
Spring Boot 백엔드 스터디 가이드

일정 : 2020.01.03(금) ~ 2020.02월 말

스터디에 대해서...

- 연락없이 불참하지 말아주세요.
 - 급한일 생겨서 참석 힘들시면, 미리 말씀만 해주시면 됩니다.
- 스터디 참여 인원 모두 **Github** 을 사용하기를 바랍니다.
- 8주차를 진행하면서 총 8 번의 **STEP** 과제를 진행합니다.
 - 단, **STEP** 과제에 대해서 너무 부담 갖지 않아도 됩니다.
 - 자신의 페이스를 유지하기..(옆사람이 너무 잘한다고 포기해버리지 말기)
 - 현재 참석하는 스터디원의 실력차이가 큰 편입니다.
 - 완전 초보이신 분들, 몇달 공부하신분, 백엔드 3년 경력이신분도 (저도 스프링 3년차)
 - 각자 실력에 맞게, 차분히 **STEP** 을 하나씩 진행해주시면 됩니다.
- 스터디 발표 준비 시간이 부족해서 문서는 깔끔하지 않습니다.
 - 죄송합니다.
- **github, Merge Request** 중심으로 진행되지만, 리뷰어는 없습니다.
 - 가능하면, 제가 리뷰를 진행하겠지만 ... 리뷰할 수있는 실력이 되지 않습니다.
 - 그래도 시간이 되면 틈틈히 보겠습니다.
 - 반드시 **PR** 로 코드를 공유해주세요
- **github github MR** 가이드
 - <https://github.com/next-step/nextstep-docs/tree/master/codereview>
 - <https://github.com/spring-basic-study/openapi>

Spring Boot 백엔드 스터디 1주차

2020.01.03(금) 삼성역 11:30 ~ 13:30

목차

1. Spring VS Spring Boot
2. Spring Boot AutoConfiguration
3. Spring Boot Dependency Management
4. Spring Boot 1.0 ~ 2.2 버전사
5. Spring Boot 오해와 진실
6. 레퍼런스

1.Spring VS Spring Boot

—

1. AutoConfiguration
 - a. 일반적으로 사용하는 기능에 대한 자동 구성 제공
 - b. 별도로 재정의 및 구현 가능
2. Embedded Server 등 단독 실행 가능한 스프링 애플리케이션 구축
 - a. Tomcat, Netty 등
 - b. 마이크로서비스 애플리케이션 적합
3. Dependency Management
 - a. 라이브러리 버전 의존성 관리
4. Actuator(모니터링) 및 기타 등등

스프링부트는 스프링 프레임워크를 쉽게 사용하기 위한 기술일 뿐
(즉, 스프링부트는 내부적으로 스프링 프레임워크를 사용)

```

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}

```

스프링부트에서의 **@SpringBootApplication**

@Configuration

@EnableAutoConfiguration


@ComponentScan

를 하나의 어노테이션으로 제공하는 것

```

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
    @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {

```

 **Spring Initializr**
Bootstrap your application

Project

Maven Project **Gradle Project**

Language

Java Kotlin Groovy

Spring Boot

2.2.3 (SNAPSHOT) **2.2.2** 2.1.12 (SNAPSHOT) 2.1.11

Project Metadata

Group



com.example

Artifact

demo

> Options

Dependencies

Search dependencies to add

Web, Security, JPA, Actuator, Devtools...

Selected dependencies

No dependency selected

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

2. Spring Boot AutoConfiguration

<https://github.com/spring-projects/spring-boot/blob/master/spring-boot-project/spring-boot-autoconfigure/src/main/resources/META-INF/spring.factories>

AutoConfiguration 예시 1 - Embedded Tomcat

Spring Boot Web Starter

-->

Spring Boot Tomcat Starter

-->

Apache Tomcat Embed...

Embedded Tomcat

```
* @author Chentao Qu
*/
@ConfigurationProperties(prefix = "server", ignoreUnknownFields = true)
public class ServerProperties {

    /**
     * Server HTTP port.
     */
    private Integer port;
```

```
* @author Phillip Webb
* @since 2.0.0
*/
@Configuration(proxyBeanMethods = false)
@ConditionalOnWebApplication
@EnableConfigurationProperties(ServerProperties.class)
public class EmbeddedWebServerFactoryCustomizerAutoConfiguration {
```

```
1
2  server.port=8080
3  server.tomcat.max-threads=200
4
5  server.tomcat.max
  server.tomcat.max-connections=10000 (Maximum n... Integer
  server.tomcat.max-http-form-post-size=2MB (Ma... DataSize
  server.tomcat.max-swallow-size=2MB (Maximum a... DataSize
  server.tomcat.accesslog.max-days=-1 (Number of... Integer
  server.tomcat.max-http-post-size DataSize
  ^↓ and ^↑ will move caret down and up in the editor >> (0/0)
```

<https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html#server-properties>

별도 설정을 별도로 정의하지 않는다면
기본으로 제공되는 Default 설정으로 AutoConfiguration!!!

```
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http)
```

```
com.example.demo.DemoApplication : Started DemoApplication in 16.093 s
```

외장톰캣 vs Embedded Tomcat

1. war 빌드, 서버 배포
2. Tomcat 재시작

1. jar 빌드, 서버 배포
2. java -jar

AutoConfiguration 장단점

- 스프링을 쉽게 시작할 수 있다.
 - 복잡한 컨피그 설정을 신경쓰지 않아도 된다.
 - 비즈니스 로직에 집중할 수 있다.
 - 모르고 사용하면 결국 헤맨다.
 - 커스터마이징하기 어려울 때가 있다.
-







내부적으로 어떻게 동작하는지
이해하고 사용해야 한다!!

Dependency Management

```
plugins {  
    id 'org.springframework.boot' version '2.2.2.RELEASE'  
    id 'io.spring.dependency-management' version '1.0.8.RELEASE'  
    id 'java'  
}  
  
group = 'com.example'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = '1.8'  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    testImplementation('org.springframework.boot:spring-boot-starter-test') {  
        exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'  
    }  
}  
  
test {  
    useJUnitPlatform()  
}
```

<https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web>

Compile Dependencies (6)

Category/License		Group / Artifact	Version	Updates
Web Framework Apache 2.0		org.springframework » spring-web	5.2.2.RELEASE	✓
Web Framework Apache 2.0		org.springframework » spring-webmvc	5.2.2.RELEASE	✓
Apache 2.0		org.springframework.boot » spring-boot-starter	2.2.2.RELEASE	✓
Apache 2.0		org.springframework.boot » spring-boot-starter-json	2.2.2.RELEASE	✓
Apache 2.0		org.springframework.boot » spring-boot-starter-tomcat	2.2.2.RELEASE	✓
Apache 2.0		org.springframework.boot » spring-boot-starter-validation	2.2.2.RELEASE	✓

- 라이브러리 버전 에 대한 의존성 관리
- 현재 스프링부트 최신 릴리스 버전(2.2.2.RELEASE)
 - <https://github.com/spring-projects/spring-boot/blob/v2.2.2.RELEASE/spring-boot-project/spring-boot-dependencies/pom.xml>
- 개별적인 버전 설정도 당연히 가능
 - 단, 스프링부트에서 제공하는 버전 의존성은 어느정도 검증이 된 버전이기 때문에, 가능하면 맞춰서 사용하는게 좋다.

Spring Boot 1.X ~ 2.X

Spring Boot 오해와 진실

- Java Config 설정만 가능한가?
 - 그렇지 않다.
 - XML Config 설정도 가능하지만, 실무에서는 대부분 Java Config 구성
- 모든 레거시 프로젝트를 Spring Boot 로 전환해야 하는가?
 - 그렇지 않다.
 - 하지만, 처음 시작하는 프로젝트는 대부분 Spring Boot 로 시작
- Gradle VS Maven

Spring Boot

레퍼런스

<https://spring.io/blog>

DEMO

STEP 01

- 네이버 Open API 연동 및 콘텐츠 API 구현
 - 요구사항
 - 네이버 오픈API 를 연동해서, query(파라미터)로 검색어를 전달해서 블로그 정보를 조회한다
 - <https://developers.naver.com/docs/search/blog/>
 - <https://developers.naver.com/docs/common/openapiguide/apilist.md#%EA%B2%80%EC%83%89>
 - 네이버 오픈 API 호출은 RestTemplate 를 사용한다.
 - OpenAPI 연동 URL,인증키 등 정보를 application 파일에 설정
 - application파일에 정의한 정보를 @ConfigurationProperties, @EnableConfigurationProperties 어노테이션을 사용해서 가져오기
 - "local" profile 로 애플리케이션 실행
 - Embedded Tomcat Port, Max Thread 설정 변경해보기
 - Service or Repository 패키지 레이어로 분리해보기
 - RestTemplate 를 별도의 Bean 으로 정의하기
 - Lombok 버전을 내가 원하는 버전으로 디펜던시 변경해보기
-

2주차

Spring Boot DI (Dependency Injection)

2020.01.10(금) 삼성역 11:30 ~ 13:30

1주차 복습

(새로 오신 분들을 위한...)

AutoConfiguration, 스프링부트에 대해서 간단하게 다시 설명합니다.

1주차 과제 정답 공유

DI 를 사용하지 않는 의존 방법

- 의존 객체를 직접 생성

```
public class MovieService {  
  
    public String getNameBy(){  
        return "겨울왕국";  
    }  
}
```

```
MovieService movieService = new MovieService();  
movieService.getNameBy();
```

DI 는 의존 객체를
직접 생성하지 않고,

주입 받는 방식!!

```
public class SearchService {
```

```
    private MovieService movieService;
```

```
@  
    public SearchService(MovieService movieService){  
        this.movieService = movieService;  
    }
```

```
    public String getMovieNameBy(){
```

```
        /*
```

```
        MovieService movieService = new MovieService();
```

```
        movieService.getNameBy();
```

```
        */
```

```
        return movieService.getNameBy();
```

```
    }
```

```
SearchService searchService = new SearchService(new MovieService());  
return searchService.getMovieNameBy();
```

스프링에서는
Bean 을 주입!!!

DI 주입



```
@Service
public class SearchService {

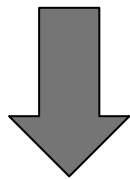
    @Autowired
    private MovieService movieService;

    public String getMovieNameBy(){

        /*
        MovieService movieService = new MovieService();
        movieService.getNameBy();
        */

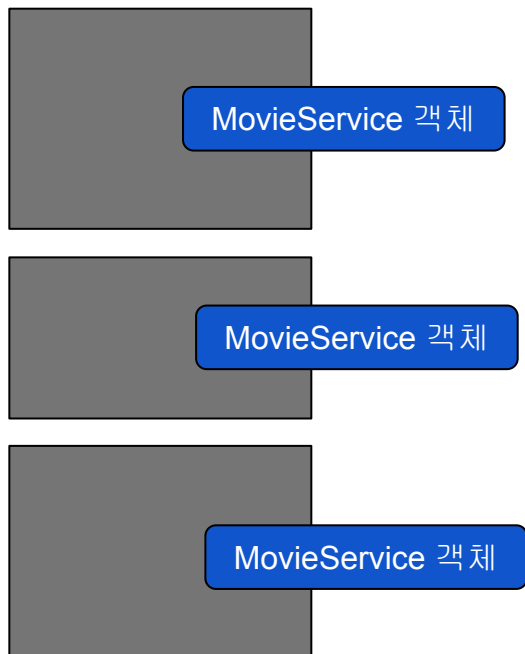
        return movieService.getNameBy();
    }
}
```

Bean

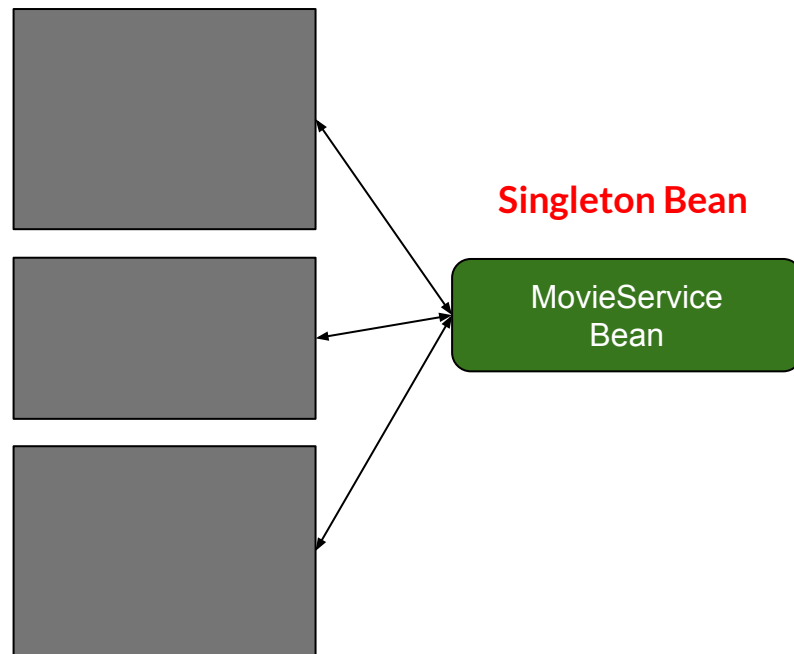


```
@Service
public class MovieService {

    public String getNameBy(){
        return "겨울왕국";
    }
}
```



Spring Bean 의존성 주입



Bean 정의 방법, DI 주입 방법

- **Bean 정의 방법**

- @Bean & @Configuration 조합으로
- @Component, @Service, @Repository, @Controller 사용

- **DI 방법**

- 필드 인젝션
 - @Autowired
 - 생성자 주입
 - Setter 주입
 - Lombok 라이브러리
-

고전적인 Bean 정의 방식 (XML)

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" data-bbox="252 273 969 528">
  <property name="driverClassName" value="${jdbc.driverClassName}" />
  <property name="url" value="${jdbc.url}" />
  <property name="username" value="${jdbc.username}" />
  <property name="password" value="${jdbc.password}" />
</bean>
```

- 스프링 3.X 이전 프로젝트에서 주로 사용, 그 이후 버전에서도 종종 사용
 - Bean 과 Bean 연결 관계를 XML 로 정의
 - 유지보수 어렵고 운영하기 어렵다는 단점
-

@ComponentScan

```
@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

```
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM, classes = AutoConfiguration.ExcludeFilters.DefaultExcludedFilters.class) })
public @interface SpringBootApplication {

    /**
     * Exclude specific auto-configuration classes such that they are not
     * @return the classes to exclude
     */
}
```


@ComponentScan

- @ComponentScan 어노테이션
 - @Component, @Service, @Repository, @Controller 어노테이션이 선언된 클래스를 자동으로 찾아서 Bean 으로 등록해주는 역할
- @ComponentScan 의 범위
 - basePackage 등 스캔하기 위한 설정을 추가할 수 있다.
 - 루트 경로에 basePackage 설정 없이 사용한다면 ...
- <https://www.baeldung.com/spring-component-scanning>

@Component, @Service, @Controller, @Repository

- @Component
 - @Controller, @Service, @Repository 어노테이션도 결국 **@Component**
- 모두 Bean 인스턴스가 생성되지만, 클래스 역할에 따라서 선언해주면 됨

```
40  * @since 2.5
41  * @see Component
42  * @see Repository
43  */
44  @Target({ElementType.TYPE})
45  @Retention(RetentionPolicy.RUNTIME)
46  @Documented
47  @Component
48  public @interface Service {
49
50  /**
51   * The value may indicate a suggestion for a logical
52   * to be turned into a Spring bean in case of an au
```

@Service

```
@Service  
public class BlogService {
```

(소문자)blogService 라는 이름의 Bean 이 생성 됨.

Bean 의 이름을 선언할수도 있음(

Bean 인스턴스 생성하는, 첫번째 방법에 대한 설명
끝!!

빈정의 방법(2) - @Configuration, @Bean

```
*/  
@Target(ElementType.TYPE)  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
@Component  
public @interface Configuration {
```

④ Configuration

@Configuration

```
public class 블라블라Config{
```

@Bean

```
public BlogService blogService(){  
    return new BlogService ();  
}
```

@Configuration

```
public class 블라블라Config{
```

```
    @Bean(name = "customService")
```

```
    public BlogService blogService(){
```

```
        return new BlogService ();
```

```
}
```

Bean 인스턴스 생성하는, 두번째 방법에 대한 설명
끝!!

스프링에서의 DI 주입 방법

- 필드 인젝션 주입
- 생성자 주입
- Setter 주입
- Lombok 라이브러리 사용

필드 인젝션

```
@Autowired
```

```
private BlogService blogService;
```



생성자 주입

```
private final BlogService blogService;  
  
public SearchController(BlogService blogService) {  
    this.blogService = blogService;  
}
```

Lombok

```
@RequiredArgsConstructor  
@RestController  
@RequestMapping("/api")  
public class SearchController {  
  
    private final BlogService blogService;  
  
    @GetMapping("/blog")
```

스프링에서의 DI 주입 심화

- @Qualifier
- @Primary
- Map Bean 주입
- 등등..

중요한 내용이지만, 시간 관계상 넘어갈게여..

스프링 Bean 라이프사이클..

- **@PostConstruct**
 - Bean 객체가 생성된 직후 초기화 작업이 필요한 경우 사용한다.
메서드 상단에 선언하면 된다.
- InitializingBean, DisposableBean
- 등등 각자 공부하시면 됩니다.

@Lazy

- ...

스프링 부트 2.X에서는

- Spring Boot Lazy Initialization

```
spring.main.lazy-initialization=true
```

Live Coding!! 함께해요
옆사람 잘 모르면 알려주기
짜증내지 말고 알려주기

STEP 01

- 네이버 Open API 연동 및 콘텐츠 API 구현
 - 요구사항
 - 네이버 오픈API 를 연동해서, query(파라미터)로 검색어를 전달해서 블로그 정보를 조회한다
 - <https://developers.naver.com/docs/search/blog/>
 - <https://developers.naver.com/docs/common/openapiguide/apilist.md#%EA%B2%80%EC%83%89>
 - 네이버 오픈 API 호출은 RestTemplate 를 사용한다.
 - OpenAPI 연동 URL,인증키 등 정보를 application 파일에 설정
 - application파일에 정의한 정보를 @ConfigurationProperties, @EnableConfigurationProperties 어노테이션을 사용해서 가져오기
 - "local" profile 로 애플리케이션 실행
 - Embedded Tomcat Port, Max Thread 설정 변경해보기
 - Service or Repository 패키지 레이어로 분리해보기
 - RestTemplate 를 별도의 Bean 으로 정의하기
 - Lombok 버전을 내가 원하는 버전으로 디펜던시 변경해보기
-

STEP 02

- 영화 Open API 를 추가로 연동하고, RestController 에서 영화 검색 정보를 제공한다.
 - 1회 쿼리 요청에 영화 정보와, 블로그 정보를 함께 제공하는 API 를 만든다.
 - **영화 Open API 와 블로그 Open API 조회를 위해 사용하는 RestTemplate 호출 구문의 중복을 최소화 해보자.**
 - RestTemplate 의 쓰레드풀, Timeout 등의 설정을 변경해보자.
 - 설정정보는 application 프로퍼티에 설정해서 불러온다.
 - 블로그, 또는 영화 정보 조회 시 LIMIT 로 개수를 제한해본다.
 - LIMIT 개수는 별도의 상수로 정의해보자
-

3주차

Spring Boot Rest API, MVC

- 코드를 참고해서 보고싶다.
- 어렵다.
- github PR 보냈는데 안왔다고 함

3주차

Spring Boot Rest API, MVC

3주차는 부담 없이 편하게...

(시간에 쫓기지 말고.. 그냥 천천히 말하는게 좋을 듯)

주제에 얽매이지 말고... 그냥 하고 싶은얘기...

이런저런 편하게 대화하면서 하는게 좋을 듯...

복습

Spring Bean 생성, 주입 방법

4 주차에 진행할 내용 미리 소개

스프링부트 TDD

5,6,7 주차에 진행할 내용 미리 소개

Redis 에 대해서
(캐시가 왜 필요한지?)

MQ(Kafka, RabbitMQ) 에 대해서

RestTemplate

= 스프링에서 제공하는 http 통신에 유용하게 쓸 수 있는 템플릿
(...Template 로 끝나는 애들이 또 있나요?)

RestTemplate
JdbcTemplate
RedisTemplate
KafkaTemplate

이런 애들은 왜 제공될까요???

내부 통신 라이브러리가 별도로 존재함!!

예) RedisTemplate... 예로 설명

Spring Data XXX

도대체 이런 모듈은 왜 만들어서 제공하는지???

[중요]

스프링 추상화 전략

Q & A

3. Rest API

3.1 HTTP, Rest API

- HTTP 에 대한 기본 개념 이해
- Rest API 설계

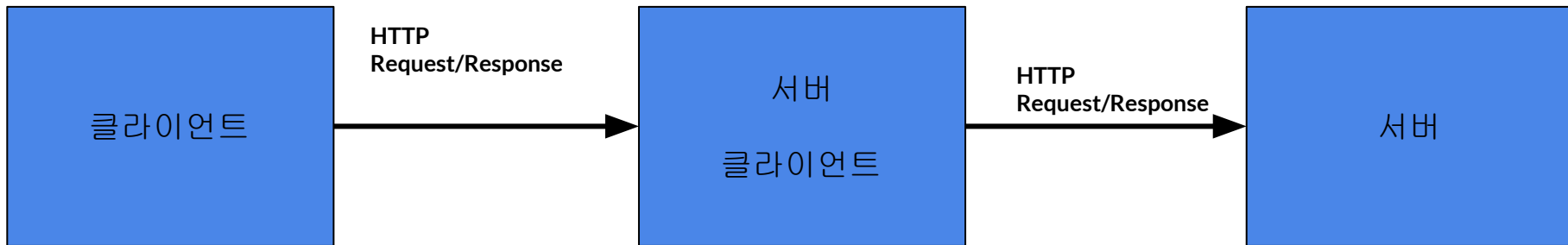
웹을 지탱하는 기술

HTTP

3.1.1 HTTP

- 설명...발표 문서 정리!!
 - 웹은 HTTP 라는 프로토콜을 사용하여 서버간 통신을 한다. HTTP 는 클라이언트와 서버의 역할이 명확하게 나누어져있다. 클라이언트 측에서 요청(Request)를 보내고, 서버 측에서 응답(Response)가 되돌아 오는 구조이다. 물론, 클라이언트와 서버는 서로 역할을 바꿀 수도 있다. 클라이언트는 한 번 응답을 보내면 자신이 보낸 응답에 대해서 기억하지 않는다. 보통 스테이트리스(Stateless) 라고도 부르는 HTTP의 중요한 특징이다. 물론, 상태를 유지하기 위해서 쿠키(Cookie) 등의 기술을 사용할 수는 있다.
-

클라이언트, 서버 아키텍처



IE, 크롬 등 브라우저
PostMan
curl 등등..

여러분이 지금 만들고 있는
애플리케이션

네이버 Open API

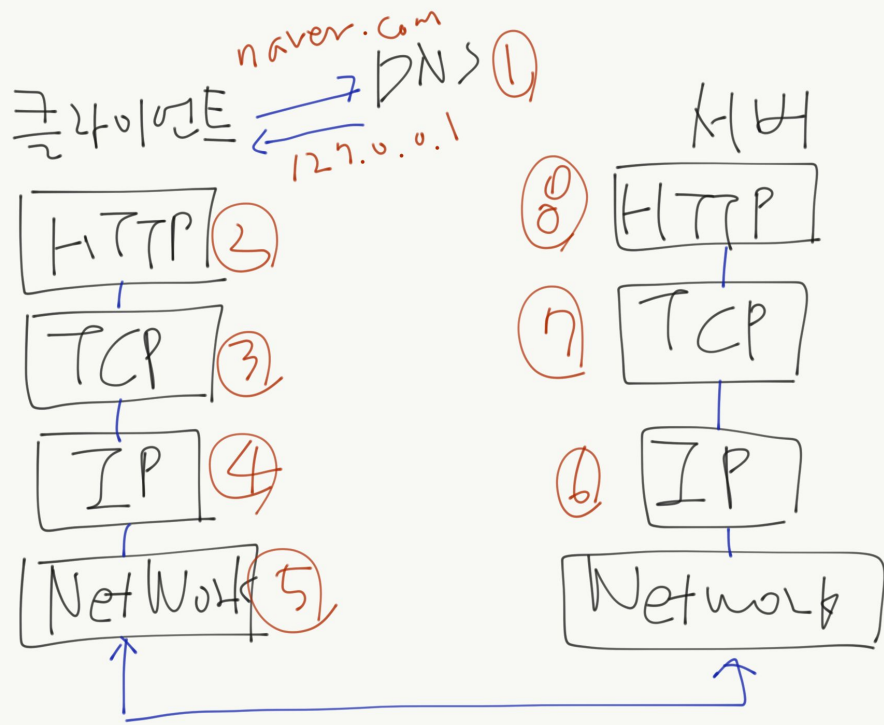
클라이언트에서 일어나는 일

1. 요청 메시지 구축
2. 요청 메시지 송신
3. (응답이 돌아올 때까지 대기)
4. 응답메시지 수신
5. 응답메시지 해석
6. 클라이언트의 목적을 달성하기 위한 처리 진행

서버에서 일어나는 일

1. (요청을 대기)
2. 요청 메시지 수신
3. 요청 메시지 해석
4. 적절한 애플리케이션 프로그램으로 처리를 위임
5. 애플리케이션 프로그램으로부터 결과를 취득
6. 응답 메시지 구축
7. 응답 메시지 송신

TCP/IP



TCP/IP

1. 클라이언트는 DNS 를 통해서 `naver.com` 의 IP 주소를 알 수 있다.
2. 클라이언트는 애플리케이션 (HTTP) 계층에서 HTTP 메시지를 작성한다.
3. 클라이언트는 전송계층 (TCP)에서 HTTP 를 패킷으로 분해한다.
- 4,5,6. 클라이언트는 IP 를 통해서 상대가 어디에 있는지 찾아 중계해 가면서 전송한다.
7. 서버는 전송계층 (TCP) 에서 패킷을 수신하고 조립한다.
8. 서버는 클라이언트의 요청을 처리한다.

HTTP 메시지

- 시작 줄(스타트라인, 리퀘스트 라인, 상태라인 등..번역서마다 다름)
- 헤더
- 본문

(1)HTTP 메시지, 시작 줄

Request 메시지

<메서드> <요청 URL> <버전>

GET /resource/a.png 1.1

Response 메시지

<버전> <상태 코드> <사유 구절, 또는 텍스트 구문 (번역서마다 다름...)>

HTTP 1.1 200 OK

(2) HTTP 메시지, 헤더

- 메시지 헤더는 본문에 대한 부가적인 정보를 표현
- **Date** : 메시지를 생성한 시간
- **Content-Type** : 메시지의 바디 내용이 어떠한 종류인가에 대한 타입 지정
 - 미디어 타입 및 문자 인코딩
 - "**Content-Type** : application/xml; charset=utf-8"
 - 미디어 타입, 문자 인코딩, 언어 태그 는 서버에서 지정하는 헤더인데 만약 클라이언트가 원하는 값이 있다면 **Accept** 관련 헤더로 요청
- **Content-Language** : 언어 태그

(3)HTTP 메시지, 본문

- 메시지 실제 정보를 담고 있다.

HTTP 상태코드..

-

쿠키

- HTTP 는 스테이트리스(Stateless)한 프로토콜
 - 서버가 클라이언트의 애플리케이션 상태를 보존하지 않는다
- 쿠키를 사용해서 HTTP 프로토콜의 단점을 해결

3.1.2 Rest API

- Rest API 에 대해서...

Rest API 설계 가이드

- **중요!!!!**
- Rest API 는
 - REST(Representational State Transfer)는 웹서비스에서 통신을 위한 아키텍처
- 메서드
 - GET : 지정된 URI에서 리소스의 표현을 조회한다.
 - POST : 지정된 URI에 신규 리소스를 생성한다.
 - PUT : 지정된 URI에 리소스를 생성하거나 업데이트한다.
 - PATCH : 리소스의 부분 업데이트한다.
 - DELETE : 지정된 URI의 리소스를 제거한다.

10분 이상 설명예정! 각자 따로 읽어보아야 함

<https://docs.microsoft.com/ko-kr/azure/architecture/best-practices/api-design>

스웨거 등 간단하게 설명

리소스	POST	GET	PUT	DELETE
/customers	새 고객 만들기	모든 고객 검색	고객 대량 업데이트	모든 고객 제거
/customers/1	오류	고객 1에 대한 세부 정보 검색	고객 1이 있는 경우 고객 1의 세부 정보 업데이트	고객 1 제거
/customers/1/orders	고객 1에 대한 새 주문 만들기	고객 1에 대한 모든 주문 검색	고객 1의 주문 대량 업데이트	고객 1의 모든 주문 제거

GraphQL(짧게 소개만...)

크롬 또는 Postman 으로 HTTP 요청

Chrome DevTools Network tab showing a failed GET request to `blog?q=%EC%95%84%EB%B0%94%ED%83%80`.

General

- Request URL: `http://localhost:8080/api/blog?q=%EC%95%84%EB%B0%94%ED%83%80`
- Request Method: GET
- Status Code: 500
- Remote Address: `[::1]:8080`
- Referrer Policy: no-referrer-when-downgrade

Chrome DevTools Network tab showing the same failed GET request. The **Response Headers** and **Request Headers** are visible.

Response Headers

- Connection: close
- Content-Language: ko-KR
- Content-Length: 471
- Content-Type: text/html; charset=UTF-8
- Date: Sat, 11 Jan 2020 17:15:49 GMT

Request Headers

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*
- Accept-Encoding: gzip, deflate, br
- Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
- Cache-Control: no-cache
- Connection: keep-alive
- Host: localhost:8080
- Pragma: no-cache
- Sec-Fetch-Mode: navigate
- Sec-Fetch-Site: none

3.2 스프링 에서 Rest API

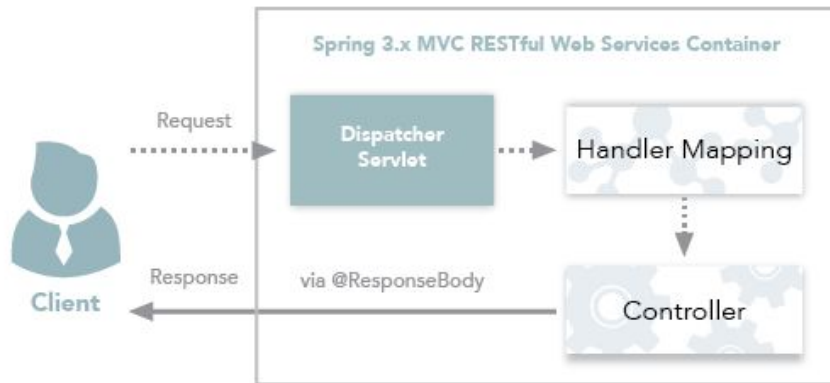
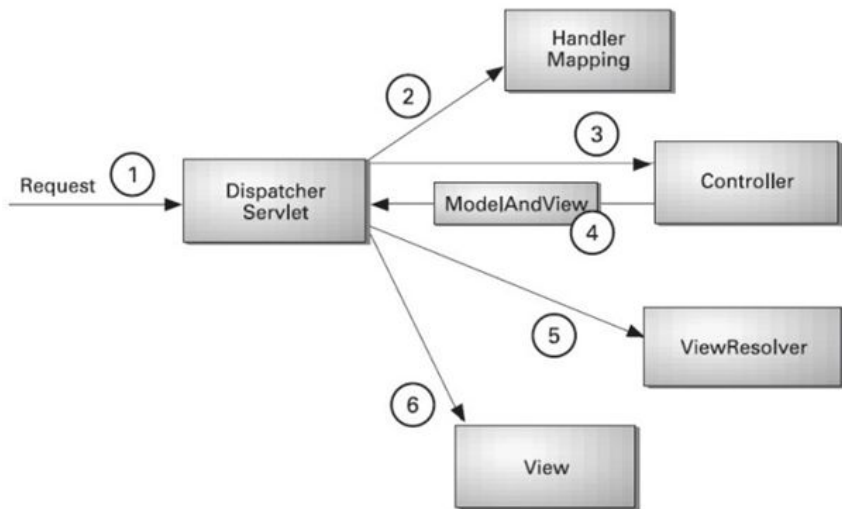
- RestController
- ResonseEntity
-

<https://www.baeldung.com/rest-with-spring-series>

3.2.1 @RestController

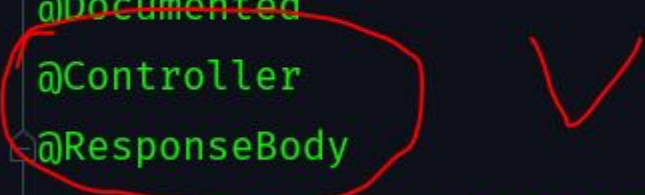
- @RequestBody, @ResponseBody
 - @RequestBody
 - JSON--> 객체 : HTTP 요청 시
 - , @ResponseBody
 - 객체 --> Json : HTTP 응답 시
 - @RestController 은 @Controller + @ResponseBody
 - JSON 포맷으로 데이터를 응답해줄 수 있음
 - @Controller vs @RestController
 - Controller 는 화면(뷰)를 리턴해주고, RestController 는 Json/XML 의 데이터를 리턴해준다.
 - Controller 도 ResponseBody 를 사용하면 Json 리턴이 가능함 (예전방식)
-

Spring MVC



@RestController

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@RestController
@ResponseBody
public @interface RestController {
```



```
@GetMapping("/articles")  
public BlogDTO findByQuery(@RequestParam(name = "q") String query){
```

3.2.2 ResponseEntity

- HTTP 상태코드, Header 등 직접 제어하고 싶은 경우 사용

Result:

▼ result = {BlogDTO@6369} "BlogDTO(total=0, items=[])"

▼ total = {Integer@6374} 0

value = 0

items = {ArrayList@6375} size = 0

▼ General

Request URL: http://localhost:8080/api/blog/arti
3216574415

Request Method: GET

Status Code: 204

Remote Address: [::1]:8080

Referrer Policy: no-referrer-when-downgrade

▼ Response Headers

[view source](#)

코드가 중요하지는 않음. 이런 방식으로 ResponseEntity 를 활용할수 있다는 사실을 인지하기 위한 샘플 코드..

```
@GetMapping("/articles")
public ResponseEntity<BlogDTO> findByQuery(@RequestParam(name = "q") String query) {

    BlogDTO blogDTO = blogService.findByQuery(query);

    if(blogDTO.getItems().isEmpty()){
        return new ResponseEntity<>(blogDTO, HttpStatus.NO_CONTENT);
    }
}
```

```
* @see <a href="https://www.iana.o
```

```
* @see <a href="https://en.wikiped
```

```
*/
```

```
public enum HttpStatus {
```

```
    // 1xx Informational
```

3.2.2 @RequestMapping

- d

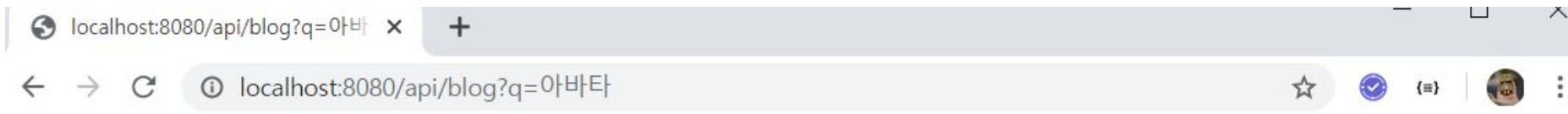
3.2.3 스프링부트에서 어떻게 오류처리를 하는지...

- BasicErrorController
 - Content type 에 따라서 오류처리 방식이 다름

3.2.4 @RestControllerAdvice

- d

Exception 처리가 제대로 안된 경우



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Jan 12 02:15:49 KST 2020

There was an unexpected error (type=Internal Server Error, status=500).

401 Unauthorized: [{"errorMessage":"Not Exist Client ID : Authentication failed. (인증에 실패했습니다).","errorCode":"024"}]

STEP 03

필수 요구사항(필수 요구사항을 구현하면 PR 보내도 됨)

1. 영화 데이터 조회 시 디폴트로 평점 높은 순으로 정렬(Sort)해서 제공한다.
2. 영화 데이터 조회 시 디폴트로 평점이 0.0 인 경우는 제외하고 제공한다.
3. 네이버 오픈 API 인증 실패 시 Exception(예외처리)를 구현한다.(@RestControllerAdvice, @ExceptionHandler 어노테이션을 활용해본다.)

선택 요구사항(시간 여유 있으신 분들만 해주세요)

1. 통합검색 API에서 블로그, 영화 조회 시 순차호출 방식이 아닌 병렬 프로그래밍으로 동시에 호출해서 데이터를 조합한다.
 2. DTO를 정의해서 데이터를 매핑한다. (예) 네이버 블로그 OpenAPI 에 조회하는 응답 Response Type 클래스가 Blog 라고 가정하면, 신규 클래스인 BlogDTO 클래스를 정의하고, 네이버 OpenAPI 를 통해서 가져온 데이터를 BlogDTO 에 매핑해서 사용한다.
-

4주차 예고

Spring Boot Test

4.1 TDD in Spring Boot

1. TDD 에 대해서
2. 통합테스트 VS 단위테스트
3. 스프링부트에서의 테스트

—

4.2 스프링부트 단위테스트

- 1.
- 2.

—

4.3 스프링부트 통합테스트

1. ○ ○

— O

5주차 예고

JPA or Spring Data

...

주제 고민 중..

- O

6주차 예고

Redis

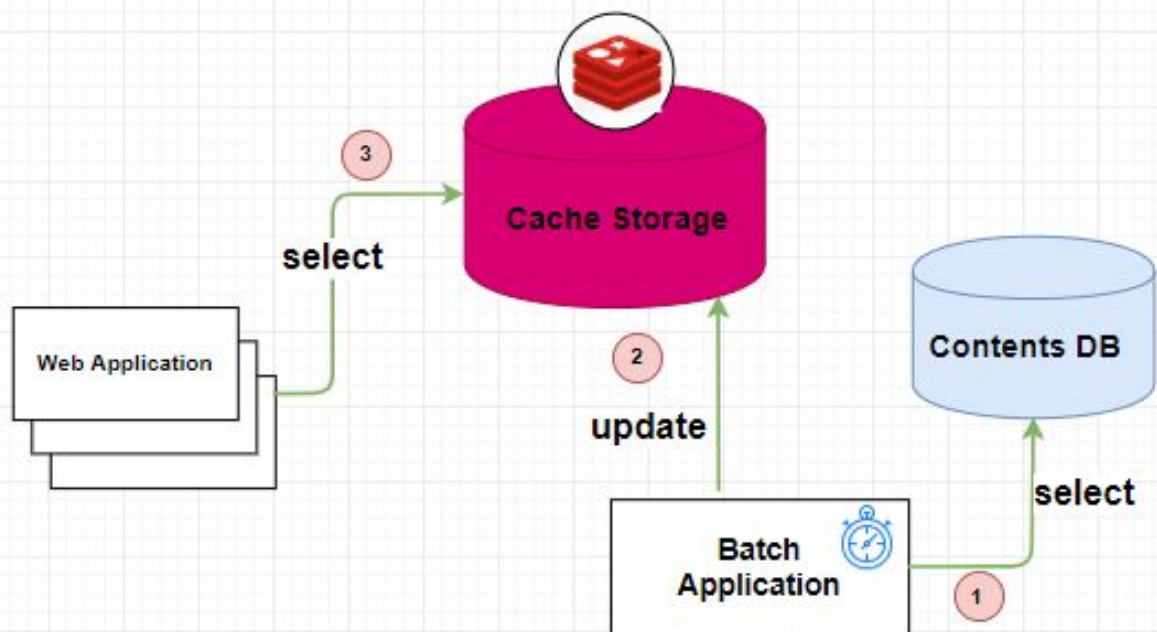
6.1 캐싱 패턴

- 공용 캐시가 필요한 이유
- 캐시 디자인 패턴

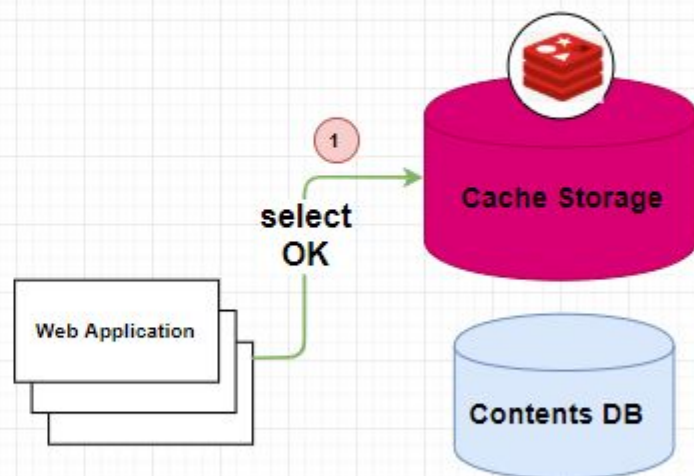
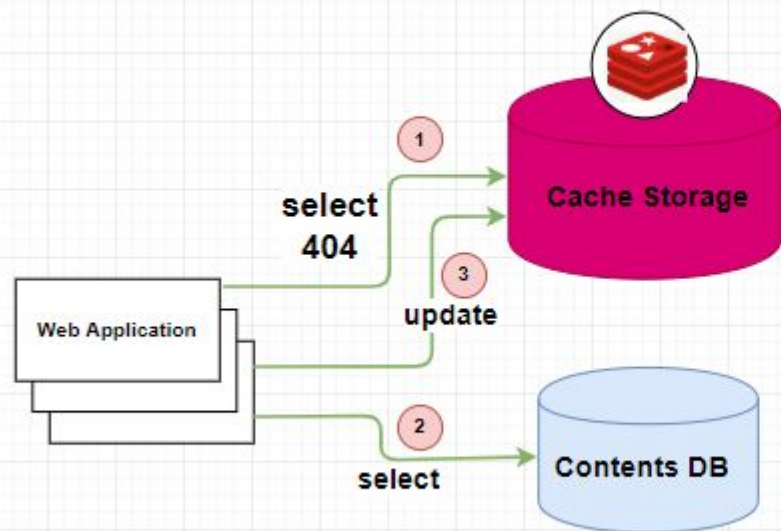
공용 캐시가 필요한 이유 설명.. 그림으로 하는게 좋을 듯

Cache Design Pattern

Inline Cache Pattern



Cache Aside Pattern



—

6.2 Redis

- Redis

Redis 에 대한 전반적인 설명

데이터 저장 타입

가용성을 위한 센티널, 클러스터 등
어떻게 동작하는지 등..

6.3 Redis in SpringBoot

- RedisTemplate
- Lettuce, Jedis, Redisson 등 라이브러리 소개
- Spring Boot Data Redis

—

6.3.1 RedisTemplate

—

6.3.2 Lettuce

—

6.3.3 Spring Data Redis

STEP 06(미리보기)

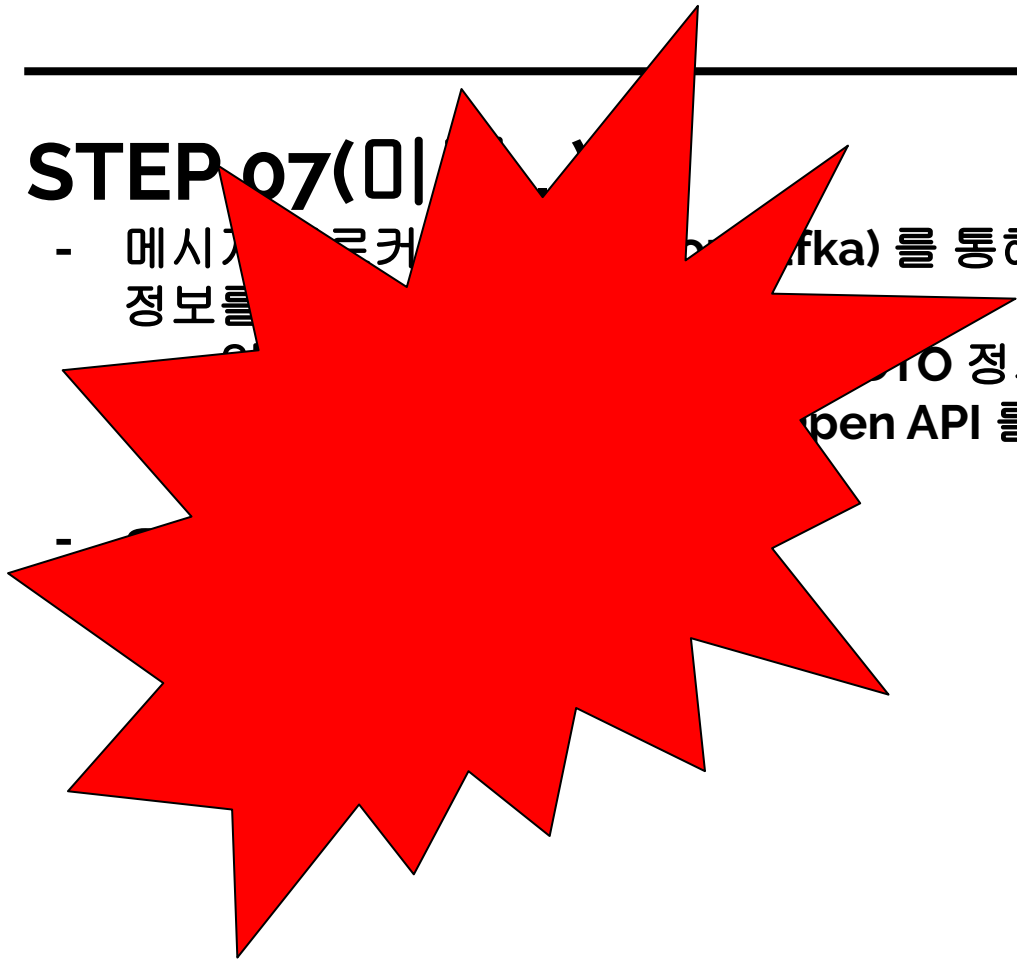
- OpenStack을 통해 서버 데이터를 Redis 캐시 스토리지로 옮긴다.
- Redis 캐시 서버가 없다면 Redis 캐시 서버를 설치한다.
- Redis 캐시 서버가 있으면 데이터를 사용하기 전에 Redis 캐시 서버에 데이터를 저장한다. 반대로 데이터가 Redis 캐시 서버에 없다면 Redis에 신규 데이터를 저장한다. (Redis에 신규 데이터를 저장할 때, Redis에 이미 저장된 데이터를 덮어쓰지 않는다. Redis는 LRU(Less Recently Used) 방식으로 오래된 데이터를 지운다. Redis에 오래된 데이터가 많으면 Redis가 느려진다. Redis에 오래된 데이터가 많으면 Redis가 느려진다. Redis에 오래된 데이터가 많으면 Redis가 느려진다.)
- Redis 캐시 서버가 없다면 Redis 캐시 서버를 설치한다.

7주차 예고

MQ(Message Queue)

STEP 07(미리보기)

- 메시지 큐 커넥터 (kafka) 를 통해서 캐시 정보를 외부로 전달 (API 정의..)
- open API 를 호출 후



8주차 예고

스터디 최종 마무리
