**@RequestMapping**

* 요청 주소에 따라 이를 처리할 컨트롤러 또는 컨트롤러 메서드를 연결하는 어노테이션
* 클래스에 작성 시 : 메서드에 작성될 @RequestMapping의 공통 주소
* 메서드에 작성 시 : 요청을 처리할 메서드 연결
* 속성
  + value : 주소(URL)
  + method : 요청 방식( RequestMethod.GET/POST)
  + 메서드 레벨에서 속성을 아무것도 작성하지 않고 주소(URL)만 작성하는 경우  
    -> method 관계없이 주소만 일치하면 매핑

**@PostMapping**

* @RequestMapping의 POST 버전 (POST 요청만 매핑 처리)

**@GetMapping**

* @RequestMapping의 GET 버전 (GET 요청만 매핑 처리)

**ViewResolver(응답 화면 해결사)**

* 컨트롤러의 처리 결과에 따라 보여질 응답 화면을 결정하는 역할
* src/main/resources/spring/appServlet/**servlet-context.xml** 파일에 작성됨
* forward를 쉽게 작성할 수 있도록 하는 역할
* prefix(접두사) 속성 : JSP 파일이 모여 있는 위치의 공통 경로 ( /WEB-INF/views/ )
* suffix(접미사) 속성 : 확장자명 (.jsp)
* Controller에서 반환된 문자열(String) 값 앞에 prefix, 뒤에 suffix 를 붙여서  
  만들어진 경로에 위치한 jsp 파일에 forward를 진행

**@Controller**

* 요청에 따라 알맞은 클래스, 메서드와 매핑하고  
  메서드 처리 결과에 따라 알맞은 응답 화면으로 forward, redirect하는 역할
* 컨트롤러(요청/응답 제어) 역할임을 명시  
  + Bean 등록

**Bean (객체(instance)와 비슷한 의미)**

* Spring의 특징 중 IOC(제어 역전)에 의해서  
  개발자가 직접 생성해서 관리하는 객체가 아닌  
  **Spring Container(== Spring Framework)가 생성하고 관리하는 객체**

**@RequestParam(value="name", required="fasle", defaultValue="1")**

[속성]

value : 전달 받은 input 태그의 name 속성값

required : 입력된 name 속성값 파라미터 필수 여부 지정(기본값 true)

-> required = true인 파라미터가 존재하지 않는다면 400 Bad Request 에러 발생

-> required = true인 파라미터가 null인 경우에도 400 Bad Request

defaultValue : 파라미터 중 일치하는 name 속성 값이 없을 경우에 대입할 값 지정.

-> required = false인 경우 사용

**web.xml**

* 배포 서술자(Deployment Descriptor)
* 배포 시 수행할 구문, 설정을 작성하는 문서
* 서버 실행 시 **가장 먼저** 읽는 파일!!

**root-context.xml**

* 프로젝트 전반적으로 사용할   
  DB 연결 관련 설정, 트랜잭션 처리, 파일 업로드, AOP 관련 내용을 작성하는 파일

**@Autowired(자동 연결)**

* 등록된 Bean 중에서 아래 작성된 필드와 타입이 일치하거나  
  상속 관계에 있는 Bean을 얻어와 필드에 주입하는 어노테이션
* 필드에 주입되는 객체를 직접 만드는 것이 아닌  
  스프링이 만들고 관리하는 객체를 주입 받는 것 **== 의존성 주입(DI)**

**Model 객체**

* Spring에서 데이터를 전달하는 목적으로 사용하는 객체
* 기본 scope == request scope
* Model.addAttribute(“key”, value); 구문으로 값 세팅 후 전달

**@SessionAttributes**

* 매개변수로 작성된 문자열과 Model에 추가된 속성의 key값 같으면  
  해당 Model 속성의 scope를 Session으로 변경
* 클래스 레벨로 작성(클래스 위에 작성)

**암호화**

* 문자열, 파일 등을 알아보기 힘들게 바꾸는 것
* 방법  
  1) 암호화 – 복호화  
  2) 복호화가 불가능한 암호화  
  3) 같은 문자열이라도 다른 결과가 나타나는 암호화

**BCrypt 암호화 방식**

* 입력된 문자열에 랜덤한 값(salt)을 더해서 암호화를 진행  
  -> 암호화 결과가 항상 다름
* 단순 비교 방법을 사용할 수 없고  
  BCrypt에서 제공하는 matches() 메서드 이용해야지만 비교 가능
* Spring에서 사용 시 pom.xml에서 아래 3가지를 추가
  + spring-security-core
  + spring-security-web
  + spring-security-config

**UncheckedException의 특징**

* 예외처리를 강요하지 않음
* 예외가 발생한 경우 호출부로 예외를 던짐(throws)

**스프링에서 트랜잭션을 처리하는 방법(코드기반, 선언적)**

1) <tx:advice> -> AOP를 이용한 방식(XML에 작성)

2) @Transactional 어노테이션을 이용한 방식(클래스 또는 인터페이스에 작성)

- 인터페이스를 구현한 클래스로 선언된 빈은 인터페이스 메소드에 한해서 트랜잭션이 적용됨

\* 트랜잭션 처리를 위해서는 트랜잭션 매니저가 bean으로 등록되어 있어야 함.

-> root-context.xml 작성

+ serlvelt-context.xml에 <tx:annotation-driven/> 추가

- 정상 여부는 RuntimeException이 발생했는지 기준으로 결정되며,

RuntimeException 외 다른 Exception(대표적으로 SQLException 등)에도 트랜잭션 롤백처리를 적용하고 싶으면

@Transactional의 rollbackFor 속성을 활용하면 된다