掲示板アプリケーション仕様書

管理番号	
システムID	
システム名称	
改定日	
改訂者	

目次

1.掲示板アプリケーションの概要

■概要：

本アプリケーションは、複数の掲示板を作成・管理できる掲示板型Webアプリケーションです。ユーザーはスレッド投稿やコメントを通じて交流することが可能です。
Struts2フレームワークをベースに、MVCアーキテクチャで構成されており、管理者と一般ユーザーのロールによる画面遷移や権限制御が実装されています。
また、ユーザーごとのスレッド・投稿の表示／編集／削除機能を備え、ユーザー体験を重視した設計となっています。

■対象ユーザー：

・管理者：掲示板、投稿、ユーザーの管理が可能
・一般ユーザー：掲示板の閲覧、スレッドや投稿の作成・編集・削除が可能

■技術構成：

・フレームワーク:Struts2
・データベース:MySQL（JDBC接続）
・画面技術（View）：JSP（Struts2タグライブラリ活用）
・アーキテクチャ:MVCモデル採用
・セッション管理：ログインセッションによるアクセス制御あり
・DBアクセス：DAOパターンによりデータベース操作をロジック層と分離

以上

2.掲示板アプリケーションの概要の定義

■概要（掲示板アプリケーションの構成定義）

本掲示板アプリケーションは、以下の4階層構造で構成され、それぞれに応じたアクターと作成タイミングが定義されています。
各階層は上位階層に属する形でネストされ、ユーザー同士の対話・情報共有が可能な構造を持ちます。

■階層定義

階層	概要	アクター	作成タイミング	例
掲示板（Board）	掲示板全体をカテゴリ単位で管理	管理者	サイト初期設定時、または新カテゴリ追加時	「プログラミング掲示板」
スレッド（Thread）	掲示板内の個別の話題	登録ユーザーまたは管理者	新しい話題について議論を始めたいとき	「Javaの使い方」
投稿（Post）	スレッド内での意見・質問	登録ユーザーまたは管理者	スレッドに対して意見・質問を投稿するとき	「Javaのクラスについて教えてください」
コメント（Comment）	投稿に対する返信・補足情報	登録ユーザーまたは管理者	特定の投稿に反応・補足したいとき	「クラスはOOPの基本です」

■階層構造（概念図）

```
掲示板（Board）：プログラミング掲示板
└ スレッド（Thread）：Javaの使い方
    ├投稿（Post）：Javaのクラスについて教えてください
    │ ├ コメント（Comment）：クラスはOOPの基本です
    │ └ コメント（Comment）：具体的にどの部分がわかりませんか？
    └ 投稿（Post）：Javaの例外処理について
        └ コメント（Comment）：try-catch文を使います
        └ スレッド（Thread）：Pythonのフレームワーク
        └ 投稿（Post）：DjangoとFlaskの違いは何ですか？
```

■用語定義

用語	定義
掲示板（Board）	複数のスレッドをまとめるカテゴリ単位の管理領域
スレッド（Thread）	掲示板内の議題や話題単位のトピック
投稿（Post）	スレッド内での意見・情報共有を行うメッセージ
コメント（Comment）	投稿に対する反応・補足を行うメッセージ

以上

3. 画面遷移図（ユーザー側）

[1.ログイン画面]
↓
[2.利用者ポータル画面]（※以下の遷移先は「掲示板一覧」「スレッド一覧」「ユーザー個別のスレッド一覧」、「ユーザー個別の投稿一覧」が表示されていることが前提）
└→ [3.掲示板詳細画面]
│　　└→ [4.スレッド作成画面]
│　　└→ [5.スレッド詳細画面]
│　　└→ [6.スレッド編集画面]
│　　├→ [7.投稿作成セクション]
│　　└→ [8.投稿編集/削除画面]
│
└→ [5.スレッド詳細画面]
│　　└→ [6.スレッド編集画面]
│　　├→ [7.投稿作成セクション]
│　　└→ [8.投稿編集/削除画面]
│
│
└→ [9.ユーザー個別のスレッド詳細画面]
│　　└→ [6.スレッド編集画面]
│　　├→ [7.投稿作成セクション]
│　　└→ [8.投稿編集/削除画面]
│
└→ [10.ユーザー個別の投稿に紐づくスレッド詳細画面]
　　└→ [6.スレッド編集画面]
　　├→ [7.投稿作成セクション]
　　└→ [8.投稿編集/削除画面]

＜各画面の説明＞
1.ログイン画面
・役割：ユーザーがシステムにログインするための画面。
・操作: ユーザー名とパスワードを入力し、「ログイン」ボタンをクリック。

2.利用者ポータル画面
・目的: ユーザーが以下の画面ににアクセスするためのメイン画面。
・掲示板詳細画面
・スレッド詳細画面
・ユーザー個別のスレッド詳細画面
・ユーザー個別の投稿に紐づくスレッド詳細画面
・操作: 各掲示板やスレッドのリンクをクリックして詳細画面へ移動。

3.掲示板詳細画面
・目的: 特定の掲示板内のスレッド一覧を表示。
・操作: 掲示板一覧の掲示板リンクをクリックし、掲示板詳細画面へ移動。

4..スレッド作成画面
・目的: 新しいスレッドを作成するための画面。
・操作: 掲示板詳細画面の「スレッド作成」リンクをクリックし、スレッド作成画面へ移動
スレッドタイトルを入力し、「作成」ボタンをクリック。

5.スレッド詳細画面
・目的: 特定のスレッド内のスレッドタイトルと投稿を表示。

・操作: 掲示板詳細画面のスレッド一覧からリンクをクリック
投稿閲覧し、新規投稿作成や、投稿に対する返信が可能。

6.スレッド編集画面
・目的: 自分のスレッドを編集するための画面。
・操作: スレッド詳細画面の「スレッド編集」ボタンをクリック
スレッドタイトルを編集し、「スレッド編集」ボタンをクリック。

7.投稿作成セクション
・目的: 新しい投稿を作成するためのセクション。
・操作: 内容を入力し、「投稿書込み」ボタンをクリック。
※投稿作成セクションは単独画面ではなく、スレッド詳細画面内の入力エリア（フォーム）である。

8.投稿編集/投稿削除の画面
・目的: 自分の投稿の編集と削除をするための画面。
・操作(画面遷移):投稿一覧の該当投稿の「投稿編集削除」ボタンをクリックして、画面へ遷移
・操作(投稿編集): 投稿内容を編集し、「投稿更新」ボタンをクリック。
・操作(投稿削除): 削除する投稿内容を確認の上、「投稿削除」ボタンをクリック。

9.ユーザー個別のスレッド詳細画面
・目的: 自分が作成したスレッドのスレッドタイトルと投稿を表示する
・操作: ユーザー個別のスレッド一覧からスレッドのリンクをクリックして、ユーザー個別のスレッド詳細画面へ移動。

10.ユーザー個別の投稿に紐づくスレッド詳細画面
・目的: 自分が作成した投稿に紐づく親スレッドの詳細画面を表示する
・操作: ユーザー個別の投稿に紐づくスレッド一覧から、リンクをクリックして、詳細画面へ移動

以上

4. 機能カテゴリー覧

No	機能カテゴリ	主な内容
1	共通基盤機能	セッション管理、認証情報保持、画面遷移時の共通処理など
2	認証機能	ログイン／ログアウト処理
3	管理機能	掲示板・ユーザー・投稿の管理
4	ユーザーポータル機能	ユーザー向けの掲示板・スレッド・投稿の一覧表示
5	掲示板機能	詳細表示やスレッド作成
6	スレッド機能	詳細表示と編集／削除
7	投稿機能	投稿処理・編集・削除

■BaseActionクラス

・概要

本アプリケーションでは、すべてのActionクラスが共通して持つ機能を集約するために、BaseActionクラスを設計・導入している。
Struts2フレームワークにおける共通親クラスとして機能し、アクション全体で共通して使用される基本処理を提供する。
セッション管理、リクエスト/レスポンス処理、エラーメッセージ管理などを一元化することにより、個別のアクションでの重複コードを排除し、メンテナンス性を高める。

・機能定義

項目	内容
機能名	共通アクションクラス（BaseAction）
対象画面	すべてのJSP画面（例: BulletinboardManagementScreen.jsp, PostEditScreen.jsp など）
呼び出しAction	各種アクションクラス（例 ListAction, CancelAction など）の親クラスとして機能
データ構造	下記参照
付加情報	- Struts2の SessionAware, ServletRequestAware, ServletResponseAware インターフェースを実装し、各種オブジェクトを管理。

・データ構造

◆共通プロパティ

項目名	形	説明
session	Map<String, Object>	ユーザーセッション情報を保持するマップ
request	HttpServletRequest	リクエストオブジェクト
response	HttpServletResponse	レスポンスオブジェクト
errorMessage	String	エラーメッセージ（必要に応じて設定される）
message	String	一般メッセージ（例: 成功メッセージなど）
userId	Integer	セッション中のユーザーID（ログイン済みの場合）
logger	Logger	デバッグやエラー記録のためのロガーインスタンス

・処理の流れ

1.各アクション（例: ListAction, CancelAction）がBaseActionを継承。
2.Struts2がAction実行時に、SessionAware・ServletRequestAware・ServletResponseAwareを通じて以下を自動セット。
・setSession(Map<String, Object> session)
・setServletRequest(HttpServletRequest request)
・setServletResponse(HttpServletResponse response)
3.各アクション内では、BaseActionで保持されたsession や request などを活用。
・例: セッションからログインユーザー情報を取得。
・例: 共通メッセージやエラー処理を呼び出し。
4.共通メソッドが必要な場合はBaseAction内で実装し、各子アクションが簡単に呼び出す。

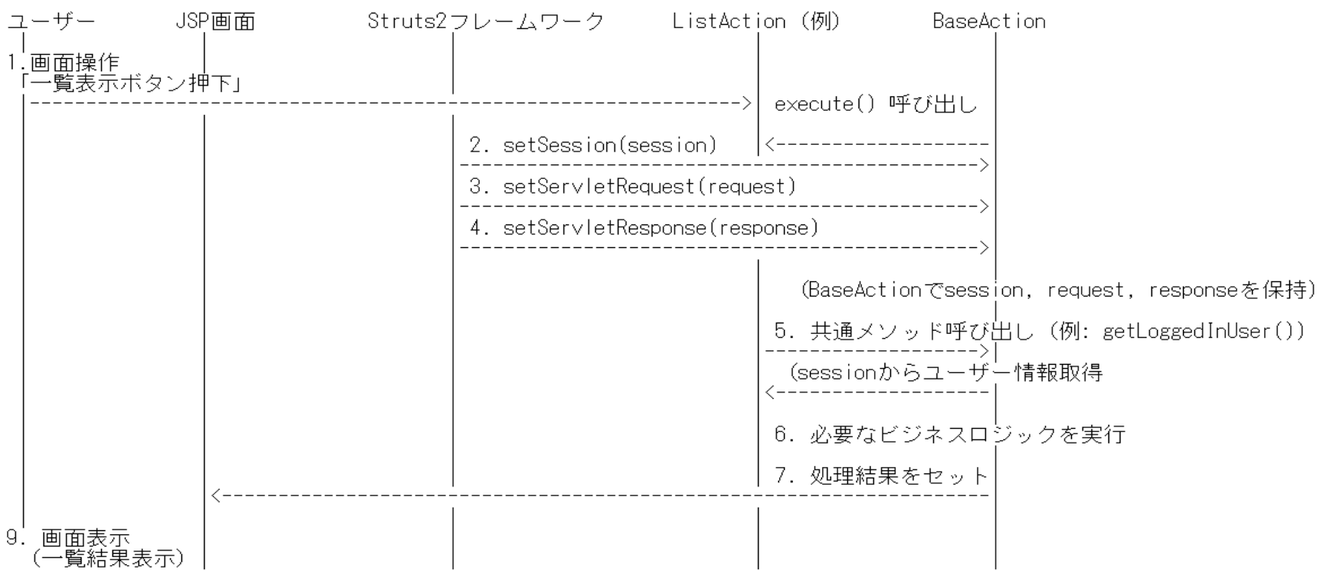
・シーケンス図（テキスト版）

＜BaseActionを継承した通常の処理フロー＞
1.ユーザー → JSP画面 → 任意のAction（例: ListAction）: execute() 呼び出し。
2.Struts2フレームワーク → BaseAction: setSession() / setServletRequest() / setServletResponse() を自動実行。
3.ListAction → BaseAction: 共通メソッドの呼び出し（例 ユーザー情報取得）。
4.ListAction → JSP画面: 処理結果を返却。

・クラス図

```
＜BaseAction（共通親クラス）＞
＋--------------------＋
| BaseAction          |（親アクション）
|--------------------|
| + setSession()     |
| + setServletRequest()|
| + setServletResponse()|
|--------------------|
|  session           |
|  request           |
|  response          |
|  errorMessage      |
|  message           |
＋--------------------＋
|
|  extended by
v
＋--------------------＋
| ListAction         |（子アクション）
|--------------------|
| + execute()        |
＋--------------------＋
```

■BaseActionクラスを利用した通常の処理シーケンス図



補足説明:

1. ユーザーがJSP画面上で操作し、アクション（例：ListAction）が呼ばれます。

2～4. Struts2フレームワークが自動的にSessionAware・ServletRequestAware・ServletResponseAwareを通してBaseActionのsetterを呼び、必要なオブジェクトを注入します。

（補足）

・Struts2のインターフェース動作（SessionAwareなど）

Struts2は、アクションが SessionAware などのインターフェースを実装していると、

1.インターフェースを確認
2.そのアクションインスタンス（ここではListAction）に setSession() などを呼び出す

public class ListAction extends BaseAction implements SessionAware, ServletRequestAware, ServletResponseAware {
// executeなど
}

しかし、実際は上記の通り、インターフェースを実装しているのは ListAction ですが、実際の setSession() の実装は **BaseAction** にあります。

つまり、
・フレームワークは ListActionを見て「SessionAwareを実装している」と認識
・実際に setSession(session) を呼ぶ
・そのメソッドは BaseAction に定義されているので BaseActionのメソッドが呼ばれる

5. ListActionはBaseActionを継承しているため、BaseAction内の共通メソッド（例：ログインユーザー情報取得）を簡単に呼び出します。
6〜7. ListActionはビジネスロジックを実行し、必要なデータをリクエスト属性などにセットします。
9. 処理結果がJSPに返り、画面が更新されます。

※この仕組みにより、複数のアクション（ListActionやCreateActionなど）で共通のセッション・リクエスト処理をBaseActionに一元化でき、コードの重複を防ぎ、保守性が向上する。

以上

■認証機能(ログイン)

・概要

ユーザー認証とセッション管理を行い、正規のログインユーザーだけが各機能へアクセスできるよう制御します。
入力されたユーザーIDとパスワードから、一致するユーザー情報をDBから取得し、認証処理を行う。
認証後にセッションにユーザー情報を保存し、権限ごとの画面に遷移させる。

・機能定義

項目	内容
機能名	ログイン機能
対象画面	login.jsp
呼び出しAction	LoginAction
データ構造	下記参照
付加情報	LoginActionから以下のクラスのメソッドが呼び出される

・データ構造

◆users（ユーザー情報）

項目名	形	説明
user_id	int	ユーザーID
user_name	varchar(20)	ユーザー名
password	varchar(60)	パスワード
auth_type	int	ユーザー権限
delete_flag	int	ユーザー削除フラグ
delete_day	datetime(6)	ユーザー削除日

・処理の流れ

1.ログイン画面でユーザー名とパスワードを入力後「ログイン」ボタンをクリック。
2.LoginActionを呼び出し。
3.入力値のバリデーション処理後、UserDAOクラスクラスの「authenticate(String user_name, String password)」メソッドを呼び出し
4.ユーザー認証処理後、認証結果の判定を行う。
5.認証結果の判定後、ユーザー情報をセッションに保存。
6.ユーザーの権限タイプに基づき適切な結果を返す（adminあるいはuser）

・シーケンス図（テキスト版）

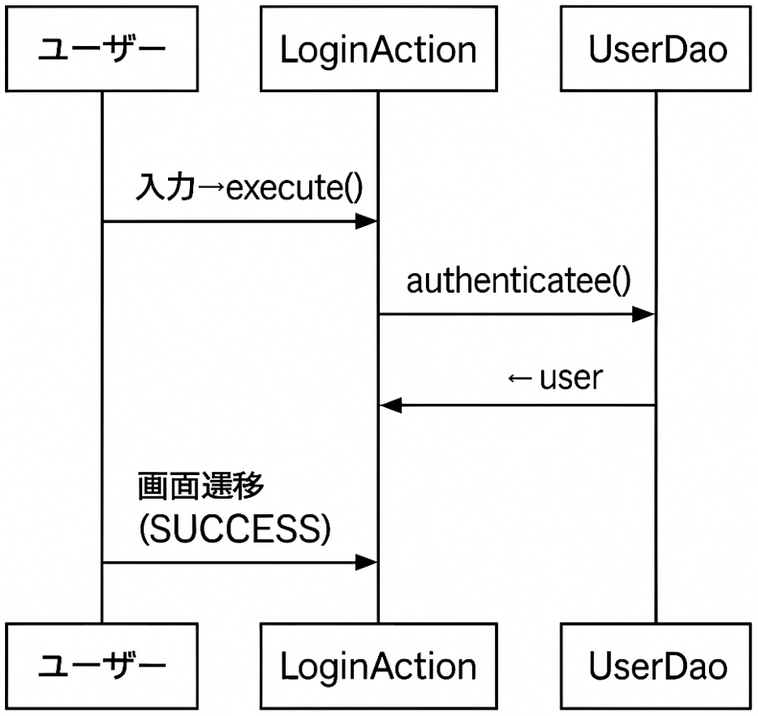
1.ユーザー → LoginAction : execute() 呼び出し
2.LoginAction → UserDao : authenticate(String user_name, String password)
3.UserDao → LoginAction: **user を返却**
4.LoginAction → ユーザー : 画面遷移 (SUCCESS)

・クラス図

＋----------------＋
| LoginAction |（アクション）
|-----------------|
＋ execute()
- userDao
＋--------+--------＋
| uses
v

＋----------＋
| UserDao |（データアクセス層）
|----------|
| + authenticate(String user_name, String password) |
＋----------＋

・シーケンス図（認証機能(ログイン)）



以上

■認証機能(ログアウト)

・概要

現在のセッションを取得し、セッションを無効化する。
セッション無効化後、ログイン画面に遷移させる。

・機能定義

項目	内容
機能名	ログアウト機能
対象画面	login.jsp
呼び出しAction	LogoutAction
データ構造	下記参照
付加情報	特になし

・データ構造

◆users（ユーザー情報）

項目名	形	説明
user_id	int	ユーザーID
user_name	varchar(20)	ユーザー名
password	varchar(60)	パスワード
auth_type	int	ユーザー権限
delete_flag	int	ユーザー削除フラグ
delete_day	datetime(6)	ユーザー削除日

・処理の流れ

1.任意の画面で「ログアウト」ボタンをクリック。
2.LogoutActionを呼び出し。
3.現在のセッションを取得。
4.セッションを無効化
5.ログイン画面に遷移する。

以上

■掲示板管理機能

※本ページについて

本ページで説明する管理機能は、掲示板・ユーザー・投稿の単純な CRUD操作が中心です。利用者側機能と異なり、特別な結合処理や複雑なロジックは含まれておりません。
そのため、機能の詳細な仕様説明は割愛し、必要最小限のポイントのみを記載しています。

・目的

アプリ内で扱う掲示板(Board)の情報を管理する機能。
管理者が新規で掲示板を作成し、既存掲示板の編集や削除を行いながら、掲示板の一覧表示通じて全体を把握できえう。

・関連クラス・DAO一覧

処理内容	クラス名・DAO
掲示板作成画面への遷移	MoveCreateBulletinboardAction
掲示板作成処理(DB登録)	InsertBulletinboardAction
掲示板編集画面への遷移	EditBulletinboardAction
掲示板更新処理(DB更新)	UpdateBulletinboardAction
掲示板削除処理	DeleteBulletinboardAction
キャンセル処理(一覧へ戻る)	CancelAction, ListAction, BoardDAO

・補足
アクションごとにクラスを分離(InsertBulletinboardAction など)することで、単一責任の原則(SRP)を実現。
CancelAction や ListAction は単なる戻り操作ではなく、最新状態の掲示板一覧表示を担保する重要な導線。
BoardDAO を利用した一覧取得処理により、常にDBの最新情報に同期されている安心設計。
ロガーによる デバッグ補助と障害発見のしやすさを重視。

・【使用技術】

- Java + Struts2：アクション毎に責務を分離し、保守性を意識
- JDBC：PreparedStatementによるSQL実行(SELECT / INSERT / UPDATE / DELETE)
- DAOパターン：DBアクセスをBoardDAOに集約
- ログ出力：Log4jを用いて操作ログ／エラーを記録

■ユーザー管理機能

・目的

システムにアクセスするユーザー情報(ID、名前、パスワード、権限など)を管理する機能。
管理者がユーザーの登録・編集・削除を行えるようにすることで、システムのセキュリティと柔軟な運用を実現。
「キャンセル」ボタン処理時にも最新のユーザー情報一覧を常に反映し、管理画面としての整合性を保つ。

・主な機能と処理内容

機能	アクション / DAO	処理概要
ユーザー作成画面表示	MoveCreateUserAction	ユーザー新規作成画面へ遷移
ユーザー情報の登録処理	InsertUserAction	入力されたユーザー情報をDBに登録
ユーザー編集画面表示	EditUserAction	選択されたユーザーの情報を編集画面に表示
ユーザー情報更新処理	UpdateUserAction	編集されたユーザー情報をDBに更新
ユーザー削除処理	DeleteUserAction	指定されたユーザーを論理削除(delete_flagの更新)
キャンセルボタン処理(一覧へ戻る)	UserCancelAction	編集・削除中のキャンセル時にユーザー一覧を再取得・表示(常に最新状態)

・使用技術

Java(Struts2)：MVCアーキテクチャを活用しアクションごとに機能を分離
JDBC(PreparedStatement)：SQLインジェクション対策を施した安全なDBアクセス
DAOパターン(UserListDAO)：データアクセスを一元化し保守性を向上
Log4j：ログ出力によるデバッグ支援と障害時の原因調査
セッション管理：ログイン情報の保持や権限に応じた画面制御に活用
論理削除(delete_flag)：物理削除せずにデータを管理

・補足

再利用性の高いDAO設計
→ 一覧取得処理はUserListDAO に集約し、通常遷移・キャンセル遷移の両方で使用することでロジックを一本化。
キャンセル時の最新データ反映
→「キャンセル」ボタンで元の一覧画面に戻る際、常に最新のデータを表示するようにしてユーザー体験を向上。
論理削除でデータ保持・安全性確保
→ 削除処理は delete_flag による制御を採用し、後から復元可能な設計に。
明確なアクション設計
→ 機能単位でアクションクラスを分け、処理の見通しを良くし、保守性の高い構成に。

以上

■ユーザーポータル機能（掲示板一覧表示機能）

・概要
全掲示板情報を取得し、ユーザーポータル画面（userPortal.jsp）に表示する機能。

・機能定義

項目	内容
機能名	掲示板一覧取得
対象画面	userPortal.jsp
呼び出しAction	MoveBulletinboardManagementAction
データ構造	bulletinboards（セッションまたはActionクラスのフィールドで保持されるリスト）
付加情報	・ユーザー認証処理後、LoginActionクラスから「user」の値が渡され、ユーザーポー

・データ構造
◆bulletinboards（掲示板情報）

項目名	形	説明
bulletinboard_id	int	掲示板ID
bulletinboard_title	varchar(191)	掲示板タイトル
bulletinboard_content	text	掲示板本文
user_id	int	掲示板作成者のユーザーID
bulletinboard_delete_flag	int	掲示板削除フラグ
bulletinboard_delete_day	datetime(6)	掲示板削除日

・処理の流れ（フロー）
1.ユーザがログイン後、セッションにuser_id を保持
2. userPortal.jsp のページがロードされる際に、MoveBulletinboardManagementAction が実行される
3.MoveBulletinboardManagementAction 内で以下の処理を実行：
a. DAOクラス（例：BulletinboardDao）を呼び出して、全掲示板情報をDBから取得
b. 取得した掲示板情報リストをbulletinboards フィールドに保持
4.Struts2 により bulletinboards が userPortal.jsp に渡される
5.userPortal.jsp にて <s:iterator> タグ等を使って、掲示板一覧を表示
※各掲示板タイトルをクリックすることで、該当の掲示板詳細画面へ遷移するリンクを設定

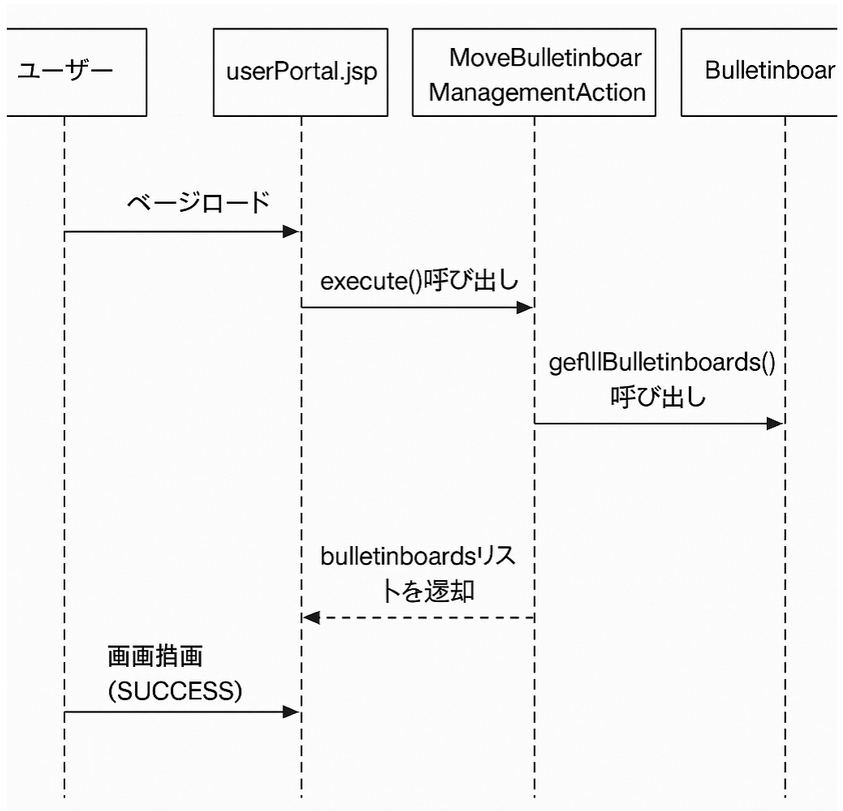
・シーケンス図（テキスト版）
1.userPortal.jsp → MoveBulletinboardManagementAction : execute() 呼び出し
2.MoveBulletinboardManagementAction → BulletinboardDao : getAllBulletinboards()
3.BulletinboardDao → MoveBulletinboardManagementAction : bulletinboardsリストを返却
4.MoveBulletinboardManagementAction → userPortal.jsp : データ転送（SUCCESS表示）

・クラス図（テキスト版）
```
＋----------------------------------------＋
| MoveBulletinboardManagementAction |（アクション）
|----------------------------------------|
```

＋ execute()
・bulletinboardDao
＋----------+----------＋
| uses
v

＋----------＋
| BulletinboardDao |（データアクセス層）
|----------|
| + getAllBulletinboards() |
＋----------＋

・シーケンス図



以上

■ユーザーポータル機能（スレッド一覧表示機能）

・概要

全スレッド情報を取得し、スレッドごとの投稿数とともにユーザーポータル画面(userPortal.jsp)に表示する機能。

・機能定義

項目	内容
機能名	スレッド一覧取得
対象画面	userPortal.jsp
呼び出しAction	GetThreadAction
データ構造	threads（セッションまたはActionクラスのフィールドで保持されるリスト）
付加情報	・ユーザー認証処理後、LoginActionクラスから「user」の値が渡され、ユーザーポータ

・データ構造

項目名	型	説明
thread_id	int	スレッドID
thread_title	String	スレッドタイトル
create_date	Timestamp	スレッド作成日時
user_id	int	作成者のユーザーID
post_count	int	このスレッドに紐づく投稿数

・処理の流れ（フロー）

1. ログイン成功後、セッションにuser_id を保持
2. userPortal.jsp のページがロードされる際にGetThread アクションが呼び出される
3. GetThread 内で以下の処理を実行：

a. 全スレッドを取得(threads リストを操作）

b. PostDaoクラスの「getPostCountByThreadId(thread.getThread_id())」メソッドが呼び出され、各スレッドに対し、紐づく投稿数をカウント。

c.各スレッドの投稿件数を「threads リスト」に入れる。

4. 処理結果をフィールドに保持し、Struts2 により JSP へ渡される
5. JSP にて全スレッド一覧と投稿件数を表示

・シーケンス図（テキスト版）

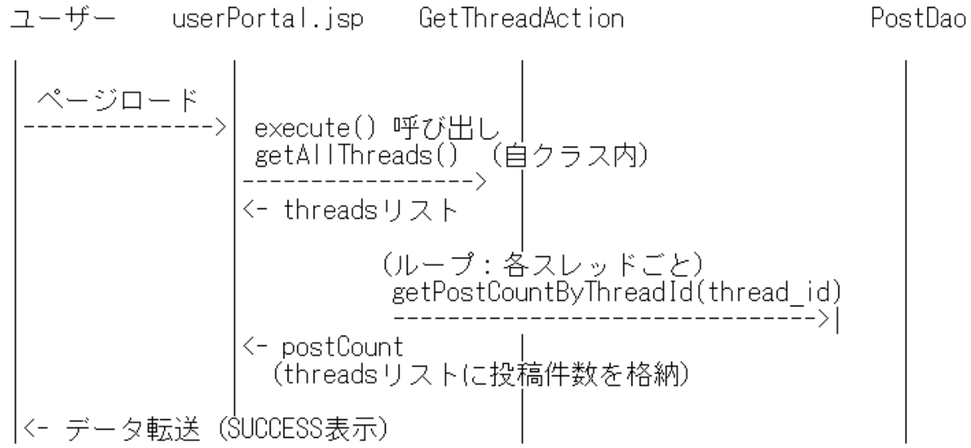
1.userPortal.jsp → GetThread : execute() 呼び出し
2.GetThread : getAllThreads() : threadsリストを返却 ※自クラス内のメソッド呼び出しと処理結果の受け取り
3.GetThread → PostDao : getPostCountByThreadId(int thread_id)
4.PostDao → GetThread : postCountの値を返却
5.GetThread → userPortal.jsp : データ転送（SUCCESS表示）

・クラス図

```
＋---------------------------------＋
| GetThreadAction |（アクション）
|---------------------------------|
＋ execute()
＋ getAllThreads()
・postDao
＋----------------+---------------＋
            | uses
            v
＋--------------------------＋
| PostDao |（データアクセス層）
|--------------------------|
| + getPostCountByThreadId(int thread_id)
＋--------------------------＋
```

以上

・シーケンス図



■ユーザーポータル機能（ユーザー個別のスレッド一覧表示機能）

・概要

ログイン中のユーザーに紐づいたスレッド情報を取得し、スレッドごとの投稿数とともにユーザーポータル画面(userPortal.jsp`)に表示する機能。

・機能定義

項目	内容
機能名	ユーザー個別のスレッド一覧取得
対象画面	userPortal.jsp
呼び出しAction	GetThreadByIdAction
データ構造	以下を参照
付加情報	・ユーザー認証処理後、LoginActionクラスから「user」の値が渡され、ユー

・データ構造

項目名	型	説明
thread_id	int	スレッドID
thread_title	String	スレッドタイトル
create_date	Timestamp	スレッド作成日時
user_id	int	作成者のユーザーID
post_count	int	このスレッドに紐づく投稿数

post_count は、表示用にAction内で動的に追加される想定

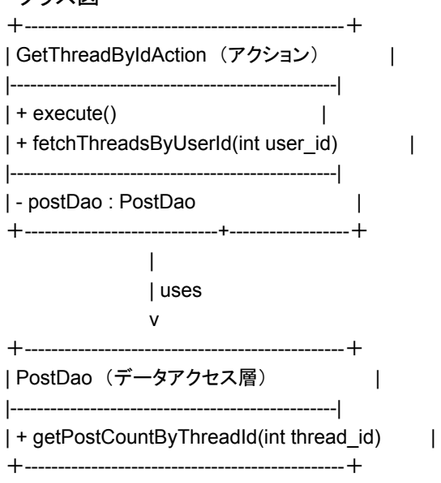
・処理の流れ（フロー）

1. ログイン成功後、セッションにuser_id を保持
2. userPortal.jspのページがロードされる際に、GetThreadByIdActionアクションが呼び出される
3. GetThreadByIdAction 内で以下の処理を実行：
a. user_id を基にユーザーに紐づくスレッドを取得（threads リストを操作）
b. 各スレッドに対し、紐づく投稿数をカウント（PostDaoクラスの「getPostCountByThreadId(thread.getThread_id())」メソッドを呼び出し処理する）
4. 処理結果をフィールドに保持し、Struts2 により JSP へ渡される
5. JSP にてスレッド一覧と投稿件数を表示

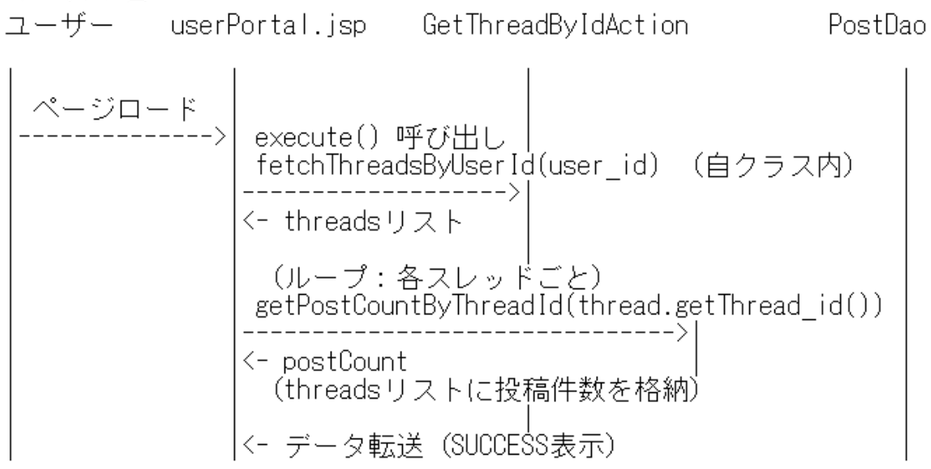
・シーケンス図（テキスト版）

1.userPortal.jsp → GetThreadByIdAction : execute() 呼び出し
2.GetThreadByIdAction : fetchThreadsByUserId(int user_id) : threadsリストを返却 ※自クラス内のメソッド呼び出しと処理結果の受け取り
3.GetThreadByIdAction → PostDao : getPostCountByThreadId(thread.getThread_id())
4.PostDao → GetThreadByIdAction : postCountの値を返却　※postCountの値をthreadsリストに格納する
5.GetThreadByIdAction → userPortal.jsp : データ転送（SUCCESS表示）

・クラス図



・シーケンス図



■ユーザーポータル機能（ユーザー個別の投稿一覧表示機能）

・概要
ログイン中のユーザーに紐づいた投稿情報を取得し、ユーザーポータル画面（userPortal.jsp）に表示する機能。

・機能定義

項目	内容
機能名	ユーザー個別の投稿一覧取得
対象画面	userPortal.jsp
呼び出しAction	GetPostByIdAction
データ構造	以下を参照
付加情報	JSP側で投稿に紐づく投稿Dを表示

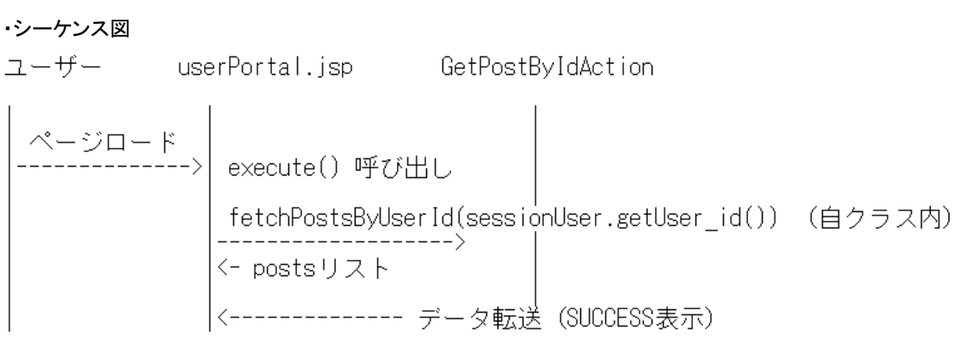
・データ項目

項目名	型	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	スレッドID
user_id	int	ユーザーID
post_delete_flag	int	投稿削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime	投稿タイムスタンプ

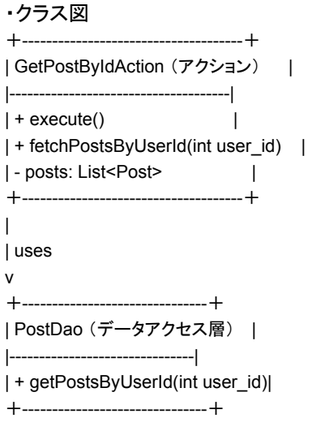
・処理の流れ（フロー）
1. ログイン成功後、セッションにuser_id を保持
2. userPortal.jspのページがロードされる際に、GetPostByIdActionアクションが呼び出される
3. GetPostByIdAction 内で以下の処理を実行：
a. user_id を基に画面表示で使用するユーザー個別の投稿リストを取得（posts リストを操作）
4. 処理結果をフィールドに保持し、Struts2 により JSP へ渡される
5. JSP にてユーザー個別の投稿を表示

・シーケンス図（テキスト版）
1.userPortal.jsp → GetPostByIdAction : execute() 呼び出し
2.GetPostByIdAction : fetchPostsByUserId(sessionUser.getUser_id()) : postsリストを返却 ※自クラス内のメソッド呼び出しと処理結果の受け取り
3.GetPostByIdAction → userPortal.jsp : データ転送（SUCCESS表示）

・シーケンス図



・クラス図



・補足

SQL文の解説　(該当のSQL文)

```
SELECT
 posts.*,
 threads.thread_id
FROM
 posts.*,
JOIN
 threads
 ON posts.thread_id = threads.thread_id
WHERE
 (posts.post_delete_flag IS NULL OR posts.post_delete_flag = 0)
 AND posts.user_id = ?;
```

1. SELECT句の threads.thread_id の部分

```
SELECT
 posts.*,
 threads.thread_id
```

※ここで、posts.* に加えて threads.thread_id を明示的に取得することで、投稿に紐づくスレッドの情報も取得している。

2. JOIN句の ON posts.thread_id = threads.thread_id の部分

```
JOIN
 threads
 ON posts.thread_id = threads.thread_id
```

※「 threads 」テーブルと「posts」テーブルを結合する

※この JOIN により、「投稿とスレッドがthread_id をキーに結び付けられている」

※そのため、「対応するスレッド情報」を取得できる仕組みが成立しています。

まとめ：

SELECT にスレッド情報を含めたいからthreads.thread_id を追加。

どのスレッド情報を紐づけるか明確にするためにJOIN threads ON posts.thread_id = threads.thread_id。

■掲示板詳細表示機能

・概要

利用者ポータル画面より、掲示板一覧から、該当の掲示板のリンクをクリックした際に、
掲示板詳細画面に遷移する機能

・機能定義

項目	内容
機能名	掲示板詳細表示機能
対象画面	BulletinboardDetailScreen.jsp
呼び出しAction	BoardDetailActio
データ構造	下記参照
付加情報	BoardDetailActioから以下のクラスのメソッドが呼び出される

・データ構造

◆bulletinboard（掲示板情報）

項目名	形	説明
bulletinboard_id	int	掲示板ID
bulletinboard_title	varchar(191)	掲示板タイトル
bulletinboard_content	text	掲示板本文
user_id	int	掲示板作成者のユーザーID
bulletinboard_delete_flag	int	掲示板削除フラグ
bulletinboard_delete_day	datetime(6)	掲示板削除日

◆Thread（スレッド情報）

項目名	形	説明
thread_id	int	スレッドID
thread_title	varchar(191)	スレッドタイトル
bulletinboard_id	int	親の掲示板ID
user_id	int	スレッド作成ユーザーID
thread_delete_flag	int	スレッド削除フラグ
thread_delete_day	datetime(6)	スレッド削除日

◆Post（投稿情報・リスト形式）

項目名	形	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	投稿の親スレッドID
user_id	int	投稿を作成したユーザーID
post_delete_flag	int	投稿の削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime(6)	投稿のタイムスタンプ

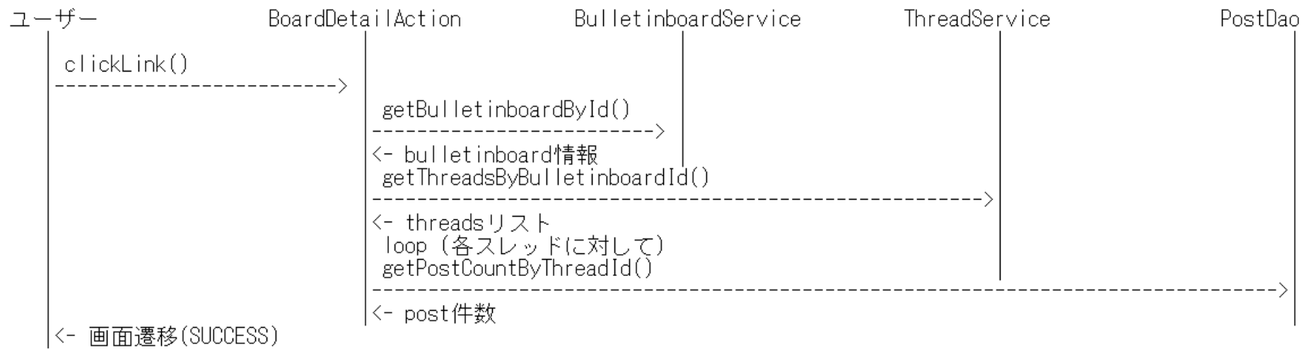
・処理の流れ

1.ユーザーがポータル画面で「掲示板一覧」の該当リンクをクリック。
2.BoardDetailActionを呼び出し。
3.リクエストパラメータとして受け取った文字列の掲示板IDを数値に変換
4.掲示板IDから掲示板の詳細情報を取得する。
5.各スレッドの投稿件数を取得する
6.BulletinboardDetailScreen.jspへ遷移し、取得データを表示する。

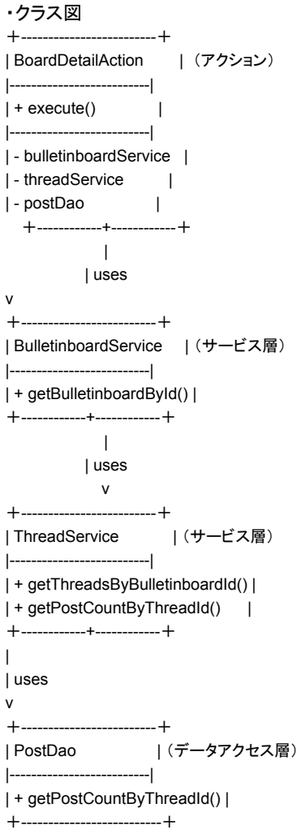
・シーケンス図（テキスト版）

1.ユーザー → BoardDetailAction : execute() 呼び出し
2.BoardDetailAction → BulletinboardService : getBulletinboardById(int id)
3.BulletinboardService → BoardDetailAction : bulletinboard情報を返却
4.BoardDetailAction → ThreadService : getThreadsByBulletinboardId(int bulletinboardId)
5.ThreadService → BoardDetailAction : threadsを返却
6.BoardDetailAction → PostDao : getPostCountByThreadId(int thread_id)
7.PostDao → BoardDetailAction : threadsのpostCountの値を返却
8.BoardDetailAction → ユーザー : 画面遷移 (SUCCESS)

・シーケンス図



・クラス図


BoardDetailAction （アクション）
+ execute()
- bulletinboardService
- threadService
- postDao
uses
BulletinboardService （サービス層）
+ getBulletinboardById()
uses
ThreadService （サービス層）
+ getThreadsByBulletinboardId()
+ getPostCountByThreadId()
uses
PostDao （データアクセス層）
+ getPostCountByThreadId()

以上

■スレッド作成機能

・概要

掲示板詳細画面より、「スレッド作成」ボタンをクリックする。スレッド作成画面に遷移される。
スレッド作成画面にて「スレッドタイトル」を入力し、「作成」ボタンをクリックする。スレッド作成処理が実行される。
スレッドの作成処理後、掲示板詳細画面のスレッド一覧が最新状態で表示される。

・機能定義

項目	内容
機能名	スレッド作成機能
対象画面	BulletinboardDetailScreen.jsp
呼び出しAction	MoveCreateThreadAction
データ構造	下記参照
付加情報	①MoveCreateThreadActionから以下の処理が実行される a.URLパラメータとして渡されたbulletinboard_id を取得。 ② ①の処理を受けて、Struts2フレームによってCreateThreadScreenに処理がリダイレクトされる、 リダイレクト後、CreateThreadScreenAction呼び出され以下が実行される a.bulletinboard_idを文字列から整数に変換して、クラスのフィールドbulletinboard_idにセットし、値を保持 ③スレッド作成画面にて「スレッドタイトル」入力し、「作成」ボタンをクリック後InsertThreadActionが 以下の処理を実効する。 a.取得した情報を元に、スレッド登録処理を実効する。 b.処理成功時にSUCCESSの文字列を返す ④ ③の処理後、ThreadListActionへ処理がリダイレクトされ、以下が実行される a.掲示板詳細画面とそれに紐づくスレッド一覧の再取得処理を行う。 ・BulletinboardServiceクラスの「getBulletinboardById(bulletinboardId)」メソッド呼び出し※掲示板情報の取得 ・ThreadServiceクラスの「getThreadsByBulletinboardId(bulletinboardId)」メソッド呼び出し※掲示板に紐づくスレッド一覧取得 ・PostDaoクラスの「getPostCountByThreadId(thread.getThread_id())」メソッド呼び出し※スレッドに紐づく投稿カウント用

・データ構造

◆bulletinboard（掲示板情報）

項目名	形	説明
bulletinboard_id	int	掲示板ID
bulletinboard_title	varchar(191)	掲示板タイトル
bulletinboard_content	text	掲示板本文
user_id	int	ユーザーID
bulletinboard_delete_flag	int	掲示板削除フラグ
bulletinboard_delete_day	int	掲示板削除日

◆Thread（スレッド情報）

項目名	形	説明
thread_id	int	スレッドID
thread_title	varchar(191)	スレッドタイトル
bulletinboard_id	int	親の掲示板ID
user_id	int	スレッド作成ユーザーID
thread_delete_flag	int	スレッド削除フラグ
thread_delete_day	datetime(6)	スレッド削除日

◆Post（投稿情報・リスト形式）

項目名	形	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	投稿の親スレッドID
user_id	int	投稿を作成したユーザーID
post_delete_flag	int	投稿の削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime(6)	投稿のタイムスタンプ

・処理の流れ

＜スレッド作成画面の表示＞
1.掲示板詳細画面より、「スレッド作成」ボタンをクリックする。
2.BulletinboardDetailScreen.jspにより、Struts2フレームワークを介して「MoveCreateThreadAction」クラスを呼び出し。
3.MoveCreateThreadActionクラス内にて以下を実効。
a.MoveCreateThreadActionにリダイレクトされたとき、MoveCreateThreadAction クラス内でURLパラメータとして渡された「bulletinboard_id」 を取得、「SUCCESS」を返す。
4. 3.の処理結果からStruts2フレームワークによって処理が「CreateThreadScreenAction」にリダイレクトされる。以下が実行される。
a.MoveCreateThreadActionクラスから、処理がリダイレクションされるbulletinboard_id」パラメータが引き継がれる。
HTTPリクエストのURLパラメータとして渡された「bulletinboard_id」を文字列から整数に変換して、クラスのフィールドbulletinboard_idにセット」し、値を保持する。
値を保持し、「SUCCESS」を返す。
5. 4.の処理後、「スレッド作成」画面（CreateThreadScreen.jsp）が表示される。
＜スレッド登録処理＞
6.「スレッド作成」（CreateThreadScreen.jsp）画面にて「スレッドタイトル」入力後に、「作成」ボタンクリックするとStruts2フレームワークにより「InsertThreadAction」が呼び出される。以下の処理が実行される。
a.スレッド作成画面のスレッド情報と、掲示板IDを取得し保持する。合わせてセッションからユーザーIDを取得する。スレッド情報はthreadモデルに保存。
b.入力値のバリデーション処理を実効。
c.メイン処理にて、SQLの登録処理を実効する際に、各プレースフォルダとthreadオブジェクトのプロパティ（掲示板ID、スレッドタイトル等）をバインドする。SQLを実効する。
d.「SUCCESS」を返す
7. 6.の処理からStruts2フレームにて処理をThreadListActionクラスへリダイレクトする。
＜スレッド作成処理後のスレッド一覧再取得処理＞
8.ThreadListActionクラスにて以下を実効。
a.ThreadListActionクラス内の「getBulletinboardDetails(int bulletinboardId)」メソッドにて、「BulletinboardService」クラスの「getBulletinboardById(bulletinboardId)」メソッドが呼び出される。
取得した掲示板IDから、掲示板情報を取得し、bulletinboardモデルに保存する。
b.ThreadListActionクラス内で、「getThreadsByBulletinboardId」メソッドが「ThreadService」クラスの「getThreadsByBulletinboardId(bulletinboardId)」メソッドが呼び出される。
スレッドデータ取得処理後、threadsに保存。
c.ThreadListActionクラス内で、「各スレッドに対して投稿件数を取得し、それをスレッドオブジェクトに格納する。」処理を実効し、結果の投稿件数をthreads」リストに代入。
d.「SUCCESS」を返す
9.掲示板詳細画面（BulletinboardDetailScreen.jsp）に遷移し、掲示板情報とスレッド一覧が最新の状態で表示される。

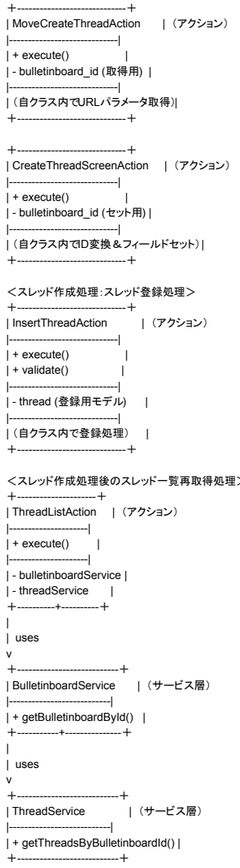
・シーケンス図（テキスト版）

＜スレッド作成処理：スレッド作成画面の表示＞
1.ユーザー → BulletinboardDetailScreen.jsp → MoveCreateThreadAction : execute() 呼び出し
2.MoveCreateThreadAction ： 自クラス内でURLパラメータとして渡された「bulletinboard_id」 を取得、「SUCCESS」を返す。
3.struts.xml → CreateThreadScreenAction ※フレームワークにより処理がリダイレクト
4.CreateThreadScreenAction：自クラス内で「bulletinboard_id」を文字列から整数に変換して、クラスのフィールドbulletinboard_idにセット、「SUCCESS」を返す。
＜スレッド作成処理：スレッド登録処理＞
1.CreateThreadScreen.jsp → InsertThreadAction: execute() 呼び出し
2.InsertThreadAction ： 自クラス内で掲示板ID、スレッドIDとフォームから受け取った入力値をthreadモデルに保存
3.InsertThreadAction → validate()：バリデーション処理を実効
4.InsertThreadAction ： 自クラス内で登録処理を実効
5.InsertThreadAction → 「SUCCESS」を返す
6.struts.xml → ThreadListAction ※フレームワークにより処理がリダイレクト
＜スレッド作成処理後のスレッド一覧再取得処理＞
1.ThreadListAction → BulletinboardService : getBulletinboardById(bulletinboardId)
2.BulletinboardService → ThreadListAction：bulletinboard
3.ThreadListAction → ThreadService : getThreadsByBulletinboardId(bulletinboardId)
4.ThreadService→ ThreadListAction：threads

5.ThreadListAction：各スレッドに対して投稿件数を取得し、それをスレッドオブジェクトに格納する。処理を実効し、結果の投稿件数者threads」リストに代入。
6.ThreadListAction →「SUCCESS」を返す
7.struts.xml → BulletinboardDetailScreen.jsp → ユーザー：画面遷移 (SUCCESS) ※フレームワークにより処理がリダイレクト

・クラス図

＜スレッド作成処理：スレッド作成画面の表示＞

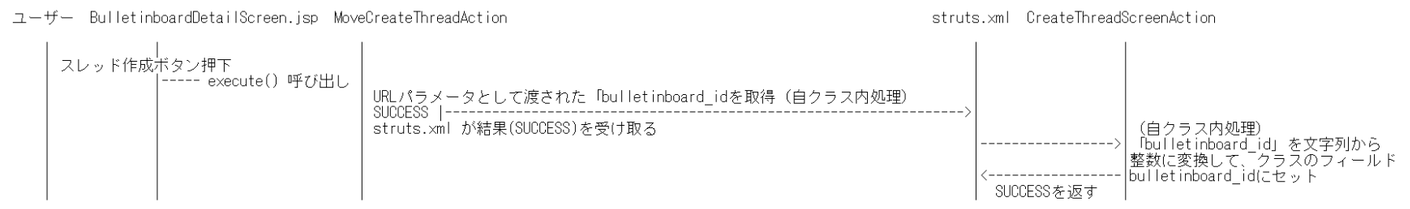


＜スレッド作成処理：スレッド登録処理＞

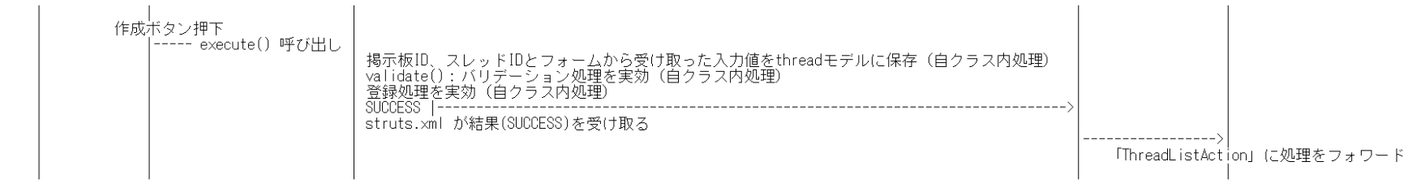
＜スレッド作成処理後のスレッド一覧再取得処理＞

・シーケンス図

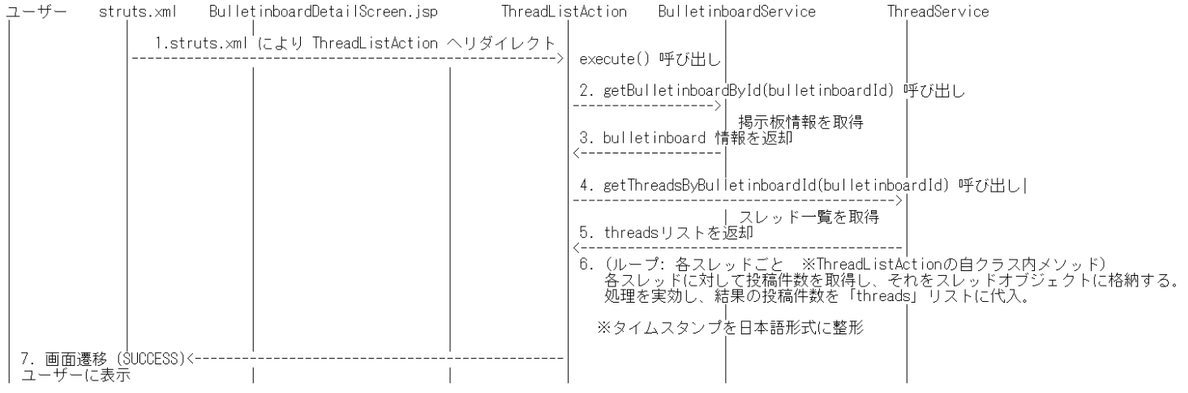
＜スレッド作成処理：スレッド作成画面の表示処理＞



＜スレッド作成処理：スレッド登録処理＞



＜スレッド作成処理：スレッド作成処理後のスレッド一覧再取得処理＞



■スレッド詳細表示機能

・概要
利用者ポータル画面より、スレッド一覧から、該当のスレッドのリンクをクリックした際に、
スレッド詳細画面に遷移する機能

・機能定義

項目	内容
機能名	スレッド詳細表示機能
対象画面	threadDetailScreen.jsp
呼び出しAction	ThreadDetailAction
データ構造	下記参照
付加情報	ThreadDetailActionから以下のクラスのメソッドが呼び出される

・データ構造
◆Thread（スレッド情報）

項目名	形	説明
thread_id	int	スレッドID
thread_title	varchar(191)	スレッドタイトル
bulletinboard_id	int	親の掲示板ID
user_id	int	スレッド作成ユーザーID
thread_delete_flag	int	スレッド削除フラグ
thread_delete_day	datetime(6)	スレッド削除日

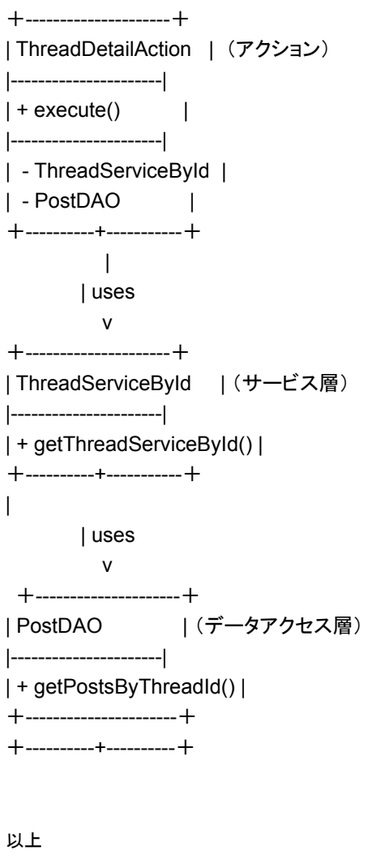
◆Post（投稿情報・リスト形式）

項目名	形	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	投稿の親スレッドID
user_id	int	投稿を作成したユーザーID
post_delete_flag	int	投稿の削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime(6)	投稿のタイムスタンプ

・処理の流れ
1.ユーザーがポータル画面で「スレッド一覧」の該当リンクをクリック。
2.ThreadDetailActionを呼び出し。
3.リクエストパラメータとして受け取った文字列のスレッドIDを数値に変換
4.スレッドIDからスレッドの詳細情報を取得する。
5.スレッドに紐づく投稿一覧を取得する。
6.threadDetailScreen.jspへ遷移し、取得データを表示する。

・シーケンス図（テキスト版）
1.ユーザー → ThreadDetailAction : execute() 呼び出し
2.ThreadDetailAction → ThreadServiceById : getThreadServiceById(int id)
3.ThreadServiceById → ThreadDetailAction : thread情報を返却
4.ThreadDetailAction → PostDAO : getPostsByThreadId(thread_id)
5.PostDAO → ThreadDetailAction : postListを返却
6.ThreadDetailAction → ユーザー : 画面遷移 (SUCCESS)

・クラス図



以上

・シーケンス図

■ユーザー個別のスレッド詳細表示機能

・概要

利用者ポータル画面より、ユーザー個別のスレッド一覧から、該当のスレッドのリンクをクリックした際に、
ユーザー個別のスレッド詳細画面に遷移する機能

・機能定義

項目	内容
機能名	ユーザー個別のスレッド詳細表示機能
対象画面	threadDetailScreen.jsp
呼び出しAction	ThreadDetailAction
データ構造	下記参照
付加情報	ThreadDetailActionから以下のクラスのメソッドが呼び出される

・データ構造

◆Thread（スレッド情報）

項目名	形	説明
thread_id	int	スレッドID
thread_title	varchar(191)	スレッドタイトル
bulletinboard_id	int	親の掲示板ID
user_id	int	スレッド作成ユーザーID
thread_delete_fla	int	スレッド削除フラグ
thread_delete_da	datetime(6)	スレッド削除日

◆Post（投稿情報・リスト形式）

項目名	形	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	投稿の親スレッドID
user_id	int	投稿を作成したユーザーID
post_delete_flag	int	投稿の削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime(6)	投稿のタイムスタンプ

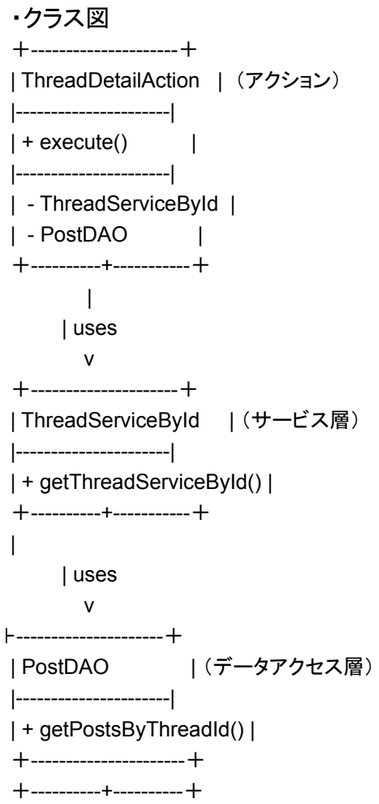
・処理の流れ

1.ユーザーがポータル画面で「〇〇さんのスレッド一覧」の該当リンクをクリック。
2.ThreadDetailActionを呼び出し。
3.リクエストパラメータとして受け取った文字列のスレッドIDを数値に変換
4.スレッドIDからスレッドの詳細情報を取得する。
5.スレッドに紐づく投稿一覧を取得する。
6.threadDetailScreen.jspへ遷移し、取得データを表示する。

・シーケンス図（テキスト版）

1.ユーザー → ThreadDetailAction : execute() 呼び出し
2.ThreadDetailAction → ThreadServiceById : getThreadServiceById(int id)
3.ThreadServiceById → ThreadDetailAction : thread情報を返却
4.ThreadDetailAction → PostDAO : getPostsByThreadId(thread_id)
5.PostDAO → ThreadDetailAction : postListを返却
6.ThreadDetailAction → ユーザー : 画面遷移 (SUCCESS)

・クラス図



以上

・シーケンス図

■ユーザー個別の投稿詳細表示機能

・概要

利用者ポータル画面より、ユーザー個別の投稿一覧から、該当の投稿の「投稿を見る」リンクをクリックした際に、
ユーザー個別の投稿に紐づく、親スレッドの詳細画面に遷移する機能

・機能定義

項目	内容
機能名	ユーザー個別の投稿詳細表示機能
対象画面	threadDetailScreen.jsp
呼び出しAction	UserPostThreadDetailAction
データ構造	下記参照
付加情報	UserPostThreadDetailActionから以下のクラスのメソッドが呼び出される

・データ構造

◆Thread（スレッド情報）

項目名	形	説明
thread_id	int	スレッドID
thread_title	varchar(191)	スレッドタイトル
bulletinboard_id	int	親の掲示板ID
user_id	int	スレッド作成ユーザーID
thread_delete_fla	int	スレッド削除フラグ
thread_delete_da	datetime(6)	スレッド削除日

◆Post（投稿情報・リスト形式）

項目名	形	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	投稿の親スレッドID
user_id	int	投稿を作成したユーザーID
post_delete_flag	int	投稿の削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime(6)	投稿のタイムスタンプ

・処理の流れ

1.ユーザーがポータル画面で「投稿を見る」をクリック。
2.UserPostThreadDetailActionを呼び出し。
3.投稿IDから親スレッドIDを取得する。
4.親スレッドIDからスレッド情報を取得する。
5.親スレッドIDからスレッドに紐づく投稿一覧を取得する。
6.threadDetailScreen.jspへ遷移し、取得データを表示する。

・シーケンス図（テキスト版）

1.ユーザー → UserPostThreadDetail : execute() 呼び出し
2.UserPostThreadDetail → UserPostDao : getThreadIdByPostId(postId)
3.UserPostDao → UserPostThreadDetail : **threadId を返却**
4.UserPostThreadDetail → UserPostThreadService : getThreadServiceById(threadId)
5.UserPostThreadService → UserPostThreadDetail : **thread 情報を返却**

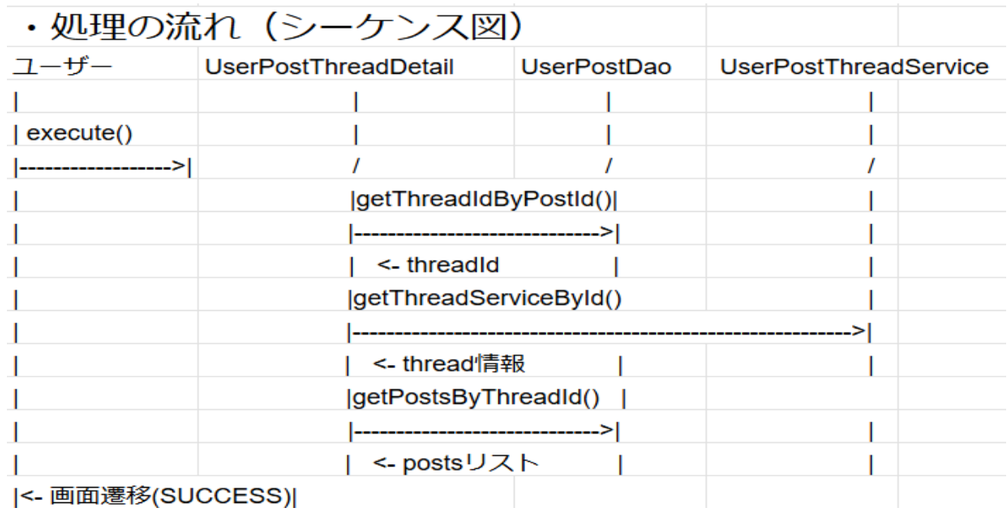
6.UserPostThreadDetail → UserPostDao : getPostsByThreadId(threadId)
7.UserPostDao → UserPostThreadDetail : **postsリストを返却**
8.UserPostThreadDetail → ユーザー : 画面遷移 (SUCCESS)

・クラス図

```
＋--------------------＋
| UserPostThreadDetail |（アクション）
|--------------------|
| + execute()        |
|--------------------|
|  - userPostDao     |
|  - userPostThreadService |
＋----------+----------＋
           |
           | uses
           v
＋-----------------＋
| UserPostDao     |（データアクセス層）
|-----------------|
| + getThreadIdByPostId() |
| + getPostsByThreadId()  |
＋----------+----------＋
           |
           | uses
           v
＋-----------------＋
| UserPostThreadService |（サービス層）
|----------------------|
| + getThreadServiceById() |
＋----------+----------＋
```

以上

・処理の流れ（シーケンス図）



■スレッド編集機能

・概要
スレッド詳細画面より、「スレッド編集」ボタンをクリックする。編集画面に遷移し、編集フォームに既存のデータが表示される。
スレッド編集画面にフォームに表示されている既存データを変更し、「スレッド編集」ボタンをクリックする。スレッドの編集処理が実行される。
スレッドの編集処理後、スレッド詳細画面の投稿一覧が最新状態で表示される。

・機能定義

項目	内容
機能名	スレッド編集機能
対象画面	ThreadEditScreen.jsp
呼び出しAction	EditThreadAction
データ構造	下記参照
付加情報	①EditThreadActionから以下のクラスのメソッドが呼び出される a.自クラス内の「getThreadById(int thread_id)」メソッド　※編集対象のデータ取得 ②UpdateThreadActionのメインメソッドにてスレッドのアップデート処理が実行される ③の処理を受けて、Struts2フレームワークによってPostListActionクラスに処理がフォワードされる。 a.ThreadServiceByIdクラス「getThreadServiceById」メソッド ※スレッド情報を取得 b.PostDaoクラスの「getPostsByThreadId(thread_id)」メソッド ※投稿一覧取

・データ構造
◆Thread（スレッド情報）

項目名	形	説明
thread_id	int	スレッドID
thread_title	varchar(191)	スレッドタイトル
bulletinboard_id	int	親の掲示板ID
user_id	int	スレッド作成ユーザーID
thread_delete_flag	int	スレッド削除フラグ
thread_delete_day	datetime(6)	スレッド削除日

◆Post（投稿情報・リスト形式）

項目名	形	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	投稿の親スレッドID
user_id	int	投稿を作成したユーザーID
post_delete_flag	int	投稿の削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime(6)	投稿のタイムスタンプ

・処理の流れ
1.スレッド詳細画面より、「スレッド編集」ボタンをクリックする。
2.threadDetailScreen.jspにより、Struts2フレームワークを介してEditThreadActionクラスを呼び出し。
3.EditThreadActionクラス内にて以下を実効。
　a.「getThreadById(int thread_id)」メソッドにて、スレッドIDを基に、既存データをデータベースから取得する。取得したデータを「thread」モデルに保存し、「SUCCESS」を返す。
　b.「ThreadEditScreen.jsp」にてa.の処理にてモデルに保存されている既存データへアクセスし、フォームに「投稿内容」を表示させる。
　c.「ThreadEditScreen.jsp」の画面にて「スレッドタイトル」を編集後、「スレッド編集」ボタンをクリックする。
5.「スレッド編集」ボタンクリック後、Struts2フレームワークにより「UpdateThreadAction」が呼び出される。以下の処理が実行される。
　a.フォームから送られて来たデータを「thread」モデルオブジェクトに一旦、保存する。
　b.メイン処理にて、SQLのアップデートを実効する際に、各プレースフォルダとthreadオブジェクトのプロパティ(スレッドID、スレッドタイトル)をバインドする。SQLを実効する。
　c.「SUCCESS」を返す
6. 5.の処理からStruts2フレームにて処理をPostListActionクラスへリダイレクトする。
5.PostListActionクラスにて以下を実効。
　a.PostListActionクラス内の「getThreadDetails(int thread_id)」メソッドにて、「ThreadServiceById」クラスの「getThreadServiceById」メソッドが呼び出される。
　　スレッドIDに紐づく、スレッド情報取得し、threadモデルに保存する。
　b.PostListActionクラス内で、「PostDao」クラスの「getPostsByThreadId(thread_id)」メソッドが呼び出される。
　　投稿データ取得処理後、postListに保存。
　c.「投稿のタイムスタンプを日本語の日付フォーマットで整形して返すメソッド」を呼び出し、結果を「formattedTimestamp」変数に代入。
6.「SUCCESS」を返す。

・シーケンス図（テキスト版）
＜スレッド編集処理：編集フォームに既存データを表示する＞
1.ユーザー → threadDetailScreen.jsp → EditThreadAction : execute() 呼び出し
2.EditThreadAction→ getThreadById(int thread_id) : 自クラス内でメソッド呼び出し、**thread情報を取得**
3.EditThreadAction→ ThreadEditScreen.jsp：thread情報を表示（DBに登録されているスレッド内容）
＜スレッド編集処理：編集データをDBに登録する＞
1.ThreadEditScreen.jsp → UpdateThreadAction: execute() 呼び出し
2.UpdateThreadAction → メイン処理にてDBの編集処理が実行される
3.UpdateThreadAction →「SUCCESS」を返す
＜スレッド編集処理後の投稿一覧再取得処理＞
1.struts.xml → PostListAction ※フレームワークにより処理がリダイレクト
2.PostListAction→ ThreadServiceById : getThreadServiceById
3.ThreadServiceById → PostListAction : **thread 情報を返却**
4.PostListAction→ PostDao : getPostsByThreadId(thread_id)
5.PostDao → PostListAction : **postListリストを返却**
6.PostListAction：String formattedTimestamp = post.getFormattedPostTimestamp();
　※「投稿のタイムスタンプを日本語の日付フォーマットで整形して返すメソッド」を呼び出し、結果を
　「formattedTimestamp」変数に代入。投稿の分、繰り返される。
7.PostListAction → threadDetailScreen.jsp → ユーザー : 画面遷移 (SUCCESS)

・クラス図
＜スレッド編集処理：編集フォームに既存データを表示する＞
```
＋-----------------＋
| EditThreadAction　| （アクション）
|-----------------|
| + execute()　　|
| - getThreadById(int thread_id)　| （自クラス内メソッド）
|-----------------|
|  thread（取得したスレッドデータ)|
```

```
+-----------------+
```

＜スレッド編集処理：データベースに登録＞

```
+--------------------+
| UpdateThreadAction   |（アクション）
|--------------------|
| + execute()        |
|--------------------|
|（自クラス内でDB編集処理）|
+--------------------+
```

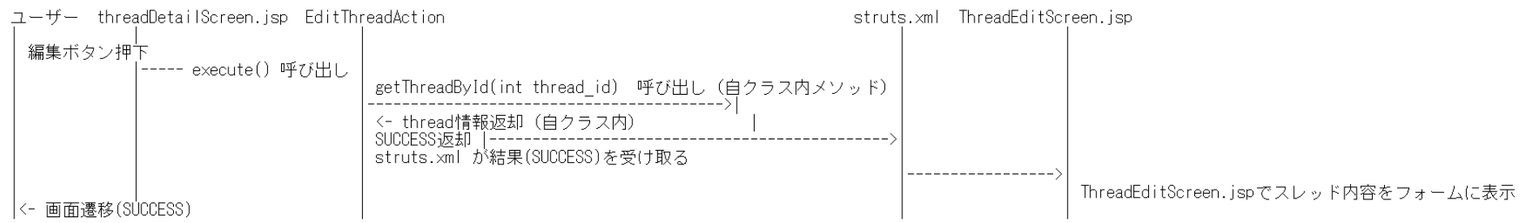
＜投稿編集処理後の投稿一覧再取得処理＞

```
+--------------------+
| PostListAction     |（アクション）
|--------------------|
| + execute()        |
|--------------------|
| - threadService    |
| - postDao          |
+----------+---------+
|
|  uses
v
+----------------------+
| ThreadServiceById    |（サービス層）
|----------------------|
| + getThreadServiceById()|
+----------+-----------+
|
|  uses
v
+-----------------+
| PostDao         |（データアクセス層）
|-----------------|
| + getPostsByThreadId() |
+----------+----------+
|
| formats
v
+-----------------+
| Post            |（エンティティ）
|-----------------|
| + getFormattedPostTimestamp() |
+-----------------+
```

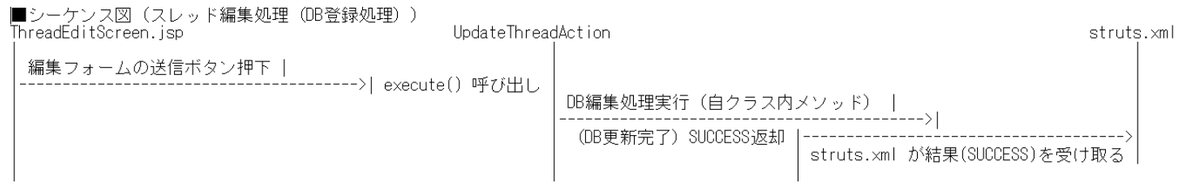
以上

・シーケンス図

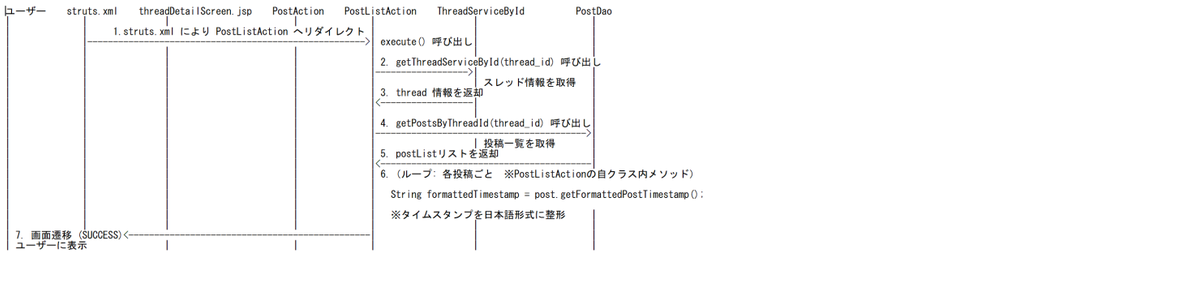
＜スレッド編集処理：編集フォームに既存データを表示する＞



＜スレッド編集処理：編集データをDBに登録する＞



＜スレッド編集処理後の投稿一覧再取得処理＞



■投稿処理機能

・概要
スレッド詳細画面より、「投稿内容：」セクションに投稿内容を入力後、「投稿書き込み」ボタンをクリックし投稿を登録する。
投稿処理後、スレッドの投稿一覧が最新の状態で表示される。

・機能定義

項目	内容
機能名	ユーザー個別の投稿詳細表示機能
対象画面	threadDetailScreen.jsp
呼び出しAction	PostAction
データ構造	下記参照
付加情報	・PostActionから以下のクラスのメソッドが呼び出される ・PostDaoクラスの「save(post)」メソッド ・PostListActionクラスに処理がフォワードされる、投稿一覧再取得 ・ThreadServiceByIdクラス「getThreadServiceById」メソッド ※スレッド情報を取得 ・PostDaoクラスの「getPostsByThreadId(thread_id)」メソッド ※投稿一覧取

・データ構造
◆Thread（スレッド情報）

項目名	形	説明
thread_id	int	スレッドID
thread_title	varchar(191)	スレッドタイトル
bulletinboard_id	int	親の掲示板ID
user_id	int	スレッド作成ユーザーID
thread_delete_flag	int	スレッド削除フラグ
thread_delete_day	datetime(6)	スレッド削除日

◆Post（投稿情報・リスト形式）

項目名	形	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	投稿の親スレッドID
user_id	int	投稿を作成したユーザーID
post_delete_flag	int	投稿の削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime(6)	投稿のタイムスタンプ

・処理の流れ
1.スレッド詳細画面より、「投稿内容：」セクションに投稿内容を入力後、「投稿書き込み」ボタンをクリック。
2.threadDetailScreen.jspより、PostActionクラスを呼び出し。
3.PostActionクラス内にて以下を実効。
  a.「save(User post)」メソッドにて、投稿情報をデータベースに保存する。
4.Struts2フレームワーク(struts.xml)にて、PostListActionに処理をリダイレクトする。
5.PostListActionクラスにて以下を実効。
  a.PostListActionクラス内の「getThreadDetails(int thread_id)」メソッドにて、「ThreadServiceById」クラスの「getThreadServiceById」メソッドが呼び出される。
    スレッドIDに紐づく、スレッド情報取得し、threadモデルに保存する。
  b.PostListActionクラス内で、「PostDao」クラスの「getPostsByThreadId(thread_id)」メソッドが呼び出される。
    投稿データ取得処理後、postListに保存。
  c.「投稿のタイムスタンプを日本語の日付フォーマットで整形して返すメソッド」を呼び出し、結果を「formattedTimestamp」変数に代入。

6.「SUCCESS」を返す。

・シーケンス図（テキスト版）

1.ユーザー → threadDetailScreen.jsp → PostAction : execute() 呼び出し
2.PostAction : save(User post) ※DB登録処理
3.struts.xml → PostListAction ※フレームワークにより処理がリダイレクト
4.PostListAction→ ThreadServiceById : getThreadServiceById
5.ThreadServiceById → PostListAction : **thread 情報を返却**
6.PostListAction→ PostDao : getPostsByThreadId(thread_id)
7.PostDao → PostListAction : **postListリストを返却**
8.PostListAction：String formattedTimestamp = post.getFormattedPostTimestamp();
※「投稿のタイムスタンプを日本語の日付フォーマットで整形して返すメソッド」を呼び出し、結果を「formattedTimestamp」変数に代入。投稿の分、繰り返される。
9.PostListAction → threadDetailScreen.jsp → ユーザー : 画面遷移 (SUCCESS)

・クラス図

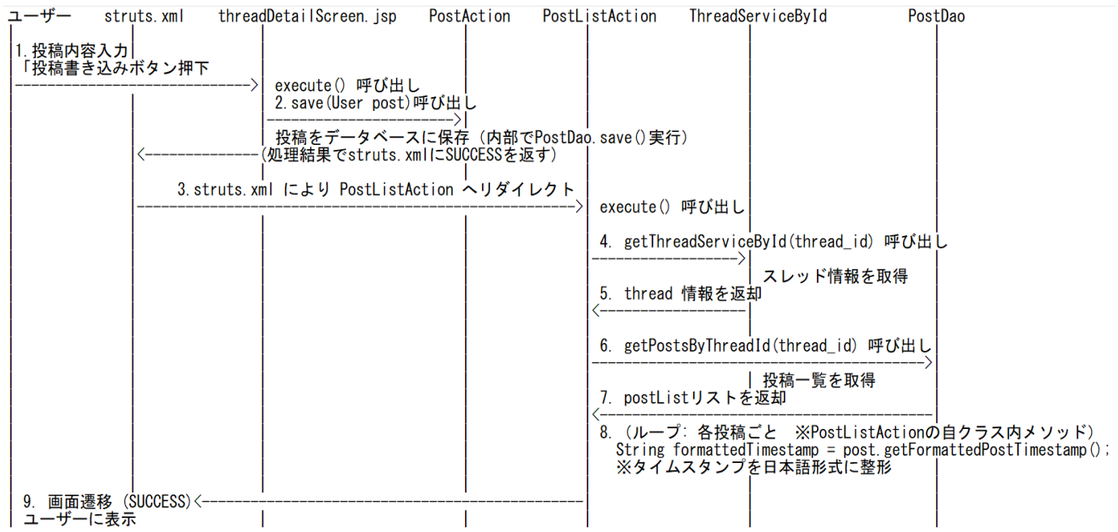
＜投稿登録処理＞

```
＋------------------＋
| PostAction |（アクション）
|------------------|
| + execute() |
| - savePost() |（自クラス内メソッド）
|------------------|
| post（新規投稿データ)|
＋------------------＋
```

＜投稿一覧再取得処理＞

```
＋---------------------＋
| PostListAction |（アクション）
|---------------------|

       ＋ execute()
- threadService
- postDao
＋----------+----------＋

uses
v
＋------------------------＋
ThreadServiceById
------------------------
＋ getThreadServiceById()
＋----------+-------------＋

uses
v
＋------------------＋
PostDao
```

・シーケンス図



＋ getPostsByThreadId()
＋----------+----------＋

formats
v
＋------------------＋
Post

＊ getFormattedPostTimestamp()
＋------------------＋

以上

■投稿編集機能

・概要

スレッド詳細画面より、スレッド詳細画面の各投稿の「投稿編集/削除」ボタンをクリックする。編集画面に遷移し、編集フォームに既存のデータが表示される。
投稿編集画面にフォームに表示されている既存データを変更し、「投稿更新」ボタンをクリックする。投稿の編集処理が実行される。
投稿の編集処理後、スレッド詳細画面の投稿一覧が最新状態で表示される。

・機能定義

項目	内容
機能名	投稿編集機能
対象画面	PostEditScreen.jsp
呼び出しAction	EditPostAction
データ構造	下記参照
付加情報	①EditPostActionから以下のクラスのメソッドが呼び出される a.自クラス内の「getPostById(int post_id)」メソッド　※編集対象のデータ取得 ②UpdatePostActionのメインメソッドにて投稿のアップデート処理が実行される ③の処理を受けて、Struts2フレームワークによってPostListActionクラスに処理がフォワードされる。 a.ThreadServiceByIdクラス「getThreadServiceById」メソッド ※スレッド情報を取得 b.PostDaoクラスの「getPostsByThreadId(thread_id)」メソッド ※投稿一覧取

・データ構造

◆Thread（スレッド情報）

項目名	形	説明
thread_id	int	スレッドID
thread_title	varchar(191)	スレッドタイトル
bulletinboard_id	int	親の掲示板ID
user_id	int	スレッド作成ユーザーID
thread_delete_flag	int	スレッド削除フラグ
thread_delete_day	datetime(6)	スレッド削除日

◆Post（投稿情報・リスト形式）

項目名	形	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	投稿の親スレッドID
user_id	int	投稿を作成したユーザーID
post_delete_flag	int	投稿の削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime(6)	投稿のタイムスタンプ

・処理の流れ

1.スレッド詳細画面より、スレッド詳細画面の各投稿の「投稿編集/削除」ボタンをクリックする。
2.threadDetailScreen.jspにより、Struts2フレームワークを介してEditPostActionクラスを呼び出し。
3.EditPostActionクラス内にて以下を実効。
　a.「getPostById(int post_id)」メソッドにて、投稿IDを基に、既存データをデータベースから取得する。取得したデータを「post」モデルに保存し、「SUCCESS」を返す。
　b.「PostEditScreen.jsp」にてa.の処理にてモデルに保存されている既存データへアクセスし、フォームに「投稿内容」を表示させる。
4.「PostEditScreen.jsp」の画面にて「投稿内容」を編集後、「投稿更新」ボタンをクリックする。
5.「投稿更新」ボタンクリック後、Struts2フレームワークにより「UpdatePostAction」が呼び出される。以下の処理が実行される。
　a.フォームから送られて来たデータを「post」モデルオブジェクトに一旦、保存する。
　b.メイン処理にて、SQLのアップデートを実効する際に、各プレースフォルダと、postオブジェクトのプロパティ(投稿ID、投稿内容)をバインドする。SQLを実効する。
　c.「SUCCESS」を返す
6. 5.の処理からStruts2フレームにて処理をPostListActionクラスへリダイレクトする。
5.PostListActionクラスにて以下を実効。
　a.PostListActionクラス内の「getThreadDetails(int thread_id)」メソッドにて、「ThreadServiceById」クラスの「getThreadServiceById」メソッドが呼び出される。
　　スレッドIDに紐づく、スレッド情報取得し、threadモデルに保存する。
　b.PostListActionクラス内で、「PostDao」クラスの「getPostsByThreadId(thread_id)」メソッドが呼び出される。
　　投稿データ取得処理後、postListに保存。
　c.「投稿のタイムスタンプを日本語の日付フォーマットで整形して返すメソッド」を呼び出し、結果を「formattedTimestamp」変数に代入。
6.「SUCCESS」を返す。

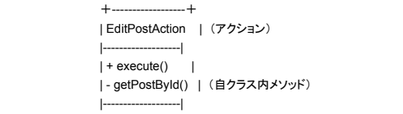
・シーケンス図（テキスト版）

＜投稿編集処理：編集フォームに既存データを表示する＞
1.ユーザー → threadDetailScreen.jsp → EditPostAction : execute() 呼び出し
2.EditPostAction→ getPostById(int post_id) : 自クラス内でメソッド呼び出し、**post情報を取得**
3.EditPostAction→ PostEditScreen.jsp：**post情報を表示（DBに登録されている投稿内容）**
＜投稿編集処理：編集データをDBに登録する＞
1.PostEditScreen.jsp → UpdatePostAction: execute() 呼び出し
2.UpdatePostAction → メイン処理にてDBの編集処理が実行される
3.UpdatePostAction →「SUCCESS」を返す
＜投稿編集処理後の投稿一覧再取得処理＞
1.struts.xml → PostListAction ※フレームワークにより処理がリダイレクト
2.PostListAction→ ThreadServiceById : getThreadServiceById
3.ThreadServiceById → PostListAction : **thread 情報を返却**
4.PostListAction→ PostDao : getPostsByThreadId(thread_id)
5.PostDao → PostListAction : **postListリストを返却**
6.PostListAction：String formattedTimestamp = post.getFormattedPostTimestamp();
　※「投稿のタイムスタンプを日本語の日付フォーマットで整形して返すメソッド」を呼び出し、結果を
　　「formattedTimestamp」変数に代入。投稿の分、繰り返される。
7.PostListAction → threadDetailScreen.jsp → ユーザー : 画面遷移 (SUCCESS)

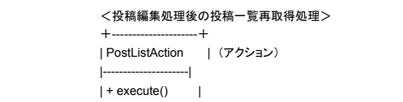
・シーケンス図

・クラス図

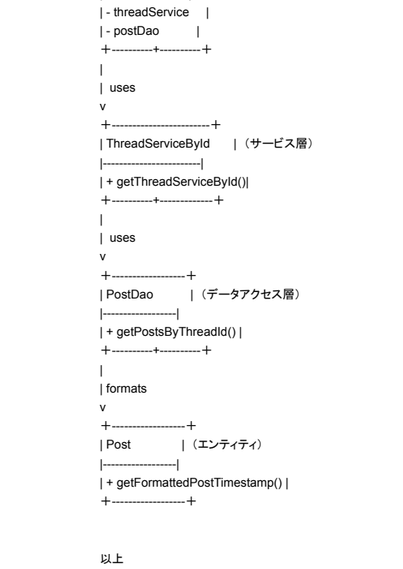
＜投稿編集処理：編集フォームに既存データを表示する＞



＜投稿編集処理後の投稿一覧再取得処理＞

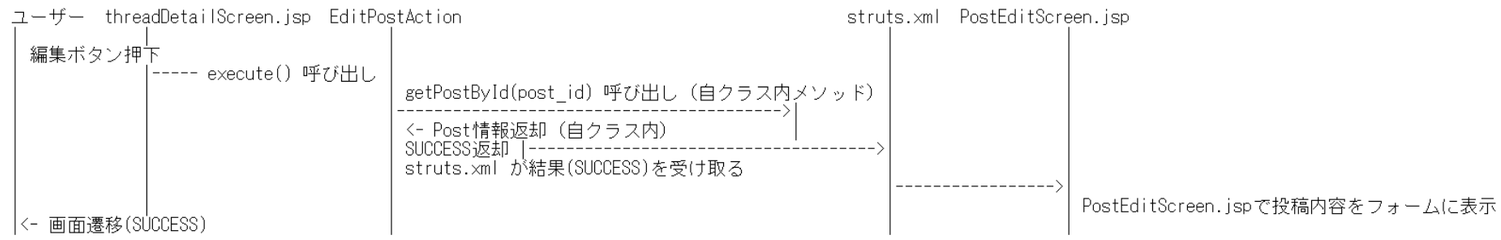


＜投稿編集処理後の投稿一覧再取得処理＞

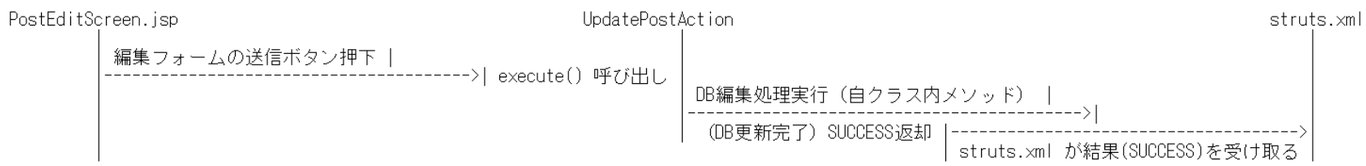


以上

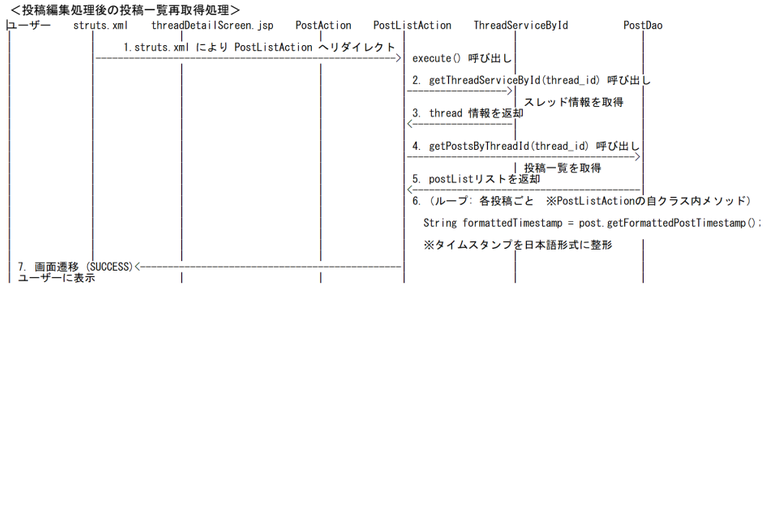
＜投稿編集処理：編集フォームに既存データを表示する＞



＜投稿編集処理：編集データをDBに登録する＞



＜投稿編集処理後の投稿一覧再取得処理＞



■投稿削除機能

・概要
スレッド詳細画面より、スレッド詳細画面の各投稿の「投稿編集削除」ボタンをクリックする。「投稿編集投稿削除」の画面が表示される。
「投稿削除」ボタンをクリックする。
投稿の削除処理後、スレッド詳細画面の投稿一覧が最新状態で表示される。

・機能定義

項目	内容
機能名	投稿削除機能
対象画面	PostEditScreen.jsp
呼び出しAction	DeletePostAction
データ構造	下記参照
付加情報	①EditPostActionから以下のクラスのメソッドが呼び出される a.自クラス内の「getPostById(int post_id)」メソッド　※編集対象のデータ取得 ②DeletePostActionのメインメソッドにて投稿の削除処理が実行される ③の処理を受けて、Struts2フレームワークによってPostListActionクラスに処理がフォワード される。 a.ThreadServiceByIdクラス「getThreadServiceById」メソッド ※スレッド情報を取得 b.PostDaoクラスの「getPostsByThreadId(thread_id)」メソッド ※投稿一覧取

・データ構造
◆Thread（スレッド情報）

項目名	形	説明
thread_id	int	スレッドID
thread_title	varchar(191)	スレッドタイトル
bulletinboard_id	int	親の掲示板ID
user_id	int	スレッド作成ユーザーID
thread_delete_flag	int	スレッド削除フラグ
thread_delete_day	datetime(6)	スレッド削除日

◆Post（投稿情報・リスト形式）

項目名	形	説明
post_id	int	投稿ID
post_content	text	投稿内容
thread_id	int	投稿の親スレッドID
user_id	int	投稿を作成したユーザーID
post_delete_flag	int	投稿の削除フラグ
post_delete_day	datetime(6)	投稿削除日
post_timestamp	datetime(6)	投稿のタイムスタンプ

・処理の流れ
1.スレッド詳細画面より、スレッド詳細画面の各投稿の「投稿編集削除」ボタンをクリックする。
2.threadDetailScreen.jspにより、Struts2フレームワークを介してEditPostActionクラスを呼び出し。
3.EditPostActionクラス内にて以下を実効。
　a.「getPostById(int post_id)」メソッドにて、投稿Dを基に、既存データをデータベースから取得する。取得したデータを「post」モデルに保存し、「SUCCESS」を返す。
　b.「PostEditScreen.jsp」にてa.の処理にてモデルに保存されている既存データへアクセスし、フォームに「投稿内容」を表示させる。
4.「PostEditScreen.jsp」の画面にて「投稿削除」ボタンをクリックする。
5.「投稿削除」ボタンクリック後、Struts2フレームワークにより「DeletePostAction」が呼び出される。以下の処理が実行される。
　a.メイン処理にて削除用のSQL文が実行される。
　b.「SUCCESS」を返す
6. 5.の処理からStruts2フレームにて処理をPostListActionクラスへリダイレクトする。
5.PostListActionクラスにて以下を実効。
　a.PostListActionクラス内の「getThreadDetails(int thread_id)」メソッドにて、「ThreadServiceById」クラスの「getThreadServiceById」メソッドが呼び出される。
　　スレッドIDに紐づく、スレッド情報取得し、threadモデルに保存する。
　b.PostListActionクラス内で、「PostDao」クラスの「getPostsByThreadId(thread_id)」メソッドが呼び出される。
　　投稿データ取得処理後、postListに保存。
　c.「投稿のタイムスタンプを日本語の日付フォーマットで整形して返すメソッド」を呼び出し、結果を「formattedTimestamp」変数に代入。
6.「SUCCESS」を返す。

・シーケンス図（テキスト版）
＜投稿編集処理：編集フォームに既存データを表示する＞
1.ユーザー → threadDetailScreen.jsp → EditPostAction : execute() 呼び出し
2.EditPostAction→ getPostById(int post_id) : 自クラス内でメソッド呼び出し、**post情報を取得**
3.EditPostAction→ PostEditScreen.jsp：**post情報を表示（DBに登録されている投稿内容）**
＜投稿削除処理：データをDBから削除する＞
1.PostEditScreen.jsp → DeletePostAction: execute() 呼び出し
2.DeletePostAction → メイン処理にてDBの削除処理が実行される
3.DeletePostAction →「SUCCESS」を返す
＜投稿削除処理後の投稿一覧再取得処理＞
1.struts.xml → PostListAction ※フレームワークにより処理がリダイレクト
2.PostListAction→ ThreadServiceById : getThreadServiceById
3.ThreadServiceById → PostListAction : **thread 情報を返却**
4.PostListAction→ PostDao : getPostsByThreadId(thread_id)
5.PostDao → PostListAction : **postListリストを返却**
6.PostListAction：String formattedTimestamp = post.getFormattedPostTimestamp();
　※「投稿のタイムスタンプを日本語の日付フォーマットで整形して返すメソッド」を呼び出し、結果を
　　「formattedTimestamp」変数に代入。投稿の分、繰り返される。
7.PostListAction → threadDetailScreen.jsp → ユーザー : 画面遷移 (SUCCESS)

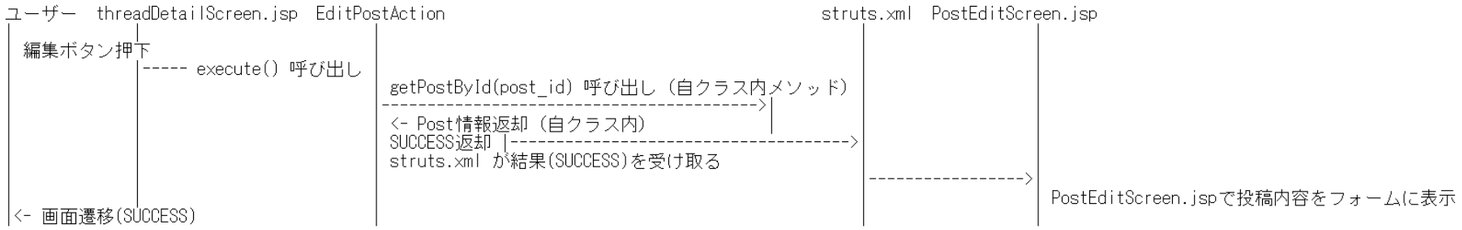
・クラス図
＜投稿削除処理：編集フォームに既存データを表示する＞
＋------------------＋
| EditPostAction　| （アクション）
|------------------|

```
| + execute()      |
| - getPostById()  | （自クラス内メソッド）
|------------------|
|  post（取得した投稿データ)|
+------------------+

＜投稿削除処理後の投稿一覧再取得処理＞
+--------------------+
| DeletePostAction   | （アクション）
|--------------------|
| + execute()        |
|--------------------|
| （自クラス内でDB削除処理）|
+--------------------+

＜投稿削除処理後の投稿一覧再取得処理＞
+--------------------+
| PostListAction     | （アクション）
|--------------------|
| + execute()        |
|--------------------|
| - threadService    |
| - postDao          |
+----------+---------+
|
|  uses
v
+----------------------+
| ThreadServiceById    | （サービス層）
|----------------------|
| + getThreadServiceById()|
+----------+-----------+
|
|  uses
v
+-----------------+
| PostDao         | （データアクセス層）
|-----------------|
| + getPostsByThreadId() |
+----------+------+
|
| formats
v
+-----------------+
| Post            | （エンティティ）
|-----------------|
| + getFormattedPostTimestamp() |
+-----------------+
```

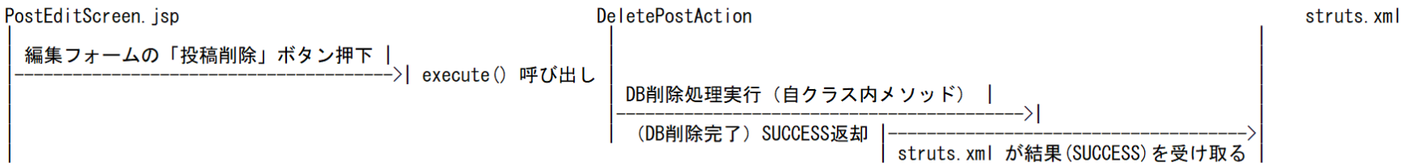
以上

・シーケンス図

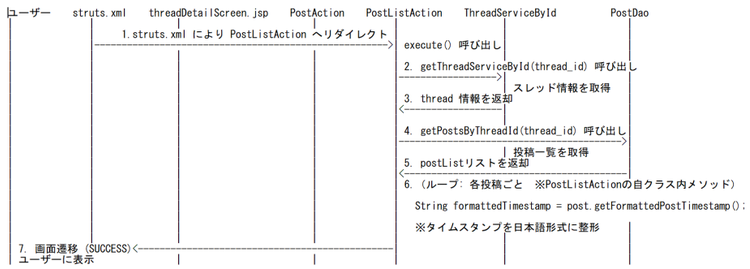
＜投稿削除処理:フォームに既存データを表示する＞



＜投稿編集処理:編集データをDBに登録する＞



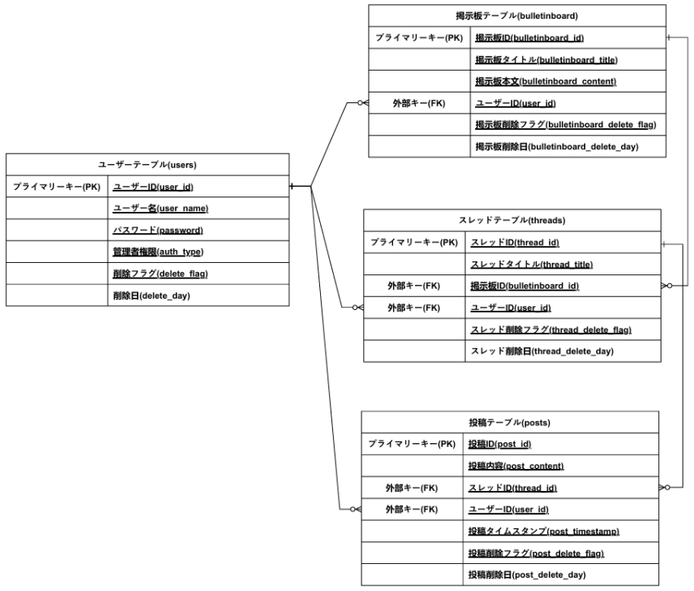
＜投稿削除処理後の投稿一覧再取得処理＞



6. ロールと権限マトリクス

画面 / 機能	管理者	一般ユーザー
掲示板作成	○	×
スレッド投稿	○	○
投稿編集	投稿者本人のみ	投稿者本人のみ

7. データモデル


掲示板テーブル(bulletinboard)
プライマリーキー(PK)
掲示板ID(bulletinboard_id)
掲示板タイトル(bulletinboard_title)
掲示板本文(bulletinboard_content)
外部キー(FK)
ユーザーID(user_id)
掲示板削除フラグ(bulletinboard_delete_flag)
掲示板削除日(bulletinboard_delete_day)
ユーザーテーブル(users)
プライマリーキー(PK)
ユーザーID(user_id)
ユーザー名(user_name)
パスワード(password)
管理者権限(auth_type)
削除フラグ(delete_flag)
削除日(delete_day)
スレッドテーブル(threads)
プライマリーキー(PK)
スレッドID(thread_id)
スレッドタイトル(thread_title)
外部キー(FK)
掲示板ID(bulletinboard_id)
外部キー(FK)
ユーザーID(user_id)
スレッド削除フラグ(thread_delete_flag)
スレッド削除日(thread_delete_day)
投稿テーブル(posts)
プライマリーキー(PK)
投稿ID(post_id)
投稿内容(post_content)
外部キー(FK)
スレッドID(thread_id)
外部キー(FK)
ユーザーID(user_id)
投稿タイムスタンプ(post_timestamp)
投稿削除フラグ(post_delete_flag)
投稿削除日(post_delete_day)