Dispatcher Servlet

디스패처 서블랫이란?

- dispatch : 보내다.
- HTTP 프로토콜로 들어오는 모든 요청을 가장 먼저 받아 적합한 컨트롤러에 위임해주는 프론트 컨트롤러

> 클라이언트 요청 -> 서블릿 컨테이너로 전달 -> 디스패처 서블릿이 먼저 받음 -> 먼저 공통작업 처리 -> 해당 요청을 처리해야하는 컨트롤러를 찾아서 작업을 위임

장점

- 과거 : 모든 서블릿을 url 매핑을 위해 web.xml에 등록
- Dispatcher-servlet : 해당 어플리케이션으로 들어오는 모든 요청을 핸들링, 공통작업을 처리
- 컨트롤러만 구현하면, 디스패처 서블릿이 알아서 적합한 컨트롤러로 위임을 해주는 구조

단점

- 장작 자원의 처리
    - 모든 요청을 처리하다보니 HTML/CSS/JAVASCRIPT 등과 같은 정적 파일에 대한 요청마저 모두 가로채는 까닭에 정적자원을 불러오지 못하는 상황 발생
        - 2가지 해결방법
            - 정적 자원 요청과 애플리케이션 요청을 분리
            - 애프리케이션 요청을 탐색, 없으면 정적 자원 요청으로 처리

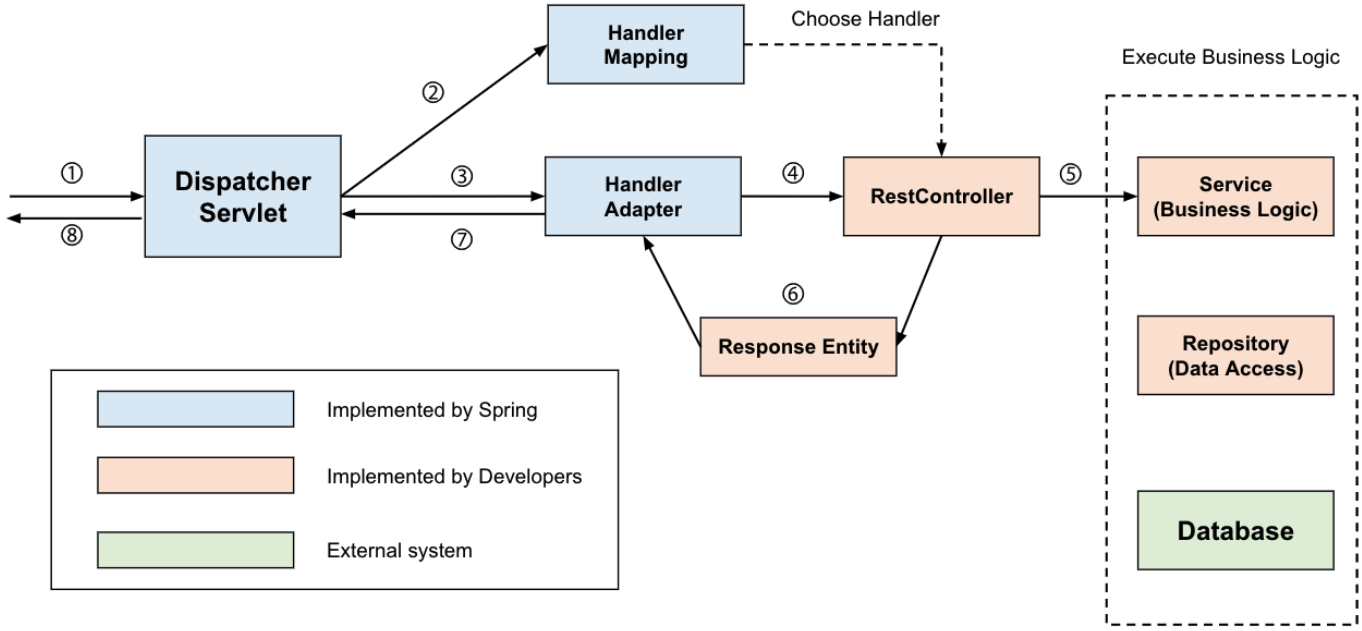
정적 자원 요청과 애플리케이션 요청을 분리

- 해결책 2가지
    - /apps의 url로 접근하면 디스패처 서블릿이 담당
    - /resources의 url로 접근하면 디스패처 서블릿이 컨트롤 할 수 없으므로 담당하지 않는다.
- 단점
    - 코드가 지저분, 모든 요청에 대한 저런 url을 붙여주어야함

애플리케이션 요청을 탐색하고 없으면 정적 자원 요청으로 처리

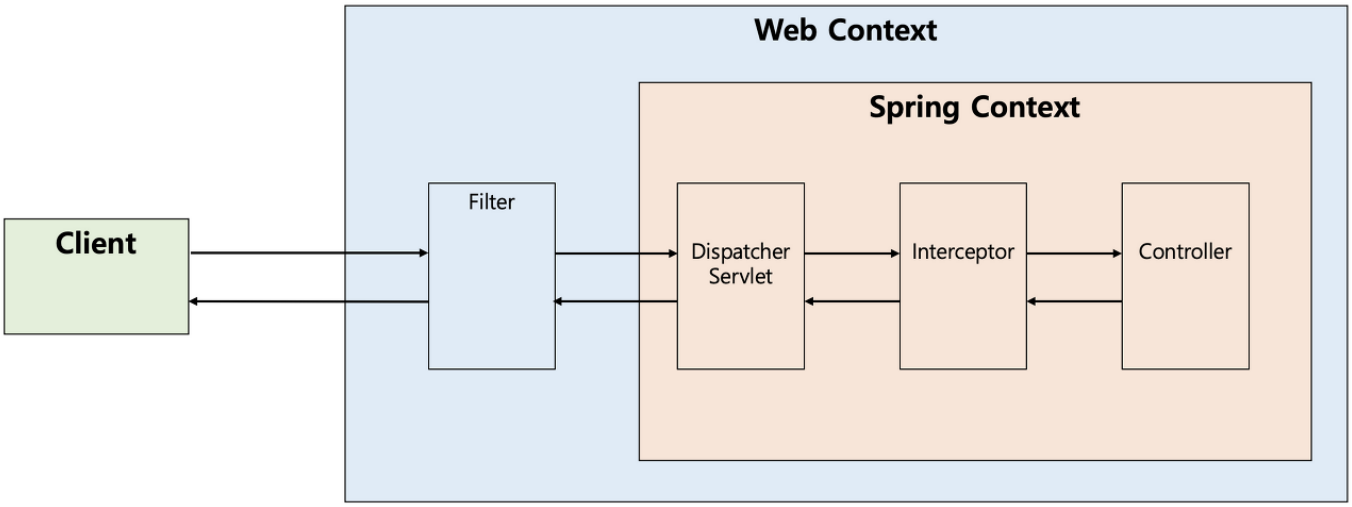
- 처리할 컨트롤러를 먼저 찾음, 요청에 대한 컨트롤러를 찾을수 없는 경우 2차적으로 설정된 자원 경로를 탐색해 자원을 탐색
- 확장 용이, 효율적인 리소스 관리

디스패처 서블릿의 동작 과정



1. 클라이언트 요청을 디스패처 서블릿이 받음
2. 요청 정보를 통해 요청을 위임할 컨트롤러를 찾음
3. 요청을 컨트롤러로 위임할 핸들러 어댑터를 찾아서 전달
4. 핸들러 어댑터가 컨트롤러로 요청을 위임
5. 비즈니스 로직을 처리
6. 컨트롤러가 반환값을 반환
7. 핸들러 어댑터가 반환값을 처리
8. 서버의 응답을 클라이언트로 반환

1. 클라이언트의 요청을 디스패처 서블릿이 받음



- 디스패처 서블릿 : 가장 먼저 요청을 받는 프론트 컨트롤러
- 서블릿 컨텍스트(WebContext) -> 필터 -> 스프링 컨텍스트 -> 디스패처 서블릿

2. 요청 정보를 통해 요청을 위임할 컨트롤러를 찾음

- 디스패처 서블릿은 요청 정보를 바탕으로,요청을 처리할 컨트롤러를 찾고 해당 메소드를 호출해야 함
- @Controller. @RequestMapping 관련 어노테이션을 조합해 컨트롤러를 생성
- 처리할 컨트롤러는 주로 HandlerMapping의 구현체 중 하나인, RquestMappingHandlerMapping이 찾아줌.

    - HashMap으로 (요청정보(Key), 처리대상(Value))을 관리
    - 요청을 처리할 대상은 먼트롤러를 가지고 있는 HandlerMethod 객체
- HandlerMethod 필요한 이유
    - 컨트롤러, 컨트롤러의 메소드 등을 포함한 부가정보가 필요하기 때문
- 찾아온 핸들러메소드는 HandlerMethodExecutionChain으로 감싸서 반환
    - 컨트롤러로 요청을 넘겨주기전, 처리해야하는 인터셉터 등을 포함하기 위해

3. 요청을 컨트롤러로 위임할 핸들러 어댑터를 찾아서 전달

- 디스패처 서블릿은 컨트롤러로 요청을 직접 위임하는것이 아니라 핸들러 어댑터를 통해 컨트롤러로 요청을 위임
    - 어댑터 인터페이스를 통해 컨트롤러를 호출하는 이유는 컨트롤러 구현 방식이 다양하기 때문
    - 컨트롤러 구현 방식에 상관없이 요청을 위임 가능

4. 핸들러 어댑터가 컨트롤러로 요청을 위임

- 핸들러어댑터가 컨트롤러로 요청을 넘기기 전, 공통적인 전처리 과정이 필요
    - 요청에 매칭되는 인터셉터들 실행
    - @RequestParam, @Requestbody 등으로 파라미터를 준비하는 ArgumentResolver도 실행하는 등, 다양한 공통작업들 수행
- 전처리 작업이 완료되면 파라미터 값들과 함께 컨트롤러로 요청을 위임

5. 비즈니스 로직을 처리

- 컨트롤러는 서비스를 호출하고 우리가 작성한 비즈니스 로직들이 진행

6. 컨트롤러가 반환값을 반환

- 비즈니스 로직 처리후 컨트롤러가 반환 값을 반환
    - ResponseEntity : 응답 데이터를 사용하는 경우
    - View 이름 : 응답 페이지를 보여주는 경우

7. 핸들러 어댑터가 반환값을 처리

- 핸들러 어댑터는 컨트롤러가 받은 반환값을 응답 처리기인 ReturnValueHandler가 후처리한 후, 디스패처 서블릿으로 돌려줌.
- 반환값
    - ResponseEntity : HttpEntityMethodProcessor가 MessageConverter를 사용해 응답 객체를 직렬화 -> 응답 상태를 설정
    - View 이름 : 뷰를 반환하기 위한 준비 작업을 처리

8. 서버의 응답을 클라이언트로 반환함

- 디스패처 서블릿을 통해 반환되는 응답은 다시 필터들을 거쳐 클라이언트에게 반환됨.
- 응답이 데이터면 그로 반환
- 응답이 화면이면 View 이름에 맞는 View를 찾아서 반환해주는 ViewResolver가 적절한 화면을 내려줌
