Spring – 자바의 프레임 워크

Intellij를 기반으로 작성.

Start.spring.io 에서 기본적인 프로젝트를 다운받는다.(spring boot)

스프링기반의 프로젝트를 만들어주는 사이트

Maven, gradle – 필요한 라이브러리를 가져와주고, 빌드 라이프? 사이클을 관리해주는 툴

과거에는 메이븐이었지만 요즘에는 그레이들을 선호

그룹명은 보통 기업 도메인. 우리는 아무거나 적어도 된다. 예제는 hello

Artifact는 빌드된 결과물(프로젝트명 같은것). Hello-spring

Dependencies는 어떠한 라이브러리를 사용할 것인지를 묻는 것.

우리는 web프로젝트 이므로 spring web을 선택한다.

또, thymeleaf라는 템플릿 엔진을 사용한다.

템플릿 엔진 - HTML 코드에서 고정적으로 사용되는 부분은 템플릿으로 만들어두고 동적으로 생성되는 부분만 템플릿 특정 장소에 끼워넣는 방식으로 동작할 수 있도록 해준다.

이후 generate하면 프로젝트를 다운받는다.

Build.grdadle을 프로젝트로 열기로 연다.

Main->java->hello-spring의 hellospringApplication.java의 main메소드를 빌드하면 8080포트로 서버가 열린다.

접속에 성공하면 기본적인 설정이 끝이난다.

Spring은 내장 tomcat이 있어서 자동으로 서버를 열게 된다.

intellij설정에서 빌드를 gradle이 아닌 intellij를 통해 빌드되도록 수정한다.

기본설정 끝

System.print.out 보다는 log출력을 할 것.

Src- main-java-hello-spring밑에 controller라는 패키지를 만든다.

패키지 안에 HelloController.java파일을 만들고

@Controller  
public class HelloController {  
 @GetMapping("hello")// /hello라는 호출이 들어오면 이하의 메소드 실행  
 public String hello(Model model){  
 model.addAttribute("data","hello");  
 return "hello";  
 }  
}

이렇게 수정한다.

여기서의 controller와 model은 MVC모델의 그 controller와 model이다.

GetMapping의 Get은 Get과 Post방식이다.

이 코드는 Get방식으로 /hello가 넘어오면 Model객체에 Data키에 hello라는 value를 생성하고

hello라는 템플릿 페이지를 return 한다. 해당 페이지는 아래에서 설명한다.

src – main – resources – templates에 hello.html파일을 만든다.

<!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>

여기서 th는 thymeleaf이며

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

는 타임리프 엔진이 템플릿 엔진이 선언되어 있다.

따라서 타임리프 문법을 사용할 수 있다.

p테그의 th:text는 값이 있다면 해당 내용으로 출력되지만 없다면 p태그의 내용이 출력된다.

빌드하기

인텔리j에 git bash를 터미널로 설정한 상태를 기준으로 설명한다.

프로젝트 폴더에서 ./gradlew build 입력

이후 cd build/libs 이동

Java -jar hello-spring-0.0.1-SNAPSHOT.jar 실행 확인

서버 배포시 해당 파일만 복사해서 서버에 넣어서 실행하면 된다.

웹 개발 방식

1. 정적 컨텐츠

서버에서 하는 것 없이 파일을 그대로 내려준다

1. MVC와 템플릿 엔진

서버상에서 동적으로 프로그래밍 해서 주는 것.

1. API

안드로이드나 아이폰 등의 개발시 json등의 데이터 형식으로 전달.

뷰,리엑트등이나 서버간 통신에서 사용

스프링은 정적 컨텐츠를 기본으로 지원한다. static폴더상의 파일에 대해서.

MVC

@GetMapping("hello-mvc")//get방식으로 hello-mvc접속  
public String helloMvc(@RequestParam("name") String name, Model model){//입력된 파라미터를 name키에 name value로 저장  
 model.addAttribute("name",name);//모델에 name키에 name 값으로 저장  
 return "hello-template";//리턴  
}

접속시 주소를 localhost:8080/hello-mvc?name=spring!으로 해보자.

template폴더의 hello-template.html 파일

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

받아온 name객체의 내용을 표시한다.

Api

@GetMapping("hello-string")//get방식으로 hello-mvc접속  
@ResponseBody //html이 아닌 http의 body에 직접 내용 주입  
public String helloString(@RequestParam("name") String name){//입력된 파라미터를 name키에 name value로 저장  
 return "hello" + name;//리턴  
}

이후 localhost:8080/hello-string?name=hello 로 접속하면

화면상에 hello hello 라고 출력되지만 소스보기를 누르면 아무런 html태그없이 글자만 있다는 것을 알 수 있다.

@GetMapping("hello-api")//get방식으로 hello-api접속  
@ResponseBody //html이 아닌 http의 body에 직접 내용 주입  
public Hello helloApi(@RequestParam("name") String name){//입력된 파라미터를 name키에 name value로 저장  
 Hello hello = new Hello();  
 hello.setName(name);  
 return hello;  
}  
  
static class Hello{//클래스 내부에서 사용할 수 있는 클래스 객체  
 private String name;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

스태틱 클래스로 클래스를 만들어 미리 만든 기능을 사용할 수 있다.

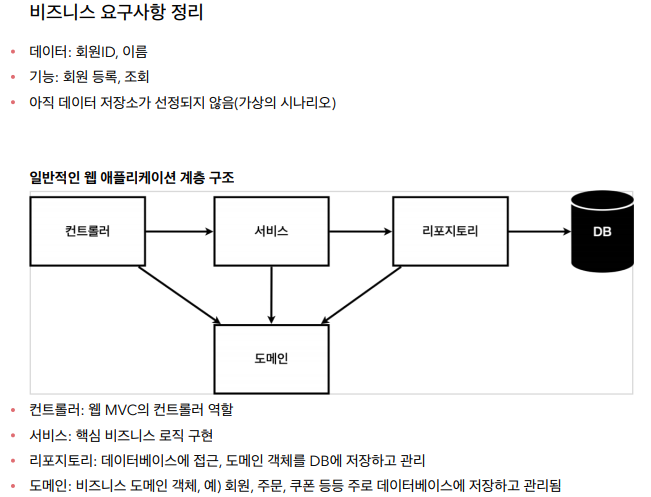
Localhost:8080/hello-api?name=tt로 접속할 경우

String name값이 Hello타입의 hello의 name에 저장된다.(set name메소드)

그리고 hello 객체를 리턴하면(클래스가 Hello이므로 Hello객체 리턴)

Json 형식으로 {name:tt}가 출력 되는 것을 볼 수 있다.

회원관리 예제



도메인 패키지를 만들어 회원정보를 관리한다.

리포지토리 패키지를 만들어 DB를 관리한다. 단 현 예제에서는 db를 결정하지 못하였다는 설정이므로 인터페이스로 기능을 설정한 후 우선 파일 형식으로 관리한다.

도메인

public class Member {  
 private Long id;//회원이 저장하는 아이디가 아닌 시스템에서 저장하는 아이디  
 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;  
 }  
}

Member.java 라는 멤버관리를 위한 파일이다.

아이디와 이름으로 구성되며, getter와 setter를 이용하여 값을 가져오고 지정한다.

리포지토리

public interface MemberRepository {  
 Member save(Member member);//저장기능 저장하면 저장된 회원 반환  
 Optional<Member> findById(Long id);  
 Optional<Member> findByName(String name);  
 List<Member> findAll();//모든 회원 리스트 반환  
}

MemberRepository라는 인터페이스를 만든다.

아직 db형식을 결정하기 전이므로 어떤 방식에서도 사용할 수 있도록 틀을 잡는다.

인터페이스에서는 자료형, 기능의 이름, 인자 등만을 지정한다.

Optional은 null 처리를 위한자료형이다.

우선 예제에서는 메모리(파일)로 저장한다.

public class MemoryMemberRepository implements MemberRepository{  
  
 private static Map<Long,Member> *store* = new HashMap<>();  
 private static long *sequence* = 0L;//키값을 생성  
  
 @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));//NULL이 반환되더라도 optional로 감싸서 반환 가능 추후처리 용의  
 }  
  
 @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());  
 }  
}

위의 인터페이스를 implemets하고 각 기능을 구현한다.

Map은 키와 value를 갖는 자료형으로 객체과 비슷하다고 생각한다.

Map형식 변수 store는 Key로 Long형식으로 구분하고, value로 Member형을 갖는다.

즉 id를 이용해서 회원을 관리한다고 볼 수 있다.

Long sequence는 간단하게 키값을 생성하기 위한 숫자형이고, long형임을 지정하기 위해 0L로 지정한다. 0으로 그냥 쓰면 int형으로 인식된다.

save기능에서 member를 인자로 받는다.

여기서 회원의 id는 회원이 직접 입력 하는 것이 아닌 시스템이 부여한다고 가정한다.

Sequence의 값을 증가시켜 해당 값을 id로 해당 회원에게 부여한다.

그 후 store변수에 해당 회원 정보를 추가한뒤 회원정보를 반환한다.

findById기능을 구현한다.

store변수에서 get메소드를 통해 입력받은 찾으려는 id를 찾아내어 value인 회원정보를 반환한다. 하지만 null일수 있기 때문에 Optional의 메소드인 ofNullable로 감싸 null이 반환이 가능하게 만든다.

findByName을 구현한다.

Key가 아닌 value를 기준으로 찾는 것이기 때문에 모든 회원정보를 확인해야 한다.

반복문 대신 store에 저장된 값들을 가져오는 메소드인 values를 이용하여 저장된 모든 Member객체들을 가져오고, stream메소드를 통해 전체를 loop한다.

Filter를 통해서 stream을 통해 순회하는 값을 member로 지정하고 -> 다음에 걸러내려는 기준을 적용한다. 해당 member의 이름을 가져와서 찾으려는 String name과 같은지 확인하기위해

-> member.getName().equals(name)을 사용한다. 이 값이 참인 어떠한 값이든 반환하기 위해 마지막에 .findAny()를 붙인다.

Store의 value인 member가 Optional로 감싸져서 반환된다.

Findall 기능을 구현한다

모든 회원정보를 반환하기 위해 List형식을 사용하고(자료형 Member)

반환은 간단하게 return new ArrayList<>(store.values()); 로 한다.

Values로 인해 store의 모든 value인 member들이 array에 추가되어 반환된다.