



“ 긍정적인 변화를 만드는 개발자 곽영현입니다.”



Github : <https://github.com/YoungHoney>.

블로그 : <https://younghoney.github.io/>

기본정보

Personal Information

이름	곽영현
생년월일	2000. 05. 15
연락처	010-9889-5275
메일	yhyh5275@naver.com
병역	대한민국 육군 병장 만기전역 (2020.03~2021.09)

학력사항

Education

대학교	2019.03~2025.02 중앙대학교 소프트웨어학과	4.1/4.5
고등학교	2016.03 ~ 2019.02 한솔고등학교 졸업	-

주요 수강 과목

Major

Course

알고리즘(3-1)	DP, Greedy, Graph에 관한 알고리즘을 학습	A+
운영체제(3-1)	OS의 자원관리, 프로세스, 동시성 제어에 대해 학습	A+
리눅스 시스템 응용 설계(3-2)	리눅스 커널의 구성과 시스템콜, 프로세스와 스레드, 동시성 제어와 스케줄링 알고리즘을 학습하고 실습을 진행	A+
데이터베이스시스템(4-1)	관계형 DB의 인덱싱, 쿼리최적화를 학습하고 간단한 블록단위 I/O기반의 DB시스템을 설계 및 구현	A
네트워크응용설계(4-1)	TCP의 신뢰성 보장, 흐름제어, 혼잡제어의 원리를 학습하고 go언어로 소켓 통신 프로그램을 만들며 실습을 진행	A

자격증 및 어학

Certifications &

Language Proficiency

자격증	TOPCIT(608, 수준3)
어학	TOEIC Speaking AL(160) TOEIC 880

기술 스택

Skill Set

Programming Language	Java
DB	MariaDB
Framework	Spring
Collaborations	Git, Github Jira, Confluence
Cloud	AWS

주요 프로젝트

우리조상알기 - 캡스톤디자인(1)

“우리 집안 우리 가족 조상정보 찾기 서비스”

작업 기간 : 2023.09.01-2023.12.21

작업인원 : 백엔드 2(본인 포함), 프론트1

맡은 역할 : ERD설계, 서비스 코드 작성, 배포

사용 메뉴얼 링크



주요 기술 키워드

1. Java
2. Spring, Spring JPA
3. AWS EC2, RDS-MariaDB
4. Azure OpenAI API
5. Git, Github

이지백엔드 - 캡스톤디자인(2)

“쉬운 백엔드 개발을 위한 자동 배포, 배포관리 서비스”

작업 기간 : 2024.03.01-2024.06.14

작업인원 : 백엔드 2(본인 포함), 프론트1

맡은 역할 : TypeScript를 사용한 Terraform 스크립트 실행 및 관리

주요 기술 키워드

1. Terraform
2. AWS EC2
3. Git, Github

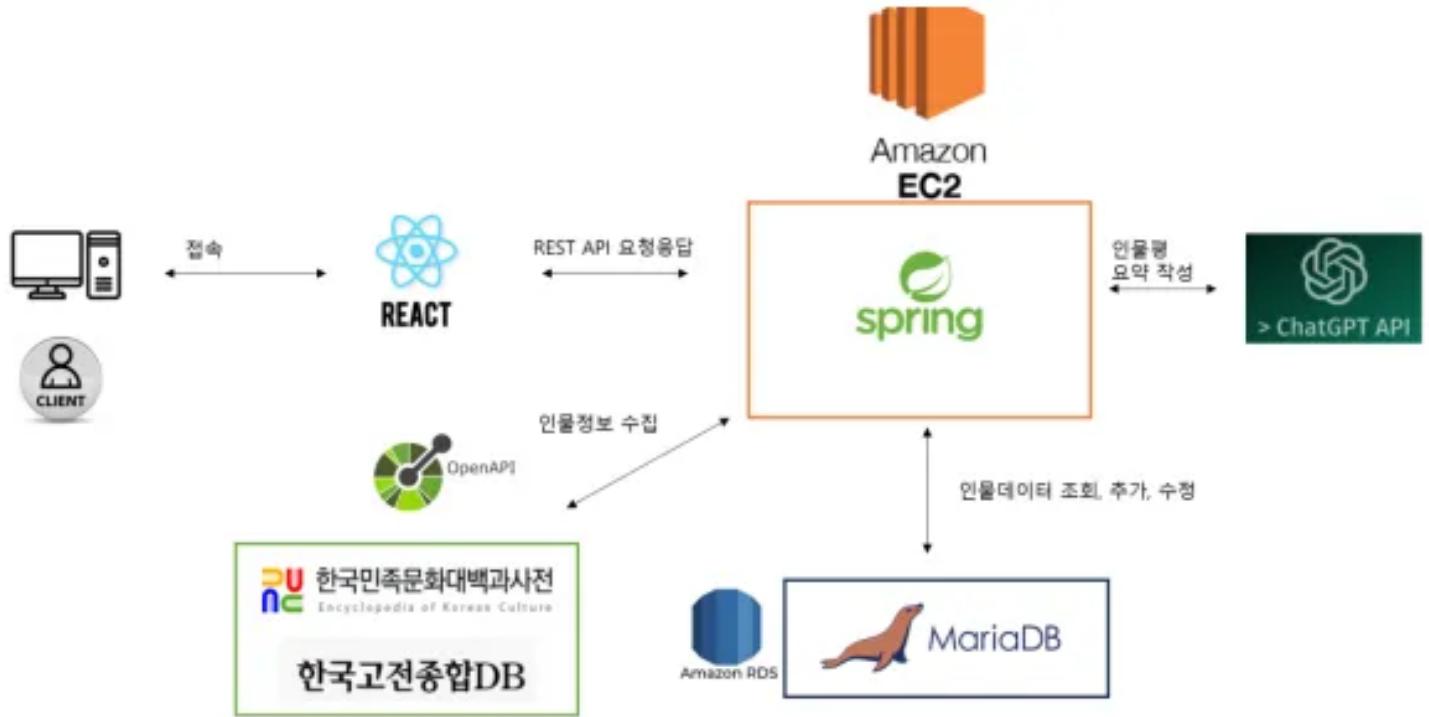
Line/Armeria의 다형성 기능 지원 기여

문서화 기능에 상속, 다형성 추가하기 ([#6313](#))

작업 기간 : 2025.08.10 - (진행중)

우리조상알기

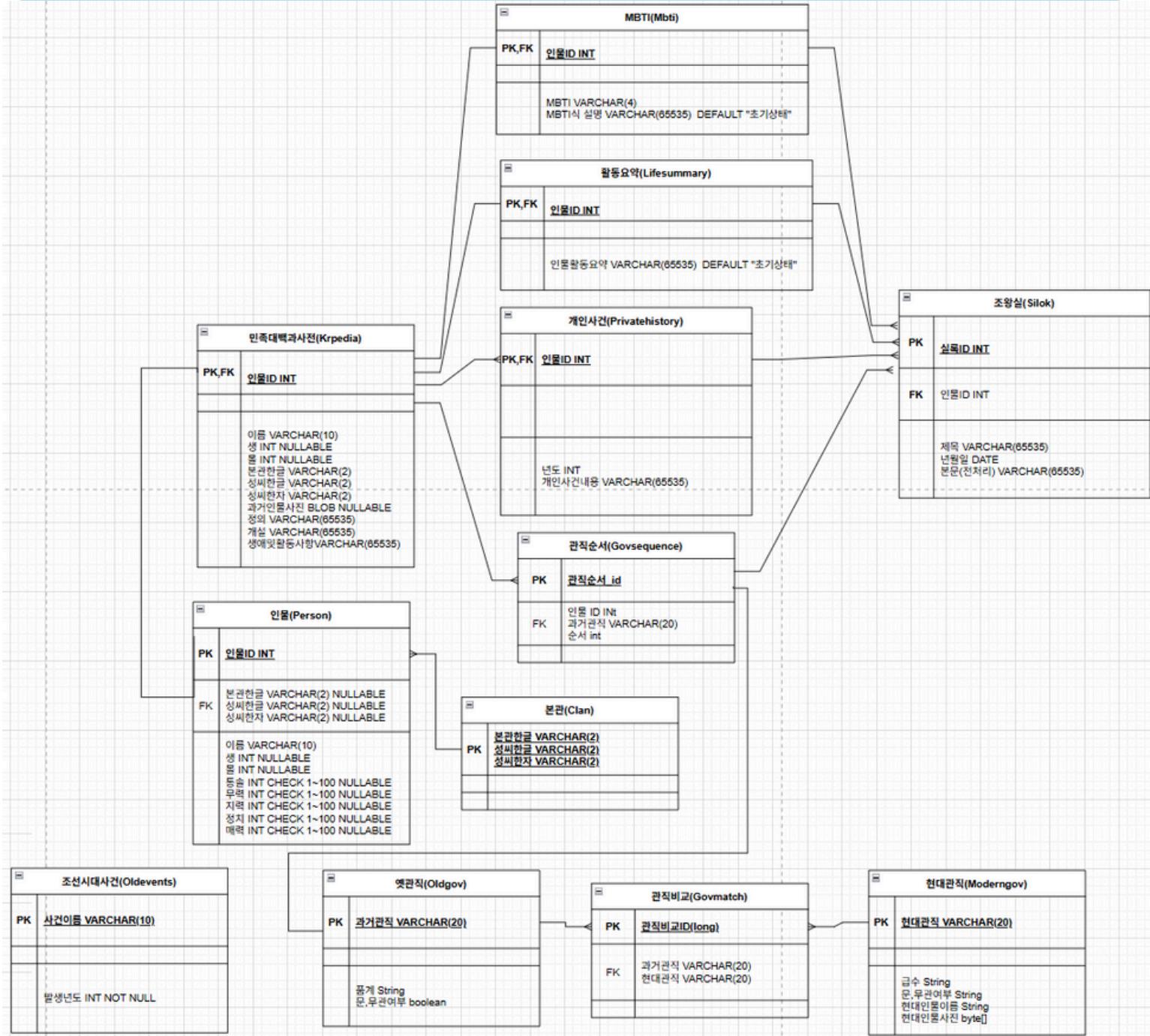
내가 담당한 역할 (설계, 기능)



이 프로젝트는 **Java Spring** 기반의 **모놀리식 아키텍처**로, **계층형 구조**를 가진 단일 백엔드 서버가 React 클라이언트로부터의 요청을 수신하여, OpenAPI 사양의 외부 API 서버로부터 동기 방식으로 데이터를 조회하고 MariaDB에 저장한 뒤, Azure OpenAI API와의 동기 통신을 통해 가공한 데이터를 클라이언트에 반환합니다.

우리조상알기

내가 담당한 역할 (설계, 기능)



프로젝트에서 사용된 ERD스키마입니다. 외부 API서버에서 민족대백과사전, 조왕실, 인물, 본관 테이블을 만들고 Azure OpenAI API를 통해 MBTI, 활동요약, 개인사건, 관직순서 테이블의 데이터를 구성합니다. 옛관직과 현대관직, 조선시대사건 데이터는 사전에 다운받아 구성합니다.

우리조상알기

```
@Entity  
@Data  
public class Oldgov {  
    @Id  
    @Column(name="oldname")  
    private String name;  
  
    @Column(name="oldrank")  
    private String rank;  
    private boolean iswarrior;  
  
    @OneToMany(mappedBy = "oldgov")  
    private List<Govmatch> govmatches=new ArrayList<>();  
  
}  
  
@Entity  
@Data  
public class Moderngov {  
    @Id  
    @Column(name="modernname")  
    private String name;  
  
    @Column(name="modernrank")  
    private String rank;  
  
    private boolean iswarrior;  
  
    @Column(name="modernpersonname")  
    private String personname;  
  
    @Column(name="modernpersonpicture")  
    private String personpicture;  
  
    @OneToMany(mappedBy = "moderngov")  
    private List<Govmatch> govmatches=new ArrayList<>();  
}
```

```
@Entity  
@Data  
public class Govmatch {  
  
    @Id  
    @GeneratedValue  
    @Column(name="Govmatch_id")  
    private Long id;  
  
    @ManyToOne(fetch=FetchType.LAZY)  
    @JoinColumn(name="oldname")  
    private Oldgov oldgov;  
  
    @ManyToOne(fetch=FetchType.LAZY)  
    @JoinColumn(name="modernname")  
    private Moderngov moderngov;  
  
}
```

앞서 전처리된 관직 데이터의 경우, 위와 같이 사용됩니다. 조선시대의 관직정보를 담은 Oldgov, 현대 공무원의 계급을 나타내는 moderngov는 다대일, 일대다의 관계로 Govmatch에 의해 이어집니다.

우리조상알기

내가 담당한 역할 (트러블 슈팅)

DB연동과 관련하여, H2를 사용하다가 AWS RDS-MariaDB로 전환하면서, 분명히 인코딩 방식을 한글-한자에 맞게 수정했는데 적용되지 않았습니다. AWS홈페이지에서도 매개변수를 수정하고, Workbench를 통해 DB에 직접 접속하여 수정했는데도 반영되지 않았습니다.

```
@Entity
@Getter @Setter
public class Lifesummary {
    @Id
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="person_id")
    private Krpedia krpedia;

    @Lob
    @Column(name="lifesummarycontents",columnDefinition = "TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci")
    private String contents;

    @OneToMany(mappedBy = "lifesummary")
    private List<Silok> siloks=new ArrayList<>();
```

결국엔 위 코드의 contents와 같이 columnDefinition="TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci"라고 직접적으로 추가를 해서 해결했습니다.

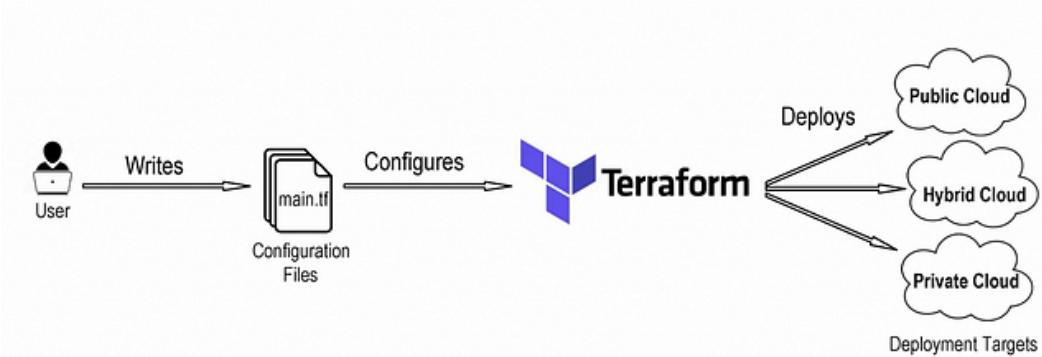
우리조상알기

또, Spring JPA에서 다대다 관계를 피하는게 좋다고 알고있었습니다. 그래서 인물과 그 본관(가문)에 대한 관계를 동명이인을 고려하여 다대다 관계로 상정했는데, 이 경우 그 관계의 복합키를 따로 클래스로 정의해서 다대다 관계를 다대일, 일대다 관계로 만들게 되었습니다.

```
@Entity  
@Getter @Setter  
public class Person {  
  
    @Id @GeneratedValue  
    @Column(name="person_id")  
    private Long id;  
  
    @ManyToOne(fetch=FetchType.EAGER)  
    @JoinColumns(value = {@JoinColumn(name = "clanHangul"),  
        @JoinColumn(name = "surnameHangul"),  
        @JoinColumn(name = "surnameHanja")})  
    private Clan clan; //본관과 양방향 다대일  
  
    @Entity  
    @Getter @Setter  
    public class Clan {  
  
        @Transient  
        @JsonIgnore  
        private static final char[] CHO =  
            {'ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅂ', 'ㅁ', 'ㅂ', 'ㅅ',  
             'ㅇ', 'ㅈ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ'};  
  
        @EmbeddedId  
        // @Column(length=2)  
        private ClanId clanid;  
  
        @JsonIgnore  
        @OneToMany(mappedBy = "clan",fetch=FetchType.EAGER)  
        private List<Person> persons=new ArrayList<>();  
  
        @Data  
        @Embeddable  
        public class ClanId implements Serializable {  
  
            @Column  
            private String clanHangul;  
            @Column  
            private String surnameHanja;  
            @Column  
            private String surnameHangul;
```

이지백엔드

내가 담당한 역할 (설계, 기능)



한 사용자가 하나의 Key Pair로 여러 인스턴스를 사용하는 경우를 가정하였고 따라서 작성해야 할 Terraform 스크립트는 다음과 같이 나뉩니다.

1. 한 사용자에 대한 공통 스크립트, 예를들어 여러 인스턴스에 접속하기위한 KeyPair 스크립트
 2. 한 사용자에 대한 개별 스크립트, 예를들어 한 사용자가 사용하는 여러 인스턴스에 대한 인스턴스 스크립트
- 1과 같은경우 Terraform으로 별도의 .tf형식의 파일로 작성했으나, 2와 같은경우 유연한 설정을 위하여 TypeScript코드 내부에서 Terraform 스크립트를 작성하였습니다.

```
private_key.tf ×  
terras > private_key.tf  
1 resource "tls_private_key" "cicd_make_key" {  
2   algorithm = "RSA"  
3   rsa_bits  = 4096  
4 }  
5  
6 resource "aws_key_pair" "cicd_make_keypair" {  
7   key_name    = "cicd_key"  
8   public_key  = tls_private_key.cicd_make_key.public_key_openssh  
9 }  
10  
11 resource "local_file" "cicd_downloads_key" {  
12   filename = "cicd_key.pem"  
13   content  = tls_private_key.cicd_make_key.private_key_pem  
14 }
```

이지백엔드

```
TS deployService.ts 9+ ×
src > services > TS deployService.ts > ...
1 import { exec, spawn } from 'child_process';
2 import * as fs from 'fs';
3 import * as path from 'path';
4
5
6
7 class DeployService {
8
9     private ami = 'ami-01ed8ade75d4eee2f';
10    private instanceType = 't2.micro';
11    private privateKeyPath = 'terras/cicd_key.pem';
12
13
14
15    createTerraformConfig(instanceName: string, ami: string, instanceType: string): Promise<void>
16        return new Promise((resolve, reject) => {
17            const config = `

18
19 resource "aws_iam_role" "${instanceName}_role" {
20     name = "${instanceName}_role"
21
22     assume_role_policy = <<EOF
23 {
24     "Version": "2012-10-17",
25     "Statement": [
26         {
27             "Effect": "Allow",
28             "Principal": {
29                 "Service": "ec2.amazonaws.com"
30             },
31             "Action": "sts:AssumeRole"
32         }
33     ]
34 }
35 EOF
36 }

37
38 resource "aws_iam_role_policy_attachment" "${instanceName}_role_attach" {
39     role      = aws_iam_role.${instanceName}_role.name
40     policy_arn = "arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy"
41 }
```

한편, EC2인스턴스를 실행시키고 코드를 배포하기 전에 몇몇 사전작업이 필요하므로 ssh를 통해 접속하여 각종 명령어를 실행했습니다. 아래는 서버 코드가 담긴 .zip파일을 업로드한 경우, scp로 전송하고 unzip하는 코드입니다.

```
executeUnzipAndListFiles(instanceIp: string, zipFilePath: string, targetDirectory: string, privateKeyPath: string): Promise<string> {
    return new Promise((resolve, reject) => {
        const sshCommand = `ssh -i ${privateKeyPath} -o StrictHostKeyChecking=no ubuntu@${instanceIp} "unzip -o ${zipFilePath} -d ${targetDirectory} && ls ${targetDirectory}"`;
        exec(sshCommand, (error, stdout, stderr) => {
            if (error) {
                reject(error);
                return;
            }
            if (stderr && !stderr.includes("Permanently added")) {
                console.warn(`stderr: ${stderr}`);
            }
            resolve(stdout);
        });
    });
}
```

이지백엔드

인스턴스의 배포 이후 인스턴스 상태확인과 간단한 버전관리 기능이 제공됩니다. 이때, Terraform의 기능으로서 서버를 닫거나 재시작하는 것보다 TypeScript 코드를 이용하여 ssh로 접속하는 것이 더 편리하다고 생각하여 이 방법으로 서버관리를 진행했습니다. 아래는 8080포트에 열린 서버를 닫는 코드입니다.

```
executeClosePortCommand(instanceIp: string, privateKeyPath: string): Promise<string> {
  return new Promise((resolve, reject) => {
    const sshCommand = `ssh -i ${privateKeyPath} -o StrictHostKeyChecking=no ubuntu@${instanceIp} "sudo lsof -t -i :8080 | xargs sudo kill -9"`;
    exec(sshCommand, (error, stdout, stderr) => {
      if (error) {
        reject(error);
        return;
      }
      if (stderr) {
        reject(new Error(stderr));
        return;
      }
      resolve(stdout);
    });
  });
}
```

내가 담당한 역할 (트러블 슈팅)

인스턴스를 생성 후 SCP, SSH접속을 하기위해 privatekey파일이 필요합니다. 그러나 배포된 Ubuntu(=Linux)환경에선 키 파일의 권한이 너무 넓게 설정되어있으면 보안문제가 발생하여 연결을 차단하므로 적절히 제한해야 합니다.

linux에서는 chmod를 사용하여 권한설정이 쉽지만 윈도우는 다소 복잡했습니다. “icacls” 명령어를 사용하는 일련의 명령어들인데, 다음과 같습니다.

```
icacls "..../key.pem" /reset
icacls "..../key.pem" /grant:r "%USERNAME%:R"
icacls "..../key.pem" /inheritance:r
```

Line/Armeria 기여

이슈 발견 및 DocService 구조 분석

Issue 링크



line/armeria

#6313 Support for inheritance and polymorphism in...

2 comments

icepeppermint opened on July 17, 2025

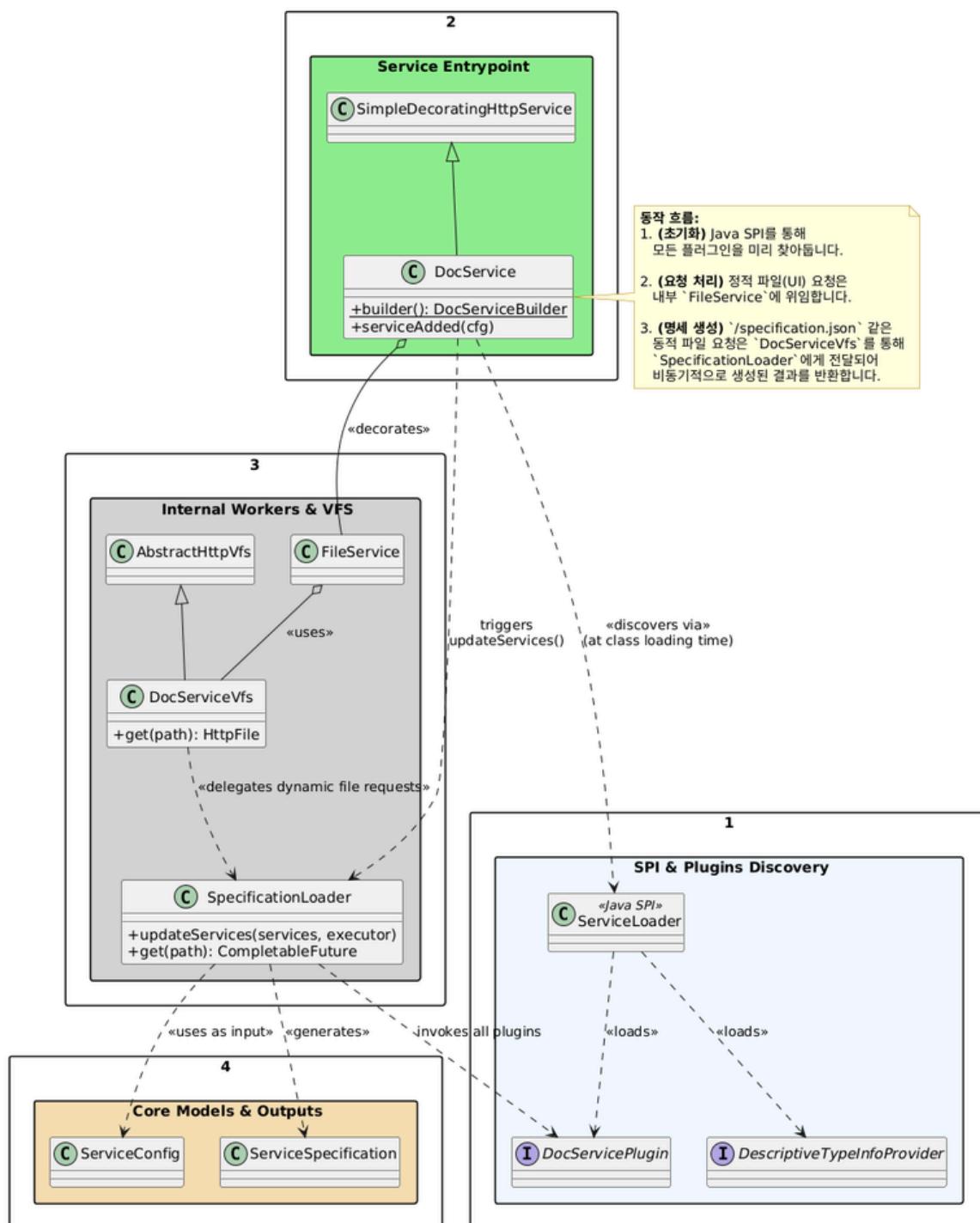


Support for inheritance and polymorphism in DocService · Issue #6313 · line/armeria

I'm using DocService to document our API. Currently, the response classes of the annotated HTTP service use inheritance and polymorphism. There doesn't seem to be any support for this case ...

Github

DocService Architecture and Flow (Improved Readability)



Line/Armeria 기여

병렬 빌드 관련 Discussion

Discussion 원본 링크



병렬 빌드 과정에서 Flaky Test와 JVM 종료 관련 에러가 발생하여 Discussion에 보고했습니다.

line/armeria

#6369 Request Guidance on Build Issues in my feature...

YoungHoney opened on August 26, 2025

Hello ~ I'm currently working on #6313 to implement Jackson polymorphism support for DocService as requested in the community. The good news is that the core feature implementation is comple...

[GitHub](#)

Summary of Investigation

To demonstrate the instability, I ran the full `clean build --parallel` process three times on my feature branch. Here are the results:

Run #	Full Build Result	Failing Task(s) & Error Type	Individual Re-run Result
1	Failed	:kubernetes:shadedTest :grpc:shadedTest (AssertionException)	Succeeded
2	Failed	:xds:shadedTest (TimeoutException)	Succeeded
3	Failed	:spring:boot2-webflux-autoconfigure:shadedTest :spring:boot3-webflux-autoconfigure:shadedTest (JVM did not terminate cleanly)	Failed

minwoox last month Maintainer

영어로 된 원본 주석 - 한국어로 번역

:spring:boot2-webflux-autoconfigure:shadedTest
:spring:boot3-webflux-autoconfigure:shadedTest
(JVM did not terminate cleanly)

This will be fixed by this PR:
[#6379](#)

[Unmark as answer](#) [↑ 1](#) [1](#)

YoungHoney last month Author

영어로 된 원본 주석 - 한국어로 번역

Thank you !

Line/Armeria 기여

상속, 다형성 기능을 추가한 PR 제출

PR 원본 링크



line/armeria

#6370 **feat(docservice): Support Jackson polymorphism...**

7 comments 8 reviews 19 files +1406 -670

YoungHoney · August 26, 2025 · 1 commit

feat(docservice): Support Jackson polymorphism annotations by YoungHoney · Pull Request #6370

Motivation This pull request implements support for Jackson's polymorphism annotations (@JsonTypeInfo, @JsonSubTypes) in DocService, as requested in the community (issue #6313). Currently, ...

[GitHub](#)

- Added `JacksonPolymorphismTypeInfoProvider`: A new provider that uses pure Java reflection to safely inspect `@JsonTypeInfo` and `@JsonSubTypes` annotations. It is registered via Java's SPI mechanism to be discoverable by `DocService`.
- Added `DiscriminatorInfo`: A new data class to hold polymorphism metadata extracted from the annotations.
- Consolidated Type Utilities: General-purpose type conversion logic (e.g., `toTypeSignature`) was moved from a separate `DocServiceTypeUtil` into `AnnotatedDocServicePlugin` for better cohesion.
- Updated `StructInfo`: Modified to include `oneOf` and `discriminator` fields to carry polymorphism information.
- Updated `JsonSchemaGenerator`: The generator now recognizes the new fields in `StructInfo` and correctly produces JSON Schema with `oneOf` and `discriminator` properties.
- Added `PolymorphismDocServiceExample`: A new example service to demonstrate and manually verify the feature.



minwoox reviewed 3 weeks ago

[View reviewed changes](#)

minwoox left a comment

[Contributor](#) [...](#)

Left a few more suggestions. 😊
Please, keep up the great work. 👍



1

core/src/main/java/com/linecorp/armeria/server/docs/DiscriminatorInfo.java [Outdated](#)

[Show resolved](#)

core/src/main/java/com/linecorp/armeria/server/docs/DocServiceTypeUtil.java

[Show resolved](#)

core/src/main/java/com/linecorp/armeria/server/docs/JacksonPolymorphismTypeInfoProvider.java

[Show resolved](#)

core/src/main/java/com/linecorp/armeria/server/docs/JacksonPolymorphismTypeInfoProvider.java

[Outdated](#)

[Show resolved](#)

내용이 많아 PR 관련 내용을 전부 담지 못했습니다. 더 자세한 진행 상황은 PR 링크 참고 부탁드립니다.