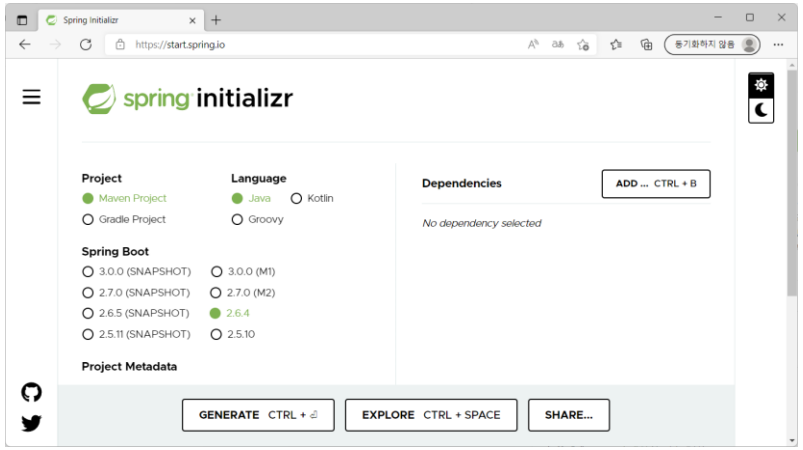
PART5 – 스프링에서 스프링부트로

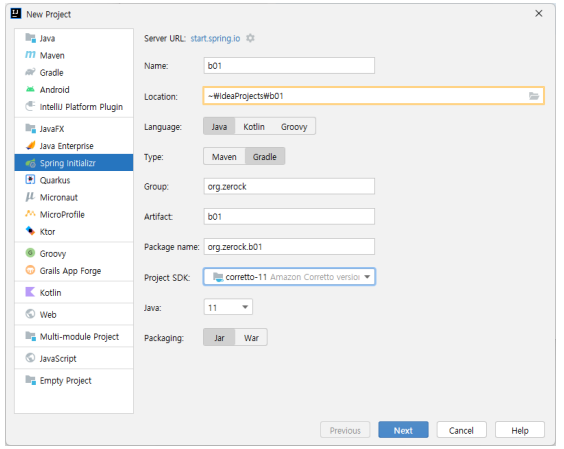
SpringBoot

- **Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".**
- 스프링 프레임워크을 더 편리하게 사용할 수 있도록 하는 도구의 역할
- '스프링 프레임워크' == '스프링 부트' 로 인식되는 과정중
- 점차 스프링기반의 개발방식에서 스프링 부트를 이용하는 형식으로 변화 중
- 편리한 기능들
  - 자동 설정 기능
  - 내장 **WAS**
  - 개발 도구의 지원 등

프로젝트의 생성방식

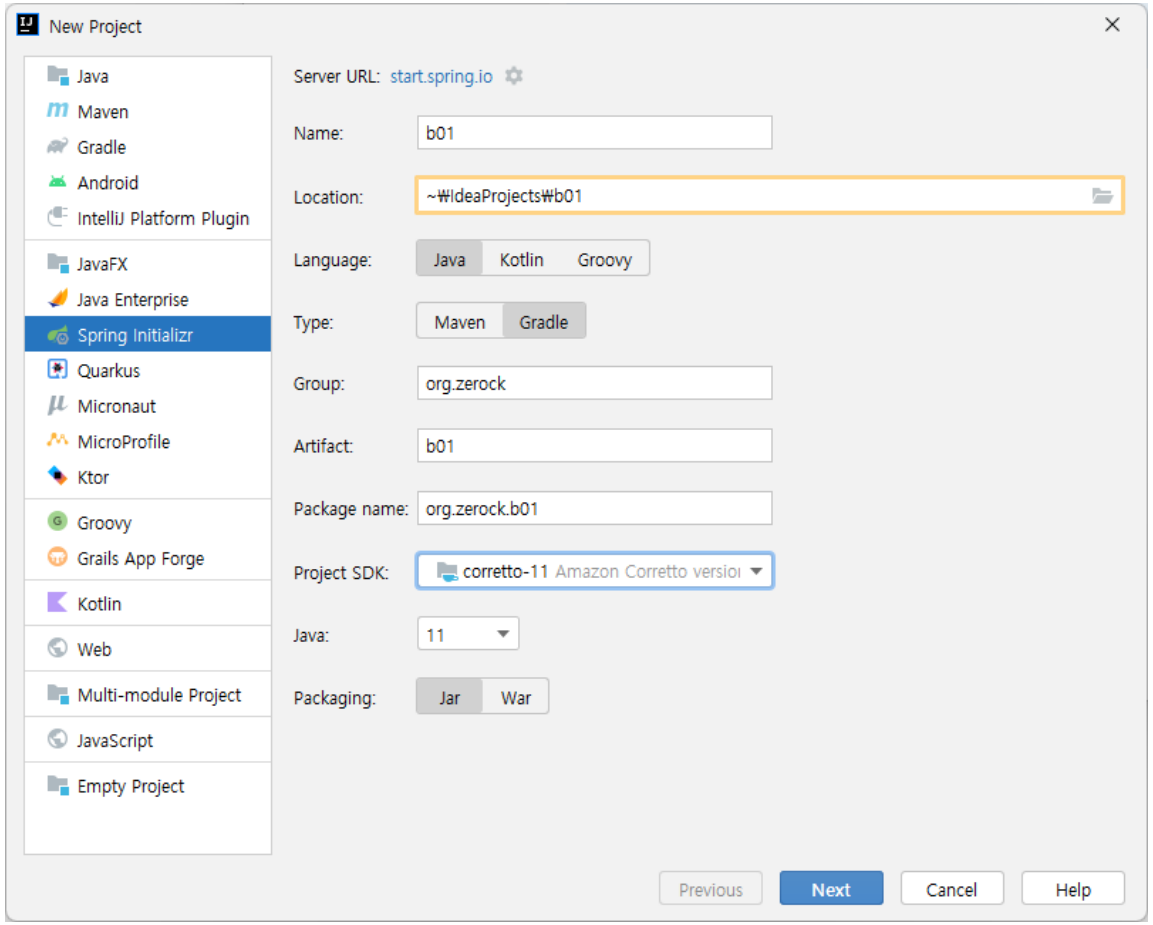
- Spring Initializr
  - 개발도구 지원
    - eclipse
    - Intellij
    - VSCode
  - web상에서 프로젝트 생성후 다운로드
- Maven/Gradle





프로젝트의 생성

- 프로젝트 생성시 추가하는 의존성 라이브러리
  - Spring Boot DevTools
  - Lombok
  - Spring Web
  - Thymeleaf
  - Spring Data JPA
  - MariaDB Driver



DataSource 설정

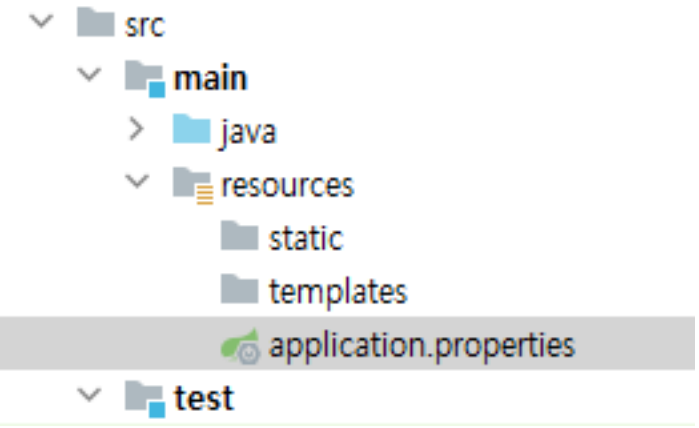
- 자동 설정기능으로 인해 의존성 라이브러리를 추가한 것 만으로 자동으로 관련 설정을 사용하게 된다.

- HikariCP를 기본으로 사용

- Spring Data JPA에서 사용할 데이터베이스 관련 설정 필요

- 설정 파일
  - application.properties 혹은 application.yml 형식

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

Reason: Failed to determine a suitable driver class
```




```
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://localhost:3306/webdb
spring.datasource.username=webuser
spring.datasource.password=webuser
```

```
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.5.Final
o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MariaDB103Dialect
o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.
j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database
o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
```

추가적으로 필요한 설정

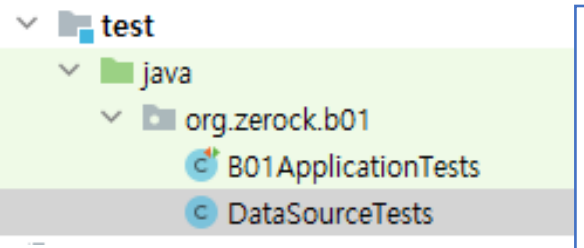
-Lombok 관련 설정

-자동 재시작 설정

-Intellij내 데이터베이스 설정

-로그레벨 설정

테스트 환경과 의존성 주입




```java
package org.zerock.b01;

import lombok.Cleanup;
import lombok.extern.log4j.Log4j2;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;

@SpringBootTest
@Log4j2
public class DataSourceTests {

  @Autowired
  private DataSource dataSource;

  @Test
  public void testConnection() throws SQLException {

    @Cleanup
    Connection con = dataSource.getConnection();

    log.info(con);

    Assertions.assertNotNull(con);
  }
}
```

```
JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed
org.zerock.b01.DataSourceTests           : Started DataSourceTests in 2.249 seconds (JVM running for 3.271)
org.zerock.b01.DataSourceTests           : HikariProxyConnection@2086483651 wrapping org.mariadb.jdbc.MariaDbConnection@31de27c
j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

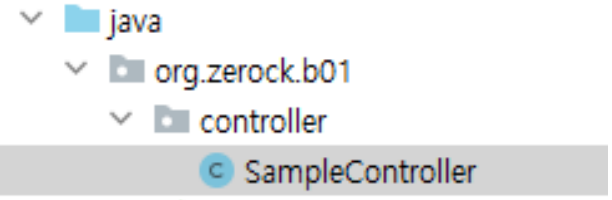
Spring Data JPA를 위한 설정

- spring.jpa.hibernate.ddl-auto=update
  spring.jpa.properties.hibernate.format_sql=true
  spring.jpa.show-sql=true

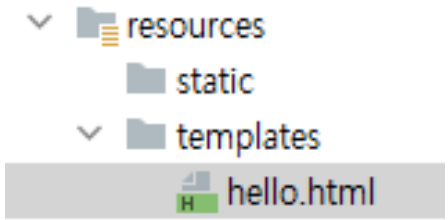
속성값	의미
none	DDL을 하지 않음
create-drop	실행할때 DDL을 실행하고 종료시에 만들어진 테이블등을 모두 삭제
create	실행할때마다 새롭게 테이블등을 생성
update	기존과 다르게 변경된 부분이 있을때는 새로 생성
validate	변경된 부분만 알려주고 종료

스프링부트에서의 웹 개발

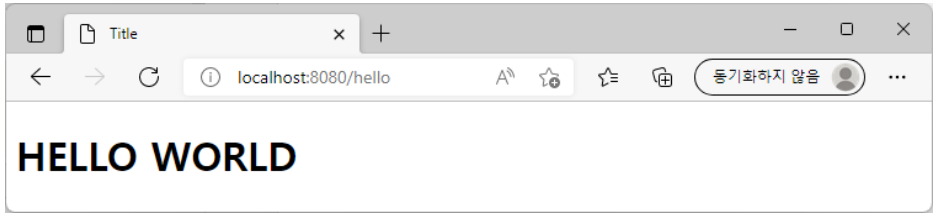
- web.xml이나 servlet-context.xml이 없는 환경에서 개발
- 설정을 위한 @Configuration이나 상속등을 사용
- 스프링부트는 기본적으로 JSP를 지원하지 않음



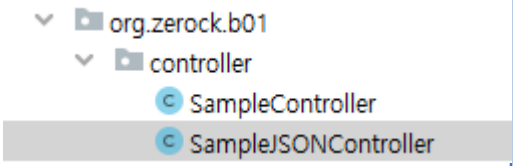

```
package org.zerock.b01.controller;


import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
@Log4j2
public class SampleController {

    @GetMapping("/hello")
    public void hello(Model model) {

        log.info("hello.....");

        model.addAttribute("msg", "HELLO WORLD");
    }
}
```




```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <h1 th:text="${msg}"></h1>
</body>
</html>
```

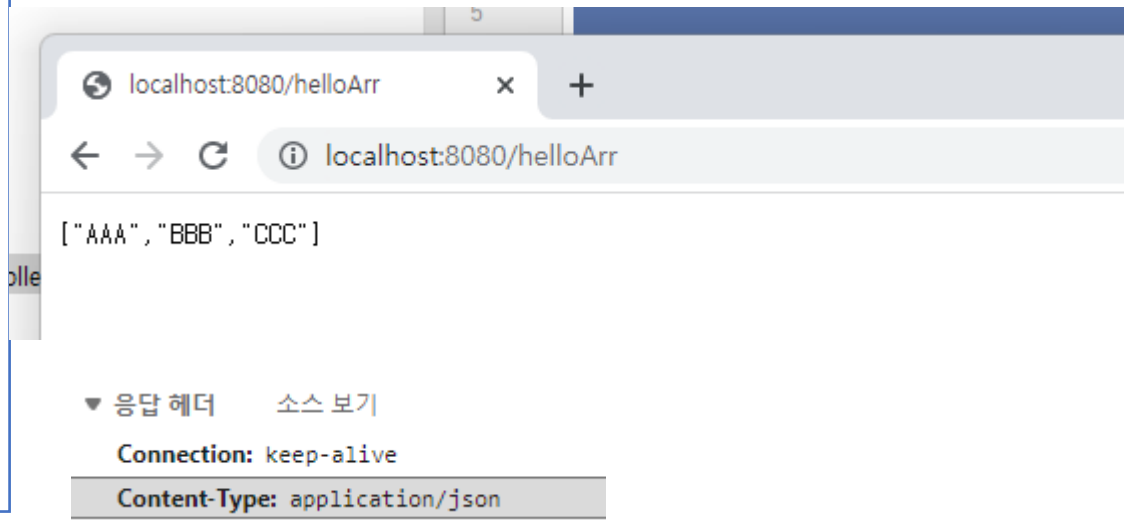


JSON 데이터 만들기




```java
package org.zerock.b01.controller;
import lombok.extern.log4j.Log4j2;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
@Log4j2
public class SampleJSONController {
    @GetMapping("/helloArr")
    public String[] helloArr() {
        log.info("helloArr.....");
        return new String[]{"AAA", "BBB", "CCC"};
    }
}
```

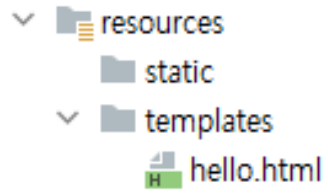
별도의 라이브러리 없이 스프링부트는 기본적으로 json 처리를 지원



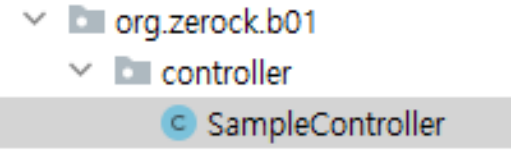
Thymeleaf

Thymeleaf 소개

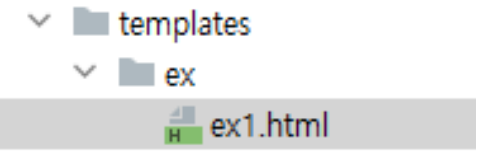
- Thymeleaf의 특징
  - JSP의 경우 서블릿으로 변환된 후에 실행되는 방식
  - Thymeleaf는 서버사이드 템플릿 엔진
  - HTML의 구조에 추가적인 태그없이 선언적으로 데이터 바인딩 처리



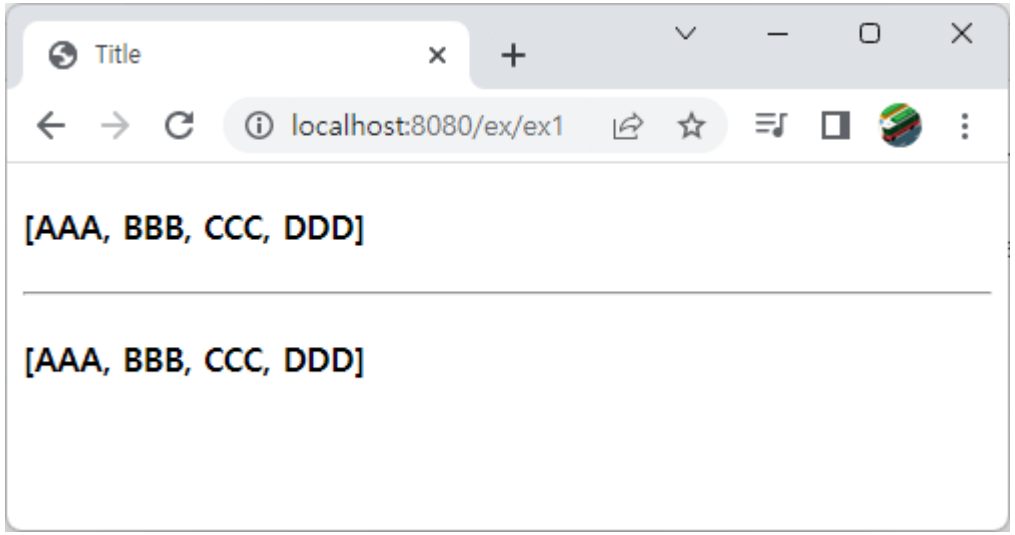

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <h1 th:text="${msg}"></h1>
</body>
</html>
```




```java
@GetMapping("/ex/ex1")
public void ex1(Model model){

    List<String> list = Arrays.asList("AAA","BBB","CCC","DDD");

    model.addAttribute("list", list);

}
```



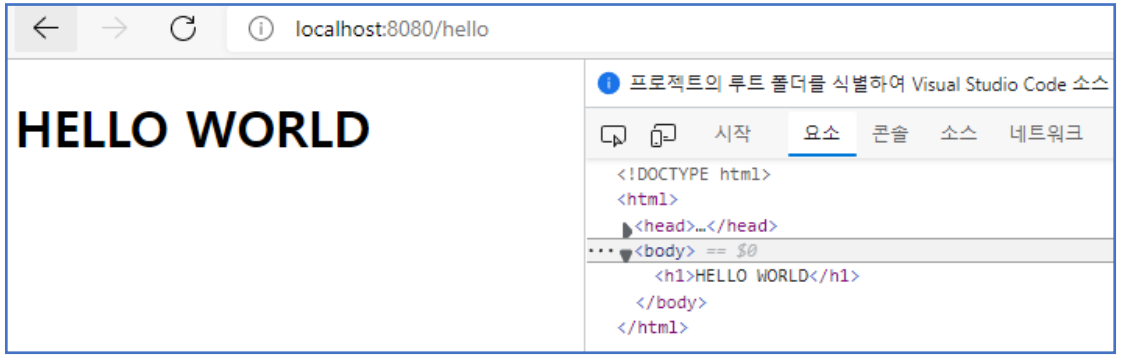

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h4>[[${list}]]</h4>
<hr/>
<h4 th:text="${list}"></h4>
</body>

</html>
```



Thymeleaf 주석처리

- 주석 처리를 해야 할 때에는 '<!--/* ... */-->' 를 이용

```
<body>
  <h1 th:text="${msg}"></h1>

<!--/*  <h3 th:each="${sos}">SOS</h3> */-->

  <!--/* ${aaaa + bbb } */-->
  <!--/*
   <div>
      <h1>AAAA</h1>
   </div>
  */-->

</body>
```



th:with를 이용한 변수 선언

```
<div th:with="num1 = ${10}, num2 = ${20}">
    <h4 th:text="${num1 + num2}"></h4>
</div>
```

반복문과 제어문 처리

th:each를 이용해서 배열/리스트/컬렉션 처리

반복문의 status 변수

- 반복문에서 자주 사용하는 인덱스 번호나 개수등을 사용
- index/count/size/first/odd/even

th:if / th:unless / th:swith 를 이용한 제어 처리

```
<ul>
    <li th:each="str,status: ${list}">
        <span th:if="${status.odd}"> ODD -- [[${str}]]</span>
        <span th:unless="${status.odd}"> EVEN --
[[${str}]]</span>
    </li>
</ul>
```

Thymeleaf를 이용한 링크 처리

절대 경로/컨텍스트 경로를 자동으로 처리

'@'를 이용해서 처리

쿼리 스트링 처리

```
<a th:href="@{/hello(name='AAA', age= 16)}">Go to /hello</a>
```

```
<a href="/hello?name=AAA&age=16">Go to /hello</a>
```

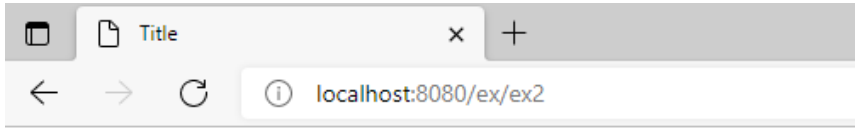
```
<a th:href="@{/hello(types=${ {'AA','BB','CC'} }, age= 16)}">Go to
/hello</a>
```

```
<a href="/hello?types=AA&types=BB&types=CC&age=16">Go to /hello</a>
```

Thymeleaf의 특별한 기능

- 인라인 처리
  - JavaScript의 경우 변수를 자동으로 JavaScript Object 형태로 출력

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div th:text="${list}"></div>
    <div th:text="${map}"></div>
    <div th:text="${dto}"></div>

    <script th:inline="javascript">

        const list = [[${list}]]

        const map = [[${map}]]

        const dto = [[${dto}]]

        console.log(list)
        console.log(map)
        console.log(dto)

    </script>

</body>
</html>
```



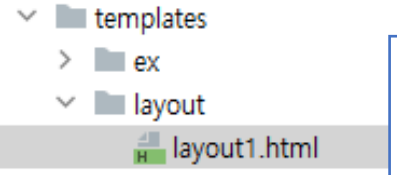
[Data1, Data2, Data3, Data4, Data5, Data6, Data7, Data8, Data9]
{A=AAAA, B=BBBB}
org.zerock.b01.controller.SampleController$SampleDTO@cf7179e

```
<script>
        const list = ["Data1","Data2","Data3","Data4","Data5","Data6","Data7","Data8","Data9"]

        const map = {"A":"AAAA","B":"BBBB"}

        const dto = {"p1":"Value -- p1","p2":"Value -- p2","p3":"Value -- p3"}

        console.log(list)
        console.log(map)
        console.log(dto)
</script>
```

Thymeleaf의 레이아웃 기능

- <th:block>을 이용해서 필요한 부분만을 작성하는 방식

```
implementation 'nz.net.ultraq.thymeleaf:thymeleaf-layout-dialect:3.1.0'
```




```
<!DOCTYPE html>
<html
xmlns:layout="http://www.ultraq.net.nz/th
ymeleaf/layout"

xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Layout page</title>
</head>
<body>

<div>
    <h3>Sample Layout Header</h3>
</div>

<div layout:fragment="content">
    <p>Page content goes here</p>
</div>

<div>
    <h3>Sample Layout Footer</h3>
</div>


<th:block layout:fragment="script" >

</th:block>

</body>
</html>
```

layout:fragment를 이용해서 변경이 가능한 부분을 지정하고 나중에 다른 내용물로 변경 가능

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"

xmlns:layout="http://www.ultraq.net.nz/thymeleaf/lay
out"
      layout:decorate="~{layout/layout1.html}">

<div layout:fragment="content">

    <h1>ex3.html</h1>

</div>
```