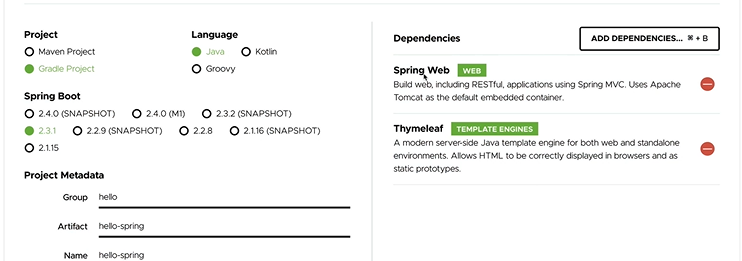
//1-5

Spring를 하기 위한 준비물 (Java 11, intellij / eclipse)

프로젝트 생성

요즘은 스프링부트를 통하여 프로젝트를 만드는 것이 더 쉽기 때문에 start.spring.io를 통해서 프로젝트를 생성한다. 다음과 같이 설정한다.



생성하면 zip파일이 나오는데 압축을 풀고 intellij에서 import를 한다.

Import -> Existing Gradle Project -> 생성 폴더 -> 생성된 프로젝트를 확인한다.

//2-1

프로젝트를 실행시키면 console에 정보가 나오며 chrome 주소에 localhost:8080을 입력하게 되면 에러페이지가 나온다면 성공적으로 한 것 이다.

Resources -> static -> index.html생성 후 내용을 입력하면 홈페이지에 hello가 출력된다.

<!DOCTYPE HTML>  
<html>  
<head>  
 <title>Hello</title>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
</head>  
<body>  
Hello  
<a href="/hello">hello</a>  
</body>  
</html>

Main -> java -> hellospring -> package(controller)생성 -> javaclass(HelloController) 생성 -> 입력

@Controller  
public class HelloController {  
  
 @GetMapping("hello")  
 public String hello(Model model) {  
 model.addAttribute("data", "hello!!");  
 return "hello";  
 }  
}

Resources -> templates -> hello.html 생성 -> 입력

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>Hello</title>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
</head>  
<body>  
<p th:text="'안녕하세요. ' + ${data}" >안녕하세요. 손님</p>  
</body>  
</html>

정적 컨텐츠 -> 파일을 그대로 웹 브라우저에 전달해주는 것

MVC와 템플릿엔진 -> 서버에서 변형 후 html을 바꿔서 내려주는 방식

API->객체반환 / json이라는 구조포맷을 통하여 클라이언트에 전달하는 방식, 서버끼리 통신할 때

스프링 부트 정적 컨텐츠

Resources -> static -> hello-static.html (생성) / <http://localhost:8080/hello-static.html>를 입력하면 바로 확인이 가능하다.

<!DOCTYPE HTML>  
<html>  
<head>  
 <title>static content</title>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
</head>  
<body>  
정적 컨텐츠 입니다.  
</body>  
</html>

MVC : Model, View, Controller

Controller 추가

@GetMapping("hello-mvc")  
public String helloMvc(@RequestParam("name") String name, Model model) {  
 model.addAttribute("name", name);  
 return "hello-template";  
}

View

Resources -> template -> hello-template.html 생성

<html xmlns:th="http://www.thymeleaf.org">  
<body>  
<p th:text="'hello ' + ${name}">hello! empty</p>  
</body>  
</html>

// hello! Empty 는 없어도 되는 부분이다

view-source:/ 경로를 입력하면 서버 연결 없이 확인 했을 때 hello! Empty가 출력된다.

실행 -> http://localhost:8080/hello-mvc?name=(spring, 아무거나)

API

Controller 추가

@GetMapping("hello-string")  
@ResponseBody // http의 body부에 데이터를 직접 넣어주겠다.  
public String helloString(@RequestParam("name") String name){  
 return "hello " + name; //"hello spring"  
}

실행 : <http://localhost:8080/hello-string?name=spring> / 페이지 소스를 보면 Html태그가 없음

Controller 추가

@GetMapping("hello-api")  
@ResponseBody  
public Hello helloApi(@RequestParam("name") String name) {  
 Hello hello = new Hello();  
 hello.setName(name);  
 return hello;  
}  
  
 static class Hello {  
 private String name;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
 }  
}

실행 -> <http://localhost:8080/hello-api?name=spring> / 출력은 비슷하지만 json방식으로 출력됨

Json = key, value로 이루어진 단순한 구조

회원 도메인과 저장소 만들기

회원 객체

package hellospring.hellospring.domain;  
  
public class Member {  
  
 private Long id;  
 private String name;  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

회원 저장소 인터페이스

package hellospring.hellospring.repository;  
  
public interface MemberRepository {  
 Member save(Member member);  
 Optional<Member> findById(Long id);  
 Optional<Member> findByName(String name);  
 List<Member> findAll();  
}

회원 저장소 메모리 구현체

package hellospring.hellospring.repository;

public class MemoryMemberRepository implements MemberRepository{  
  
 private static Map<Long, Member> *store* = new HashMap<>();  
 private static long *sequence* = 8L;  
  
 @Override  
 public Member save(Member member) {  
 member.setId(++*sequence*);  
 *store*.put(member.getId(), member);  
 return member;  
 }  
  
 @Override  
 public Optional<Member> findById(Long id) {  
 return Optional.*ofNullable*(*store*.get(id));  
 }  
  
 @Override  
 public Optional<Member> findByName(String name) {  
 return *store*.values().stream()  
 .filter(member -> member.getName().equals(name))  
 .findAny();  
 }  
  
 @Override  
 public List<Member> findAll() {  
 return new ArrayList<>(*store*.values());  
 }  
  
 public void clearStore() {  
 *store*.clear();  
 }  
}

이것들을 검증하기 위해서 test를 작성하는 것이다.

Test -> repository(package)생성 -> MemoryMemberRepositoryTest(java class)생성

전체 or 부분으로 실행하면 확인이 가능하다

public class MemroyMemberRepositoryTest {  
  
 MemoryMemberRepository repository = new MemoryMemberRepository();  
  
 @AfterEach  
 public void afterEach() {  
 repository.clearStore();  
 }//실행 후 repository초기화  
 @Test  
 public void save(){  
 Member member = new Member();  
 member.setName("spring");  
  
 repository.save(member);  
  
 Member result = repository.findById(member.getId()).get();  
 //Assertions.assertEquals(member, result);  
 *assertThat*(member).isEqualTo(result);  
 }  
  
 @Test  
 public void fineByName(){  
 Member member1 = new Member();  
 member1.setName("spring1");  
 repository.save(member1);  
  
 Member member2 = new Member();  
 member2.setName("spring2");  
 repository.save(member2);  
  
 Member result = repository.findByName("spring1").get();  
  
 *assertThat*(result).isEqualTo(member1);  
 }  
 @Test  
 public void findAll() {  
 //given  
 Member member1 = new Member();  
 member1.setName("spring1");  
 repository.save(member1);  
 Member member2 = new Member();  
 member2.setName("spring2");  
 repository.save(member2);  
 //when  
 List<Member> result = repository.findAll();  
 //then  
 *assertThat*(result.size()).isEqualTo(2);  
 }  
}

회원 서비스 개발

package hellospring.hellospring.service;  
  
import hellospring.hellospring.domain.Member;  
import hellospring.hellospring.repository.MemberRepository;  
import hellospring.hellospring.repository.MemoryMemberRepository;  
  
import java.util.\*;  
  
public class MemberService {  
 private final MemberRepository memberRepository = new MemoryMemberRepository();  
  
 //회원가입  
 public long join(Member member) {  
 //같은 이름이 있는 중복 회원x  
 validateDuplicateMember(member);//중복회원검증  
 memberRepository.save(member);  
 return member.getId();  
 }  
  
 private void validateDuplicateMember(Member member) {  
 memberRepository.findByName(member.getName())  
 .ifPresent(m -> {  
 throw new IllegalStateException("이미 존재하는 회원입니다.");  
 });  
 }  
 //전체 회원 조회  
 public List<Member> fineMembers(){  
 return memberRepository.findAll();  
 }  
  
 public Optional<Member> findOne(Long memberId){  
 return memberRepository.findById((memberId));  
 }  
}

회원 서비스 테스트

기존 회원 서비스 코드 수정

private final MemberRepository memberRepository;  
public MemberService(MemberRepository memberRepository) {  
 this.memberRepository = memberRepository;  
}

Ctrl + Shift + T를 통하여 쉽게 Test파일을 생성할 수 있다.

package hellospring.hellospring.service;  
  
import hellospring.hellospring.domain.Member;  
import hellospring.hellospring.repository.MemoryMemberRepository;  
import org.assertj.core.api.Assertions;  
import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.util.Optional;  
  
import static org.assertj.core.api.Assertions.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class MemberServiceTest {  
  
 MemberService memberService;  
 MemoryMemberRepository memoryMemberRepository;  
  
 @BeforeEach  
 public void beforeEach() {  
 memoryMemberRepository = new MemoryMemberRepository();  
 memberService = new MemberService(memoryMemberRepository);  
 }  
  
 @AfterEach  
 public void afterEach() {  
 memoryMemberRepository.clearStore();  
 }  
  
 @Test  
 void 회원가입() {  
 //given  
 Member member = new Member();  
 member.setName("spring");  
  
 //when  
 Long saveId = memberService.join(member);  
 //than  
 Member findMember = memberService.findOne(saveId).get();  
 *assertThat*(member.getName()).isEqualTo(findMember.getName());  
 }  
  
 @Test  
 public void 중복\_회원\_예외() {  
 //given  
 Member member1 = new Member();  
 member1.setName("spring");  
  
 Member member2 = new Member();  
 member2.setName("spring");  
  
 //when  
 memberService.join(member1);  
 IllegalStateException e = *assertThrows*(IllegalStateException.class, () -> memberService.join(member2));  
  
 *assertThat*(e.getMessage()).isEqualTo("이미 존재하는 회원입니다.");  
  
// try {  
// memberService.join(member2);  
// fail();  
// } catch (IllegalStateException e) {  
// assertThat(e.getMessage()).isEqualTo("이미 존재하는 회원입니다.");  
// }  
 //than  
  
 }  
}