11강 MVC 2

<spring:message> 태그로 메시지 출력하기

- 사용자 화면에 보여질 문자열은 보통 JSP 코드에 직접 넣는 것이 일반적이다.
- 그러나 문자열을 변경하고 싶을 때, 또는 다국어 지원이 필요할 때 해당 문자열을 언어에 알맞게 변경해 표시 해주어야 하는데 이때 하드 코딩이 되어있다면 언어별로 별도의 뷰를 만들어야 하는 상황도 생길수 있다.

- 이런 상황을 해결하기 위한 가장 좋은 방법은 뷰 코드에서 사용할 문자열을 별도의 파일로 보관하고 언어에 따라 알맞은 문자열을 읽어와서 출력하는 것이다.

- 스프링은 자체적으로 이런 기능을 제공한다.

<spring:message> 태그로 메시지 출력하기

- 문자열을 별도의 파일에 작성하고 JSP 코드에서 이를 사용하려면 다음의 작업을 한다.

  - 메시지 파일을 작성(.properties)
  - 메시지 파일에서 값을 읽어오는 MessageSource 빈 을 설정
  - JSP 코드에서 <spring:message>태그를 사용해서 메시지를 출력

<spring:message> 태그로 메시지 출력하기

- 메시지 파일을 작성(.properties) : label.properties

```
member.register=회원가입

term=약관
term.content=약관 내용
term.agree=약관 동의
next.btn=다음단계

member.info=회원 정보
email=이메일
name=이름
password=비밀번호
password.confirm=비밀번호 확인
register.btn=가입 완료

register.done=<strong>{0}님</strong> 회원 가입을 완료했습니다.

go.main=메인으로 이동
```

<spring:message> 태그로 메시지 출력하기

- 메시지 파일에서 값을 읽어오는 MessageSource 빈 을 설정 : spring-mvc.xml에 추가

```
<bean id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>classpath:message/label</value>
            <!-- <value>message.second</value>   -->
        </list>
    </property>
    <property name="defaultEncoding" value="UTF-8"/>
</bean>
```

<spring:message> 태그로 메시지 출력하기

- JSP 코드에서 <spring:message>태그를 사용해서 메시지를 출력
  - Step1,2,3모두 바꿔보자 : step1.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title><spring:message code="member.register"/></title>
</head>
<body>
    <h2><spring:message code="term"/></h2>
    <p><spring:message code="term.content"/></p>
    <form action="step2" method="POST">  <!-- register/step1 -->
        <label>
            <input type="checkbox" name="agree" value="true">
            <spring:message code="term.agree"/>
        </label>
        <input type="submit" value="<spring:message code="next.btn"/>">
    </form>
</body>
</html>
```

<spring:message> 태그로 메시지 출력하기

- JSP 코드에서 <spring:message>태그를 사용해서 메시지를 출력
  - Step1,2,3모두 바꿔보자 : step2.jsp

```
<h2><spring:message code="member.info"/></h2>
    <form:form action="step3" commandName="formData">
        <p>
            <label><spring:message code="email"/> : <br>
                <form:input path="email" />
            </label>
        </p>
        <p>
            <label><spring:message code="name"/> : <br>
                <form:input path="name" />

            </label>
        </p>
        <p>
            <label><spring:message code="password"/> : <br>
                <form:password path="password" />
            </label>
        </p>
        <p>
            <label><spring:message code="password.confirm"/> : <br>
                <form:password path="confirmPassword" />
            </label>
        </p>
        <input type="submit" value="<spring:message code="register.btn"/>">
    </form:form>
```

<spring:message> 태그로 메시지 출력하기

- JSP 코드에서 <spring:message>태그를 사용해서 메시지를 출력
  - Step1,2,3모두 바꿔보자 : step3.jsp

```
<p><%-- <strong>${formData.name}님</strong>
     회원 가입을 완료했습니다. --%>
     <spring:message code="register.done" arguments="${formData.name}"/>
</p>
<p><a href="<c:url value='/main'/>">[<spring:message code="go.main"/>]</a></p>
```

커맨드 객체의 값 검증과 에러 메시지 처리

- 회원 가입처리 부분의 코드에서 정상적이지 않은 값을 입력해도 정상적으로 동작하는 문제가 있었다.

- 올바르지 않은 이메일을 입력하거나, 이름을 생략해도 가입이 가능하다.

⇒입력 값 검증이 없다.

- 또는 중복된 이메일 주소를 입력해서 가입이 실패 한 경우 사용자에게 가입 실패의 원인을 알려주지 않아 사용자는 혼란을 겪게 된다.

=> 에러 메시지 처리

- 스프링에는 두가지 문제를 처리하기 위해 다음과 같은 방법을 제공한다.

  - 커맨드 객체를 검증하고 결과를 에러 코드로 저장
  - JSP에서 에러 코드로부터 메시지 출력

커맨드 객체의 값 검증과 에러 메시지 처리

- 커맨드 객체 검증과 에러 코드 지정하기
- 커맨드 객체의 값이 올바른지 검사하려면 다음 두 인터페이스를 사용한다.
  - org.springframework.validation.Validator
  - org.springframework.validation.Errors

```
public interface Validator{
        Boolean supports(Class<?>   clazz);
        void validate(Object target, Errors errors);
}
```

  - support 메서드는 Validator가 검증할 수 있는 타입인자인지 검사 목적으로 사용
  - validate 메서드는 첫 번째로 받은 객체를 검증하고 그 결과를 Errors에 담는 기능을 정의한다.

  - 예제2

  - Validator 구현 클래스를 제작한다.
  - 커맨드 객체를 검증하도록 controller 클래스를 수정한다.

커맨드 객체의 값 검증과 에러 메시지 처리

- 예제2-1 : spring.validator.RegisterRequestValidator.java

```java
public class RegisterRequestValidator implements Validator{

    @Override
    public boolean supports(Class<?> clazz) {
        return RegisterRequest.class.isAssignableFrom(clazz);
    }
    //파라미터로 전달받은 객체가 RegisterRequest로 변환 가능한지 확인
    //  스프링에서 자동 검사 진행

    private static final String emailExp =
            "^[0-9a-zA-Z]([-_.]?[0-9a-zA-Z])*@[0-9a-zA-Z]([-_.]?[0-9a-zA-Z])*.[a-zA-Z]{2,3}$";

    private Pattern pattern;

    public RegisterRequestValidator() {
        pattern = Pattern.compile(emailExp);
    }
```

커맨드 객체의 값 검증과 에러 메시지 처리

- 예제2-1 : spring.validator.RegisterRequestValidator.java

```java
@Override
public void validate(Object target, Errors errors) {

    RegisterRequest regReq = (RegisterRequest)target;

    if(regReq.getEmail()==null || regReq.getEmail().trim().isEmpty()) {
        errors.rejectValue("email","required");
    }else {
        Matcher matcher = pattern.matcher(regReq.getEmail());
        if(!matcher.matches()) {
            errors.rejectValue("email", "bad");
        }
    }

    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "required");
    ValidationUtils.rejectIfEmpty(errors, "password", "required");
    ValidationUtils.rejectIfEmpty(errors, "confirmPassword", "required");

    if(!regReq.getPassword().isEmpty()) {
        if(!regReq.isPasswordEqualToConfirmPassword()) {
            errors.rejectValue("confirmPassword", "nomatch");
        }
    }
}
```

커맨드 객체의 값 검증과 에러 메시지 처리

- 예제2-2 : RegisterController.java 변경

```
@RequestMapping(value="/register/step3", method=RequestMethod.POST)    //  =>   /register/step3
public String handlerStep3(@ModelAttribute("formData") RegisterRequest regReq,Errors errors) {

    new RegisterRequestValidator().validate(regReq, errors);

    if(errors.hasErrors()) {
        return "register/step2";
    }

    try {
        memberRegisterService.regist(regReq);
        return "register/step3";
    }catch(AlreadyExistingMemberException e) {
        errors.rejectValue("email", "duplicate");
        return "register/step2";
    }

}
```

커맨드 객체의 값 검증과 에러 메시지 처리

- 커맨드 객체의 특정 프로퍼티가 아닌 커맨드 객체 자체가 잘못되었을 경우 커맨드 객체 자체에 에러 코드를 추가한다.

```
try{
...인증 처리 코드
}catch(IdPasswordNotMatchingException e){
    errors.reject("notMatchingIdPassword");
    return "login/loginForm";
}
```

- Errors타입의 파라미터를 추가할때 주의 점은 반드시 커맨드객체 파라미터 다음에 와야 한다는 것이다. 그렇지 않으면 예외가 발생한다.

```
@RequestMapping(value = "/register/step3", method = RequestMethod.POST)
public String handleStep3(Errors errors, RegisterRequest regReq) {
//                              커맨드 객체 앞에 Errors 파라미터가 들어오면 예외
```

커맨드 객체의 값 검증과 에러 메시지 처리

- Errors인터페이스 주요 메서드

reject(String errorcode)
reject(String errorcode, String defaultMessage)
reject(String errorcode, Object[] errorArgs, String defaultMessage)
rejectValue(String field, String errorCode)
rejectValue(String field, String errorCode, String defaultMessage)
rejectValue(String field, String errorCode, Object[] errorArgs, String defaultMessage)

- 에러 코드에 해당하는 메시지가 {0},{1}과 같이 인덱스 기반 변수를 포함하는 있는 Object 배열 타입의 errorArgs 파라미터를 이용해서 변수에 삽입될 값을 전달한다.

- 에러 코드에 해당하는 메시지가 존재하지 않을 때 예외를 발생 시키는 대신 defaultMessage를 출력한다.

커맨드 객체의 값 검증과 에러 메시지 처리

- ValidationUtils 클래스는 RejectIfEmpty()메서드와 rejectIfEmptyOrWhitespace()메서드를 제공한다.

rejectIfEmpty(Errors errors, String field, String errorCode)
rejectIfEmpty(Errors errors, String field, String errorCode, Object[] errorArgs)
rejectIfEmptyOrWhitespace(Errors errors, String field, String errorCode)
rejectIfEmptyOrWhitespace(Errors errors, String field, String errorCode, Object[] errorArgs)

- rejectIfEmpty메서드의 field에 해당하는 프로퍼티의 값이 null이거나 빈 문자열("")인 경우 에러 코드로errorCode 를 추가한다.
- rejectIfEmptyOrWhitespace null이거나 빈 문자열인 경우 공백문자로 값이 구성된 경우 에러 코드를 추가한다.

커맨드 객체의 값 검증과 에러 메시지 처리

- 커맨드 객체의 에러 메시지 출력하기
- 에러 코드를 지정하는 이유는 에러메시지를 출력하기 위함

- JSP에서는 스프링이 제공하는 <form:errors>태그를 사용해서 에러 메시지를 출력할 수 있다.

커맨드 객체의 값 검증과 에러 메시지 처리

- <form:errors>태그의 주요 속성

- element - 각 에러 메시지를 출력할 때 사용될 HTML태그, 기본값은 span
- delimiter - 각 에러 메시지를 구분할 때 사용될 HTML 태그 기본값은 br이다.
- path 속성을 지정하지 않으면 글로벌 에러에 대한 메시지 출력

```
<form:errors path="userid" element="div" delimeter="" />
```

커맨드 객체의 값 검증과 에러 메시지 처리

- 예제3 : 에러 표시를 위한 step2.jsp파일 수정

```
<h2><spring:message code="member.info"/></h2>
    <form:form action="step3" commandName="formData">
        <p>
            <label><spring:message code="email"/> : <br>
                <form:input path="email" />
                <%-- <input type="text" name="email" id="email" value="${formData.email}"> --%>
                <form:errors path="email"/>
            </label>
        </p>
        <p>
            <label><spring:message code="name"/> : <br>
                <form:input path="name" />
                <form:errors path="name"/>
            </label>
        </p>
        <p>
            <label><spring:message code="password"/> : <br>
                <form:password path="password" />
                <form:errors path="password"/>
            </label>
        </p>
        <p>
            <label><spring:message code="password.confirm"/> : <br>
                <form:password path="confirmPassword" />
                <form:errors path="confirmPassword"/>
            </label>
        </p>
        <input type="submit" value="<spring:message code="register.btn"/>">
    </form:form>
```

커맨드 객체의 값 검증과 에러 메시지 처리

- 에러코드에 해당하는 메시지 코드를 찾을 때 다음의 규칙을 따른다.
  - 에러코드.커맨드객체이름.필드명
  - 에러코드.필드명
  - 에러코드.필드타입
  - 에러코드

- 프로퍼티 타입이 list나 목록인 경우 다음의 순서를 사용해서 메시지 코드를 생성한다.
  - 에러코드.커맨드객체이름.필드명[인덱스].중첩필드명
  - 에러코드.커맨드객체이름.필드명.중첩필드명
  - 에러코드.필드명[인덱스].중첩필드명
  - 에러코드.필드명.중첩필드명
  - 에러코드.중첩필드명
  - 에러코드.필드타입
  - 에러코드

커맨드 객체의 값 검증과 에러 메시지 처리

- 예시

 - errors.rejectValue("email","required")코드로 "email"프로퍼티에 "required"에러 코드 추가 , 커맨드 객체의 이름이 "requestRequest"라면 다음 순서로 메시지코드를 검색한다.

 - required.requestRequest.email
 - required.email
 - required.String
 - required

- 에러 코드에 맞는 메시지를 추가해준다.

커맨드 객체의 값 검증과 에러 메시지 처리

- 예제4 : 에러 표시를 위한 properties파일 수정

```
required=필수 항목입니다.
required.email=이메일은 필수항목입니다.
bad.email=이메일이 올바르지 않습니다.
duplicate.email=중복된 이메일입니다
nomatch.confirmPassword=비밀번호와 확인이 일치하지 않습니다.
```

로그인 처리 위한 코드 준비

- 예제 5

- vo
  - AuthInfo
  - LoginCommand
- service
  - AuthService
- validator
  - LoginCommandValidator
- controllor
  - LoginController
- jsp
  - loginForm
  - loginSuccess

-Properties

-xml

로그인 처리 위한 코드 준비

- 예제 5 - vo
  - AuthInfo.java

```java
public class AuthInfo {

    private Long id;
    private String email;
    private String name;

    public AuthInfo(Long id, String email, String name) {
        this.id = id;
        this.email = email;
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public String getEmail() {
        return email;
    }

    public String getName() {
        return name;
    }
}
```

로그인 처리 위한 코드 준비

- 예제 5 - vo
  - LoginCommand.java

```java
public class LoginCommand {

    private String email;
    private String password;
    private boolean rememberEmail;

    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public boolean isRememberEmail() {
        return rememberEmail;
    }
    public void setRememberEmail(boolean rememberEmail) {
        this.rememberEmail = rememberEmail;
    }
```

로그인 처리 위한 코드 준비

- 예제 5 - service
  - AuthService.java

```java
public class AuthService {

    private MemberDao memberDao;

    public void setMemberDao(MemberDao memberDao) {
        this.memberDao = memberDao;
    }

    public AuthInfo authenticate(String email, String password) {
        Member member = memberDao.selectByEmail(email);

        if(member==null) {
            throw new IdPasswordNotMatchingException();
        }
        if(!member.getPassword().equals(password)) {
//      if(!member.matchPassword(password)) {
            throw new IdPasswordNotMatchingException();
        }

        return new AuthInfo(member.getId(), member.getEmail(), member.getName());
    }
}
```

로그인 처리 위한 코드 준비

- 예제 5 - validator
  - LoginCommandValidator.java

```java
public class LoginCommandValidator implements Validator{

    @Override
    public boolean supports(Class<?> clazz) {
        // TODO Auto-generated method stub
        return LoginCommand.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "email", "required");
        ValidationUtils.rejectIfEmpty(errors, "password", "required");
    }
```

로그인 처리 위한 코드 준비

- 예제 5 - controllor
  - LoginController.java

```java
@Controller
@RequestMapping("/login")
public class LoginController {

    private AuthService authService;

    public void setAuthService(AuthService authService) {//스프링으로부터 주입
        this.authService = authService;
    }

    @RequestMapping(method=RequestMethod.GET)
    public String form(Model model){
        model.addAttribute("loginCommand", new LoginCommand());
        return "login/loginForm";
    }
```

로그인 처리 위한 코드 준비

- 예제 5 - controllor
  - LoginController.java

```java
@RequestMapping(method=RequestMethod.POST)
public String submit(LoginCommand loginCommand, Errors errors) {
    //1. 아이디, 비밀번호가 입력이 되기는 했는지 검증
    new LoginCommandValidator().validate(loginCommand, errors);

    if(errors.hasErrors()) {// 로그인 실패
        return "login/loginForm";
    }
    //2. 입력받은 이메일이 DB에 저장이 되었는가? /DB에서 가져온 비밀번호와 일치하는가?

    try {
        AuthInfo authInfo = authService.authenticate(
                loginCommand.getEmail(),
                loginCommand.getPassword());

        //로그인 성공시 처리할 코드

        return "login/loginSuccess";
    }catch(IdPasswordNotMatchingException e) {
        // 이메일이 없거나 또는 비밀번호가 틀렸거나
        errors.reject("idPasswordNotMatching");
        return "login/loginForm";
    }
}
```

로그인 처리 위한 코드 준비

- 예제 5 - jsp
  - loginForm.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"  %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"  %>
```

```
<title><spring:message code="login.title"/></title>
</head>
<body>
    <form:form commandName="loginCommand">
    <form:errors />
    <p>
        <label><spring:message code="email"/>:<br>
            <form:input path="email"/>
            <form:errors path="email"/>
        </label>
    </p>
    <p>
        <label><spring:message code="password"/>:<br>
            <form:password path="password"/>
            <form:errors path="password"/>
        </label>
    </p>

    <input type="submit" value="<spring:message code="login.btn"/>" />
    </form:form>
</body>
```

로그인 처리 위한 코드 준비

- 예제 5 - jsp
  - loginSuccess.jsp

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

```
<title><spring:message code="login.title"/></title>
</head>
<body>
    <p>
        <spring:message code="login.done"/>
    </p>
    <p>
        <a href="<c:url value='/main'/>">
            [<spring:message code="go.main"/>]
        </a>
    </p>
</body>
```

로그인 처리 위한 코드 준비

- 예제 5 - Properties
  - label.properties 추가

```
login.title=로그인
login.btn=로그인하기
idPasswordNotMatching=아이디와 비밀번호가 일치하지 않습니다.
login.done=로그인 성공
```

로그인 처리 위한 코드 준비

- 예제 5 - XML설정 등록
  - spring-member.xml 추가

```
<bean id="authSvc" class="spring.service.AuthService">
    <property name="memberDao" ref="memberDao"/>
</bean>
```

  - spring-controller.xml 추가

```
<bean class="spring.controller.LoginController">
    <property name="authService" ref="authSvc"/>
</bean>
```

로그인 처리 위한 코드 준비

- 예제 5 – Main.jsp 다음 링크 추가

```
<p>환영합니다</p>
<p><a href="<c:url value='/register/step1'/>">[회원 가입하기]</a></p>
<p><a href="<c:url value='/login'/>">[로그인]</a></p>
```

GET과 POST 방식에 동일한 커맨드 객체 사용하기

- 웹 개발을 할 때 일반적으로 Get요청시 폼을 보여주고, Post 요청시 폼 전송을 처리하는 경우가 많다.

- 이때 입력폼과 폼 전송 처리에서 동일한 커맨드 객체를 사용하려면 다음과 같이 처리한다.

```
@Controller
@RequestMapping("/login")
public class LoginController {

    @RequestMapping(method = RequestMethod.GET)
    public String form(Model model){
            model.addAttribute("loginCommand", new LoginCommand());
            return "login/loginForm";
    }
    @RequestMapping(method = RequestMethod.POST)
    public String submit(LoginCommand loginCommand, Errors errors){
            ……
    }
```

GET과 POST 방식에 동일한 커맨드 객체 사용하기

- form()메서드 에서 직접 커맨드 객체를 생성하고 submit()메서드는 스프링 MVC가 생성한 커맨드 객체를 전달 받는다.
- form() 메서드도 커맨드 객체를 전달 받도록 구현하면 보다 간결하게 form()메서드를 구현 할 수 있다

```
@Controller
@RequestMapping("/login")
public class LoginController {

	@RequestMapping(method = RequestMethod.GET)
	public String form(LoginCommand loginCommand){
		return "login/loginForm";
	}
	@RequestMapping(method = RequestMethod.POST)
	public String submit(LoginCommand loginCommand, Errors errors){
		.....
	}
```

로그인 처리 위한 코드 준비

- 예제 6 - controllor
  - LoginController.java

```java
@Controller
@RequestMapping("/login")
public class LoginController {

    private AuthService authService;

    public void setAuthService(AuthService authService) {//스프링으로부터 주입
        this.authService = authService;
    }

    @RequestMapping(method=RequestMethod.GET)
    public String form(LoginCommand loginCommand) {
        return "login/loginForm";
    }
```

컨트롤러에서 HttpSession 사용하기

- 로그인을 했다면 로그인한 상태를 유지해야 한다.
- 로그인 상태를 유지하기 위해서 로그인 정보가 저장되어야 하는데 그 방법은 두가지 이다.
  - 쿠키를 이용하는 방법
  - 세션을 이용하는 방법

컨트롤러에서 HttpSession 사용하기

- 스프링 컨트롤러에서 HttpSession을 사용하려면 다음 두가지 방법 중 하나를 사용하면 된다.
  - @RequestMapping 적용 메서드에 HttpSession파라미터를 추가한다.

```
@RequestMapping(method=RequestMethod.POST)
public String form(LoginCommand loginCommand, Errors errors, HttpSession session){
  ....//session을 사용하는 코드
}
```

  - @RequestMapping 적용 메서드에 HttpServletRequest파라미터를 추가하고, HttpServletRequest를 이용해서 HttpSession을 구한다.

```
@RequestMapping(method=RequestMethod.POST)
public String submit(LoginCommand loginCommand, Errors errors, HttpServletRequest req){
    HttpSession session =req.getSession();
    ....//session을 사용하는 코드
}
```

예제6

컨트롤러에서 HttpSession 사용하기

- 예제 7 -Logincontroller.java 추가

```java
@RequestMapping(method=RequestMethod.POST)
public String submit(LoginCommand loginCommand, Errors errors,
        HttpSession session, HttpServletResponse response) {
    //1. 아이디, 비밀번호가 입력이 되기는 했는지 검증
    new LoginCommandValidator().validate(loginCommand, errors);

    if(errors.hasErrors()) {// 로그인 실패
        return "login/loginForm";
    }
    //2. 입력받은 이메일이 DB에 저장이 되었는가? /DB에서 가져온 비밀번호와 일치하는가?

    try {
        AuthInfo authInfo = authService.authenticate(
                loginCommand.getEmail(),
                loginCommand.getPassword());

        // 로그인 성공시 인증 정보를 저장 => 세션 저장
        session.setAttribute("authInfo", authInfo);

        return "login/loginSuccess";
    }catch(IdPasswordNotMatchingException e) {
        // 이메일이 없거나 또는 비밀번호가 틀렸거나
        errors.reject("idPasswordNotMatching");
        return "login/loginForm";
    }
}
```

컨트롤러에서 HttpSession 사용하기

- 예제 7 -Main.jsp 변경

```
<c:if test="${empty authInfo}">
    <p>702 홈페이지에 오신 것을 환영합니다.</p>
    <p>
        <a href="<c:url value='/register/step1'/>">[회원 가입하기]</a> <a
            href="<c:url value='/login'/>">[로그인]</a>
    </p>
</c:if>
<c:if test="${! empty authInfo}">
    <p>${authInfo.name}님환영합니다.</p>
    <p>
        <a href="<c:url value='/edit/changePassword'/>">[비밀번호 변경]</a> <a
            href="<c:url value='/logout'/>">[로그아웃]</a>
    </p>
</c:if>
```

컨트롤러에서 HttpSession 사용하기

- 예제 8 - 로그 아웃 기능을 구성해 본다.
  - logoutController.java

```java
@Controller
public class LogoutController {

    @RequestMapping("/logout")
    public String logout(HttpSession session) {
        session.invalidate();

        return "redirect:/main";
    }
}
```

  - Spring-controller.xml에 빈 등록한다.

```xml
<bean class="spring.controller.LogoutController" />
```

비밀번호 변경 기능 구현

- 예제 8

- vo
  - ChangePwdCommand
- service
  - ChangePasswordService
- validator
  - ChangePwdCommandValidator
- controllor
  - ChangePwdController
- jsp
  - changePwdForm
  - changedPwd

-Properties

-xml

비밀번호 변경 기능 구현

- 예제 8 - vo
  - ChangePwdCommand

```
public class ChangePwdCommand {

    private String currentPassword;
    private String newPassword;

    public String getCurrentPassword() {
        return currentPassword;
    }
    public void setCurrentPassword(String currentPassword) {
        this.currentPassword = currentPassword;
    }
    public String getNewPassword() {
        return newPassword;
    }
    public void setNewPassword(String newPassword) {
        this.newPassword = newPassword;
    }

}
```

비밀번호 변경 기능 구현

- 예제 8 – service => 기존 코드 사용
  - ChangePasswordService

```java
public class ChangePasswordService {

    private MemberDao memberDao;

    public ChangePasswordService(MemberDao memberDao) {
        this.memberDao = memberDao;
    }

    @Transactional
    public void changePassword(String email, String oldPwd, String newPwd) {
        Member member = memberDao.selectByEmail(email);
        if (member == null)
            throw new MemberNotFoundException();

        member.changePassword(oldPwd, newPwd);

        memberDao.update(member);
    }
```

비밀번호 변경 기능 구현

- 예제 8- validator
  - ChangePwdCommandValidator

```
public class ChangePwdCommandValidator implements Validator{

    @Override
    public boolean supports(Class<?> clazz) {
        return ChangePwdCommand.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "currentPassword", "required");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "newPassword", "required");

    }
```

비밀번호 변경 기능 구현

- 예제 8- controllor
  - ChangePwdController

```
@Controller
@RequestMapping("edit/changePassword")
public class ChangePwdController {

    private ChangePasswordService changePasswordService;

    public ChangePwdController(ChangePasswordService changePasswordService) {
        this.changePasswordService=changePasswordService;
    }

     // 1.    Model객체를 이용하는 방법
    @RequestMapping(method=RequestMethod.GET)
    public String form(Model model) {
        model.addAttribute("changePwdCommand", new ChangePwdCommand());
        return "edit/changePwdForm";
    }
```

비밀번호 변경 기능 구현

- 예제 8- controllor
  - ChangePwdController

```
//2.   @ModelAttribute 애노테이션을 이용하는 방법
@RequestMapping(method=RequestMethod.GET)
public String form(
        @ModelAttribute("changePwdCommand") ChangePwdCommand changePwdCommand) {
    return "edit/changePwdForm";
}
```

비밀번호 변경 기능 구현

- 예제 8- controllor
  - ChangePwdController

```
//  3.   Post하고 같은 커맨드 객체를 사용하는 경우 model을 생략 가능
@RequestMapping(method=RequestMethod.GET)
public String form(ChangePwdCommand changePwdCommand, HttpSession session) {
    return "edit/changePwdForm";
}
```

비밀번호 변경 기능 구현

- 예제 8- controllor
  - ChangePwdController

```
@RequestMapping(method=RequestMethod.POST)
public String submit(ChangePwdCommand changePwdCommand,
        Errors errors,HttpSession session) {
    new ChangePwdCommandValidator().validate(changePwdCommand, errors);

    if(errors.hasErrors()) {
        return "edit/changePwdForm";
    }

    AuthInfo authInfo = (AuthInfo)session.getAttribute("authInfo");

    try {
        changePasswordService.changePassword(
                authInfo.getEmail(),
                changePwdCommand.getCurrentPassword(),
                changePwdCommand.getNewPassword());

        return "edit/changedPwd";

    }catch(IdPasswordNotMatchingException err) {//  입력한 기존 비밀번호가 저장된 비밀번호와 다른 경우
        errors.rejectValue("currentPassword", "notMatching");
        return "edit/changePwdForm";
    }

}
```

비밀번호 변경 기능 구현

- 예제8 - jsp
  - edit/changePwdForm.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title><spring:message code="change.pwd.title" /></title>
</head>
<body>
    <form:form commandName="changePwdCommand">
        <p>
            <label><spring:message code="currentPassword" />:<br>
                <form:password path="currentPassword" />
                <form:errors path="currentPassword" />
            </label>
        </p>
        <p>
            <label><spring:message code="newPassword" />:<br>
                <form:password path="newPassword" />
                <form:errors path="newPassword" />
            </label>
        </p>
        <input type="submit" value="<spring:message code="change.btn"/>">
    </form:form>
</body>
```

비밀번호 변경 기능 구현

- 예제8 – jsp
  - edit/changedPwd.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title><spring:message code="change.pwd.title" /></title>
</head>
<body>
    <p>
        <spring:message code="change.pwd.done" />
    </p>
    <p>
        <a href="<c:url value='/main'/>">
        [<spring:message code="go.main"/>]</a>
    </p>
</body>
```

비밀번호 변경 기능 구현

- 예제8 - Properties : label.properties 추가

```
change.pwd.title=비밀번호 변경
currentPassword=현재 비밀번호
newPassword=새 비밀번호
change.btn=변경하기
notMatching.currentPassword=비밀번호를 잘못 입력했습니다.

change.pwd.done=비밀번호를 변경했습니다.
```

비밀번호 변경 기능 구현

- 예제8 –xml : spring-contoller.xml 빈 등록

```
<bean class="spring.controller.ChangePwdController">
    <constructor-arg ref="changePwdSvc"/>
</bean>
```

인터셉터 사용하기

- 비밀번호 변경 폼은 일반적으로 로그인이 진행된 이후에 나와야 할 화면이다.

- 그러나 직접 주소창에 http://localhost:8090/ex09/edits/changePassword 입력하면 비밀번호 변경페이지가 나타난다.
- 로그인하지 않을 때 비밀번호 변경 폼을 요청하면 로그인 화면으로 이동시키는 것이 좋다.

```
//   3.    Post하고 같은 커맨드 객체를 사용하는 경우 model을 생략 가능
@RequestMapping(method=RequestMethod.GET)
public String form(ChangePwdCommand changePwdCommand, HttpSession session) {
    AuthInfo authInfo = (AuthInfo)session.getAttribute("authInfo");

    if(authInfo==null) {
        return "redirect:/login";
    }

    return "edit/changePwdForm";
}
```

인터셉터 사용하기

- 다만 실제 웹 애플리케이션은 로그인 이후에 진행되는 기능이 비밀번호 변경말고도 매우 많을 수 있다.
- 이런 경우 앞서 본 코드를 일일이 넣는 것은 많은 중복을 발생시킨다.

- 이때 여러 컨트롤러에 공통의 기능을 적용할 때 사용 가능한 것이 인터셉터이다.
- 스프링은 인터셉터 기능을 지원하기 위해 HandlerInterceptor 인터페이스를 제공한다.

인터셉터 사용하기

- HandlerInterceptor인터페이스 구현하기
- HandlerInterceptor인터페이스는 3가지 시점에 공통의 기능을 삽입할 수 있다.
  - 컨트롤러(핸들러) 실행 전
  - 컨트롤러(핸들러) 실행 후, 뷰를 실행하기 전
  - 뷰를 실행한 이후

- HandlerInterceptor인터페이스는 3가지 시점을 처리하기 위한 3가지 메서드를 제공한다.
  - boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)throws Exception;
  - void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView)throws Exception;
  - void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)throws Exception;

인터셉터 사용하기

- 예제 10

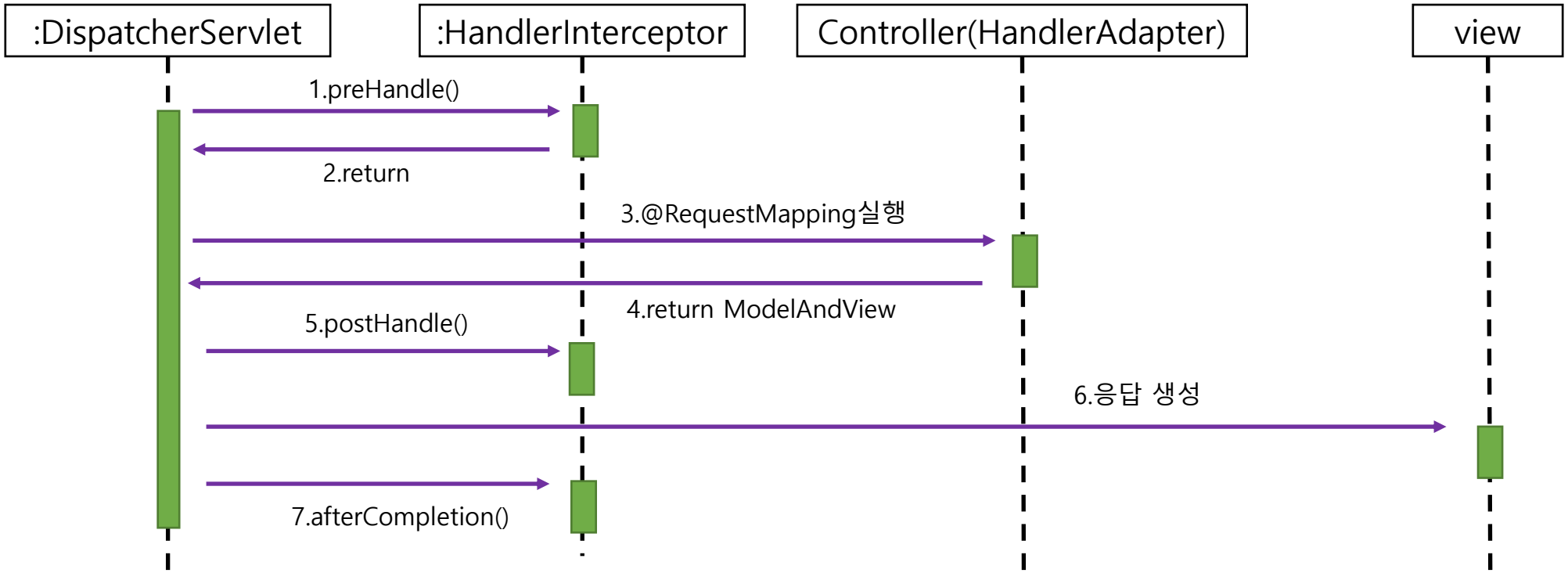
```
public class AuthCheckInterceptor extends HandlerInterceptorAdapter{

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
            throws Exception {
        HttpSession session= request.getSession();

        if(session != null) {
            Object authInfo = session.getAttribute("authInfo");
            if(authInfo != null) {
                return true;
            }
        }
        response.sendRedirect(request.getContextPath()+"/login");
        return false;
    }
}
```

인터셉터 사용하기

- preHandle메서드는 컨트롤러 객체가 실행하기 전 필요한 기능을 구현할 때 사용
  - false를 반환하면 컨트롤러를 실행하지 않는다.
- postHandle메서드는 컨트롤러가 정상적으로 실행된 이후에 추가 기능을 구현할 때 사용
- afterCompletion메서드는 뷰가 클라이언트에 응답을 전송한 이후에 실행된다.
  - 컨트롤러에서 예외가 발생하면 메서드의 매개값으로 전달한다.
  - 주로 예외의 로그를 남기거나 후처리등을 진행한다.



인터셉터 사용하기

- HandlerInterceptor인터페이스 설정하기

- 적용 위치를 지정해 주어야 하는데 이때 사용되는 태그는 <mvc:interceptor>와 <mvc:mapping> 태그이다.
- 이 태그를 controller.xml에 추가한다. -> 예제 10

- < mvc:interceptors> : 여러개의 인터셉터를 등록한다.
- <mvc:interceptor> : 하나의 인터셉터를 설정한다. 중첩된 bean태그로 인터셉터로 사용될 객체를 지정한다.
- <mvc:mapping> : 인터셉터를 적용할 경로를 지정한다.

인터셉터 사용하기

- 예제 10 :

```
<mvc:interceptors>
    <mvc:interceptor>
        <!-- 어디에 -->
        <mvc:mapping path="/edit/**"/>
        <!-- 어떤 인터셉터 -->
        <bean class="spring.intercepter.AuthCheckInterceptor" />
    </mvc:interceptor>
</mvc:interceptors>
```

인터셉터 사용하기

- 경로 패턴
  - *: 0개 또는 그 이상의 글자
  - ? : 1개 글자
  - ** : 0개 또는 그 이상의 디렉터리

- <mvc:mapping>에 지정된 경로중 일부 경로를 제외하고 싶을 때는 <mvc:exclude-mapping>을 사용한다.

```
<mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/edit/**" />
            <mvc:exclude-mapping path="/edit/help/**" />
            <bean class="interceptor.AuthCheckInterceptor" />
        </mvc:interceptor>
</mvc:interceptors>
```

컨트롤러에서 쿠키 사용하기

- 어떤 정보를 서버에 저장하는 것이 아닌 사용자 컴퓨터에 저장하는 것을 쿠키라고 한다.
- 비밀번호와 같은 민감한 개인정보를 저장하기에 보안상 문제가 있지만 사용자의 편의를 위해서 적절하게 사용할 수있다.

- 사용자의 편리함을 위해 아이디를 기억해 두었다가 다음 로그인시 아이디를 자동으로 넣어주는 기능을 구현할 때 쿠키를 사용할 수 있다.

- 이메일을 자동으로 기억하는 기능을 추가해본다.

컨트롤러에서 쿠키 사용하기

- 이메일 기억하기 기능을 구현하는 방식
  - 로그인 폼에 이메일 기억하기 옵션을 추가한다.
  - 로그인 시 이메일 기억하기 옵션을 선택하면 로그인 성공 후 쿠키에 이메일 정보를 저장한다.
  - 쿠키를 저장할 땐 삭제되지 않도록 유효시간을 길게 설정한다.
  - 이후 로그인 폼을 보여줄 때 이메일을 저장한 쿠키가 존재하면 입력폼에 이메일을 보여준다

- 이메일 기억하기 기능을 구현하기 위해 추가해야 할 부분
  - loginForm.jsp : 이메일 기억하기 선택항목 추가
  - LoginController.java - form() : 쿠키가 존재할 경우 폼에 전달할 커맨드 객체의 email프로퍼티를 쿠키의 값으로 꺼내온다.
  - LoginController.java - Submit(): 이메일 기억하기 옵션을 선택한 경우 로그인 성공후 이메일을 담고 있는 쿠키를 생성한다.

컨트롤러에서 쿠키 사용하기

- 예제 11 : LoginForm.jsp에 이메일 기억하기 체크박스를 달아둔다

```
<p>
    <label><spring:message code="rememberEmail"/>:<br>
        <form:checkbox path="rememberEmail"/>
    </label>
</p>
```

컨트롤러에서 쿠키 사용하기

- 예제 11 : 체크박스에 사용할 label를 작성한다.

```
rememberEmail=이메일 기억하기
```

컨트롤러에서 쿠키 사용하기

- 쿠키를 생성하려면 HttpServletResponse 객체가 필요하다.

```
public String submit(
    LoginCommand loginCommand, Errors errors, HttpSession session,
    HttpServletResponse response) {
.....
Cookie rememberCookie = new Cookie("REMEMBER", loginCommand.getEmail());
rememberCookie.setPath("/");
if (loginCommand.isRememberEmail()) {
    rememberCookie.setMaxAge(60 * 60 * 24 * 30);
} else {
    rememberCookie.setMaxAge(0);
}
response.addCookie(rememberCookie);    .....
```

컨트롤러에서 쿠키 사용하기

- 스프링 MVC에서 쿠키를 사용하는 방법 중 하나는 @CookieValue 애노테이션을 사용하는 것이다.
- @RequestMapping 적용 메서드의 Cookie타입 파라미터를 적용한다. 이를 통해 쉽게 쿠키 파라미터를 전달 받는다.

```
@RequestMapping(method = RequestMethod.GET)
public String form(LoginCommand loginCommand,
     @CookieValue(value = "REMEMBER", required = false) Cookie rememberCookie) {
    if (rememberCookie != null) {
        loginCommand.setEmail(rememberCookie.getValue());
        loginCommand.setRememberEmail(true);
    }
return "login/loginForm";
}
```

- value : 쿠키 이름을 가져온다

- required : false면 쿠키가 없을 때는 가져오지 않는다.(true인 경우 쿠키가 없을 때 예외 발생)

컨트롤러에서 쿠키 사용하기

- 예제 11 : LoginController를 수정한다.

```
@RequestMapping(method=RequestMethod.POST)
public String submit(LoginCommand loginCommand, Errors errors,
        HttpSession session, HttpServletResponse response) {
    //1. 아이디, 비밀번호가 입력이 되기는 했는지 검증
    new LoginCommandValidator().validate(loginCommand, errors);

    if(errors.hasErrors()) {// 로그인 실패
        return "login/loginForm";
    }
    //2. 입력받은 이메일이 DB에 저장이 되었는가? /DB에서 가져온 비밀번호와 일치하는가?

    try {
        AuthInfo authInfo = authService.authenticate(
                loginCommand.getEmail(),
                loginCommand.getPassword());

        // 로그인 성공시 인증 정보를 저장 => 세션 저장
        session.setAttribute("authInfo", authInfo);

        // 이메일 저장용 쿠키
        Cookie rememberCookie =
                new Cookie("rememberEmail",loginCommand.getEmail());

        rememberCookie.setPath("/");
        if(loginCommand.isRememberEmail()) {
            rememberCookie.setMaxAge(60*60*24*365);
        }else {
            rememberCookie.setMaxAge(0);
        }
        // 쿠키 클라이언트에 저장
        response.addCookie(rememberCookie);

        return "login/loginSuccess";
    }catch(IdPasswordNotMatchingException e) {
        // 이메일이 없거나 또는 비밀번호가 틀렸거나
        errors.reject("idPasswordNotMatching");
        return "login/loginForm";
    }
}
```

컨트롤러에서 쿠키 사용하기

- 예제 11 : LoginController를 수정한다.

```
@RequestMapping(method=RequestMethod.GET)
public String form(LoginCommand loginCommand,
        @CookieValue(value="rememberEmail", required=false)Cookie rememberEmail) {
    if(rememberEmail != null) {
        loginCommand.setEmail(rememberEmail.getValue());
        loginCommand.setRememberEmail(true);
    }

    return "login/loginForm";
}
```

컨트롤러에서 쿠키 사용하기

- 쿠키의 전송 범위

path	전송범위
미지정	쿠키를 생성했던 URL 범위에서 전송
/	웹 애플리케이션의 모든 URL 범위에서 전송
/login	/login/xx ~~요청시 전송
/edits	/edits/xx ~~요청시 전송 /edits/subDir/xx ~~요청시 전송
/edits/subDir/	/edits/subDir/xx ~~요청시 전송