VanillaWAS

순수 Java 기반 경량 웹 애플리케이션 서버

제작: 조영진

개발 동기 및 목적

기술의 발전 덕분에 많은 개발자들이 다양한 라이브러리와 프레임워크를 활용해 애플리케이션을 편리하게 개발하고 있다

특히, 우리나라에서 Back-End 개발 분야에서 가장 널리 사용되는 Spring Framework는 매우 높은 수준의 추상화와 은닉화를 통해 구현되어 있어, 사용자는 내부 동작을 몰라도 몇 줄의 코드만으로도 안정적으로 웹 애플리케이션을 구축할 수 있다.

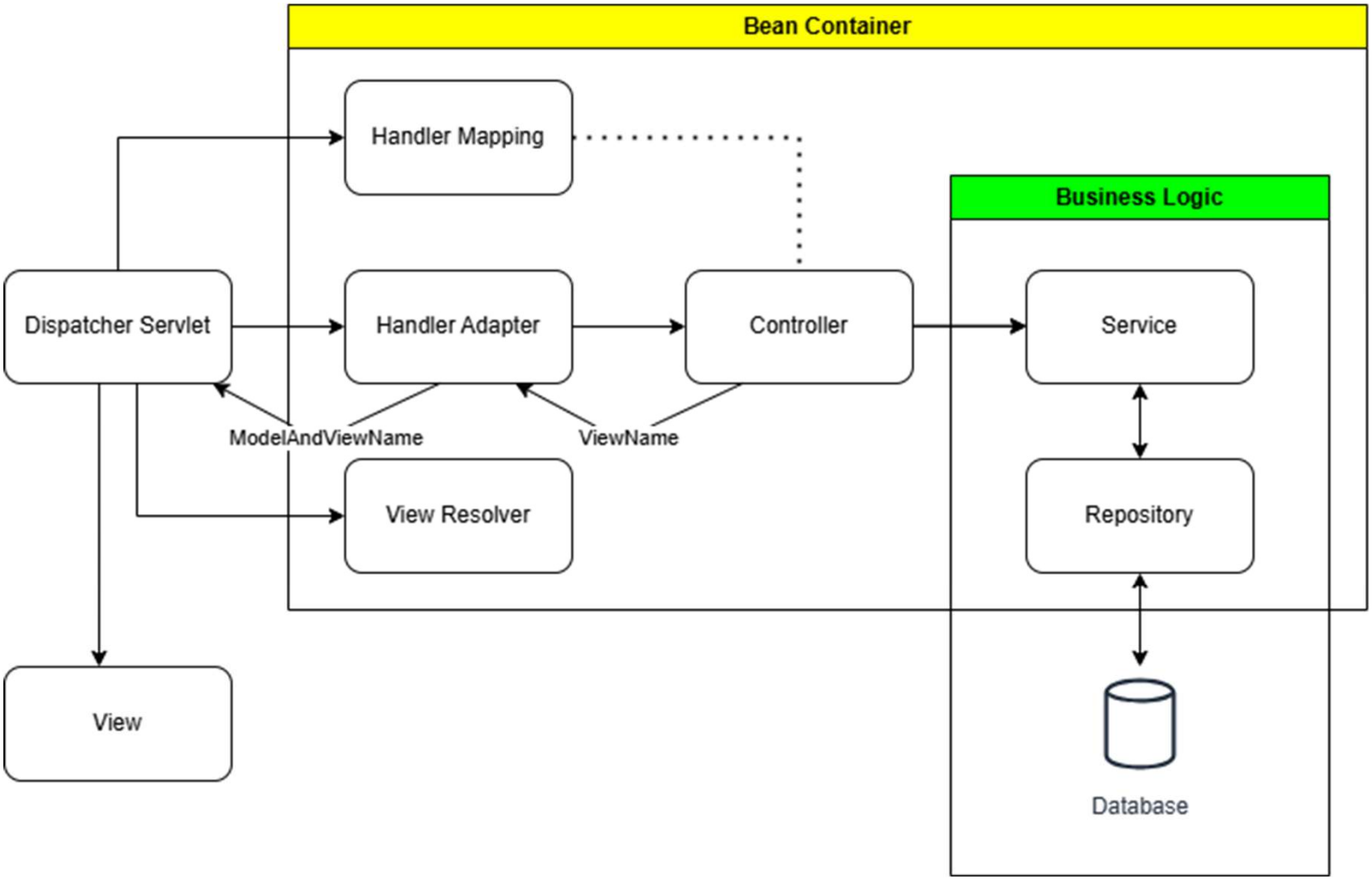
그러나 기반 기술에 대한 이해 없이 프레임워크를 단순히 사용하는 데 그치는 경우가 많아, 응용이나 확장에는 어려움을 겪는 경우가 적지 않다.

이에 본 프로젝트는 Tomcat과 같은 WAS와 Spring과 같은 프레임워크를 순수 Java로 바닥부터 직접 구현함으로써, 그 내부 동작 원리를 깊이 있게 이해하고 탐구하는 데 목적을 두고 있다.

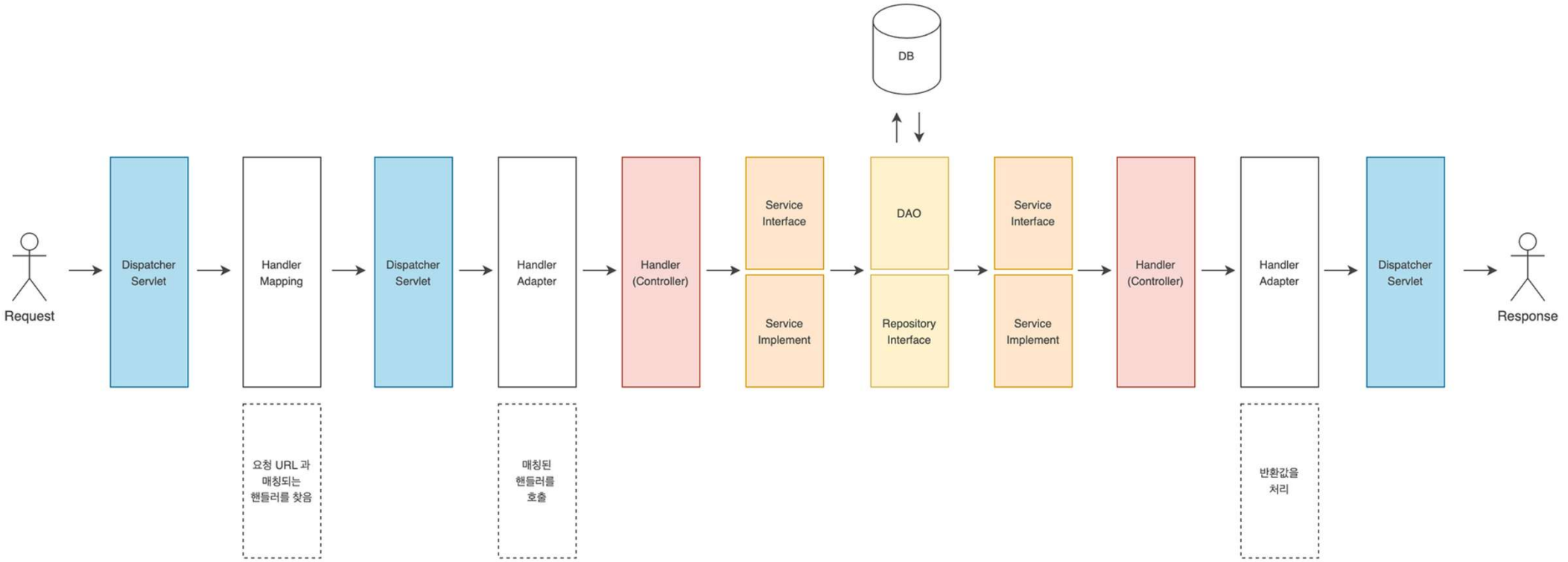
주요 기능 요약

- Multi Threading을 통한 사용자 단위 Http Request 처리
- Bean Container를 통한 IoC 및 DI 구현
- Dispatcher Servlet 기반 Front Controller 패턴 구현
- Java Reflection 활용한 Custom Annotation
- Static Resource와 Dynamic Resource 분리
- Regular Expression을 이용한 Template Engine 구현

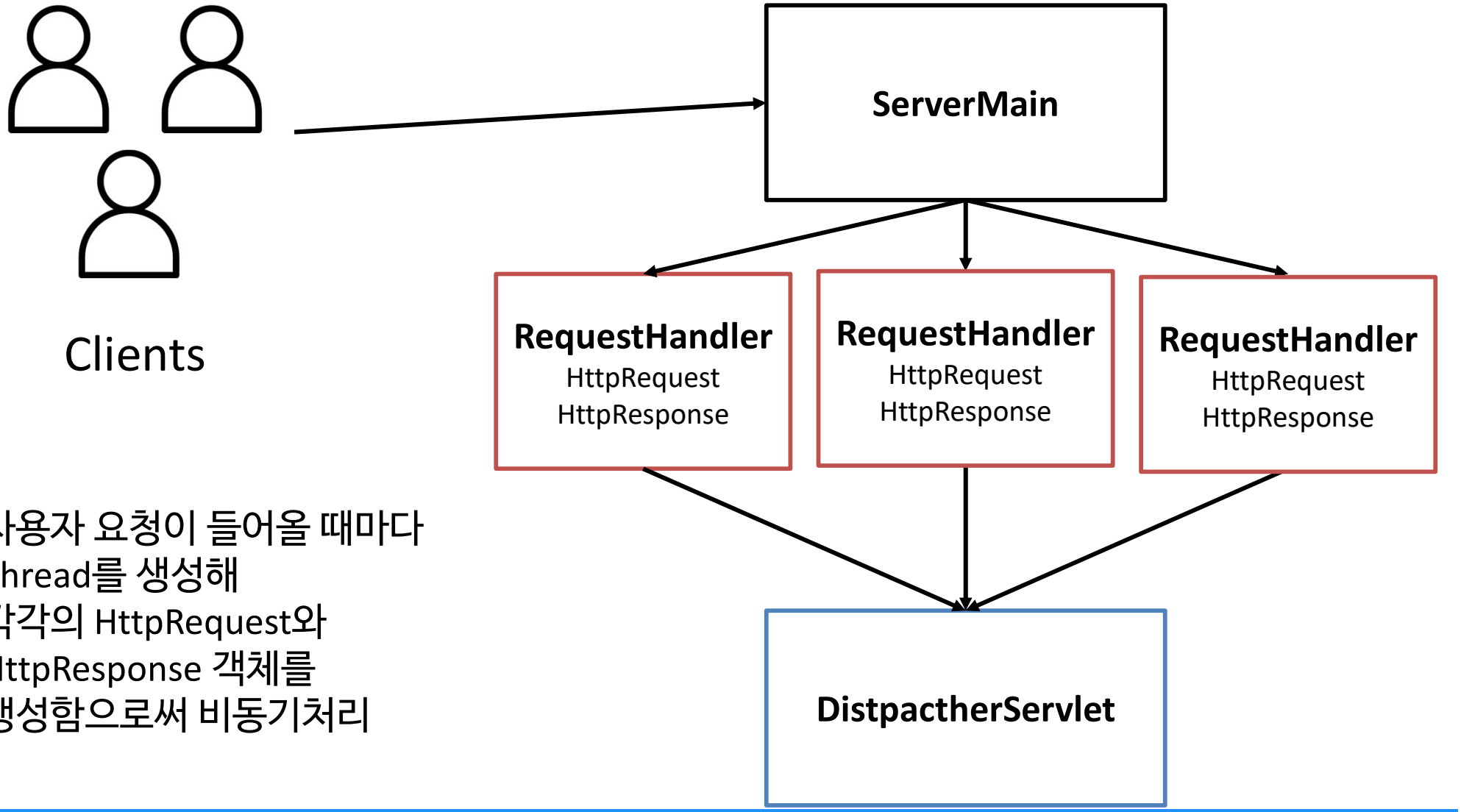
System Architecture



Request Flow



동시 요청 처리



사용자 요청이 들어올 때마다
Thread를 생성해
각각의 HttpRequest와
HttpResponse 객체를
생성함으로써 비동기처리

DI 예시: @Bean과 Autowiring

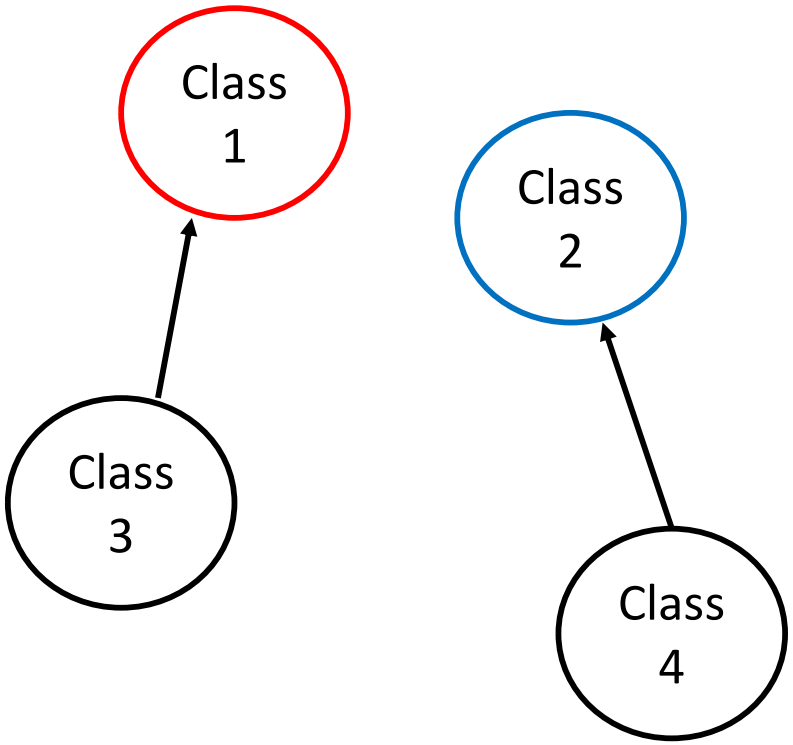
```java
@Configuration
public class Config {

    @Bean
    public BoardController boardController(BoardService boardService) {
        return new BoardController(boardService);
    }

    @Bean
    public BoardService boardService(BoardRepository boardRepository) {
        return new BoardServiceImpl(boardRepository);
    }

    @Bean
    public BoardRepository boardRepository() {
        return new MemoryBoardRepository();
    }

}
```

```java
public class BoardController {

    private final BoardService boardService;

    public BoardController(BoardService boardService) {
        this.boardService = boardService;

    }
```

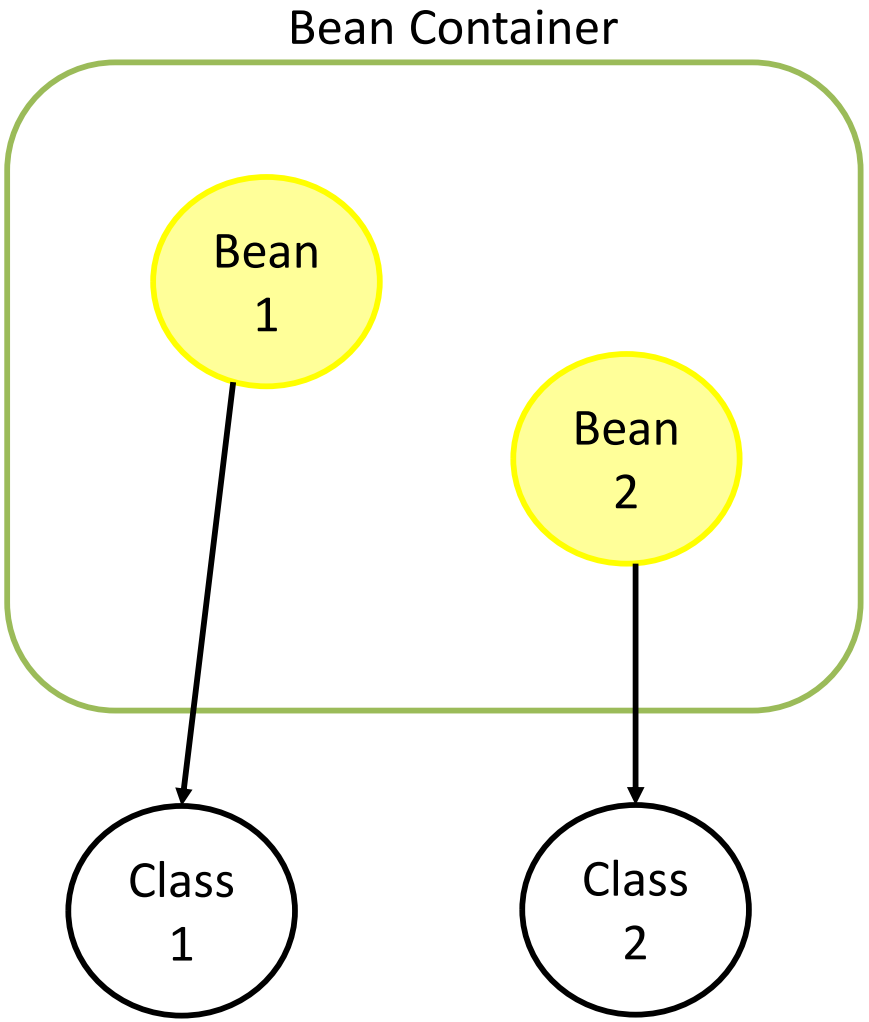
```java
public class BoardServiceImpl implements BoardService {

    private final BoardRepository boardRepository;

    public BoardServiceImpl(BoardRepository boardRepository) {
        this.boardRepository = boardRepository;
    }
}
```

클래스를 Bean으로 등록하면 Singleton과 같이 Instance를 단 한번만 생성하는 것을 보장한다.
또한 Client가 작성하는 클래스에서 의존성 주입을 통해 구체 클래스에 의존하지 않게 해주며 유연하고 결합도가 낮은 코드를 작성할 수 있게 해준다.

Bean Container



코드가 특정 클래스에 의존적임



의존성이 컨테이너로부터 자동 주입됨

Bean Container 등록 과정

```
private void init(Config config) {

    Class<? extends Config> aClass = config.getClass();
    assert aClass.isAnnotationPresent(Configuration.class);

    //구성 파일에서 Bean을 반환하는 메소드들을 가져옴
   Method[] declaredMethods = aClass.getDeclaredMethods();

    // 원하는 Bean을 반환하는 메소드를 먼저 실행하기 위해 매핑해놓음
   for (Method method : declaredMethods) {
        if (method.isAnnotationPresent(Bean.class)) {
            methodMap.put(method.getReturnType(), method);
        }
    }

    // 각 메소드들을 실행해서 Bean으로 등록 시작
   for (Method method : declaredMethods) {
        if (method.isAnnotationPresent(Bean.class)) {
            registerBean(method);
        }
    }
}
```

구성 파일에 정의된 Method를 통해 특정 Instance를 Application Runtime 동안 Bean으로 등록 및 관리

```
private void registerBean(Method method) {

    Object bean;

    // Bean이 이미 등록되어 있으면 무시
   if(beans.containsKey(method.getReturnType())) {
        return;
    }

    // 파라미터가 없으면 바로 실행 후 Bean으로 등록
   if(method.getParameterCount() <= 0) {
        bean = method.invoke(config);
    } else {
        // 파라미터가 있을 경우 그 Bean을 컨테이너에서 찾아보고 없으면 먼저 등록 처리(재귀함수)
        Class<?>[] parameterTypes = method.getParameterTypes();
        Object[] args = new Object[parameterTypes.length];

        for(int i = 0; i < parameterTypes.length; i++) {
            if(!beans.containsKey(parameterTypes[i])) {
                registerBean(methodMap.get(parameterTypes[i]));
            }
            args[i] = beans.get(parameterTypes[i]);
        }

        bean = method.invoke(config, args);
    }

    Class<?>[] beansInterfaces = bean.getClass().getInterfaces();

    if(beansInterfaces.length == 0) { // 구현체가 아닐 경우
      beans.put(bean.getClass(), bean);
    } else { // 구현체일 경우
      beans.put(beansInterfaces[0], bean);
    }

    log(bean + ": Bean 등록완료");
}
```

Custom Annotation: @Mapping과 @Param

```java
@Mapping(value = "/board", method = HttpMethod.POST)
public String writeBoard(@Param("writer") String writer,
                         @Param("content") String content) {
    Board board = new Board();
    board.setWriter(writer);
    board.setContent(content);
    boardService.createBoard(board);
    return "redirect:/board";
}
```

```java
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface Mapping {
    String value();
    HttpMethod method() default HttpMethod.GET;
}
```

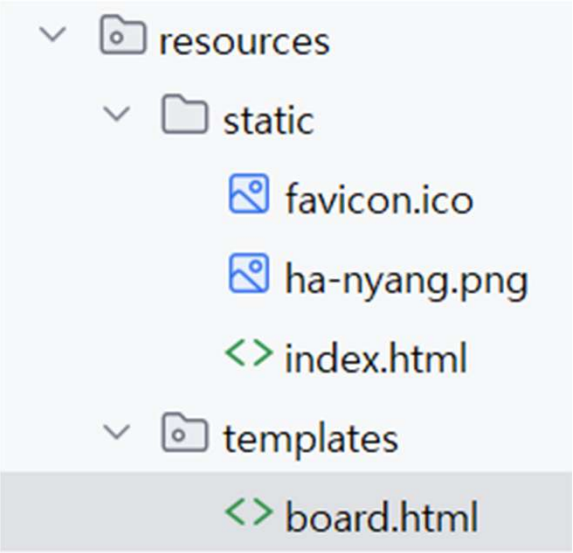
```java
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.PARAMETER)
public @interface Param {
    String value();
}
```

Custom Annotation을 통해 Controller의 Parameter를 유연하게 받을 수 있음

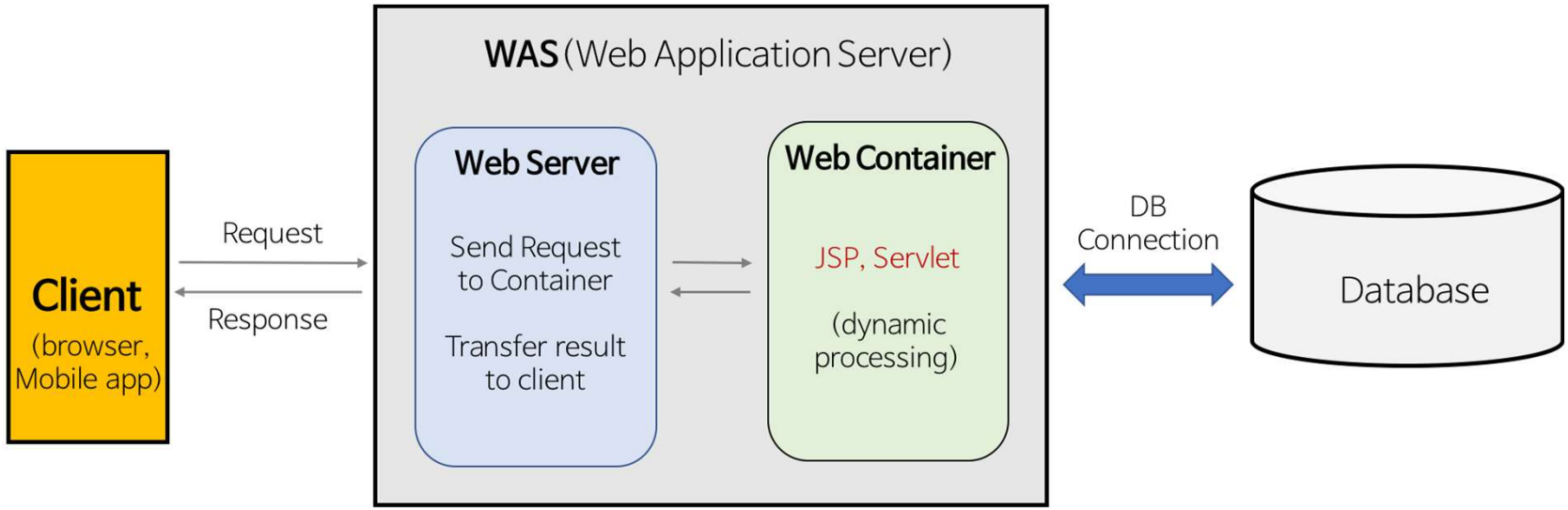
```java
public ModelAndView handle(HttpRequest request,
                           HttpResponse response,
                           ControllerAndMethod controllerAndMethod) throws IOException {

    ModelAndView mav = new ModelAndView();
    Model model = new Model();
    String viewName;

    // 컨트롤러에 넘겨줄 메소드 설정
    Object controller = controllerAndMethod.getController();
    Method method = controllerAndMethod.getMethod();

    Parameter[] parameters = method.getParameters();
    Object[] arguments = new Object[parameters.length];

    // 메소드에 넘겨줄 매개변수 설정
    for (int i = 0; i < parameters.length; i++) {
        if(parameters[i].getType() == HttpRequest.class) {
            arguments[i] = request;
        } else if(parameters[i].getType() == HttpResponse.class) {
            arguments[i] = response;
        } else if(parameters[i].getType() == Model.class) {
            arguments[i] = model;
        } else if(parameters[i].isAnnotationPresent(Param.class)) {
            String paramKey = parameters[i].getAnnotation(Param.class).value();
            arguments[i] = request.getQueryParameters().get(paramKey);
        } else {
            log(method + "컨트롤러 매개 변수 오류: " + parameters[i]);
            throw new IOException();
        }
    }
    ...
```

HandlerAdapter의 컨트롤러 메소드에 대한 가변인자 설정 로직

Static Resource와 Dynamic Resource 분리



Static Resource와 Dynamic Resource를 분리해 Web Server와 Web Apllication Server의 역할을 분리



Regular Expression을 통한 동적 HTML 생성

```
<h1>상품 목록</h1>

{{#if showItems}}
<ul>
  {{#each items}}
    <li>{{name}} - {{price}}원</li>
  {{/each}}
</ul>
{{/if}}

<p>총 상품 수: {{totalCount}}</p>
```

```
private static final Pattern TOKEN_PATTERN
= Pattern.compile(

"\\{\\{(?<type>#each|#if|/each|/if)?\\s*(?<
key>\\w+)?\\s*\\}\\}");
```

정규식 패턴을 정의하고 패턴에 일치하는 HTML 파일 내부의 문자열을 AST로 만들어 동적으로 렌더링한다.

Raspberry Pi 환경의
On-premise 서버 구축

Raspberry Pi 위에 Linux Server 환경 구축 및
포트 포워딩을 통해 공유기의 특정 포트로
들어오는 TCP 메시지를 Raspberry Pi로 전송

```
pi@raspberrypi: ~
tcp6       0      0 :::12345                :::*                    LISTEN
616/java
tcp6       0      0 :::22                   :::*                    LISTEN
-
pi@raspberrypi:~ $ htop
pi@raspberrypi:~ $ tail -f nohup.out
20:43:24.702 [pool-1-thread-3] 192.168.25.2 <-- HTTP 응답 : HttpResponse{httpStat
us=200 OK, headers={Content-Type=image/png}}
20:43:25.601 [pool-1-thread-5] 192.168.25.2 --> HTTP 요청 : HttpRequest{method=GE
T, path='/board', queryParameters={}}
20:43:25.608 [pool-1-thread-5] 192.168.25.2 <-- HTTP 응답 : HttpResponse{httpStat
us=200 OK, headers={Content-Type=text/html; charset=UTF-8}}
20:45:36.005 [pool-1-thread-1] java.io.IOException: EOF: No start line received
20:45:36.161 [pool-1-thread-4] 192.168.25.2 --> HTTP 요청 : HttpRequest{method=GE
T, path='/', queryParameters={}}
20:45:36.163 [pool-1-thread-4] 파일 전송 완료 : index.html
20:45:36.164 [pool-1-thread-4] 192.168.25.2 <-- HTTP 응답 : HttpResponse{httpStat
us=200 OK, headers={Content-Type=text/html; charset=UTF-8}}
20:45:36.186 [pool-1-thread-2] 192.168.25.2 --> HTTP 요청 : HttpRequest{method=GE
T, path='/ha-nyang.png', queryParameters={}}
20:45:36.198 [pool-1-thread-2] 파일 전송 완료 : ha-nyang.png
20:45:36.200 [pool-1-thread-2] 192.168.25.2 <-- HTTP 응답 : HttpResponse{httpStat
us=200 OK, headers={Content-Type=image/png}}
```

개발 중 문제 및 해결

- PORT 개방 후 서버로 지속적인 SSH Brute Force 공격 –> RSA 인증 및 fail2ban 설치로 해결
- Bean Container에서 Bean 누락 문제 -> Java Reflection으로 정의된 Method를 가져올 때 Method의 순서를 보장하지 않아서 생긴 문제, 순서에 상관 없이 등록 시도 중인 Bean이 의존하는 Bean이 있다면 먼저 등록 하도록 등록 Method를 재귀적으로 구현해서 해결
- Bean Container에서 Bean 중복 등록 문제 -> Config 클래스의 Method를 호출해 받은 Object 객체의 getClass()를 호출하면 구체 클래스의 메타 정보를 반환하기 때문에 Interface Type이 아닌 구체 클래스 Bean으로 등록됨. 이에 다른 Bean에서 등록 중 Interface Type Bean을 요구할 때마다 해당 Bean이 등록되지 않은 것으로 간주하고 구체 클래스를 계속 등록하는 문제 발생. getInterfaces()를 호출해 구체 클래스인지 검사해 맞을 경우 구현한 Interface 타입으로 등록 처리해서 해결.
- jar 파일로 서버 배포 시 정적 리소스를 찾을 수 없는 문제 -> Java의 File 클래스는 os의 파일 시스템에 의존하기 때문에 파일이 압축된 jar 배포 환경에서는 정상적으로 파일을 불러오지 못함. jar 내부적으로 파일에 접근 가능한 ClassLoader.getResourceAsStream()을 이용하여 해결

성과 및 시사점

- Web Browser 및 Web Server 간 통신 및 작동 원리 파악
- Apache, Nginx 등의 정적 Web Sever 및 Tomact, Jboss 등의 WAS 구조에 대한 이해
- Front Controller, DI, Annotation 처리 등 Spring Framework 핵심 메커니즘 체득
- Framework 설계 및 Java 기반 서버 개발 역량 향상

Thank you!