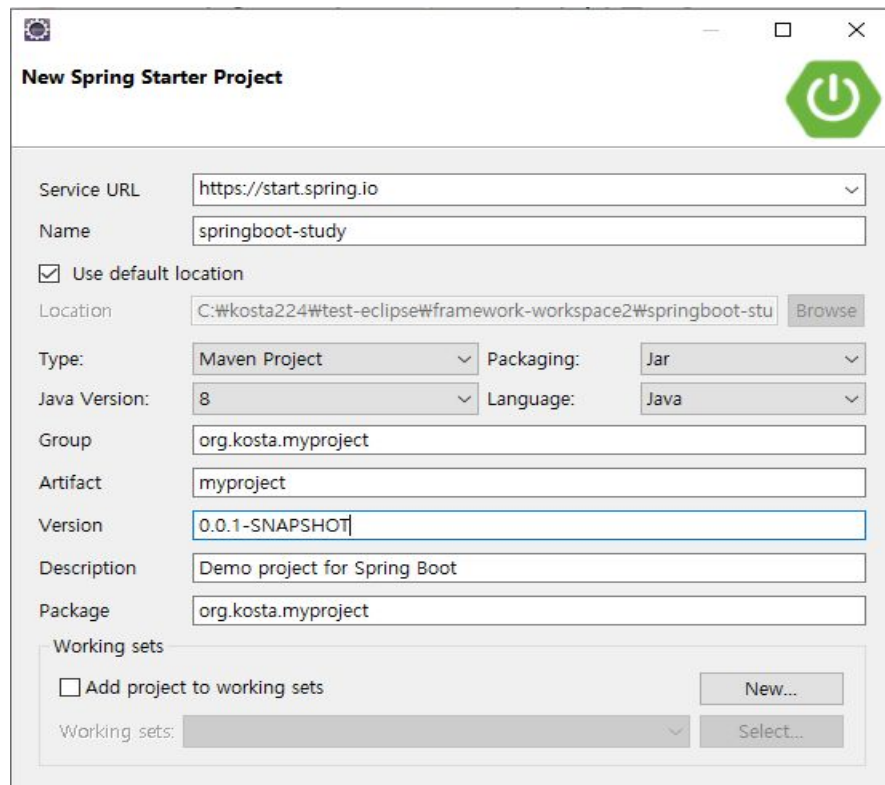
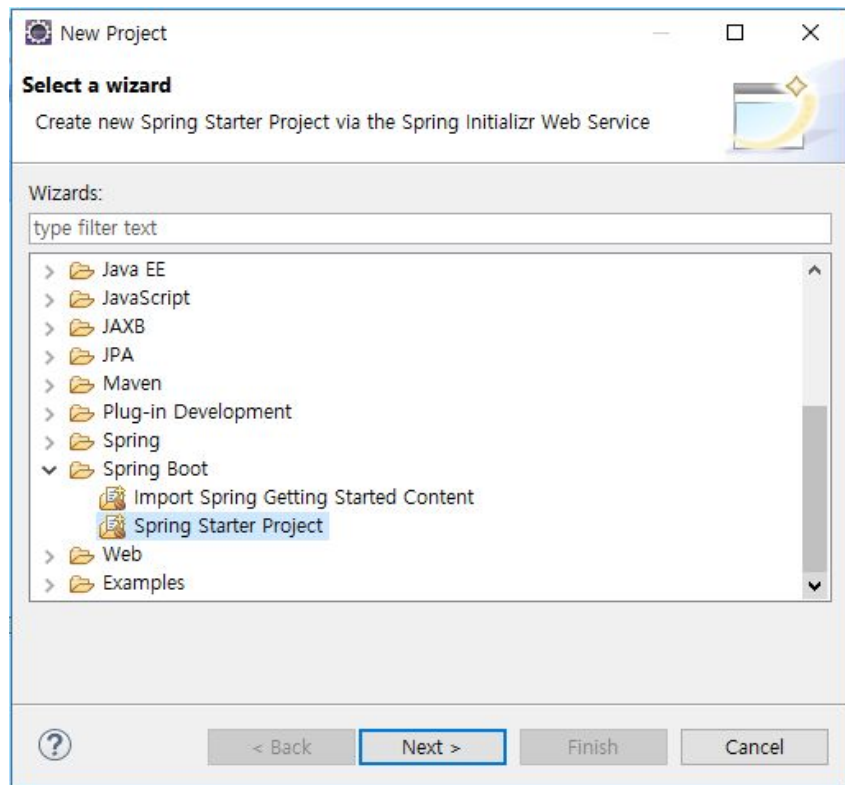
☐ 2.2.7 (SNAPSHOT) ☒ 2.2.6 ☐ 2.1.14 (SNAPSHOT)

☐ 2.1.13

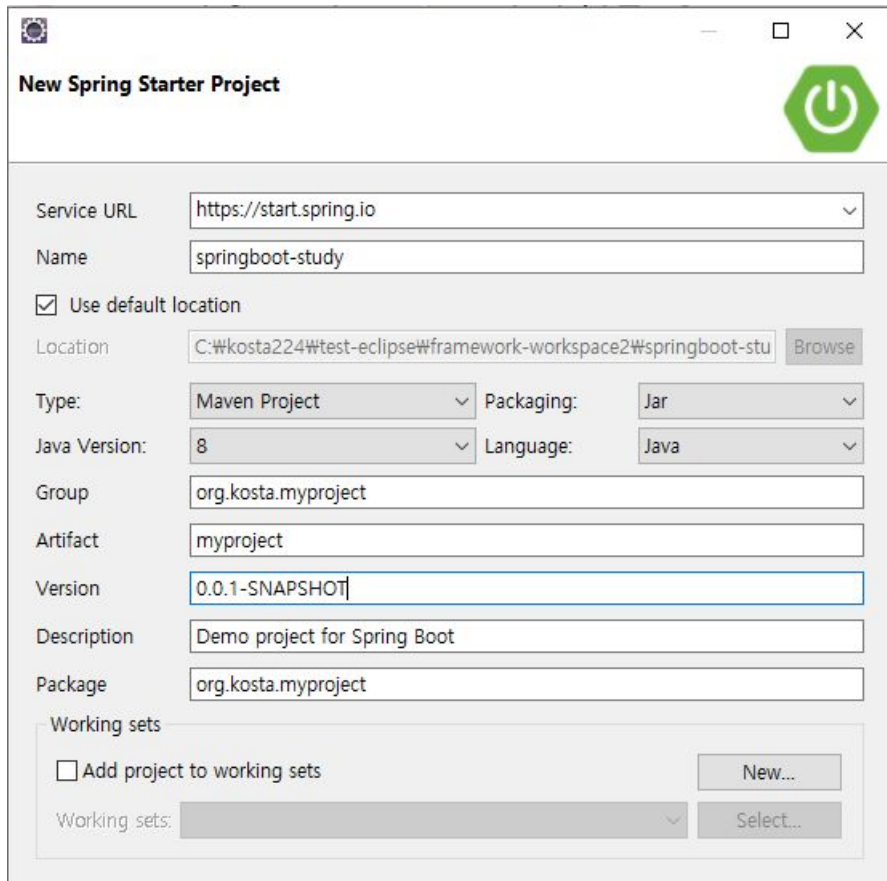
Project Metadata

Group com.example

Spring Starter Project 를 이용한 Spring Boot Project 생성 1



Spring Starter Project 를 이용한 Spring Boot Project 생성 1



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

Name : 워크스페이스에 저장되는 프로젝트 이름

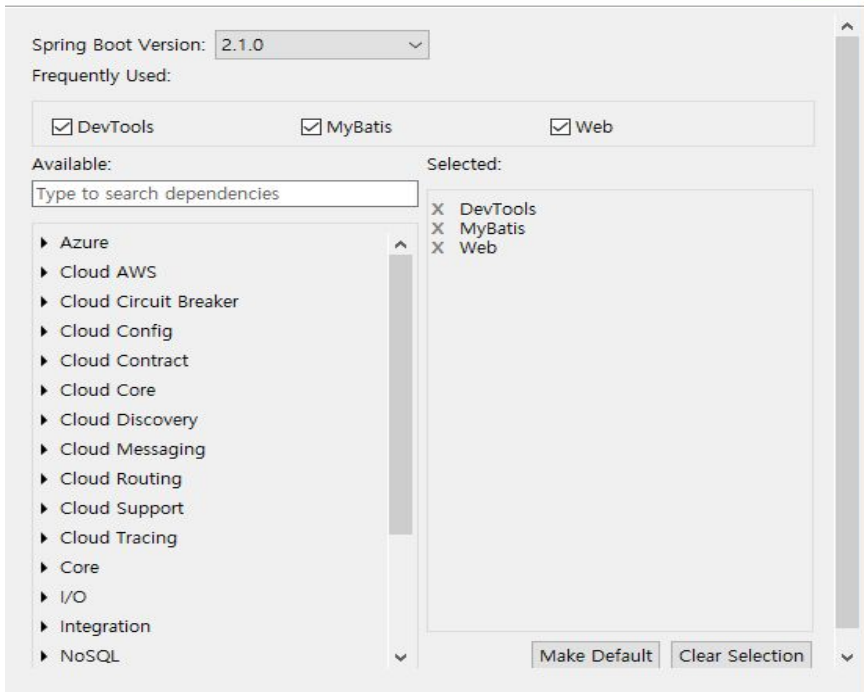
Group Id : 프로젝트의 고유 식별자, package 명명 규칙을 따른다 ex) org.apache.commons

Artifact Id : jar(or war) 파일의 이름, 소문자로 작성, project Name과 일치시킨다. 합성어일 경우 - 으로 연결한다 ex) commons-math

Spring Boot Project 생성 2

MySql 인 경우 MySql Driver도 클릭!

New Spring Starter Project Dependencies



Spring Boot Version: 2.1.0

Frequently Used:

☒ DevTools ☒ MyBatis ☒ Web

Available:

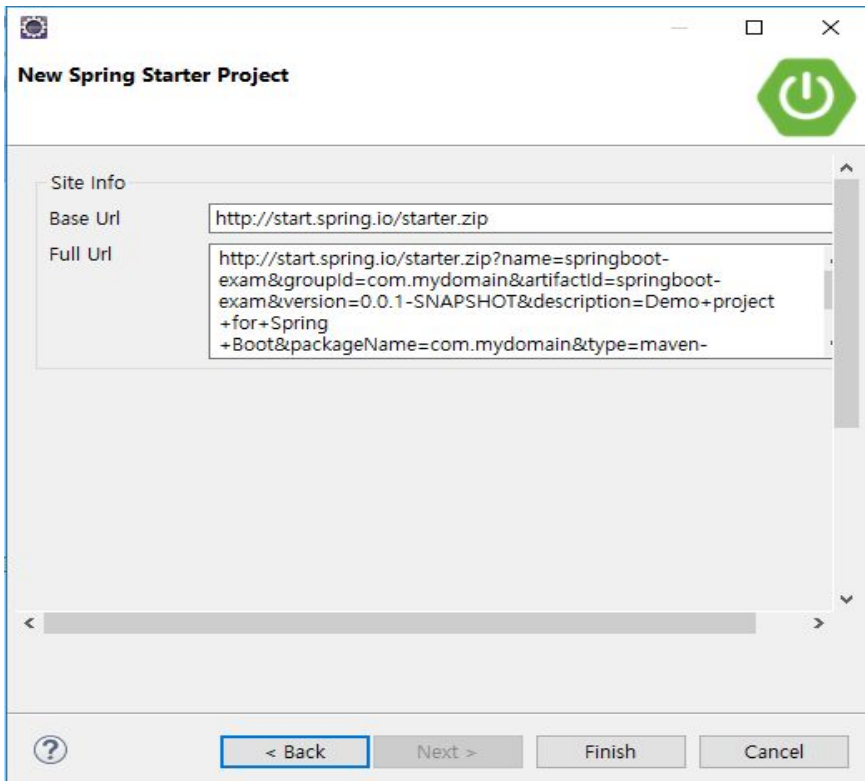
Type to search dependencies

- ▶ Azure
- ▶ Cloud AWS
- ▶ Cloud Circuit Breaker
- ▶ Cloud Config
- ▶ Cloud Contract
- ▶ Cloud Core
- ▶ Cloud Discovery
- ▶ Cloud Messaging
- ▶ Cloud Routing
- ▶ Cloud Support
- ▶ Cloud Tracing
- ▶ Core
- ▶ I/O
- ▶ Integration
- ▶ NoSQL

Selected:

- X DevTools
- X MyBatis
- X Web

Make Default Clear Selection



New Spring Starter Project

Site Info

Base Url: <http://start.spring.io/starter.zip>

Full Url: <http://start.spring.io/starter.zip?name=springboot-exam&groupId=com.mydomain&artifactId=springboot-exam&version=0.0.1-SNAPSHOT&description=Demo+project+for+Spring+Boot&packageName=com.mydomain&type=maven->

< Back Next > Finish Cancel

Spring Boot Project 환경설정 - Oracle

maven pom.xml

```
<!-- 기존 생성된 dependency외 추가 -->
<dependency>
<groupId>org.apache.tomcat.embed</groupId>
<artifactId>tomcat-embed-jasper</artifactId>
<scope>provided</scope>
</dependency>

<dependency>
<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
</dependency>
```

spring boot configuration : **application.properties**

```
# port
server.port = 8888
# dbcp setting
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521:xe
spring.datasource.username=scott
spring.datasource.password=tiger
# spring mvc view resolver setting
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
#devtools
spring.devtools.livereload.enabled=true
# mybatis
mybatis.type-aliases-package=com.mydomain.model.vo
mybatis.configuration.map-underscore-to-camel-case=true
# log level setting
logging.level.root=ERROR
```

Spring Boot Project 환경설정 - MySql

maven pom.xml

```
<!-- 기존 생성된 dependency외 추가 -->
<dependency>
<groupId>org.apache.tomcat.embed</groupId>
<artifactId>tomcat-embed-jasper</artifactId>
<scope>provided</scope>
</dependency>

<dependency>
<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
</dependency>
```

spring boot configuration : **application.properties**

```
# port
server.port = 8888
# dbcp setting
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/testdb
spring.datasource.username=love
spring.datasource.password=peace
# spring mvc view resolver setting
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
#devtools
spring.devtools.livereload.enabled=true
# mybatis
mybatis.type-aliases-package=com.mydomain.model.vo
mybatis.configuration.map-underscore-to-camel-case=true
# log level setting
logging.level.root=ERROR
```

STS에 Data Tool 설치하기

<https://download.eclipse.org/datatools/1.14.1.201712071719/repository/>

위 주소로 install new software 하면 됨

mysql jdbc driver - mysql-connector-java-5.1.0-bin.jar

<https://dev.mysql.com/downloads/connector/j/>

Drivers: MySQL JDBC Driver

Properties

General Optional

Database: database

URL: jdbc:mysql://localhost:3306/testdb

User name: love

Password: ●●●●●●

STS에 ObjectAid UML Explorer 설치하기

Help - Install New Software

Name- ObjectAid UML Explorer

Location- <http://www.objectaid.com/update/current>

ObjectAid UML Explorer 항목을 체크

****ObjectAid UML Explorer로 클래스 다이어그램을 생성하는 방법****

1. 이클립스의 **Package Explorer**에서 마우스를 원하는 프로젝트 이름 위에 올려 두고 오른쪽 버튼을 클릭-> 뜨는 팝업 메뉴에서 **New - Others...**를 선택
2. **Select a wizard** 단계에서
폴더 **ObjectAid UML Diagram->**
Class Diagram 선택 후 -> **Next** 버튼 클릭
3. **Create a new UML Class Diagram** 단계에서 **Name**에는
클래스 다이어그램 파일의 이름을 작성
4. 이후 추가시 **Package Explorer**에 있는 프로젝트의 자바 파일 항목을
클래스 다이어그램 창으로 드래그 앤 드롭

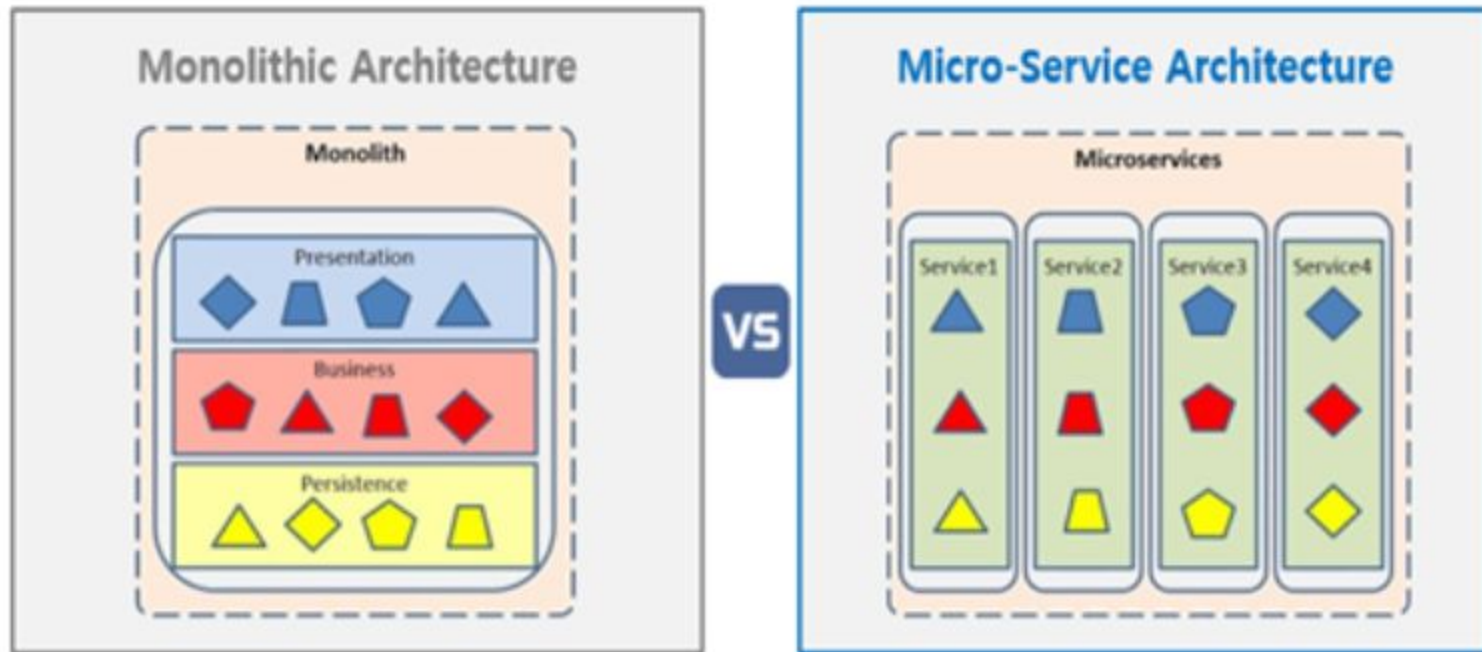
Spring Cloud



MSA , Netflix OSS, Spring Cloud

- “넷플릭스 당하다”
- 기존 비즈니스 모델 붕괴했을 때 사용하는 표현
- 2020년 현재 1억 8천명 가입자에게 고화질 동영상 서비스 제공
- 2007년 심각한 DB 손상으로 3일간 서비스 장애
- 신뢰성 높고 수평 확장가능한 클라우드 시스템을 도입의 필요성
- MSA 도입
- 7년에 걸쳐 클라우드 환경으로 이전
- 넷플릭스 OSS(Open Source Software)
- MSA 전환을 위한 기술을 오픈 소스로 공개

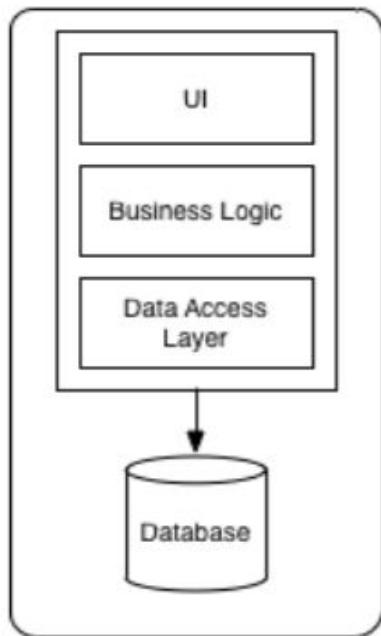
MSA



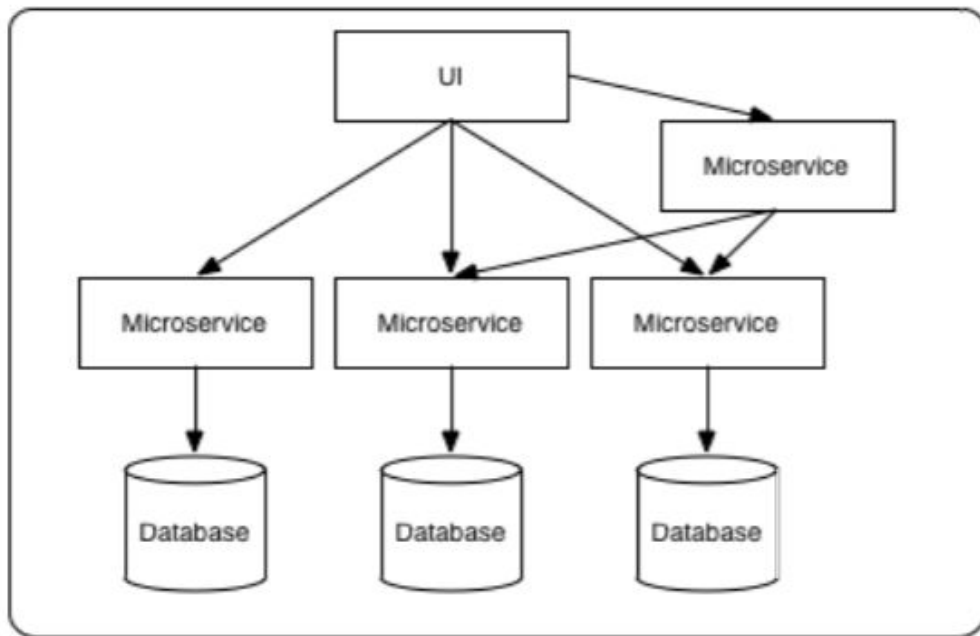
MSA (MicroService Architecture)

- Application을 독립적으로 개발,배포할 수 있는 서비스의 모음으로 설계하는 방식
- 작은 단위의 application 모음군으로 개발하는 접근방식
- 각각 자체 프로세스에서 실행
- 경량 메커니즘(**REST**)으로 통신하는 소규모 서비스 모음으로 단일 애플리케이션을 개발하는 접근방식
- 비즈니스 중심으로 구축해 독립적으로 배포
- 개별 서비스에 최적화된 프로그래밍 언어 및 **Database** 를 사용할 수 있음

Spring Cloud - MSA



Monolithic Architecture



Microservices Architecture

MSA

- 트위터, 페이스북, 이베이등이 **MSA** 방식을 도입
- 기능별로 마이크로 서비스를 개발
- 변경 사항이 있는 마이크로 서비스만 빠르게 빌드,배포
- 해당 서비스에 가장 적합한 기술과 환경을 선택할 수 있음
- 서비스 소비자와 제공자 사이의 데이터 교환을 위해
REST/JSON을 주로 사용
- 이슈 발생시 해당 서비스만 수정해서 정상화 가능

MSA의 목적

- **MicroService** 는 상호 독립적으로 구축될 수 있어 결합도가 낮음
- 특정 서비스에 집중하여 응집도 높음
- **Restful API** 와 같이 **lightweight** 한 통신을 통해 효과적인 상호 연계가 가능
- 독립적인 서비스 단위 확장(**scale-out**)을 지원하기 때문에 효율적인 시스템 자원 활용

MSA 적용시 고려사항

- 비용: 특정 서비스 아키텍처를 도입할 경우 비용을 얼마나 절감할 수 있는가
- 개발 생산성: 마이크로서비스를 요구할 만큼 시스템 복잡도가 높은가
- 개발 생산성: 복잡도를 지나치게 높인 마이크로서비스가 생산성을 저해하고 있지는 않은가
- 운영: 개발팀에게 개발과 운영을 동시에 요구할 만큼 인프라가 준비되어 있는가

(<https://www.samsungsds.com/kr/insights/msa.html>)

Service Mesh

- 마이크로 서비스 간의 통신을 최적화
- 증가한 서비스를 관리
- 서비스간 복잡한 연결구조의 장애 추적 및 전파 현상을 해결
- 이를 위해 비즈니스 로직과 분리된 네트워크 통신 인프라인

Service Mesh 의 도입이 필요

Service Mesh의 주요기능

- Configuration Management: 설정변경 시 서비스의 재빌드와 재부팅 없이 즉시 반영
- Service Discovery: API Gateway 가 서비스를 검색하는 메커니즘
- Load Balancing: 서비스간 부하 분산
- API Gateway API: API 엔드포인트 단일화 및 인증, 인가, 라우팅 기능 담당
- Circuit Breaker : 장애 전파 방지 및 대안흐름 실행

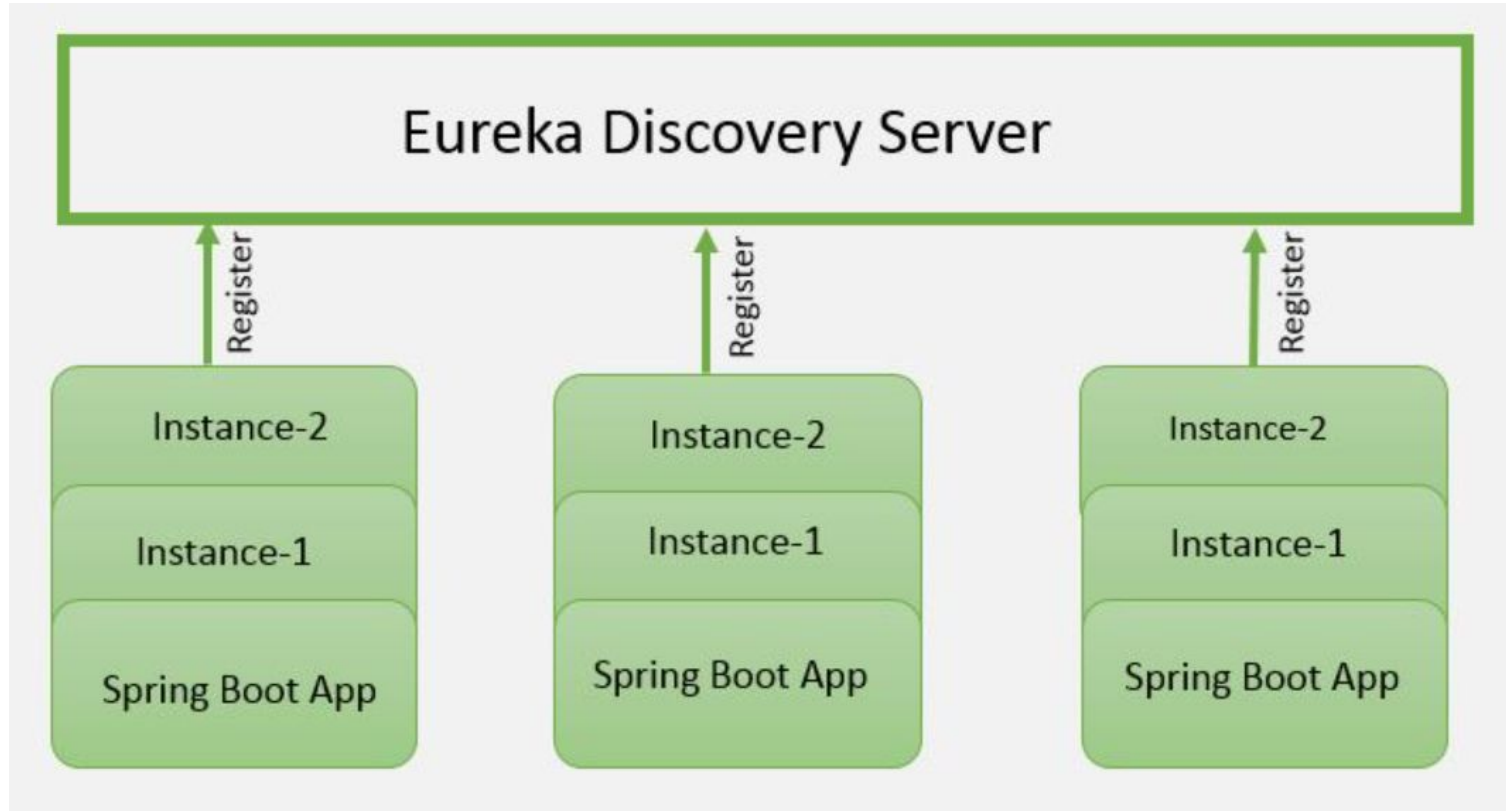
Spring Cloud 란

- 독립적인 프로젝트들로 구성된 상위 프로젝트
(Spring Cloud is an umbrella project consisting of independent projects)
- MSA 를 위한 스프링 프로젝트
- MSA 의 Service Mesh 를 위한 Netflix OSS 기반 기술을 제공
- Spring Boot를 기반으로 MSA 구축에 특화된 라이브러리들의 집합

Spring Cloud

- Spring Boot 기반에 Spring Cloud를 적용하면 분산처리 환경의 안정적인 Service Mesh 를 구현할 수 있음
- Spring Cloud는 Service Mesh를 위한 Eureka, Hystrix, Ribbon, Zuul 등 많은 넷플릭스 OSS가 통합되어 있음

Spring Cloud - Eureka



Spring Cloud - Eureka

- Service Discovery Server
- 서비스 인스턴스의 상태를 동적으로 관리
- Auto Scale-out (자동확장)
- 자동확장이란 특정 서비스에 부하 증가, 감소시
인스턴스 생성 , 삭제를 통해 시스템 자원 활용 최적화

Spring Cloud - Hystrix

- Circuit Breaker
- 특정 서비스에 문제 발생시 장애가 확산되지 않도록 차단
- **Hystrix** 서버는 각 서비스의 오류 상태나 복구 상태, 오류내용에 대해 파악
- 서비스에 이상을 감지하면 통신 중단 후 정상으로 돌아오면 통신을 연결
- **Fallback** : 정해진 시간 내에 호출이 실패할 때 예외처리를 말함
개발자는 폴백 메서드를 구현하여 시스템 장애에 유연하게 대처

Spring Cloud - Ribbon

- Load Balancer
- 로드밸런싱이란 트래픽 부하를 줄이기 위해 여러 자원에 작업을 분산
- Ribbon : Client side Load Balancer
- REST API를 호출하는 서비스에 탑재되는 SW모듈
- 서버 선택, 실패시 Skip 시간, ping 체크
- Retry 기능 내장 - 서버에게 응답받지 못했을 때 동일 서버 또는 다른 서버에 재시도하는 기능
- H/W 필요없이 SW적으로 구현

Spring Cloud - Zuul

- API Gateway
- 클라이언트 요청에 대해 적절한 서비스를 연결하고 응답하는 단일한 진입점(Front) 역할을 수행
- 요청분석,인증, 보안, 예외처리등을 단일한 진입점(Front)인 Zuul에서 처리
- 데이터 및 통계 분석 , 모니터링

Spring Cloud Practice : RESTful , RestTemplate

request ---> 카탈로그서비스(7777) <--RESTful API--> 고객서비스(9999)

RestTemplate

	화면 서비스	커스터머 서비스
서비스	Catalogs	Customers
Request	/catalogs/{customerId}	/customers/{customerId}
Response	JSON	JSON

출처: 전자정부표준프레임워크 MSA 적용 개발가이드

Spring Cloud Practice : Hystrix

request ---> 카탈로그서비스(7777) <--RESTful API--> 고객센터(9999)

RestTemplate
@HystrixCommand : fallback method

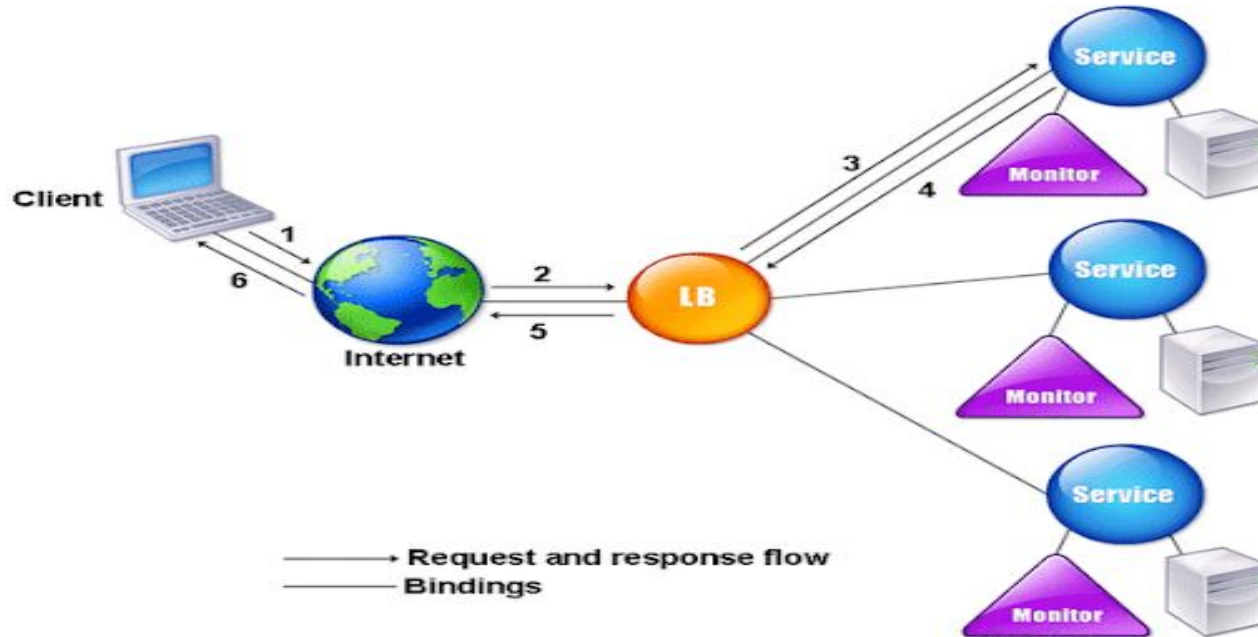
장애대비 Test를 위한
Exception 발생시켜봄

Hystrix : 장애 전파방지

해당서비스 실패로 의존성 있는 타 서비스까지 장애가 전파될 수 있다.

Hystrix의 Fallback(대비책)을 이용해 장애전파를 미연에 방지하고 빠르게 복구할 수 있도록 한다

Spring Cloud Practice : Ribbon (Load Balancer)



Spring Cloud Practice : Ribbon (Load Balancer)

- Rule (요청할 서버를 선택하는 논리)
 - Round Robbin : 서버를 하나씩 돌아가며 전달(기본값)
 - Filtering : 에러가 많은 서버를 제외
 - Weighted Response Time : 서버별 응답 시간에 따라 확률 조절
- Ping - 서버가 살아있는 지 체크
- 로드 밸런싱 대상 서버 목록

Spring Cloud Practice : Ribbon (Load Balancer)

request ---> 카탈로그서비스(7777) <--RESTful API--> 고객서비스(9999)
고객서비스(6666)

RestTemplate

@LoadBalanced

여러 서버에 접속해서 테스트

pplication.properties : ribbon 설정 추가 -> customer (localhost:9999)

Spring Cloud Practice : Eureka (Service Discovery, Registry)

Eureka : MSA 장점인 동적 서버 증설 및 축소를 위해 서비스 등록,
탐색 및 부하 분산에 사용하는 기술
마이크로 서비스들의 정보를 레지스트리 서버에 등록

Eureka 서버 : Eureka 클라이언트에 해당하는 마이크로서비스들의
상태가 등록되어 있는 레지스트리 서버임

Eureka 클라이언트 : 서비스가 시작될 때 Eureka 서버에 자신의 정보를
등록하고 이후 주기적으로 자신의 가용상태(health check)를 알리며 일정 횟수
이상의 ping이 확인되지 않으면 Eureka 서버에서 해당 서비스를 제외한다

Spring Cloud Practice : Eureka (Service Registry)

1. Eureka Server 구축 및 실행

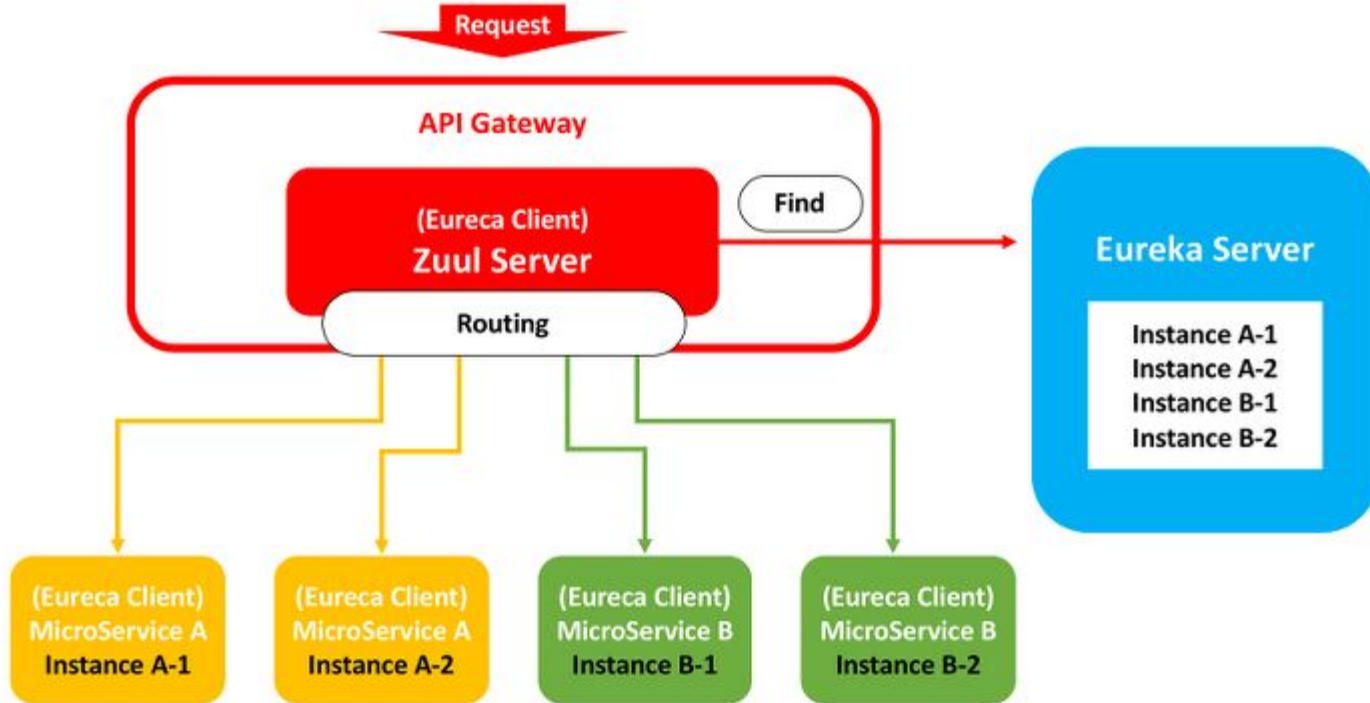
@EnableEurekaServer

2. Eureka Client 서비스

기존의 Catalog 서비스와 Customers 서비스를 Eureka Client로 적용
Eureka 서버에 서비스를 등록할 수 있도록 구현

@EnableEurekaClient

Spring Cloud Practice : API Gateway (Zuul)



<https://lion-king.tistory.com/entry/Spring-Boot-Spring-Cloud-MSA-4-Zuul>

Spring Cloud Essence 1

- MSA MicroService Architecture 서비스 단위로 독립적인 시스템을 구축 , 운영,
- 서비스별 최적화된 기술을 선택해서 개발
- 명확히 정의된 책임영역을 담당해 특정 서비스에만 집중해 응집도가 높아짐
- 서비스가 독립되어 변경 용이하고 변경시 서비스간 영향이 적다 즉 결합도를 낮추어 유지보수성 향상에 기여

Spring Cloud Essence 2

- Service Mesh : 마이크로 서비스간 통신의 최적화
- Spring Cloud : 독립적인 프로젝트들로 구성된 상위 프로젝트
(Spring Cloud is an umbrella project consisting of independent projects)

MSA 를 위한 스프링 기술

MSA 의 Service Mesh 를 위한 Netflix OSS 기반 기술을 제공
분산 시스템 구성에 필요한 다양한 기능 (설정 관리 및 공유,
서비스 등록관리,서비스 요청 라우팅) 을 제공

Spring Cloud Essence 3 : MSA에서 SpringBoot와 SpringCloud의 역할

- Spring Boot : 컴포넌트 레벨에서 마이크로 서비스 아키텍처로 설정 간소화 및 독립서비스를 지원
- Spring Cloud : MSA 컴포넌트들 간의 효율적인 분산 서비스를 지원(Load Balancing,Service Discovery 등의 Outer Architecture 영역지원)

Spring Cloud Essence 4

- Spring Cloud 의 주요 기술

Configuration Management : Spring Cloud Config Server

Service Discovery : Netflix Eureka

Circuit Breaker: Netflix Hystrix

Load Balancing : Netflix Ribbon

API Gateway : Netflix Zuul