2024/10/18 3번째

💡 Spring 핵심 : 객체를 획득하는 방법이 달라진것

DB엑세스

정보추가하기

입력폼 요청하기

GET 방식

/employee/register

입력값을 전송하기 저장요청하기

POST 방식

/employee/register

```
@Controller
@RequestMapping("/employee")
public class EmployeeController{

    @GetMapping("list")
    public String employees(Model model){
        //직원목록 조회, Model 객체에 조회된 직원목록 담기
        return "emp/list" <-- jsp페이지의 경로 및 이름이다.
    }

    @GetMapping("/register")
    public String 입력폼요청처리(Model model){
        // 입력폼에 표시할 데이터 조회
        // 조회된 데이터를 Model에 담기

        // 뷰이름 반환
        return"emp/from"; <-- jsp페이지의 경로 및 이름이다.
    }

    @PostMapping("/register")
```

```
    public String 신규직원등록요청처리(EmpRegisterForm form){
        // EmpRegisterForm으로 전달받은 입력폼 유효성 체크
        // EmpRegisterForm객체를 서비스에 전달해서 신규 직원정보 저장시키기

        // 재요청 URL("/employee/list")을 뷰이름으로 반환
redirect: 재요청하는 URL (새로 요청할 URL)
        return "redirect:/employee/list" <-- jsp 페이지 경로 및 이름이 아니다. (재요청할 URL 매핑 경로다.)

    }

}
```

```
    <!--void insertEmployee(@Param("emp") Employee employee);

      -->
    <insert id="insertEmployee">
        insert into employees
        (
            employee_id
            , first_name
            , last_name
            , email
            , phone_number
            , hire_date
            , job_id
            , salary
            <if test="emp.commissionPct > 0">
                , commission_pct
```

```
            </if>
            <if test="emp.managerId != 0">
                , manager_id
            </if>
            <if test="emp.departmentId != 0">
                , department_id
            </if>
            )
            values
            (
                employees_seq.nextval
                , #{emp.firstName}
                , #{emp.lastName}
                , #{emp.email}
                , #{emp.phoneNumber}
                , #{emp.hireDate}
                , #{emp.jobId}
                , #{emp.salary}
            <if test="emp.commissionPct > 0">
                , #{emp.commissionPct}
            </if>
            <if test="emp.managerId != 0">
                , #{emp.managerId}
            </if>
            <if test="emp.departmentId != 0">
                , #{emp.departmentId}
            </if>
            )
    </insert>
```

데이터 엑세스 작업

정보(직원)를 표현하는 객체만들기

`classs Employee{ }`

데이터베이스 엑세스 작업을 정의하는 인터페이스

```
@Mapper
public interface EmployeeMapper {

    Employee getEmployeeById(@Param("id")int id);
    List<Employee> getAllEmployees(@Param("deptId") int dep
tId);
    void insertEmployee(@Param("emp") Employee employee);

}
```

SQL매퍼파일 만들기

```
<mapper namespace="com.example.demo.mapper.EmployeeMapper">
<insert id ="insertEmployee">
    ...
</insert>
</mapper>
```

Service 직원정보 관련 업무로직이 구현된 객체

```
@Service
public class EmployeeService{
    @Autowired
    private EmployeeMapper employeeMapper;      <--- Employ
eeMapper 인터페이스 조립(주입) 데이터 엑세스
    @Autowired
    private ModelMapper modelMapper;
    // 신규 직원등록 서비스

    public void addNewEmployee(EmployeeForm form) {

    }
```

```
}
```

Controller 직원관리 요청을처리하는 객체

```
@Controller
@RequestMapping("/employee")
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;    Service 객체
조립(주입) 업무처리
    @getMapping("list")
    public String employees(Model model({
        HTTP 요청처리
    }
```

오후 실습

- 1. com.example.demo.web.controller패키지에 BookController 클래스를 정의한다 .
  - 도서목록조회
    요청URL
    localhost/book/list
    요청방식
    GET
  - 신규 도서등록 폼
    요청URL
    localhost/book/add
    요청방식
    GET
  - 도서 상세정보 조회

요청URL
localhost/book/detail

요청방식
GET
요청 파라미터
no=100
필수입력값 int no만 하면됨

- 2. com.example.demo.vo 패키지에 tb_books 테이블이 정보를 표현한 VO 클래스를 정의한다.
  lombok 라이브러리를 사용해서 기본생성자, Getter, Setter, toString를 생성시킨다.
- 3. com.example.demo.mapper 패키지에 tb_books 테이블에 대한 CRUD작업을 정의할 BookMapper 인터페이스를 정의한다.
- 4. src/main/resources/mybatis/mappers 폴더에 tb_books 테이블관련 매핑된 SQL을 정의하는 BookMapper.xml파일을 생성한다.
- 5. 전체도서목록 조회 요청을 구현하기
  - 요청URL
    localhost/book/list
    요청방식
    GET
    가. 데이터베이스 엑세스 작업 구현

    + 도서목록 화면을 확인하고, 표시될 데이터를 파악하기
      번호,제목,저자,가격 출판일 정보 표현
      * tb_books 테이블만 조회하면 된다.
    + BookMapper 인터페이스에 조회 메소드를 정의한다.
      List<Book> getAllBooks();
    + BookMapper.xml에 매핑된 SQL 구문을 작성한다.
      <select id="getBooks" resultType="...">

      </select>
    나. 업무로직 구현

    + com.example.demo.service패키지에 BookService 클래스를 정의한다.

    + 필요한 의존성을 주입기

@Autowired
private BookMapper bookMapper;

+ 업무로직 메소드 구현하기
// 전체 도서정보를 조회해서 반환하는 서비스 메소드다.
public List<Book> getAllBooks(){

}

다. 컨트롤러의 요청 핸들러 메소드 구현
+ 필요한 의존성 주입받기
@Autowired
private BookService BookService;
+ 요청처리 핸들러 메소드 구현하기
// 전체 도서정보를 반환하는 업무로직 메소드를 호출해서 도서 목록을 획득한다.
// 획득한 도서목록을 뷰페이지에 전달하기 위해서 Model객체에 담는다.
// 뷰 이름을 반환한다

- 신규도서 등록하기

  가. 신규도서 등록화면 요청 처리하기

  나. 신규도서 등록 요청 처리하기

  ```
  @PostMapping("/add")
  public String addBook(){

  }
  ```

  + 입력폼을 체크하고, 입력폼의 입력값을 저장하는 BookAddForm.java를 작성한다

    - 요청핸들러 메소드 작성하기

      ```
      @PostMapping("add")
      public String addBook(BookAddForm form){
      System.out.println(form);

      return null;
      ```

}

- 요청핸들러 메소드 작성하기
  public class BookService{
  public void addNewBook(BookAddForm form){}
  }

조회

개발순서

5←4←3←-2←1

화면 → APP → DB

sql 잘 작성되었는지 확인 값 출력 확인 등

하나씩 각개격파하기

등록

개발순서

1→2→3→4→5

화면→ app →DB

화면

값이 제데로가는지 확인

app

값이 넘어왔는지확인