14. HTTP 서버 활용

#1.인강/0.자바/6.자바-고급2편

HTTP 서버7 - 애노테이션 서블릿1 - 시작

지금까지 학습한 애노테이션 내용을 바탕으로 애노테이션 기반의 컨트롤러와 서블릿을 만들어보자.

예를 들어 다음과 같은 컨트롤러를 만들 예정이다.

```
public class AnnotationController {

    @Mapping("/")
    public void home(HttpRequest request, HttpResponse response) {
    }

    @Mapping("/site1")
    public void site1(HttpRequest request, HttpResponse response) {
    }

    @Mapping("/site2")
    public void site2(HttpRequest request, HttpResponse response) {
    }

}
```

애노테이션부터 만들어보자.

```
package was.httpserver.servlet.annotation;

import java.lang.annotation.*;
```

```
@Retention( RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Documented
public @interface Mapping {
    String value();
}
```

```
package was.httpserver.servlet.annotation;

import was.httpserver.HttpRequest;
import was.httpserver.HttpResponse;
import was.httpserver.HttpServlet;
import was.httpserver.PageNotFoundException;

import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.List;

public class AnnotationServletV1 implements HttpServlet {

    private final List<Object> controllers;

    public AnnotationServletV1(List<Object> controllers) {
        this.controllers = controllers;
    }

    @Override
    public void service(HttpRequest request, HttpResponse response) throws
IOException {
        String path = request.getPath();

        for (Object controller : controllers) {
            Method[] methods = controller.getClass().getDeclaredMethods();
            for (Method method : methods) {
                if (method.isAnnotationPresent(Mapping.class)) {
                    Mapping mapping = method.getAnnotation(Mapping.class);
                    String value = mapping.value();
                    if (value.equals(path)) {
                        invoke(controller, method, request, response);
                        return;
```

```
                    }
                }
            }
        }
        throw new PageNotFoundException("request=" + path);
    }

    private void invoke(Object controller, Method method, HttpRequest request,
HttpResponse response) {
        try {
            method.invoke(controller, request, response);
        } catch (InvocationTargetException | IllegalAccessException e) {
            throw new RuntimeException(e);
        }
    }

}
```

- 리플렉션에서 사용한 코드와 비슷하다. 차이가 있다면 호출할 메서드를 찾을 때, 메서드의 이름을 비교하는 대신에, 메서드에서 `@Mapping` 애노테이션을 찾고, 그곳의 `value` 값으로 비교한다는 점이다.
- 패키지 위치에 주의하자. 다른 프로젝트에서도 사용할 수 있다.

```
package was.v7;

import was.httpserver.HttpRequest;
import was.httpserver.HttpResponse;
import was.httpserver.servlet.annotation.Mapping;

public class SiteControllerV7 {

    @Mapping("/")
    public void home(HttpRequest request, HttpResponse response) {
        response.writeBody("<h1>home</h1>");
        response.writeBody("<ul>");
        response.writeBody("<li><a href='/site1'>site1</a></li>");
        response.writeBody("<li><a href='/site2'>site2</a></li>");
        response.writeBody("<li><a href='/search?q=hello'>검색</a></li>");
        response.writeBody("</ul>");
    }
```

```
    @Mapping("/site1")
    public void site1(HttpRequest request, HttpResponse response) {
        response.writeBody("<h1>site1</h1>");
    }

    @Mapping("/site2")
    public void site2(HttpRequest request, HttpResponse response) {
        response.writeBody("<h1>site2</h1>");
    }

}
```

- `@Mapping("/")`, `home()` : 애노테이션을 사용한 덕분에 `/` URL 요청도 처리할 수 있게 되었다.
- 물론 메서드 이름도 원하는 이름으로 변경해도 된다.

```
package was.v7;

import was.httpserver.HttpRequest;
import was.httpserver.HttpResponse;
import was.httpserver.servlet.annotation.Mapping;

public class SearchControllerV7 {

    @Mapping("/search")
    public void search(HttpRequest request, HttpResponse response) {
        String query = request.getParameter("q");

        response.writeBody("<h1>Search</h1>");
        response.writeBody("<ul>");
        response.writeBody("<li>query: " + query + "</li>");
        response.writeBody("</ul>");
    }
}
```

```
package was.v7;

import was.httpserver.HttpServer;
import was.httpserver.HttpServlet;
import was.httpserver.ServletManager;
import was.httpserver.servlet.DiscardServlet;
```

```
import was.httpserver.servlet.annotation.AnnotationServletV1;

import java.io.IOException;
import java.util.List;

public class ServerMainV7 {

    private static final int PORT = 12345;

    public static void main(String[] args) throws IOException {
        List<Object> controllers =
                            List.of(new SiteControllerV7(), new
SearchControllerV7());
        HttpServlet servlet = new AnnotationServletV1(controllers);

        ServletManager servletManager = new ServletManager();
        servletManager.setDefaultServlet(servlet);
        servletManager.add("/favicon.ico", new DiscardServlet());
        HttpServer server = new HttpServer(PORT, servletManager);
        server.start();
    }
}
```

- `/favicon.ico`의 경우 컨트롤러를 통해서 해결해도 되지만, 이미 `DiscardServlet`이라는 공통 서블릿 기능이 있으므로 해당 기능을 그대로 사용했다.

실행 결과

- 기존과 같다.

정리

애노테이션을 사용한 덕분에 매우 편리하고, 또 실용적으로 웹 애플리케이션을 만들 수 있게 되었다.

현대의 웹 프레임워크들은 대부분 애노테이션을 사용해서 편리하게 호출 메서드를 찾을 수 있는 지금과 같은 방식을 제공한다.

자바 백엔드의 사실상 표준 기술인 스프링 프레임워크도 스프링 MVC를 통해 이런 방식의 기능을 제공한다.

HTTP 서버8 - 애노테이션 서블릿2 - 동적 바인딩

우리가 만든 애노테이션 기반 컨트롤러에서 아쉬운 부분이 있다.
예를 들어서 다음 `site1()`, `site2()`의 경우 `HttpRequest request`가 전혀 필요하지 않다.
`HttpResponse response`만 있으면 된다.

```
@Mapping("/site1")
public void site1(HttpRequest request, HttpResponse response) {
    response.writeBody("<h1>site1</h1>");
}

@Mapping("/site2")
public void site2(HttpRequest request, HttpResponse response) {
    response.writeBody("<h1>site2</h1>");
}
```

컨트롤러의 메서드를 만들 때, `HttpRequest request, HttpResponse response` 중에 딱 필요한 메서드만 유연하게 받을 수 있도록 `AnnotationServletV1`의 기능을 개선해보자.

```
package was.httpserver.servlet.annotation;

import was.httpserver.HttpRequest;
import was.httpserver.HttpResponse;
import was.httpserver.HttpServlet;
import was.httpserver.PageNotFoundException;

import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.List;

public class AnnotationServletV2 implements HttpServlet {

    private final List<Object> controllers;

    public AnnotationServletV2(List<Object> controllers) {
        this.controllers = controllers;
    }

    @Override
    public void service(HttpRequest request, HttpResponse response) throws
```

```
IOException {
        String path = request.getPath();

        for (Object controller : controllers) {
            Method[] methods = controller.getClass().getDeclaredMethods();
            for (Method method : methods) {
                if (method.isAnnotationPresent(Mapping.class)) {
                    Mapping mapping = method.getAnnotation(Mapping.class);
                    String value = mapping.value();
                    if (value.equals(path)) {
                        invoke(request, response, controller, method);
                        return;
                    }
                }
            }
        }
        throw new PageNotFoundException("request=" + path);
    }

    private void invoke(HttpRequest request, HttpResponse response, Object
controller, Method method) {
        Class<?>[] parameterTypes = method.getParameterTypes();
        Object[] args = new Object[parameterTypes.length];

        for (int i = 0; i < parameterTypes.length; i++) {
            if (parameterTypes[i] == HttpRequest.class) {
                args[i] = request;
            } else if (parameterTypes[i] == HttpResponse.class) {
                args[i] = response;
            } else {
                throw new IllegalArgumentException("Unsupported parameter
type: " + parameterTypes[i]);
            }
        }

        try {
            method.invoke(controller, args);
        } catch (IllegalAccessException | InvocationTargetException e) {
            throw new RuntimeException(e);
        }
    }

}
```

- `invoke()` 부분을 보자
- 메서드의 파라미터 타입을 확인한 후에 각 타입에 맞는 값을 `args[]`에 담아서 메서드를 호출한다.

```
package was.v8;

import was.httpserver.HttpResponse;
import was.httpserver.servlet.annotation.Mapping;

public class SiteControllerV8 {

    @Mapping("/")
    public void home(HttpResponse response) {
        response.writeBody("<h1>home</h1>");
        response.writeBody("<ul>");
        response.writeBody("<li><a href='/site1'>site1</a></li>");
        response.writeBody("<li><a href='/site2'>site2</a></li>");
        response.writeBody("<li><a href='/search?q=hello'>검색</a></li>");
        response.writeBody("</ul>");
    }

    @Mapping("/site1")
    public void site1(HttpResponse response) {
        response.writeBody("<h1>site1</h1>");
    }

    @Mapping("/site2")
    public void site2(HttpResponse response) {
        response.writeBody("<h1>site2</h1>");
    }

}
```

- 자신에게 필요한 값만 매개변수로 선언하면 된다.

```
package was.v8;

import was.httpserver.HttpRequest;
import was.httpserver.HttpResponse;
import was.httpserver.servlet.annotation.Mapping;
```

```
public class SearchControllerV8 {

    @Mapping("/search")
    public void search(HttpRequest request, HttpResponse response) {
        String query = request.getParameter("q");

        response.writeBody("<h1>Search</h1>");
        response.writeBody("<ul>");
        response.writeBody("<li>query: " + query + "</li>");
        response.writeBody("</ul>");
    }
}
```

ServerMainV8 - AnnotationServletV2

```
package was.v8;

import was.httpserver.HttpServer;
import was.httpserver.HttpServlet;
import was.httpserver.ServletManager;
import was.httpserver.servlet.DiscardServlet;
import was.httpserver.servlet.annotation.AnnotationServletV2;

import java.io.IOException;
import java.util.List;

public class ServerMainV8 {

    private static final int PORT = 12345;

    public static void main(String[] args) throws IOException {
        List<Object> controllers =
                            List.of(new SiteControllerV8(), new
SearchControllerV8());
        HttpServlet servlet = new AnnotationServletV2(controllers);

        ServletManager servletManager = new ServletManager();
        servletManager.setDefaultServlet(servlet);
        servletManager.add("/favicon.ico", new DiscardServlet());
        HttpServer server = new HttpServer(PORT, servletManager);
        server.start();
```

```
    }
}
```

실행 결과

- 기존과 같다.

정리

`AnnotationServletV2`에서 호출할 컨트롤러 메서드의 매개변수를 먼저 확인한 다음에 매개변수에 필요한 값을 동적으로 만들어서 전달했다. 덕분에 컨트롤러의 메서드는 자신에게 필요한 값만 선언하고, 전달 받을 수 있다. 이런 기능을 확장하면 `HttpRequest`, `HttpResponse` 뿐만 아니라 다양한 객체들도 전달할 수 있다.
참고로 스프링 MVC도 이런 방식으로 다양한 매개변수의 값을 동적으로 전달한다.

HTTP 서버9 - 애노테이션 서블릿3 - 성능 최적화

지금까지 만든 `AnnotationServletV2`는 2가지 아쉬운 점이 있다.

- 성능 최적화
- 중복 매핑 문제

문제 1. 성능 최적화

AnnotationServletV2 코드 일부분

```
@Override
public void service(HttpRequest request, HttpResponse response) throws
IOException {
    String path = request.getPath();

    for (Object controller : controllers) {
        Method[] methods = controller.getClass().getDeclaredMethods();
        for (Method method : methods) {
            if (method.isAnnotationPresent(Mapping.class)) {
                Mapping mapping = method.getAnnotation(Mapping.class);
                String value = mapping.value();
                if (value.equals(path)) {
                    invoke(request, response, controller, method);
                    return;
                }
```

```
                }
            }
        }
        throw new PageNotFoundException("request=" + path);
    }
```

- 모든 컨트롤러의 메서드를 하나하나 순서대로 찾는다. 이것은 결과적으로 O(n)의 성능을 보인다.
- 만약 모든 컨트롤러의 메서드가 100개라면 최악의 경우 100번은 찾아야한다.
- 더 큰 문제는 고객의 요청 때 마다 이 로직이 호출된다는 점이다. 동시에 100명의 고객이 요청하면 100 * 100번 해당 로직이 호출된다.
- 이 부분의 성능을 O(n) ➜ O(1)로 변경하려면 어떻게 해야할까?

문제 2. 중복 매핑 문제

```
@Mapping("/site2")
public void site2(HttpResponse response) {
    response.writeBody("<h1>site2</h1>");
}

@Mapping("/site2")
public void page2(HttpResponse response) {
    response.writeBody("<h1>site2</h1>");
}
```

개발자가 실수로 `@Mapping`에 같은 `/site2`를 2개 정의하면 어떻게 될까?

- `/site2` ➜ `site2()`
- `/site2` ➜ `page2()`

이 경우 현재 로직에서는 먼저 찾은 메서드가 호출된다.

개발에서 가장 나쁜 것은 모호한 것이다! 모호한 문제는 반드시 제거해야 한다! 그렇지 않으면 나중에 큰 재앙(장애)으로 다가온다.

최적화 구현

```
package was.httpserver.servlet.annotation;

import was.httpserver.HttpRequest;
import was.httpserver.HttpResponse;
```

```
import was.httpserver.HttpServlet;
import was.httpserver.PageNotFoundException;

import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class AnnotationServletV3 implements HttpServlet {

    private final Map<String, ControllerMethod> pathMap;

    public AnnotationServletV3(List<Object> controllers) {
        this.pathMap = new HashMap<>();
        initializePathMap(controllers);
    }

    private void initializePathMap(List<Object> controllers) {
        for (Object controller : controllers) {
            for (Method method : controller.getClass().getDeclaredMethods()) {
                if (method.isAnnotationPresent(Mapping.class)) {
                    String path = method.getAnnotation(Mapping.class).value();
                    // 중복 경로 체크
                    if (pathMap.containsKey(path)) {
                        ControllerMethod controllerMethod = pathMap.get(path);
                        throw new IllegalArgumentException("경로 중복 등록, path="
+ path + ", method=" + method + ", 이미 등록된 메서드=" + controllerMethod.method);
                    }

                    pathMap.put(path, new ControllerMethod(controller,
method));
                }
            }
        }
    }

    @Override
    public void service(HttpRequest request, HttpResponse response) throws
IOException {
        String path = request.getPath();
        ControllerMethod controllerMethod = pathMap.get(path);
```

```
        if (controllerMethod == null) {
            throw new PageNotFoundException("request=" + path);
        }

        controllerMethod.invoke(request, response);
    }

    private static class ControllerMethod {
        private final Object controller;
        private final Method method;

        public ControllerMethod(Object controller, Method method) {
            this.controller = controller;
            this.method = method;
        }

        public void invoke(HttpRequest request, HttpResponse response) {
            Class<?>[] parameterTypes = method.getParameterTypes();
            Object[] args = new Object[parameterTypes.length];

            for (int i = 0; i < parameterTypes.length; i++) {
                if (parameterTypes[i] == HttpRequest.class) {
                    args[i] = request;
                } else if (parameterTypes[i] == HttpResponse.class) {
                    args[i] = response;
                } else {
                    throw new IllegalArgumentException("Unsupported parameter
type: " + parameterTypes[i]);
                }
            }

            try {
                method.invoke(controller, args);
            } catch (IllegalAccessException | InvocationTargetException e) {
                throw new RuntimeException(e);
            }
        }

    }
}
```

초기화

- `AnnotationServletV3`을 생성하는 시점에 `@Mapping`을 사용하는 컨트롤러의 메서드를 모두 찾아서 `pathMap`에 보관한다.
  - `key=경로, value=ControllerMethod` 구조이다.
- 초기화가 끝나면 `pathMap`이 완성된다.
- `ControllerMethod`: `@Mapping`의 대상 메서드와 메서드가 있는 컨트롤러 객체를 캡슐화했다. 이렇게 하면 `ControllerMethod` 객체를 사용해서 편리하게 실제 메서드를 호출할 수 있다.

실행

- `ControllerMethod controllerMethod = pathMap.get(path)`를 사용해서 URL 경로에 매핑된 컨트롤러의 메서드를 찾아온다. 이 과정은 `HashMap`을 사용하므로 일반적으로 O(1)의 매우 빠른 성능을 제공한다.

중복 경로 체크

```
// 중복 경로 체크
if (pathMap.containsKey(path)) {
    ControllerMethod controllerMethod = pathMap.get(path);
    throw new IllegalArgumentException("경로 중복 등록, path=" + path + ",
method=" + method + ", 이미 등록된 메서드=" + controllerMethod.method);
}
```

- `pathMap`에 이미 등록된 경로가 있다면 중복 경로이다. 이 경우 예외를 던진다.

ServerMainV8 - AnnotationServletV3

```
package was.v8;

import was.httpserver.HttpServer;
import was.httpserver.HttpServlet;
import was.httpserver.ServletManager;
import was.httpserver.servlet.DiscardServlet;
import was.httpserver.servlet.annotation.AnnotationServletV2;
import was.httpserver.servlet.annotation.AnnotationServletV3;

import java.io.IOException;
import java.util.List;

public class ServerMainV8 {
```

```
    private static final int PORT = 12345;

    public static void main(String[] args) throws IOException {
        List<Object> controllers = List.of(new SiteControllerV8(), new
SearchControllerV8());
        //HttpServlet servlet = new AnnotationServletV2(controllers);
        HttpServlet servlet = new AnnotationServletV3(controllers);

        ServletManager servletManager = new ServletManager();
        servletManager.setDefaultServlet(servlet);
        servletManager.add("/favicon.ico", new DiscardServlet());
        HttpServer server = new HttpServer(PORT, servletManager);
        server.start();
    }
}
```

- **ServerMainV8을 그대로 사용하고, 일부 수정한다.**
- `AnnotationServletV2` ➜ `AnnotationServletV3`을 사용하도록 변경했다.
- 다른 코드는 변경할 부분이 없으므로 이 부분만 수정하자.

실행 결과

- 기존 결과와 같다.

중복 체크 확인

`SiteControllerV8` 코드에 다음 코드를 추가해서 실행해보자.

```
// 중복 체크 확인용
@Mapping("/site2")
public void page2(HttpResponse response) {
    response.writeBody("<h1>page2</h1>");
}
```

```
package was.v8;

import was.httpserver.HttpResponse;
import was.httpserver.servlet.annotation.Mapping;

public class SiteControllerV8 {

    @Mapping("/")
```

```
    public void home(HttpResponse response) {
        response.writeBody("<h1>home</h1>");
        response.writeBody("<ul>");
        response.writeBody("<li><a href='/site1'>site1</a></li>");
        response.writeBody("<li><a href='/site2'>site2</a></li>");
        response.writeBody("<li><a href='/search?q=hello'>검색</a></li>");
        response.writeBody("</ul>");
    }

    @Mapping("/site1")
    public void site1(HttpResponse response) {
        response.writeBody("<h1>site1</h1>");
    }

    @Mapping("/site2")
    public void site2(HttpResponse response) {
        response.writeBody("<h1>site2</h1>");
    }

    // 중복 체크 확인용
/*
    @Mapping("/site2")
    public void page2(HttpResponse response) {
        response.writeBody("<h1>page2</h1>");
    }
*/

}
```

실행 결과

```
Exception in thread "main" java.lang.IllegalArgumentException: 경로 중복 등록,
path=/site2,
method=public void was.v8.SiteControllerV8.page2(was.httpserver.HttpResponse),
이미 등록된 메서드=public void
was.v8.SiteControllerV8.site2(was.httpserver.HttpResponse)

    at
was.httpserver.servlet.annotation.AnnotationServletV3.initializePathMap(Annota
tionServletV3.java:32)
    at
was.httpserver.servlet.annotation.AnnotationServletV3.<init>(AnnotationServlet
```

```
V3.java:21)
    at was.v8.ServerMainV8.main(ServerMainV8.java:19)
```

서버를 실행하는 시점에 바로 오류가 발생하고 서버 실행이 중단된 것을 확인할 수 있다. 이렇게 서버 실행 시점에 발견할 수 있는 오류는 아주 좋은 오류이다.
만약 이런 오류를 체크하지 않고, `/site2`가 2개 유지된 대로 작동한다면, 고객은 기대한 화면과 다른 화면을 보고 있을 수 있다.

3가지 오류

- **컴파일 오류**: 가장 좋은 오류이다. 프로그램 실행 전에 개발자가 가장 빠르게 문제를 확인할 수 있다.
- **런타임 오류 - 시작 오류**: 자바 프로그램이나 서버를 시작하는 시점에 발견할 수 있는 오류이다. 문제를 아주 빠르게 발견할 수 있기 때문에 좋은 오류이다. 고객이 문제를 인지하기 전에 수정하고 해결할 수 있다.
- **런타임 오류 - 작동 오류**: 고객이 특정 기능을 작동할 때 발생하는 오류이다. 원인 파악과 문제 해결에 가장 많은 시간이 걸리고 가장 큰 피해를 주는 오류이다.

정리

드디어 성능, 유연성, 오류 체크 기능까지 강화한 쓸만한 `AnnotationServletV3`을 만들어냈다.
이제 지금까지 만든 기능을 모두 활용해서 회원 정보를 관리하는 웹 애플리케이션 서비스를 만들어보자.

HTTP 서버 활용 - 회원 관리 서비스1

우리는 앞서 I/O에서 콘솔을 활용해 다음 회원 관리 기능을 개발했다.

요구사항

회원 관리 프로그램을 작성해라.
회원의 속성은 다음과 같다.

- ID
- Name
- Age

회원을 등록하고, 등록한 회원의 목록을 조회할 수 있어야 한다.

```
1.회원 등록 | 2.회원 목록 조회 | 3.종료
```

```
선택: 1
ID 입력: id1
Name 입력: name1
Age 입력: 20
회원이 성공적으로 등록되었습니다.

1.회원 등록 | 2.회원 목록 조회 | 3.종료
선택: 1
ID 입력: id2
Name 입력: name2
Age 입력: 30
회원이 성공적으로 등록되었습니다.

1.회원 등록 | 2.회원 목록 조회 | 3.종료
선택: 2
회원 목록:
[ID: id1, Name: name1, Age: 20]
[ID: id2, Name: name2, Age: 30]

1.회원 등록 | 2.회원 목록 조회 | 3.종료
선택: 3
프로그램을 종료합니다.
```

이 기능을 콘솔이 아닌 웹으로 구현해보자.
참고로 기존에 있던 `io.member` 패키지의 `Member`, `MemberRepository`와 같은 기능은 그대로 사용하자.

회원 컨트롤러

```java
package webservice;

import io.member.Member;
import io.member.MemberRepository;
import was.httpserver.HttpRequest;
import was.httpserver.HttpResponse;
import was.httpserver.servlet.annotation.Mapping;

import java.util.List;

import static util.MyLogger.log;
```

```
public class MemberController {
    private final MemberRepository memberRepository;

    public MemberController(MemberRepository memberRepository) {
        this.memberRepository = memberRepository;
    }

    @Mapping("/")
    public void home(HttpResponse response) {
        String str = "<html><body>" +
                "<h1>Member Manager</h1>" +
                "<ul>" +
                "<li><a href='/members'>Member List</a></li>" +
                "<li><a href='/add-member-form'>Add New Member</a></li>" +
                "</ul>" +
                "</body></html>";
        response.writeBody(str);
    }

    @Mapping("/members")
    public void members(HttpResponse response) {
        List<Member> members = memberRepository.findAll();

        StringBuilder page = new StringBuilder();
        page.append("<html><body>");
        page.append("<h1>Member List</h1>");
        page.append("<ul>");
        for (Member member : members) {
            page.append("<li>")
                    .append("ID: ").append(member.getId())
                    .append(", Name: ").append(member.getName())
                    .append(", Age: ").append(member.getAge())
                    .append("</li>");
        }
        page.append("</ul>");
        page.append("<a href='/'>Back to Home</a>");
        page.append("</body></html>");
        response.writeBody(page.toString());
    }

    @Mapping("/add-member-form")
    public void addMemberForm(HttpResponse response) {
        String body = "<html><body>" +
```

```
                "<h1>Add New Member</h1>" +
                "<form method='POST' action='/add-member'>" +
                "ID: <input type='text' name='id'><br>" +
                "Name: <input type='text' name='name'><br>" +
                "Age: <input type='text' name='age'><br>" +
                "<input type='submit' value='Add'>" +
                "</form>" +
                "<a href='/'>Back to Home</a>" +
                "</body></html>";
        response.writeBody(body);
    }

    @Mapping("/add-member")
    public void addMember(HttpRequest request, HttpResponse response) {
        log("MemberController.addMember");
        log("request = " + request);

        String id = request.getParameter("id");
        String name = request.getParameter("name");
        int age = Integer.parseInt(request.getParameter("age"));

        Member member = new Member(id, name, age);
        memberRepository.add(member);

        response.writeBody("<h1>save ok</h1>");
        response.writeBody("<a href='/'>Back to Home</a>");
    }
}
```

home()

첫 화면이다. 회원 목록과 신규 회원 등록 화면으로 이동하는 기능을 제공한다.

- Member List - `/members`
- Add New Member - `/add-member-form`

members()

- `memberRepository.findAll()` 기능을 사용해서 저장된 모든 회원 목록을 조회한다.
- 반복문을 사용해서 컬렉션에 담긴 회원 정보를 기반으로 다음과 같은 HTML을 생성한다.

```
<html>
<body>
<h1>Member List</h1>
<ul>
    <li>ID: id1, Name: member1, Age: 20</li>
    <li>ID: id2, Name: member2, Age: 30</li>
    ...
</ul>
<a href='/'>Back to Home</a></body>
</html>
```

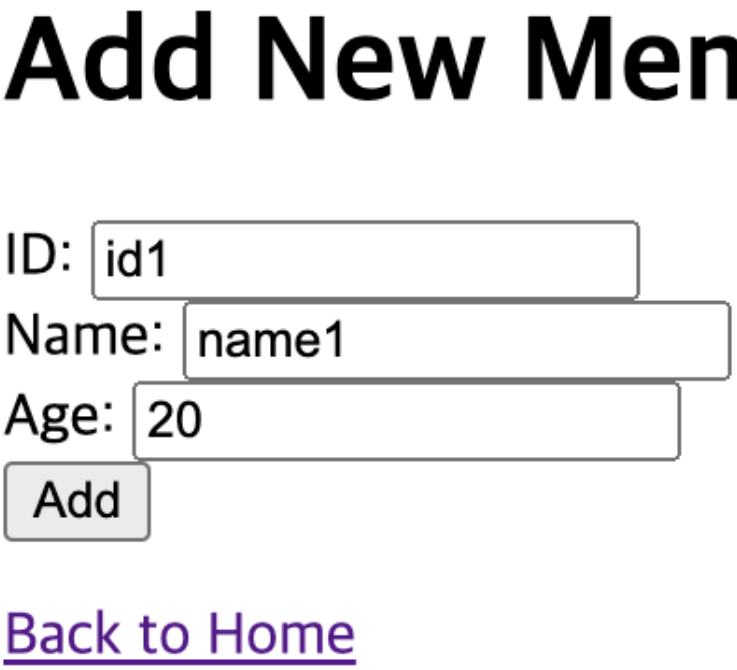
addMemberForm

회원을 저장하기 위해서는 회원을 등록하는 화면이 필요하다. HTML에서는 이것을 폼(form)이라 한다.

그리고 이런 폼을 처리하기 위한 특별한 HTML 태그들을 지원한다.

```
<html>
<body>
<h1>Add New Member</h1>
<form method='POST' action='/add-member'>
  ID: <input type='text' name='id'><br>
  Name: <input type='text' name='name'><br>
  Age: <input type='text' name='age'><br>
  <input type='submit' value='Add'>
</form>
<a href='/'>Back to Home</a></body>
</html>
```

- `<form>` 클라이언트에서 서버로 전송할 데이터를 입력하는 기능을 제공한다.
  - `method=POST` : HTTP 메시지를 전송할 때 POST 방식으로 전송한다. 참고로 `POST` 는 메시지 바디에 필요한 데이터를 추가해서 서버에 전달할 수 있다.
  - `action='/add-member'` : HTTP 메시지를 전송할 URL 경로이다.
- `<input type='text'>` 클라이언트에서 서버로 전송할 각각의 항목이다. `name` 이 키로 사용된다.
- `<input type='submit'>` 폼에 입력한 내용을 서버에 전송할 때 사용하는 전송 버튼이다.



이렇게 입력하고 `Add` 버튼을 누르면 HTTP 메시지는 다음과 같이 생성된다.

클라이언트가 생성한 HTTP 요청 메시지

```
POST /add-member HTTP/1.1
Host: localhost:12345
Content-Length: 24
Content-Type: application/x-www-form-urlencoded

id=id1&name=name1&age=20
```

- `Content-Length` : 메시지 바디가 있는 경우 메시지 바디의 크기를 표현한다.
- `Content-Type` : 메시지 바디가 있는 경우 메시지 바디의 형태를 표현한다.
  - `application/x-www-form-urlencoded` 은 HTML의 폼을 사용해서 전송한 경우이다.
  - 이것은 형식으로 `input type` 에서 입력한 내용을 `key=value` 형식으로 메시지 바디에 담아서 전송한다.
  - URL에서 `?` 이후의 부분에 `key1=value1&key2=value2` 포멧으로 서버에 전송하는 것과 거의 같은 포멧으로 전송한다.
    - `/search?q=hello` 를 떠올려보자.

addMember()

```
String id = request.getParameter("id");
String name = request.getParameter("name");
int age = Integer.parseInt(request.getParameter("age"));

Member member = new Member(id, name, age);
memberRepository.add(member);
```

메시지 바디에 담겨있는 `id=id1&name=name1&age=20` 데이터를 꺼내서 회원 객체를 생성한다. 그리고 `MemberRepository`를 통해서 회원을 저장소에 저장한다.

HTTP 서버 활용 - 회원 관리 서비스2

HttpRequest - 메시지 바디 파싱

그런데 아직 `HttpRequest`에서 메시지 바디를 파싱하도록 처리하지 않았다. 이 부분을 추가하자.

```
package was.httpserver;

import java.io.BufferedReader;
import java.io.IOException;
import java.net.URLDecoder;
import java.util.HashMap;
import java.util.Map;

import static java.nio.charset.StandardCharsets.*;
import static util.MyLogger.log;

public class HttpRequest {

    private String method;
    private String path;
    private final Map<String, String> queryParameters = new HashMap<>();
    private final Map<String, String> headers = new HashMap<>();

    public HttpRequest(BufferedReader reader) throws IOException {
        parseRequestLine(reader);
        parseHeaders(reader);
        parseBody(reader); // 추가
```

```
    }

    private void parseRequestLine(BufferedReader reader) throws IOException {
        String requestLine = reader.readLine();
        if (requestLine == null) {
            throw new IOException("EOF: No request line received");
        }

        String[] parts = requestLine.split(" ");
        if (parts.length != 3) {
            throw new IOException("Invalid request line: " + requestLine);
        }

        method = parts[0];
        String[] pathParts = parts[1].split("\\?");
        path = pathParts[0];

        if (pathParts.length > 1) {
            parseQueryParameters(pathParts[1]);
        }
    }

    private void parseQueryParameters(String queryString) {
        for (String param : queryString.split("&")) {
            String[] keyValue = param.split("=");
            String key = URLDecoder.decode(keyValue[0], UTF_8);
            String value = keyValue.length > 1 ?
URLDecoder.decode(keyValue[1], UTF_8) : "";
            queryParameters.put(key, value);
        }
    }

    private void parseHeaders(BufferedReader reader) throws IOException {
        String line;
        while (!(line = reader.readLine()).isEmpty()) {
            String[] headerParts = line.split(":");
            // "trim() 앞 뒤에 공백 제거"
            headers.put(headerParts[0].trim(), headerParts[1].trim());
        }
    }

    // 추가
    private void parseBody(BufferedReader reader) throws IOException {
```

```
        if (!headers.containsKey("Content-Length")) {
            return;
        }

        int contentLength = Integer.parseInt(headers.get("Content-Length"));
        char[] bodyChars = new char[contentLength];
        int read = reader.read(bodyChars);
        if (read != contentLength) {
            throw new IOException("Failed to read entire body. Expected " +
contentLength + " bytes, but read " + read);
        }
        String body = new String(bodyChars);
        log("HTTP Message Body: " + body);

        String contentType = headers.get("Content-Type");
        if ("application/x-www-form-urlencoded".equals(contentType)) {
            parseQueryParameters(body);
        }
    }

    public String getMethod() {
        return method;
    }

    public String getPath() {
        return path;
    }

    public String getParameter(String name) {
        return queryParameters.get(name);
    }

    public String getHeader(String name) {
        return headers.get(name);
    }

    @Override
    public String toString() {
        return "HttpRequest{" +
                "method='" + method + '\'' +
                ", path='" + path + '\'' +
                ", queryParameters=" + queryParameters +
                ", headers=" + headers +
```

```
                '}';
    }

}
```

- `Content-Length`가 있는 경우 메시지 바디가 있다고 가정하겠다.
- `Content-Length`의 길이 만큼 스트림에서 메시지 바디의 데이터를 읽어온다.
  - 만약 읽어온 길이가 다르다면 문제가 있다고 보고 예외를 던진다.
- 다음으로 `Content-Type`을 체크한다. 만약 HTML 폼 전송인 `application/x-www-form-urlencoded` 타입이라면 URL의 쿼리 스트링과 같은 방식으로 파싱을 시도한다.
- 그리고 파싱 결과를 URL의 쿼리 스트링과 같은 `queryParameters`에 보관한다.
- 이렇게 하면 URL의 쿼리 스트링이든, HTML 폼 전송이든 `getParameter()`를 사용해서 같은 방식으로 데이터를 편리하게 조회할 수 있다.

```
package webservice;

import io.member.MemberRepository;
import io.member.impl.FileMemberRepository;
import io.member.impl.ObjectMemberRepository;
import was.httpserver.HttpServer;
import was.httpserver.HttpServlet;
import was.httpserver.ServletManager;
import was.httpserver.servlet.DiscardServlet;
import was.httpserver.servlet.annotation.AnnotationServletV3;

import java.io.IOException;
import java.util.List;

public class MemberServerMain {
    private static final int PORT = 12345;

    public static void main(String[] args) throws IOException {
        MemberRepository memberRepository = new FileMemberRepository();
        MemberController memberController = new
MemberController(memberRepository);
        HttpServlet servlet = new
AnnotationServletV3(List.of(memberController));

        ServletManager servletManager = new ServletManager();
        servletManager.add("/favicon.ico", new DiscardServlet());
```

```
        servletManager.setDefaultServlet(servlet);
        HttpServer server = new HttpServer(PORT, servletManager);
        server.start();
    }
}
```

MemberRepository 인터페이스의 구현체

- `MemoryMemberRepository`
- `FileMemberRepository`
- `DataMemberRepository`
- `ObjectMemberRepository`

저장한 데이터를 확인하기 쉽도록 `FileMemberRepository`를 사용했다. 이 부분은 선택이다.

`new MemberController(memberRepository)`

- `MemberController`는 `MemberRepository`가 필요하다. 여기서 중요한 핵심은 `MemberController`는 `MemberRepository` 인터페이스에만 의존한다는 점이다. 실제 런타임에 어떤 인스턴스가 들어올지는 모른다.
- 런타임에 `MemoryMemberRepository`에서 `FileMemberRepository`로 변경하더라도 `MemberController`의 코드는 전혀 변경하지 않아도 된다.
- `MemberController` 입장에서는 `MemberRepository`의 인스턴스를 외부에서 주입 받는 것 처럼 느껴진다. 이것을 의존 관계 주입(Dependency Injection), 줄여서 DI라 한다.
- 앞서 마지막에 작성한 `AnnotationServletV3`를 사용한다.

실행 결과

- 영상 참고

정리

구성 역할, 사용 역할

- `MemberServerMain` 클래스의 역할은 아주 재미있다. 본인이 어떤 코드 블록을 만든 것이 아니라, 지금까지 있는 코드 블록들을 조립하는 일을 한다. 어떤 `MemberRepository`를 사용할 지, 어떤 컨트롤러를 사용할 지, 어떤 `HttpServlet`을 사용할 지 선택한다.
- 이것은 마치 레고 블록을 조립하는 것과 비슷한 느낌이 든다.

- 이것은 마치 필요한 컴포넌트를 구성(Configuration)하는 것 같다.
- 컴퓨터를 조립할 때 어떤 CPU를 사용할 지, 어떤 메모리, GPU를 선택할 지 고르는 것 같다.
- `MemberServerMain`는 프로젝트를 **구성하는 역할**을 담당한다.
- 나머지 클래스들은 실제 기능을 제공하는 **사용 역할**을 담당한다.
- 이렇게 구성하는 역할과 사용하는 역할을 명확하게 분리해두면 다음과 같은 장점이 있다.

- **유연성 향상**: 프로젝트의 구성을 쉽게 변경할 수 있다. 예를 들어, 다른 `MemberRepository`나 컨트롤러를 사용하고 싶을 때 `MemberServerMain`만 수정하면 된다.
- **테스트 용이성**: 각 컴포넌트를 독립적으로 테스트할 수 있다. 구성 로직과 실제 기능 로직이 분리되어 있어 단위 테스트가 더 쉬워진다.
- **코드 재사용성 증가**: 각 컴포넌트는 독립적이므로 다른 프로젝트에서도 쉽게 재사용할 수 있다.
- **관심사의 분리**: 구성 로직과 비즈니스 로직이 분리되어 각 부분에 집중할 수 있다.
- **유지보수 용이성**: 전체 시스템의 구조를 이해하기 쉬워지며, 특정 부분을 수정할 때 다른 부분에 미치는 영향을 최소화할 수 있다.
- **확장성 개선**: 새로운 기능이나 컴포넌트를 추가할 때, 기존 코드를 크게 수정하지 않고도 `MemberServerMain`에 새로운 구성을 추가할 수 있다.

이러한 설계 방식을 효과적으로 구현하려면 소프트웨어 개발의 주요 원칙들을 준수해야 한다. 특히 다형성, 개방-폐쇄 원칙(OCP), 그리고 의존관계 주입(DI) 원칙이 중요하다. 이러한 접근법은 대규모 프로젝트에서 특히 유용하다.
하지만 구성과 사용의 역할을 명확히 구분하고 소프트웨어 개발의 핵심 원칙들을 적용하는 것은 쉽지 않다. 그러나 이러한 과정을 크게 간소화해주는 도구가 있는데, 그것이 바로 스프링 프레임워크이다.
스프링은 앞서 언급한 구성과 사용의 역할 분리, 그리고 소프트웨어 개발의 핵심 원칙들을 쉽고 효과적으로 적용할 수 있게 도와주는 실무 백엔드 개발의 핵심 기술이다.