**CHAPTER1 스프링 개발 툴**

mavne이나 gradle(<https://gradle.org/releases/>) 을 이용하여 환경구성

기존 manve project에서 gradle로 변경 하려면 해당 명령어 사용하면 된다.

**gradle init --type pom**

**CHAPTER4 스프링 REST**

-REST는 HTTP를 기본으로 웹 서비스를 구현하는 아키텍처로 각광받고 있다.

**레시피 4-1 REST 서비스로 XML 발행하기**

-스프링 mvc에서 애플리케이션 데이터를 REST 서비스로 발행하는 주역은 @RequestMapping과 @PathVariable 두 애너테이션이다.

**MarshallingVeiw로 XML 만들기**

@RequestMapping("/members")  
public String getRestMembers(Model model) {  
 Members members = new Members();  
 members.addMembers(memberService.findAll());  
 model.addAttribute("members", members);  
 return "membertemplate";  
}

🡺host\_name/애플리케이션명/members URL로 접근 가능

@Bean  
public View membertemplate() {  
 return new MarshallingView(jaxb2Marshaller());  
}

🡺위의 뷰는 MarshallingView 형으로 정의 (마샬러를 이용해 렌더링 하는 범용 클래스)

-마샬링은 메모리에 있는 객체를 특정한 데이터 형식으로 변환하는 과정

@Bean  
public Marshaller jaxb2Marshaller() {  
 Jaxb2Marshaller marshaller = new Jaxb2Marshaller();  
 marshaller.setClassesToBeBound(Members.class, Member.class);  
 return marshaller;  
}

🡺classesToBeBound는 XML로 변환할 대상 클래스

@XmlRootElement  
public class Member {  
 private String name;  
 private String phone;  
 private String email;

}

🡺@XmlRootElement는 Jaxb2Marshaller가 클래스 필드를 자동으로 감지해서 XML 데이터로 변경

**콘텐트 협상**

-http://[호스트명]/[애플리케이션명]/memebrs 처럼 확장 없이 URL을 요청할 경우 MVC 뷰의 해석 로직에 달려 있다. 이런 경우 URL 확장자 또는 HTTP 헤더를 보고 뷰를 결정한다.

ex) Accept: application/xml

**@ResponseBody로 XML 만들기**

-여러 뷰로 보여주기 위해 뷰를 하나씩 추가하는건 너무 번거롭기 때문에 이런 경우 HttpMessageConverter를 이용하면 된다.

@RequestMapping("/members")  
@ResponseBody  
public Members getRestMembers(Model model) {  
 Members members = new Members();  
 members.addMembers(memberService.findAll());  
 return members;  
}

🡺@ResponseBody를 사용해서 응답 본문으로 취급

**결 과**

<http://localhost:8080/members.xml>

<members>  
 <member>  
 <email>marten@deinum.biz</email>  
 <name>Marten Deinum</name>  
 <phone>00-31-1234567890</phone>  
 </member>  
 <member>  
 <email>john@doe.com</email>  
 <name>John Doe</name>  
 <phone>1-800-800-800</phone>  
 </member>  
 <member>  
 <email>jane@doe.com</email>  
 <name>Jane Doe</name>  
 <phone>1-801-802-803</phone>  
 </member>  
</members>

<http://localhost:8080/members>

marten@deinum.bizMarten Deinum00-31-1234567890john@doe.comJohn Doe1-800-800-800jane@doe.comJane Doe1-801-802-803

**@PathVariable로 결과 거르기**

@RequestMapping("/member/{memberid}")  
@ResponseBody  
public Member getMember(@PathVariable("memberid") long memberID) {  
 return memberService.find(memberID);  
}

🡺@PathVariable을 이용해서 매개변수 사용하기

@RequestMapping("/member/\*/{memberid}")

🡺와일드카드 사용할 경우 /member/심준보/1 이런 url 호출도 가능

**ResponseEntity로 클라이언트에게 알려주기**

@RequestMapping("/member/{memberid}")  
@ResponseBody  
public ResponseEntity<Member> getMember(@PathVariable("memberid") long memberID) {  
 Member member = memberService.find(memberID);  
 if (member != null) {  
 return new ResponseEntity<Member>(member, HttpStatus.*OK*);  
 }  
 return new ResponseEntity(HttpStatus.*NOT\_FOUND*);  
}

🡺ResponseEntity는 결과 본문을 HTTP 상태 코드와 함께 집어넣은 래퍼 클래스이다.

**레시피 4-2 REST 서비스로 JSON 발행하기**

**JSON 형식으로 처리하는 이유**

🡺어느 브라우저든 자바스크립트 언어 해석기는 다 장착되어 있기 때문에

**MappingJackson2JsonView로 JSON 만들기**

@RequestMapping("/members")  
public String getRestMembers(Model model) {  
 Members members = new Members();  
 members.addMembers(memberService.findAll());  
 model.addAttribute("members", members);  
 return "jsonmembertemplate";  
}

@Bean  
public View jsonmembertemplate() {  
 MappingJackson2JsonView view = new MappingJackson2JsonView();  
 view.setPrettyPrint(true);  
 return view;  
}

🡺MappingJackson2JsonView 뷰는 잭슨2 라이브러리를 이용해 객체를 JSON으로 바꾼다. (내부적으로는 ObjectMapper 사용)

**Xml, Json 별도 뷰**

@RequestMapping(value = "/members", produces = MediaType.*APPLICATION\_XML\_VALUE*)  
public String getRestMembersByXml(Model model) {  
 Members members = new Members();  
 members.addMembers(memberService.findAll());  
 model.addAttribute("members", members);  
 return "xmlmembertemplate";  
}  
  
@RequestMapping(value = "/members", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
public String getRestMembersByJson(Model model) {  
 Members members = new Members();  
 members.addMembers(memberService.findAll());  
 model.addAttribute("members", members);  
 return "jsonmembertemplate";  
}

🡺하지만 이런 방식은 뷰 별로 중복이 되버린다.

**/members.xml**



**/members.json**



**@ResponseBody로 Json 만들기**

@RequestMapping(value = "/members")  
@ResponseBody  
public Members getRestMembersByXml(Model model) {  
 Members members = new Members();  
 members.addMembers(memberService.findAll());  
 return members;  
}

@Configuration  
@EnableWebMvc  
@ComponentScan(basePackages = "chapter4.part7")  
public class CourtRestConfiguration {  
  
}

위 **@ResponseBody로 XML 만들기** 와 동일

**왜 이런일들이 가능할까?**

-스프링 MVC가 클래스패스에 있는 것을 자동 감지하기 때문에…

-JAXB2, 잭슨 등의 라이브러리가 발견되면 해당 기술에 적합한 HttpMessageConvert를 알아서 등록

**GSON으로 JSON만들기**

compile group: 'com.google.code.gson', name: 'gson', version: '2.8.0'

Jackson 대신 해당 라이브러리 추가

**레시피 4-3 스프링으로 REST 서비스 액세스하기**

-페이로드 : 사용에 있어서 전송되는 데이터를 뜻한다. (헤더와 메타데이터 제외)

-페이로드를 쉽게 찾아 쓰게 할 목적으로 WADL(Web Application Description Language)를 제공하는 REST 서비스도 있다.

public static void main(String[] args) {  
 final String uri = "http://localhost:8080/members.json";  
 RestTemplate restTemplate = new RestTemplate();  
 String result = restTemplate.getForObject(uri, String.class);  
 System.*out*.println(result);  
}

🡺RestTeamplte 을 사용해서 서비스에 액세스 했다.

**매개변수화한 URL에서 데이터 가져오기**

private static void placeholders() {  
 final String uri = "http://localhost:8080/member/{memberId}";  
 Map<String, String> params = new HashMap<>();  
 params.put("memberId", "1");  
  
 RestTemplate restTemplate = new RestTemplate();  
 String result = restTemplate.getForObject(uri, String.class, params);  
 System.*out*.println(result);  
}

🡺HashMap을 이용하여 getForObject 메서드에 넣어서 호출한다.

**데이터를 매핑된 객체로 가져오기**

final String uri = "http://localhost:8080/members.xml";  
RestTemplate restTemplate = new RestTemplate();  
Members result = restTemplate.getForObject(uri, Members.class);

🡺결과를 Members.class로 지정

**레시피 4-4 RSS/아톰 피드 발행하기**

-RSS/아톰 피드는 정보를 발행하는데 널리 쓰이는 수단이다.

**RSS/아톰 공통적 특징**

-피드 내용을 서술하는 메타데이터 영역이 있다. (아톰 <authr>와 <title>, RSS의 <description>과 <pubDate>)

-순환 엘리먼트로 복수의 정보를 나타낼 수 있다.(dㅏ톰 <entity>, RSS의 <item>)

-버전이 다양하다.

**Atom**

public class AtomFeedView extends AbstractAtomFeedView {  
  
 @Override  
 protected void buildFeedMetadata(Map<String, Object> model, Feed feed, HttpServletRequest request) {

}  
  
 @Override  
 protected List<Entry> buildFeedEntries(Map<String, Object> model, HttpServletRequest request, HttpServletResponse response) {

}

🡺buildFeedMetadata() 메서드는 피드 데이터가 담긴 Map 객체(핸들러 메서드에 할당된 데이터)

, 피드를 처리하는데 필요한 ROME의 Feed, HTTP 요청 시 필요한 HttpServletRequest

🡺buildFeedEntries() 메서드는 피드 데이터가 담긴 Map 객체, HTTP 요청을 다루어야 할 때 필요한 HttpServletRequest, HTTP 응답을 다루어야 할 때 필요한 HttpServletResponse

**/atomfeed.atom**



**RSS**

public class RSSFeedView extends AbstractRssFeedView {  
  
 @Override  
 protected void buildFeedMetadata(Map<String, Object> model, Channel feed, HttpServletRequest request) {  
  
 }  
  
 @Override  
 protected List<Item> buildFeedItems(Map<String, Object> model, HttpServletRequest request, HttpServletResponse response)  
 throws Exception {  
  
 }  
}

🡺buildFeedMetadata() 메서드는 Channel 객체를 사용해서 RSS 피드를 만든다.

🡺buildFeedItems()는 Atom의 **buildFeedEntries**와 용도가 같다.

**/rssfeed.rss**

