12강 MVC 3

날짜를 이용한 회원 검색 기능

- 가입일을 기준으로 회원을 검색하는 기능을 구현하면서 몇몇 특징을 알아보자

- MemberDao에 selectByRegdate() 메서드를 추가한다.

```
public List<Member> selectByRegdate(Date from, Date to){

    List<Member> result = jdbcTemplate.query(
            "SELECT * FROM member WHERE REGDATE between ? AND ? ORDER BY REGDATE ASC",
            new RowMapper<Member>() {
                @Override
                public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
                    Member member = new Member(
                            rs.getString("email"),
                            rs.getString("password"),
                            rs.getString("name"),
                            rs.getTimestamp("regdate")
                            );
                    member.setId(rs.getLong("id"));
                    return member;
                }

            },from,to);
    return result;
}
```

커맨드 객체 date 타입 프로퍼티 변환 처리(@DateTimeFormat)

- 가입 일시를 기준으로 회원을 검색할 수 있도록 하기 위해 시작과 끝 날짜를 파라미터로 전달받아야 한다.
- 이를 위해 커맨드 객체를 제작해본다.

```
public class ListCommand {

        private Date from;
        private Date to;

        public Date getFrom() {
            return from;
        }
        public void setFrom(Date from) {
            this.from = from;
        }
        public Date getTo() {
            return to;
        }
        public void setTo(Date to) {
            this.to = to;
        }
}
```

커맨드 객체 date 타입 프로퍼티 변환 처리(@DateTimeFormat)

- 그러나 문자열을 Date 타입으로 변환해야 하는데 스프링은 기본타입(int, long등)으로 변환은 지원하지만 Date타입으로 변환은 추가 설정을 해야 한다.

```
public class ListCommand {

    @DateTimeFormat(pattern="yyyyMMddHH")
    private Date from;
    @DateTimeFormat(pattern="yyyyMMddHH")
    private Date to;

    public Date getFrom() {
        return from;
    }
    public void setFrom(Date from) {
        this.from = from;
    }
    public Date getTo() {
        return to;
    }
    public void setTo(Date to) {
        this.to = to;
    }
}
```

커맨드 객체 date 타입 프로퍼티 변환 처리(@DateTimeFormat)

- 컨트롤러를 제작한 후 빈 등록을 해본다.

```java
@Controller
public class MemberListController {

    private MemberDao dao;

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

    @RequestMapping("/member/list")
    public String list(
        @ModelAttribute("cmd") ListCommand listCommand,
        Model model) {


        if(listCommand.getFrom()!=null && listCommand.getTo() !=null) {

            List<Member> members =
                dao.selectByRegdate(listCommand.getFrom(), listCommand.getTo());
            System.out.println("members : " + members.size());
            model.addAttribute("members", members);
        }

        return "member/memberList";
    }
}
```

```xml
<bean class="spring.controller.MemberListController">
    <property name="memberDao" ref="memberDao" />
</bean>
```

커맨드 객체 date 타입 프로퍼티 변환 처리(@DateTimeFormat)

- 이어서 뷰 코드를 작성해 본다.

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

```jsp
<body>
    <form:form commandName="listCommand">
        <p>
            <label>from: <form:input path="from"/></label>
            ~
            <label>to: <form:input path="to"/></label>
            <input type="submit" value="조회">
        </p>
    </form:form>
    <c:if test="${!empty members}">
        <table>
            <tr>
                <th>아이디</th>
                <th>이메일</th>
                <th>이름</th>
                <th>가입일</th>
            </tr>
            <c:forEach var="m" items="${members}">
                <tr>
                    <td>${m.id}</td>
                    <td>
                        <a href="<c:url value="/member/detail/${m.id}"/>">${m.email}</a>
                    </td>
                    <td>${m.name}</td>
                    <td>
                    <fmt:formatDate value="${m.registerDate}" pattern="yyyy-MM-dd"/>
                    </td>
                </tr>
            </c:forEach>
        </table>
    </c:if>
</body>
```

커맨드 객체 date 타입 프로퍼티 변환 처리(@DateTimeFormat)

- 날짜를 입력해야 하는 뷰에 형식에 맞지 않는 날짜를 입력하면 오류가 발생한다.
- 이럴 경우 400번 에러 페이지가 나오는데 이런 에러페이지 말고 타입변환에 대한 에러 코드(typeMismatch)를 추가함으로 에러메시지를 출력할 수 있다.

```
@RequestMapping("/member/list")
public String list(
    @ModelAttribute("cmd") ListCommand listCommand,Errors errors,
    Model model) {

    if(errors.hasErrors()) {
        return "member/memberList";
    }
```

```
typeMismatch.java.util.Date=잘못된 형식
```

```
<p>
    <label>from: <form:input path="from"/></label>
    <form:errors path="from"/>
    ~
    <label>to: <form:input path="to"/></label>
    <form:errors path="to"/>
    <input type="submit" value="조회">
</p>
```

@PathVariable을 이용한 경로 변수 처리

- ID가 10인 회원 정보를 조회하기 위한 URL구성

=> http://localhost:8090/ex09/member/detail/10

- 위와 같이 해당 ID값을 요청 경로에 포함시키는 방법으로 사용한다.

- 이렇게 경로의 특정 부분의 값이 고정되지 않고 달라 질 수 있는 것이 @PathVariable이다.

@PathVariable을 이용한 경로 변수 처리

- DAO객체에 다음 메서드를 추가해본다.

```
public Member selectById(Long id) {
    String sql = "SELECT * FROM members WHERE id=?";
    List<Member> result = jdbcTemplate.query(sql,rowMapper,id);

    return result.isEmpty()?null:result.get(0);
}
```

@PathVariable을 이용한 경로 변수 처리

- 컨트롤러를 만들어본다.

```
public class MemberDetailController {
    private MemberDao memberDao;

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

    @RequestMapping("/member/detail/{id}")
    public String detail(@PathVariable("id") Long memId,Model model) {
        Member m = memberDao.selectById(memId);

        if(m==null) {
            throw new MemberNotFoundException();
        }

        model.addAttribute("member",m);
        return "member/memberDetail";
    }
}
```

@PathVariable을 이용한 경로 변수 처리

- 컨트롤러를 빈으로 등록합니다.

```
<bean class="spring.controller.MemberDetailController">
    <property name="memberDao" ref="memberDao" />
</bean>
```

@PathVariable을 이용한 경로 변수 처리

- 뷰페이지를 생성한다

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원 정보</title>
</head>
<body>
    <p>아이디: ${member.id}</p>
    <p>이메일: ${member.email}</p>
    <p>이름: ${member.name}</p>
    <p>가입일: <fmt:formatDate value="${member.registerDate}"
                pattern="yyyy-MM-dd HH:mm" /></p>
    <br>
    <p>
        <a href="<c:url value='/member/list'/>">[날짜별 회원 정보 보기 ]</a>
    </p>
</body>
</html>
```

컨트롤러 예외 처리하기

- ID가 존재할 때는 정상적으로 뷰페이지가 출력이 되지만 ID가 존재하지 않으면 500번 에러페이지를 보여준다.
- 또한 ID값은 long타입인데 문자열 등을 입력하면 역시 형변환 불가로 인한 400번 에러페이지를 보여준다.

- 이런 여러가지 에러가 발생했을 때 적절한 에러 화면을 보여 주는 방법이 존재한다.
- @ExceptionHandler 애노테이션을 사용하는 방법이다.

- @ExceptionHandler가 적용된 메서드가 존재하면 스프링 MVC는 이 메서드가 예외를 처리하도록 한다.

컨트롤러 예외 처리하기

- 컨트롤러에서 예외처리를 위한 메서드를 만든다.

```
@ExceptionHandler(TypeMismatchException.class)
public String handleTypeMismatchException(TypeMismatchException er) {

    return "member/invalidate";
}

@ExceptionHandler(MemberNotFoundException.class)
public String handleNotFoundException(MemberNotFoundException err) {
    return "member/noMember";
}
```

컨트롤러 예외 처리하기

- 각 예외 페이지를 보여주기 위한 뷰페이지를 제작한다.

```
<title>코드 오류</title>
</head>
<body>
    <p>
        잘못된 요청입니다.
    </p>
```

```
<title>회원 없음</title>
</head>
<body>
    <p>
        존재하지 않는 회원입니다.
    </p>
</body>
```

컨트롤러 예외 처리하기

- 다수의 일반 예외를 모아서 처리하기 위한 컨트롤러를 만들 수 있다

```java
@ControllerAdvice("spring")
public class CommonExceptionHandler {

    @ExceptionHandler(MemberNotFoundException.class)
    public String handlerMemberNotFoundException(MemberNotFoundException e) {
        return "member/noMember";
    }

    @ExceptionHandler(RuntimeException.class)
    public String handlerRuntimeException(RuntimeException e) {
        return "error/commonException";
    }
}
```

```xml
<bean class="spring.commonException.CommonExceptionHandler" />
```