

フレームワーク実習

09B. MyBatis 基礎 練習

■ 概要

データベースプログラミング実習の練習問題で作成した会員管理システム「JavaClub」を、Spring Boot のプロジェクトとして改めて開発していきます。

JavaClub は以下のようなページで構成される Web システムです。具体的なイメージについては「SFWP09.JavaClub.Data/JavaClubHtml」内の HTML ファイルをブラウザで開いて確認してください。

ページ	ファイル	説明
トップページ	home.html	ログインフォームを表示する。 ログイン済みの場合、フォームは表示せず、 会員一覧とお知らせ一覧へのリンクを表示 する。
会員一覧	members/list.html	JavaClub に所属する会員の情報を一覧で表 示する。会員数が 10 人を超える場合は、ペ ージを分ける。
会員の追加	members/save.html	新規会員を登録するためのフォームを表示 する。
会員の編集		既存の会員情報を編集するためのフォーム を表示する。
お知らせ一覧	news/list.html	会員向けのお知らせを一覧で表示する。
お知らせの追加	news/save.html	会員向けのお知らせを追加するためのフォ ームを表示する。
お知らせ詳細	news/detail.html	お知らせの詳細を表示する。

練習 09 では、これらのページのうち、トップページと会員に関するページを作成していきます。お
知らせに関するページは後の練習問題で作成します。

■ 準備

この練習問題では、データベースプログラミング実習の練習問題で作成した「mydb」を使用します。まだ mydb を作成していない場合は「SFWP09.JavaClub.Data/setup_mydb.sql」内の SQL 文を実行してください。

mydb の準備が整ったら、以下のような設定で、新規 Spring スターター・プロジェクトを作成しましょう。

サービス URL	https://start.spring.io
名前	SFWP.JavaClubSpring
デフォルト・ロケーションを使用	✓ を入れる
タイプ	Maven
パッケージング	Jar
Java バージョン	17
言語	Java
グループ	com.example
成果物	SFWP.JavaClubSpring
バージョン	0.0.1-SNAPSHOT
説明	会員管理システム「JavaClub」
パッケージ	com.example.app

追加する依存関係

カテゴリ	依存関係
Web	Spring Web
テンプレート・エンジン	Thymeleaf
開発ツール	Spring Boot DevTools
開発ツール	Lombok
I/O	Validation (検証)
SQL	MyBatis Framework
SQL	MySQL Driver

プロジェクトを作成したら、application.properties を編集し、データベース接続と MyBatis の設定を記述します。

application.properties

```
# データベース接続設定
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/mydb?useUnicode=true&ch
```

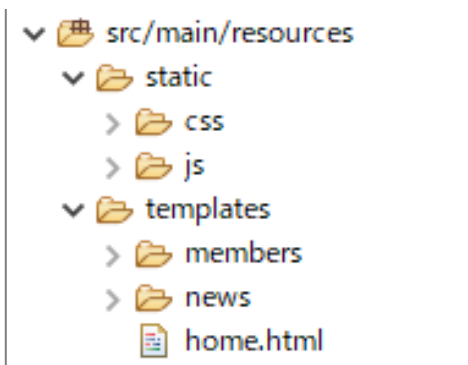
```

characterEncoding=utf8&serverTimezone=Asia/Tokyo
spring.datasource.username=root
spring.datasource.password=

# MyBatis の設定
mybatis.mapper-locations=classpath*:/mybatis/**/*.xml
mybatis.configuration.map-underscore-to-camel-case=true

```

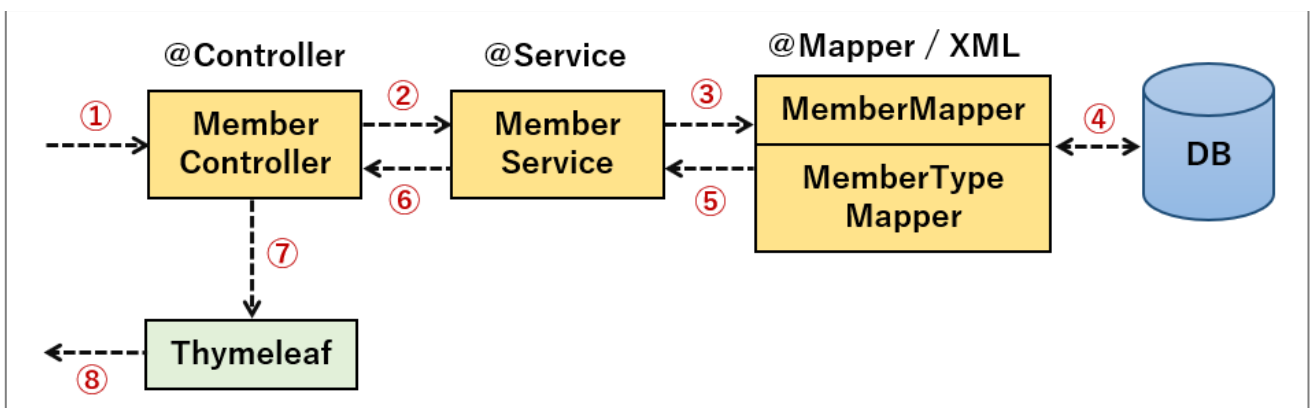
続いて「SFWP09.JavaClub.Data/JavaClubHtml」内のファイル群を src/main/resources 以下に移動します。「home.html, news フォルダ, members フォルダ」は templates 以下に配置してください。また「css フォルダ, js フォルダ」は static 以下に配置します。



これで準備は完了です。

■ 説明 : members/member_types テーブルとの連携

mydb の members テーブルや member_types テーブルと連携する流れは下図のようになります。

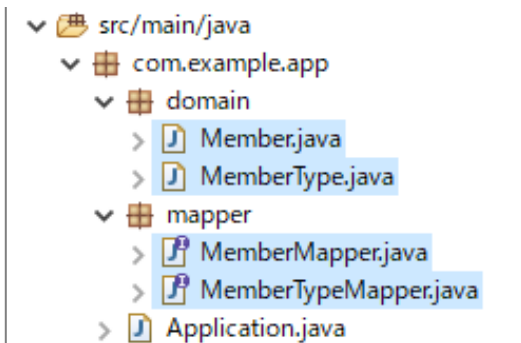


- ① リクエストに応じて、MemberController のメソッドが呼び出される
- ② MemberController は MemberService のメソッドを利用する
- ③ MemberService は MemberMapper/MemberTypeMapper を介して
- ④ データベース(members/member_types テーブル)を操作する
- ⑤ MemberMapper/MemberTypeMapper から MemberService にデータが渡される

- ⑥ MemberService から MemberController にデータが渡される
- ⑦ MemberController から Model オブジェクトを介して、ビューファイルにデータが渡る
- ⑧ HTML を生成され、レスポンスが行われる

■ 練習 09B-1 : マッパーの作成

練習 09B-1 では、MemberMapper と MemberTypeMapper、及びこれらにドメインクラスを作成します。



まずはドメインクラスから作成しましょう。com.example.app.domain パッケージを作成し、Member クラスと MemberType クラスを配置します。Member クラスについては、会員登録時に利用することを想定し、あらかじめバリデーションのためのアノテーションを付与しておきます。

Member.java (List 09B-1-1)

```
package com.example.app.domain;

import java.time.LocalDateTime;

import org.hibernate.validator.constraints.Range;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.Data;

@Data
public class Member {

    private Integer id;

    @NotBlank
    @Size(max=10)
    private String name;

    @Range(min=0, max=120)
    private Integer age;

    @Size(max=255)
    private String address;
```

```

private Integer typeId;
private String typeName;
private LocalDateTime created;
}

```

MemberType.java (List 09B-1-2)

```

package com.example.app.domain;

import lombok.Data;

@Data
public class MemberType {

    private Integer id;
    private String name;
}

```

続いて、com.example.app.mapper パッケージを作成し、MemberMapper インターフェースと MemberTypeMapper インターフェースを配置します。また、これらのインターフェースが Spring から利用されるよう main メソッドをもつクラス(Application.java)に@MapperScan アノテーションを付与しましょう (インターフェースに@Mapper アノテーションを付与しても構いません)。

MemberMapper には、データの取得／追加／更新／削除に対応する抽象メソッドを、MemberTypeMapper には、データの取得に対応する抽象メソッドを記述します。

MemberMapper.java (List 09B-1-3)

```

package com.example.app.mapper;

import java.util.List;

import com.example.app.domain.Member;

public interface MemberMapper {

    List<Member> selectAll() throws Exception;
    Member selectById(Integer id) throws Exception;
    void insert(Member member) throws Exception;
    void update(Member member) throws Exception;
    void delete(Integer id) throws Exception;
}

```

MemberTypeMapper.java (List 09B-1-4)

```
package com.example.app.mapper;

import java.util.List;

import com.example.app.domain.MemberType;

public interface MemberTypeMapper {

    List<MemberType> selectAll() throws Exception;

}
```

Application.java (List 09B-1-5)

```
package com.example.app;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

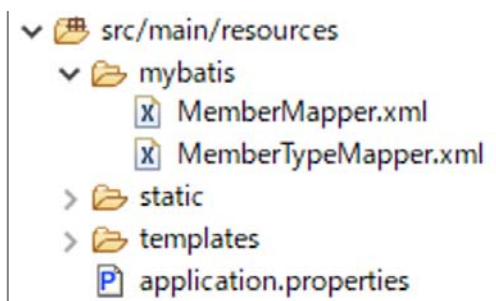
@MapperScan("com.example.app.mapper")
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

MemberMapper や MemberTypeMapper の抽象メソッドをオーバーライドして実装するためのクラスは作成しません。その代わりに、これらの抽象メソッドと紐づく SQL 文を XML ファイルに記述していきます。

src/main/resources 内に mybatis フォルダを作成します。その中に MemberMapper.xml と MemberTypeMapper.xml を作成しましょう。



XML ファイルを作成する前に、XML 内に頻繁に登場する「com.example.app.domain.Member」を「Member」と短く記述できるように application.properties に追記を行います。

application.properties (List 09B-1-6)

```
# データベース接続設定
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/mydb?useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Tokyo
spring.datasource.username=root
spring.datasource.password=

# MyBatis の設定
mybatis.mapper-locations=classpath*:mybatis/**/*.xml
mybatis.configuration.map-underscore-to-camel-case=true
mybatis.type-aliases-package=com.example.app.domain
```

MemberMapper.xml (List 09B-1-7)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.app.mapper.MemberMapper">

    <select id="selectAll" resultType="Member">
        SELECT
            members.id, members.name, members.age,
            members.address, members.created,
            member_types.name AS typeName
        FROM members
        JOIN member_types
        ON members.type_id = member_types.id
    </select>

    <select id="selectById" parameterType="int" resultType="Member">
        SELECT
            members.id, members.name, members.age,
            members.address, members.created,
            members.type_id AS typeId
        FROM members
        WHERE members.id = #{id}
    </select>

    <insert id="insert" parameterType="Member">
        INSERT INTO members (name, age, address, type_id, created)
        VALUES (#{name}, #{age}, #{address}, #{typeId}, NOW())
    </insert>

    <update id="update" parameterType="Member">
        UPDATE members
        SET name = #{name}, age = #{age},
            address = #{address}, type_id = #{typeId}
        WHERE id = #{id}
    </update>

    <delete id="delete" parameterType="int">
        DELETE FROM members WHERE id = #{id}
    </delete>
```

```
</mapper>
```

MemberTypeMapper.xml (List 09B-1-6)

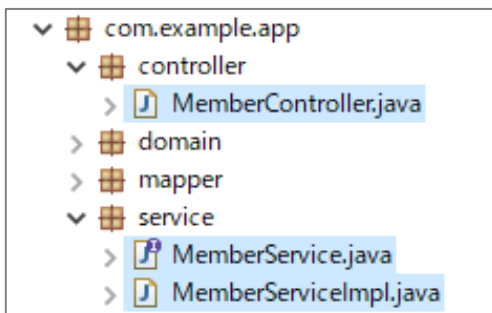
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.app.mapper.MemberTypeMapper">

    <select id="selectAll" resultType="MemberType">
        SELECT * FROM member_types;
    </select>

</mapper>
```

■ 練習 09B-2 : 会員一覧ページの作成

練習 09B-2 では、会員一覧ページを作成していきます。まずは、コントローラーが利用するサービスクラスを作成しましょう。com.example.app.service パッケージを作成し、MemberService インターフェイスと、それを実装する MemberServiceImpl クラスを配置しましょう。



MemberService.java (List 09B-2-1)

```
package com.example.app.service;

import java.util.List;

import com.example.app.domain.Member;
import com.example.app.domain.MemberType;

public interface MemberService {

    List<Member> getMemberList() throws Exception;
    Member getMemberById(Integer id) throws Exception;
    void addMember(Member member) throws Exception;
    void editMember(Member member) throws Exception;
    void deleteMember(Integer id) throws Exception;
    List<MemberType> getTypeList() throws Exception;
}
```



```
}

```

MemberServiceImpl.java (List 09B-2-2)

```
package com.example.app.service;

import java.util.List;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.app.domain.Member;
import com.example.app.domain.MemberType;
import com.example.app.mapper.MemberMapper;
import com.example.app.mapper.MemberTypeMapper;

import lombok.RequiredArgsConstructor;

@Service
@Transactional(rollbackFor = Exception.class)
@RequiredArgsConstructor
public class MemberServiceImpl implements MemberService {

    private final MemberMapper memberMapper;
    private final MemberTypeMapper memberTypeMapper;

    @Override
    public List<Member> getMemberList() throws Exception {
        return memberMapper.selectAll();
    }

    @Override
    public Member getMemberById(Integer id) throws Exception {
        return memberMapper.selectById(id);
    }

    @Override
    public void addMember(Member member) throws Exception {
        memberMapper.insert(member);
    }

    @Override
    public void editMember(Member member) throws Exception {
        memberMapper.update(member);
    }

    @Override
    public void deleteMember(Integer id) throws Exception {
        memberMapper.delete(id);
    }

    @Override
    public List<MemberType> getTypeList() throws Exception {

```

```

        return memberTypeMapper.selectAll();
    }
}

```

続いて、コントローラーを作成します。com.example.app.controller パッケージを作成し、MemberController クラスを配置します。サービスを使用して会員のリストを取得し、ビューファイルに渡すメソッドを実装しましょう。

MemberController.java (List 09B-2-3)

```

package com.example.app.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import com.example.app.service.MemberService;

import lombok.RequiredArgsConstructor;

@Controller
@RequestMapping("/members")
@RequiredArgsConstructor
public class MemberController {

    private final MemberService service;

    @GetMapping
    public String list(Model model) throws Exception {
        model.addAttribute("members", service.getMemberList());
        return "members/list";
    }
}

```

最後に「templates/members/list.html」を編集し、コントローラーから渡された会員リストを表示させましょう。併せて、CSS 等のリンクも修正します。

members/list.html (List 09B-2-4)

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>会員一覧</title>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
</head>
<body>

```

```

<div class="container">
<h1 class="mt-4">会員一覧</h1>

<!-- 操作に応じたメッセージを表示:start -->
<div class="alert alert-success mt-4">
    会員を追加しました等、操作に応じたメッセージ
</div>
<!-- 操作に応じたメッセージを表示:end -->

<p>
    <a class="mt-4 btn btn-primary" th:href="@{/members/add}">会員の追加</a>
    <a class="mt-4 ml-4 btn btn-secondary" th:href="@{/logout}">ログアウト
</a>
</p>
<table class="table">
    <tr>
        <th>ID</th>
        <th>氏名</th>
        <th>年齢</th>
        <th>住所</th>
        <th>会員種別</th>
        <th>登録日</th>
        <th colspan="2">操作</th>
    </tr>
    <tr th:each="member : ${members}">
        <td>[[${member.id}]]</td>
        <td>[[${member.name}]]</td>
        <td>[[${member.age}]]</td>
        <td>[[${member.address}]]</td>
        <td>[[${member.typeName}]]</td>
        <td>[[${#temporals.format(member.created,
                                'yyyy 年 MM 月 dd 日')}]]</td>
        <td><a class="btn btn-primary"
                th:href="@{/members/edit/{id}(id=${member.id})}">編集
            </a></td>
        <td><button class="btn btn-danger delete"
                th:data-href="@{/members/delete/{id}(id=${member.id})}"
                th:data-name="${member.name}" data-bs-toggle="modal"
                data-bs-target="#confirm-modal">削除</button></td>
    </tr>
</table>
</div>

<!-- 2 ページ以上の場合、ページネーションを表示:start -->
<div class="container">
    <ul class="pagination">
        <!-- 前のページへ -->
        <li class="page-item disabled"><a class="page-link"
            href="">&laquo;</a></li>

        <!-- ページ番号 -->

```

```

        <li class="page-item active"><a class="page-link"
href="">1</a></li>
        <li class="page-item"><a class="page-link" href="">2</a></li>
        <li class="page-item"><a class="page-link" href="">3</a></li>

        <!-- 次のページへ -->
        <li class="page-item"><a class="page-link" href="">&raquo;</a></li>
    </ul>
</div>
<!-- 2 ページ以上の場合、ページネーションを表示: end -->

<!-- Bootstrap5 Modal -->
<div class="modal fade" id="confirm-modal" tabindex="-1"
aria-labelledby="confirmModalLabel" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">会員情報の削除</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
            </div>
            <div class="modal-body">
                「<span id="delete-name"></span>」を削除します。よろしいですか?
            </div>
            <div class="modal-footer">
                <a class="btn btn-danger" id="delete-yes">はい</a>
                <button type="button" class="btn btn-secondary"
                    data-bs-dismiss="modal">いいえ</button>
            </div>
        </div>
    </div>
</div>
</div>
</div>

<script th:src="@{/js/bootstrap.bundle.min.js}"></script>
<script>
document
    .querySelectorAll(".delete")
    .forEach(btn => btn.addEventListener("click", e => {
        const name = e.target.getAttribute("data-name");
        document
            .getElementById("delete-name")
            .textContent = name;

        const href = e.target.getAttribute("data-href");
        document
            .getElementById("delete-yes")
            .setAttribute("href", href);
    }));
</script>
</body>
</html>

```

これで会員一覧ページは完成です。<http://localhost:8080/members> にアクセスし、会員情報が表示されることを確認しましょう。

実習課題 09B-2 完成時点の表示例

会員一覧							
会員を追加しました等、操作に応じたメッセージ							
会員の追加		ログアウト					
ID	氏名	年齢	住所	会員種別	登録日	操作	
1	福岡 弘義	27	静岡県	通常会員	2022年08月26日	編集	削除
2	大平 直紀	34	東京都	通常会員	2022年08月26日	編集	削除
3	田上 三起	18	栃木県	通常会員	2022年08月26日	編集	削除
4	黒川 希久	77	岡山県	プレミアム会員	2022年08月26日	編集	削除
5	大島 郁穂	65	福島県	通常会員	2022年08月26日	編集	削除
<< 1 2 3 >>							

■ 練習 09-3 : 会員削除機能の実装

会員一覧ページで削除ボタンを押すと、`http://localhost:8080/members/delete/{id}` という URL に遷移します。この URL にアクセスがあった際に、会員データを削除するメソッドをコントローラーに実装しましょう。

MemberController.java (List 09B-3-1)

```
package com.example.app.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.example.app.service.MemberService;

import lombok.RequiredArgsConstructor;

@Controller
@RequestMapping("/members")
```

```

@RequiredArgsConstructor
public class MemberController {

    private final MemberService service;

    @GetMapping
    public String list(Model model) throws Exception {
        model.addAttribute("members", service.getMemberList());
        return "members/list";
    }

    @GetMapping("/delete/{id}")
    public String delete(@PathVariable Integer id, RedirectAttributes rd)
        throws Exception {
        service.deleteMember(id);
        rd.addFlashAttribute("statusMessage", "会員情報を削除しました。");
        return "redirect:/members";
    }
}

```

次に members/list.html を修正します。

members/list.html (List 09B-3-2)

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>会員一覧</title>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
</head>
<body>
<div class="container">
<h1 class="mt-4">会員一覧</h1>

<!-- 操作に応じたメッセージを表示:start -->
<div th:if="${statusMessage}" class="alert alert-success mt-4">
    [[${statusMessage}]]
</div>
<!-- 操作に応じたメッセージを表示:end -->


<p>
    <a class="mt-4 btn btn-primary" th:href="@{/members/add}">会員の追加</a>
    <a class="mt-4 ml-4 btn btn-secondary" th:href="@{/logout}">ログアウト
</a>
</p>

... 以下略 ...

```

会員を削除するための URL にアクセスすると、会員データの削除と「会員情報を削除しました。」というフラッシュメッセージのセットが行われます。フラッシュメッセージは、リダイレクト先の会員一覧ページで一度だけ表示されます。

会員を削除した際、URL 末尾に `jsessionid` が表示されてしまう場合は、`application.properties` を修正しましょう。

 `localhost:8080/members;jsessionid=D295A2895D190D4C5252F3538B39493B`

`application.properties` (List 09B-3-3)

```
# URL 末尾の jsessionid への対応
server.servlet.session.tracking-modes=cookie

# データベース接続設定
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/mydb?useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Tokyo
spring.datasource.username=root
spring.datasource.password=

# MyBatis の設定
mybatis.mapper-locations=classpath*:mybatis/**/*.xml
mybatis.configuration.map-underscore-to-camel-case=true
mybatis.type-aliases-package=com.example.app.domain
```

■ 練習 09-4 : 会員の追加／編集機能の実装

会員の追加及び編集ページを作成していきます。これらのページは、似たような機能を有しているため、1つのビューファイル(`save.html`)で対応します。

まずは、コントローラーに追加／編集それぞれの URL に対応するメソッドを追記します。編集ページに対応するメソッドでは、URL 末尾の整数(会員 ID)を元に会員データを取得し、フォームの初期値として表示できるようにしています。

`MemberController.java` (List 09B-4-1)

```
package com.example.app.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
```

```

import com.example.app.domain.Member;
import com.example.app.service.MemberService;

import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;

@Controller
@RequestMapping("/members")
@RequiredArgsConstructor
public class MemberController {

    private final MemberService service;

    @GetMapping
    public String list(Model model) throws Exception {
        model.addAttribute("members", service.getMemberList());
        return "members/list";
    }

    @GetMapping("/add")
    public String addGet(Model model) throws Exception {
        model.addAttribute("title", "会員の追加");
        model.addAttribute("member", new Member());
        model.addAttribute("types", service.getTypeList());
        return "members/save";
    }

    @PostMapping("/add")
    public String addPost(
        @Valid Member member,
        Errors errors,
        RedirectAttributes rd,
        Model model) throws Exception {
        if(errors.hasErrors()) {
            model.addAttribute("title", "会員の追加");
            model.addAttribute("types", service.getTypeList());
            return "members/save";
        }

        service.addMember(member);
        rd.addFlashAttribute("statusMessage", "会員を追加しました。");
        return "redirect:/members";
    }

    @GetMapping("/edit/{id}")
    public String editGet(@PathVariable Integer id, Model model) throws
Exception {
        model.addAttribute("title", "会員情報の変更");
        model.addAttribute("member", service.getMemberById(id));
        model.addAttribute("types", service.getTypeList());
        return "members/save";
    }

```



```

    }

    @PostMapping("/edit/{id}")
    public String editPost(
        @PathVariable Integer id,
        @Valid Member member,
        Errors errors,
        RedirectAttributes rd,
        Model model) throws Exception {
        if(errors.hasErrors()) {
            model.addAttribute("title", "会員情報の変更");
            model.addAttribute("types", service.getTypeList());
            return "members/save";
        }

        member.setId(id); //更新に必要な会員 ID をセット
        service.editMember(member);
        rd.addFlashAttribute("statusMessage", "会員情報を更新しました。");
        return "redirect:/members";
    }

    @GetMapping("/delete/{id}")
    public String delete(@PathVariable Integer id, RedirectAttributes rd)
    throws Exception {
        service.deleteMember(id);
        rd.addFlashAttribute("statusMessage", "会員情報を削除しました。");
        return "redirect:/members";
    }
}

```

続いて、members/save.html を修正します。title 要素と h1 要素については、コントローラーから渡された文字列を出力し、会員の追加ページと編集ページで見た目を切り替えられるようにします。

members/save.html (List 09B-4-2)

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>[[${title}]]</title>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
<link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
<div class="container">
<h1 class="my-4">[[${title}]]</h1>
<form action="" method="post" th:object="${member}">
    <span th:errors="*{name}" class="error"></span>

```

```

<p class="mt-2">氏名:<input type="text" th:field="*{name}"></p>

<span th:errors="*{age}" class="error"></span>
<p class="mt-2">年齢:<input type="text" th:field="*{age}"></p>

<span th:errors="*{address}" class="error"></span>
<p class="mt-2">住所:<input type="text" th:field="*{address}"></p>

<p class="mt-2">会員種別:
    <select th:field="*{typeId}">
        <option th:each="type : ${types}" th:value="${type.id}"
            th:text="${type.name}"></option>
    </select>
</p>

<input class="btn btn-primary" type="submit">
</form>
<p class="mt-4"><a th:href="@{/members}">一覧に戻る</a></p>
</div>
</body>
</html>

```

ここまで出来たら、挙動を確認してみましょう。この時点では、SpringBoot であらかじめ用意されているエラーメッセージが表示されます。

このエラーメッセージをプロパティファイルによってカスタマイズしてみましょう。

src/main/resources 内に「validation.properties」を作成します。

validation.properties (List 09B-4-3)

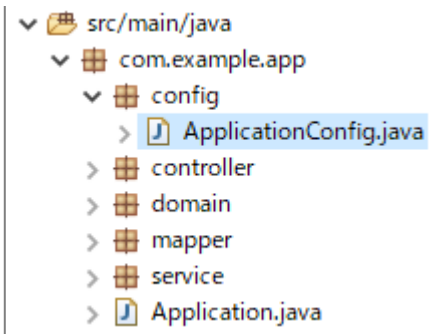
```

jakarta.validation.constraints.NotBlank.message={0}は必須項目です
org.hibernate.validator.constraints.Range.message={0}は{min}～{max}の間で入力してください
typeMismatch.java.lang.Integer={0}は整数で入力してください
jakarta.validation.constraints.Size.message={0}は{max}文字以内で入力してください

name=氏名
age=年齢
address=住所

```

最後に、このプロパティファイルを有効化します。com.example.app.config パッケージを作成し、ApplicationConfig クラスを配置します。



ApplicationConfig.java (List 09B-4-4)

```

package com.example.app.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.validation.Validator;
import
org.springframework.validation.beanvalidation.LocalValidatorFactoryBean;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class ApplicationConfig implements WebMvcConfigurer {

    // バリデーションメッセージのカスタマイズ
    @Override
    public Validator getValidator() {
        var validator = new LocalValidatorFactoryBean();
        validator.setValidationMessageSource(messageSource());
        return validator;
    }

    @Bean
    ResourceBundleMessageSource messageSource() {
        var messageSource = new ResourceBundleMessageSource();
        messageSource.setBasename("validation");
        return messageSource;
    }
}

```

これでエラーメッセージのカスタマイズができました。改めて、挙動を確認してみましょう。

エラーメッセージ時の表示例

会員の追加

氏名は必須項目です

氏名 :

年齢は整数で入力してください

年齢 :

住所 :

会員種別 : 通常会員 ▼

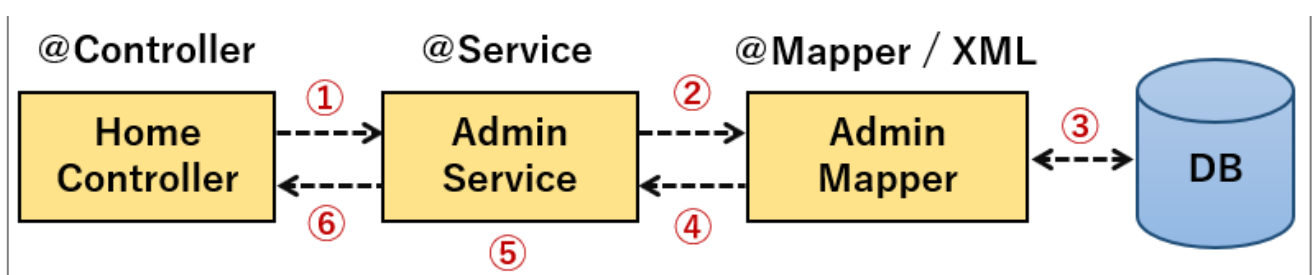
送信

■ 練習 09-5 : ログイン機能の実装

概要

あらたに HomeController クラスを作成し、トップページ(home.html)と連携させます。トップページにはログインフォームを表示させ、ログイン認証が行われた場合に、会員一覧ページとお知らせページへのリンクを表示させます。ログイン認証を経ず、会員一覧ページやお知らせページへのアクセスが行われた場合は、フィルターを使いトップページへリダイレクトします。

HomeController は、他に新たに作成する AdminService と AdminDao と連携して、ログイン認証を行います(下図)。



- ① HomeController は、フォームを通じて取得したログイン ID とパスワードを AdminService に渡す
- ② AdminService は、ログイン ID を AdminMapper に渡す
- ③ AdminMapper は、ログイン ID を元に admins テーブルから管理者データを取得する
- ④ AdminMapper は、管理者データを AdminService に渡す

- ⑤ AdminService が、フォームに入力されたパスワードと管理者データ内のパスワードが一致するか確認する
- ⑥ AdminService は、確認結果を true/false で HomeController に渡す

準備

まずは BCrypt を利用できるよう pom.xml に依存関係を追加します。

<https://mvnrepository.com/artifact/de.svenkubiak/jBCrypt/0.4.3>

pom.xml (List 09B-5-1)

... 省略 ...

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>3.0.2</version>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
</dependencies>
```

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter-test</artifactId>
  <version>3.0.2</version>
  <scope>test</scope>
</dependency>

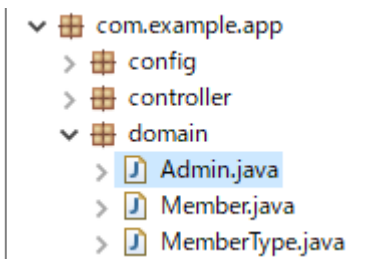
<!-- 別途追加 : BCrypt -->
<dependency>
  <groupId>de.svenkubiak</groupId>
  <artifactId>jBCrypt</artifactId>
  <version>0.4.3</version>
</dependency>
</dependencies>

```

... 省略 ...

※ 依存関係を追加した後は、アプリケーションを停止し、再実行しましょう。

続いて admins テーブルから取得するデータを保持するためのドメインクラスを作成します。
com.example.app.domain パッケージ内に Admin.java を作成しましょう。あらかじめバリデーション用のアノテーションも付与しておきます。



Admin.java (List 09B-5-2)

```

package com.example.app.domain;

import jakarta.validation.constraints.NotBlank;
import lombok.Data;

@Data
public class Admin {

    private Integer id;

    @NotBlank
    private String loginId;

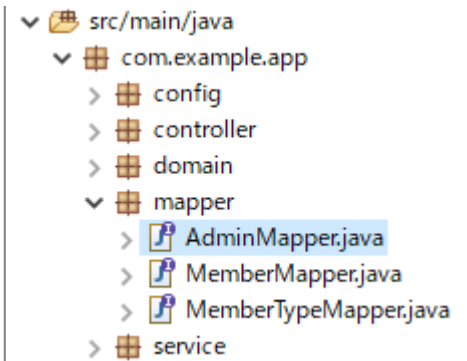
    private String loginPass;
}

```

```
}
```

Mapper の作成

com.example.app.mapper パッケージにインターフェース AdminMapper.java を作成します。
ログイン ID を元に管理者データを取得する selectByLoginId() メソッドを定義しましょう。



AdminMapper.java (List 09B-5-3)

```
package com.example.app.mapper;

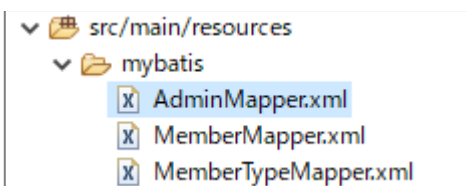
import com.example.app.domain.Admin;

public interface AdminMapper {

    Admin selectByLoginId(String loginId) throws Exception;

}
```

続いて、src/main/resources/mybatis 内に AdminMapper.xml を作成し、インターフェースの中で定義されている「selectByLoginId」に対応する SQL 文を記述します。



AdminMapper.xml (List 09B-5-4)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.app.mapper.AdminMapper">

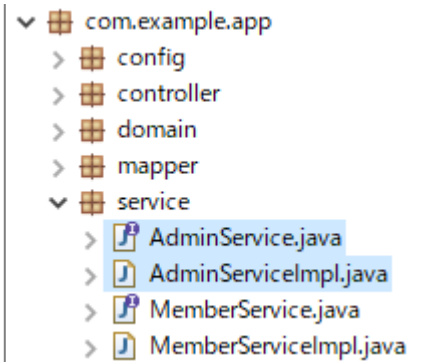
    <select id="selectByLoginId" resultType="Admin">
        SELECT * FROM admins WHERE login_id = #{loginId}
    </select>

</mapper>
```

```
</mapper>
```

Service の作成

com.example.app.service パッケージ内にインターフェース AdminService.java と、その実装クラス AdminServiceImpl.java を作成します。ログイン ID とパスワードを元に、それが正しい組み合わせか判断する isCorrectIdAndPassword() メソッドを定義しましょう。



AdminService.java (List 09B-5-5)

```
package com.example.app.service;

public interface AdminService {

    boolean isCorrectIdAndPassword(String loginId, String loginPass)
                                   throws Exception;

}
```

AdminServiceImpl.java (List 09B-5-6)

```
package com.example.app.service;

import org.mindrot.jbcrypt.BCrypt;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.app.domain.Admin;
import com.example.app.mapper.AdminMapper;

import lombok.RequiredArgsConstructor;

@Service
@Transactional(rollbackFor = Exception.class)
@RequiredArgsConstructor
public class AdminServiceImpl implements AdminService {

    private final AdminMapper mapper;

    @Override
```



```

    public boolean isCorrectIdAndPassword(String loginId, String loginPass)
    throws Exception {
        Admin admin = mapper.selectByLoginId(loginId);

        // ログイン ID が正しいかチェック
        //   ログイン ID が正しくなければ、管理者データは取得されない
        if(admin == null) {
            return false;
        }

        // パスワードが正しいかチェック
        if(!BCrypt.checkpw(loginPass, admin.getLoginPass())) {
            return false;
        }

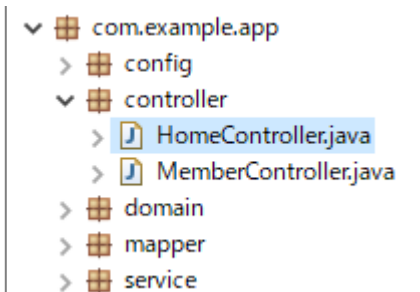
        return true;
    }
}

```

HomeController の作成

com.example.app.controller パッケージ内に、HomeController クラスを作成します。

トップページの URL に対応するメソッドに加え、ログアウトの URL に対応するメソッドも併せて実装していきます。



HomeController.java (List 09B-5-7)

```

package com.example.app.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

import com.example.app.domain.Admin;
import com.example.app.service.AdminService;

import jakarta.servlet.http.HttpSession;
import jakarta.validation.Valid;

```

```

import lombok.RequiredArgsConstructor;

@Controller
@RequiredArgsConstructor
public class HomeController {

    private final AdminService service;

    @GetMapping
    public String showHome(Model model) {
        model.addAttribute("admin", new Admin());
        return "home";
    }

    @PostMapping
    public String login(
        @Valid Admin admin,
        Errors errors,
        HttpSession session) throws Exception {
        // 入力に不備がある
        if(errors.hasErrors()) {
            return "home";
        }

        // パスワードが正しくない
        if(!service.isCorrectIdAndPassword(admin.getLoginId(),
admin.getLoginPass())) {
            errors.rejectValue("loginId", "error.incorrect_id_password");
            return "home";
        }

        // 正しいログイン ID とパスワード
        // セッションにログイン ID を格納し、リダイレクト
        session.setAttribute("loginId", admin.getLoginId());
        return "redirect:/";
    }

    @GetMapping("/logout")
    public String logout(HttpSession session) {
        // セッションを破棄し、トップページへ遷移
        session.invalidate();
        return "redirect:/";
    }
}

```

ログイン ID とパスワードの組み合わせが正しくない場合のメッセージを、validation.properties に追記します。

validation.properties (List 09B-5-8)

```
jakarta.validation.constraints.NotBlank.message={0}は必須項目です
org.hibernate.validator.constraints.Range.message={0}は{min}～{max}の間で入力してください
typeMismatch.java.lang.Integer={0}は整数で入力してください
jakarta.validation.constraints.Size.message={0}は{max}文字以内で入力してください
error.incorrect_id_password=ログイン ID またはパスワードが正しくありません

name=氏名
age=年齢
address=住所
loginId=ログイン ID
```

ビューの修正

templates/home.html に修正を加えます。セッションにログイン ID が格納されている場合は、ログイン認証済みと見なし、会員一覧ページとお知らせページへのリンクを表示させます。セッションにログイン ID が格納されていない場合は、ログインフォームを表示させます。

home.html (List 09B-5-9)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>会員管理システム</title>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
<link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
<div class="text-center mt-4">
<h1>会員管理システム</h1>

<th:block th:if="${session.loginId}">
    <a class="btn btn-primary btn-lg mt-4 mr-2" th:href="@{/members}">会員一覧</a>
    <a class="btn btn-primary btn-lg mt-4 ml-2" th:href="@{/news}">お知らせ一覧</a>
</th:block>

<th:block th:unless="${session.loginId}">
    <form class="mt-4" action="" method="post" th:object="${admin}">
        <span th:errors="*{loginId}" class="error"></span>
        <p class="mt-2">ログイン ID :
            <input type="text" th:field="*{loginId}"></p>
        <p>パスワード : <input type="password" th:field="*{loginPass}"></p>
        <input class="btn btn-primary" type="submit" value="ログイン">
    </form>
</th:block>
</div>
</body>
</html>
```

```

    </form>
</th:block>

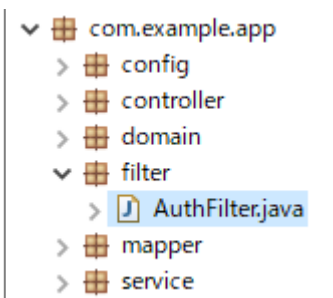
</div>
</body>
</html>

```

認証用フィルターの作成

最後に、認証用フィルターを作成すれば完成です。

com.example.app.filter パッケージを作成し、その中に AuthFilter クラスを配置します。セッションにログイン ID を保持していない場合、トップページへ強制的に遷移させる処理を記述します。



AuthFilter.java (List 09B-5-10)

```

package com.example.app.filter;

import java.io.IOException;

import jakarta.servlet.Filter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

public class AuthFilter implements Filter {

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;
        HttpSession session = req.getSession();

        if (session.getAttribute("loginId") == null) {
            res.sendRedirect("/");
            return;
        }
    }
}

```

```

        chain.doFilter(request, response);
    }
}

```

この処理を「/members」及び「/news」以下の URL で有効にします。
ApplicationConfig クラスを編集し、AuthFilter を有効化しましょう。

ApplicationConfig (List 09B-5-11)

```

package com.example.app.config;

import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.validation.Validator;
import org.springframework.validation.beanvalidation.LocalValidatorFactoryBean;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import com.example.app.filter.AuthFilter;

@Configuration
public class ApplicationConfig implements WebMvcConfigurer {

    // バリデーションメッセージのカスタマイズ
    @Override
    public Validator getValidator() {
        var validator = new LocalValidatorFactoryBean();
        validator.setValidationMessageSource(messageSource());
        return validator;
    }

    @Bean
    ResourceBundleMessageSource messageSource() {
        var messageSource = new ResourceBundleMessageSource();
        messageSource.setBasename("validation");
        return messageSource;
    }

    // 認証用フィルタの有効化
    @Bean
    FilterRegistrationBean<AuthFilter> authFilter() {
        var bean =
            new FilterRegistrationBean<AuthFilter>(new AuthFilter());
        bean.addUrlPatterns("/members/*");
        bean.addUrlPatterns("/news/*");
        return bean;
    }
}

```

これでログイン機能も完成しました。実行して挙動を確認してみましょう。

表示例: ホームページ

会員管理システム

ログインID :

パスワード :

ログイン

表示例: ホームページ (ログイン失敗時)

会員管理システム

ログインIDまたはパスワードが正しくありません

ログインID :

パスワード :

ログイン

表示例: ホームページ (ログイン成功時)

会員管理システム

会員一覧

お知らせ一覧