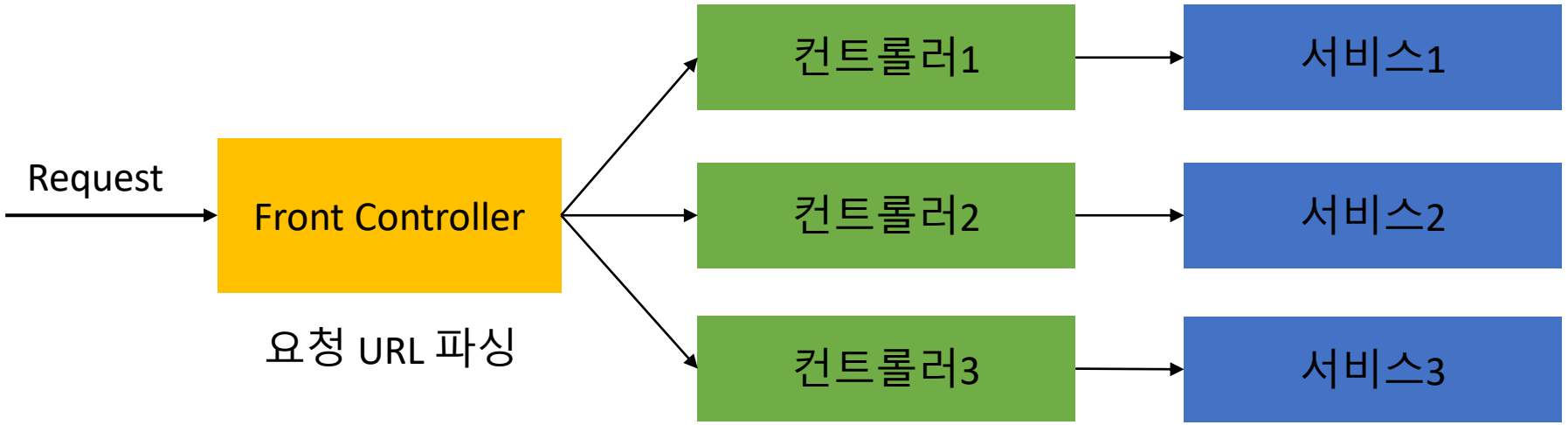
스프링 MVC

https://www.baeldung.com/

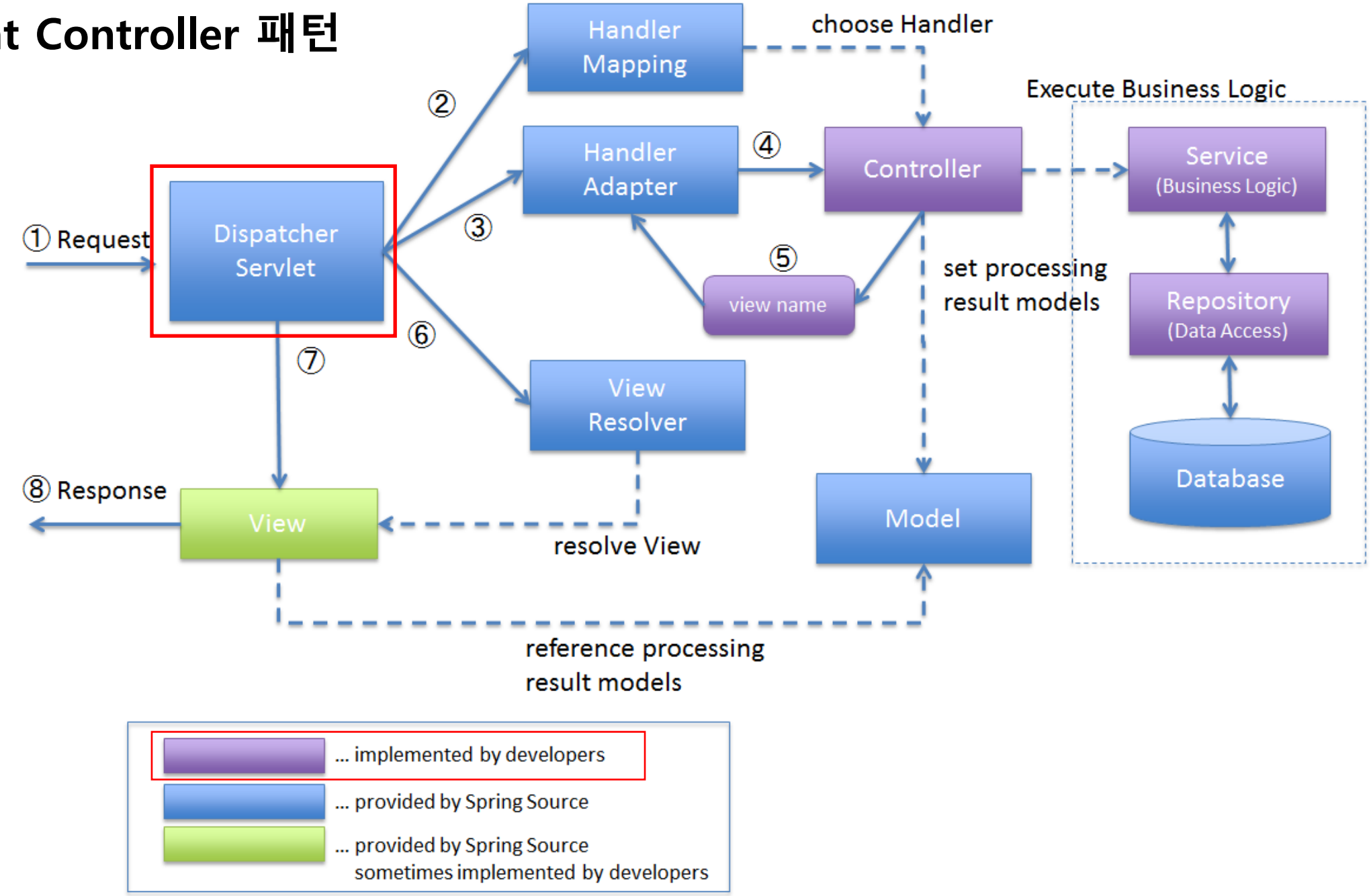
https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

Front Controller 패턴

- **모든 요청은 일단 Front Controller라는 입구를 통과하고 여기서 분기되는 구조**
- **Front Controller의 필요 기능**
  - 요청 종류 파악(Parsing)
  - 처리 가능한 컨트롤러 찾기
  - 처리 가능한 컨트롤러에 작업 위임
  - 컨트롤러 실행결과를 view에 전달
  - html에 데이터를 binding하여 응답 페이지 생성 및 전달



Front Controller 패턴



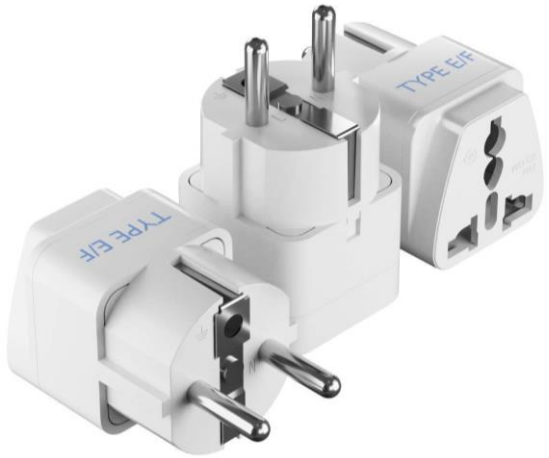
https://terasolunaorg.github.io/guideline/1.0.1.RELEASE/en/Overview/SpringMVCOverview.html

Handler Mapping

- **Handler Mapping**
  - 프론트 컨트롤러의 역할을 수행하는 DispatcherServlet은 HTTP request를 받아 처리를 담당할 컨트롤러를 찾음
  - Handler Mapping을 통해 적절한 핸들러를 찾았다면 핸들러 호출은 Handler Adapter를 통해 수행
  - 핸들러를 등록하고 찾는 방법이 다양함에 따라 Handler Mapping도 여러 종류가 존재
    - BeanNameUrlHandlerMapping
      - 요청 url과 같은 이름의 빈을 찾음: ex) 요청: "/foo", 빈 이름 "/foo" → 매칭(정규 표현식과 같이 패턴 매칭도 가능)
    - SimpleUrlHandlerMapping
      - URL 패턴에 매핑되는 지정된 Controller를 사용하는 방법
    - RequestMappingHandlerMapping: 어노테이션이 붙은 컨트롤러를 매칭

Handler Adapter

- **Handleradapter**
  - HandlerMapping을 통해 핸들러를 찾았다면 해당 핸들러의 실행은 Handleradapter를 통해 수행
  - adapter: 실행과 반환에 대한 규약만 지키면 사용자가 다양한 형태로 핸들러 정의 가능
  - invokeHandlerMethod를 통해 핸들러를 실행하고 실행 결과를 ModelAndView로 변환해서 반환
  - 종류
    - SimpleControllerHandlerAdapter
    - SimpleServletHandlerAdapter
    - AnnotationMethodHandlerAdapter
    - RequestMappingHandlerAdapter
      - RequestMappingHandlerMapping을 사용하면 선택되는 Adapter



View Resolver

- **ViewResolver**
  - ModelAndView에 실행 결과를 담았다면 응답으로 사용될 html 파일을 찾아야 함
    - ModelAndView의 setViewName, addObject 메소드를 이용하여 각각 html파일과 전달할 데이터를 저장
  - 뷰의 논리 이름을 물리이름으로 바꿈
  - "/WEB-INF/views/board.jsp" → view를 지정할 때 매번 이렇게 전체를 지정하는 것은 번거로움
    - 논리적 이름: board
    - 물리적 이름: /WEB-INF/views/board.jsp
  - 프로그래머는 prefix와 suffix를 제외한 "board"만 지정하면 됨
  - 단, 특별한 설정이 없다면 특정 경로 하위에 view 파일을 위치시켜야 함
    - ex) thymeleaf → resources/templates

템플릿 엔진

- **정의**
  - 템플릿 양식과 특정 데이터 모델에 따른 입력 자료를 합성하여 결과 문서를 출력하는 소프트웨어
- **대표 템플릿 엔진의 특징**
  - JSP: 스프링 부트에서는 권장하지 않음. 특히 JSP를 사용할 경우, WAR로 배포해야 함
  - Thymeleaf: 스프링 진영에서 권장
  - Mustache**:** 문법이 다른 템플릿 엔진보다 심플하지만 기능이 제한적

스프링 부트

자동 설정

- **@SpringBootApplication**

```
@SpringBootApplication
public class SeedstarterApplication {
```

  - @SpringBootConfiguration
    - 스프링에서는 @Configuration이 붙은 클래스를 스프링 설정 파일로 인식

```
@SpringBootApplication
        -------> @SpringBootConfiguration
                -------> @Configuration
```

  - @ComponentScan
    - @Component 어노테이션을 가진 Bean들을 스캔해서 등록
    - @Configuration, @Repository, @Service, @Controller, @RestController
  - @EnableAutoConfiguration
    - @ComponentScan으로 빈이 등록된 이후, 추가적인 Bean들을 읽어 등록

자동 설정

- **웹 관련 추가 설정이 필요할 때**
  - EnableAutoConfiguration에는 WebMvcAutoConfiguration이 포함되어 있음
  - WebMVC와 관련된 설정 정보를 가지고 있음
  - 스프링 MVC 어플리케이션 개발 시 직접 설정해주어야 했던 것들을 스프링 부트가 자동으로 설정

```
public class WebMvcAutoConfiguration {
```

  - 스프링 부트가 제공하는 MVC 기능을 모두 사용하면서 **추가적인 설정**을 하는 방법

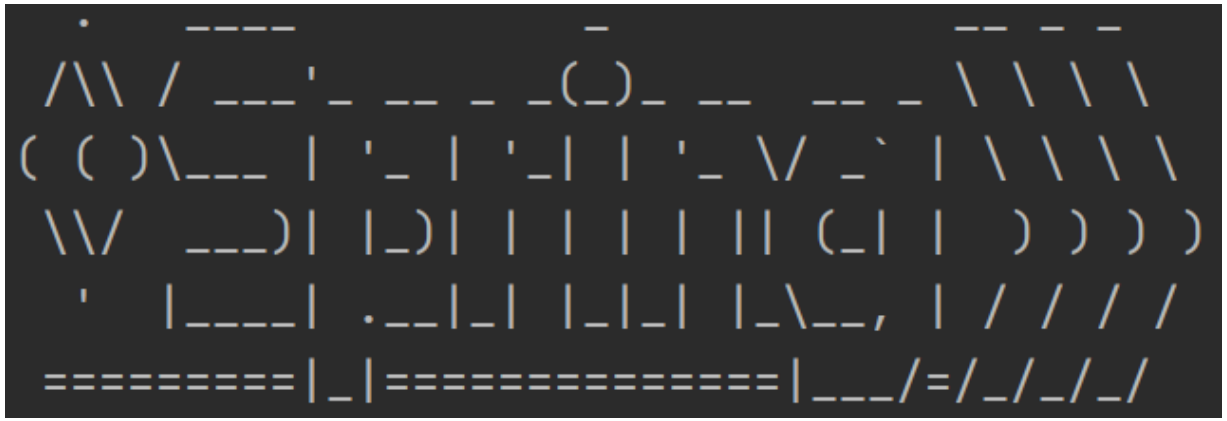
```
@Configuration
public class CustomConfig implements WebMvcConfigurer {

}
```

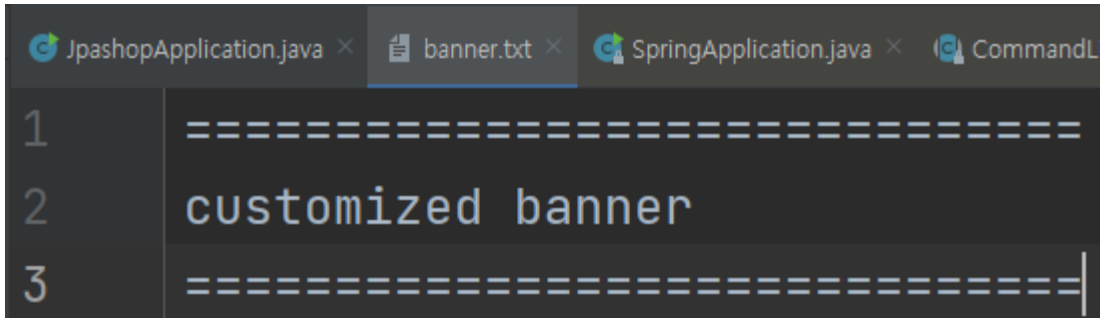
    - @EnableWebMvc은 붙이면 안됨(부트 설정이 무시됨)

SpringApplication

- **배너(banner)**



  - 수정 방법
    - src → main → resources → banner.txt 파일 생성



외부 설정

- **application.properties**
  - Spring Boot가 application.properties에 명시된 설정 정보를 기본적으로 읽음
  - key와 value로 표현
  - 'src/main/resources' 폴더에 위치

```
name = spring
```

```
@Value("${name}")
private String name;
@RequestMapping("/log-test")
public String logTest() {
    System.out.println("name=" + name);
    return name;
}
```

- 프로퍼티를 정의하는 방법은 다양하며 우선순위가 존재

외부 설정

- **YAML is a human-friendly notation used in configuration files**
- **properties 대비 장점**
  - 계층구조로 이루어져 가독성이 좋음
  - application.yml파일을 생성

```yaml
name: spring2
spring:
  output:
    ansi:
      enabled: always
```