

TERASOLUNA Server Framework for Java 5.x

内部設計／製造作業要領・規約（雛型）

第1.1.1版

2015年09月04日

**変更履歴**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版数** | **制改訂日** | **変更箇所** | **変更内容** | **変更者** |
| 1.1.1 | 2015年09月04日 |  | 初版制定 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

##### カスタマイズポイント一覧

プロジェクトごとに修正を検討する必要のあるカスタマイズポイントについて、以下の表に記述する。

|  |  |
| --- | --- |
| **カスタマイズポイント** | **説明** |
| 2. ソフトウェア概要 | システムの構成に合わせて変更する |
| 3. 開発環境 | プロジェクトで使用するものに合わせて変更する。プロジェクト内でツールの種類やバージョンが統一されていないと、ファイル破壊やデータ損失の恐れがあるため、必ず規定し統一をはかること。 |
| 5.1設計工程における成果物の単位及び命名 | TERASOLUNA Server Framework for Java 5.x 開発手順の制約として規定している命名付与基準はないため、各プロジェクトの標準に合わせて変更して構わない。 |
| 5.2 製造工程における成果物の単位及び命名 | プロジェクトの特性に合わせて成果物の種類を変更する。例えば、本書では帳票機能やシステム連携を対象外として関連する成果物を省略しているため、必要に応じて追加すること。 |
| 5.2.3 SQL関連の命名規約 | MyBatis3を利用する場合の命名規約である。JPAや他のO/R Mapperを利用する場合、それに対応する命名規約を記載すること。 |
| 6.1.1 プロジェクト構成 | プロジェクトの特性に合わせてSTSのプロジェクトを変更する。例えば、層を跨ぐ共通的な資材を格納する「atrs-common」パッケージや、Selenium試験用の資材を格納する「atrs-selenium」パッケージ等を追加する。 |
| 7.4.1.1 単項目チェック | 外部設計にてチェックルールが追加されている場合、それに合わせて該当するチェックを一覧に追加する。 |
| 7.4.1.1 単項目チェック | 文字列型の未入力を空文字とするかNullとするかはプロジェクトの指針によって異なる。今回はNullとするため@NotNullにマッピングされるが、空文字の場合は@NotEmptyとなる。空文字とした場合、「未入力は許容するが、入力された場合は6文字以上であること」という要件を、既存のアノテーションで満たすことができない点に注意が必要である。 |
| 7.4.2.3 エラー時の画面遷移 | 「共通的なエラー画面に遷移する」以外に以下の２パターンがある。プロジェクトの特定に合わせて適切なものを選択すること。 ●入力画面に遷移する 入力データを修正させるため入力画面に遷移し、エラーメッセージを表示する。 ●確認画面に遷移する 遷移元の画面を再表示するという基本ルールに従い確認画面に遷移し、エラーメッセージを表示する。 |

|  |  |
| --- | --- |
| **カスタマイズポイント** | **説明** |
| 7.7 セッション管理 | リクエストスコープを前提としているが、セッションスコープを前提とする方針も可能である。プロジェクトの特性に合わせて選択すること。なお、セッションスコープの場合、複数台のAPサーバ間におけるセッションデュプリケーションについても検討すること。 |
| 7.9 プロパティ管理 | プロパティファイルにミドルウェアのユーザIDやパスワード等を記載する場合、セキュリティ的な観点から暗号化することが望ましい。  共通バリデーションのメッセージとしてBean Validationのデフォルトメッセージを上書きして使用する場合は、ValidationMessages.propertiesを管理対象に加えること。 |
| 7.12 国際化対応 | 国際化対応を行う際は、対応する言語の種類を明記し、画面仕様や画面メッセージを設計する際の方針を追加すること。 システム連携の要件がある場合、連携データについても国際化の対応を検討すること。 |
| 7.17 認証・認可 | Spring Securityを利用しない場合、認証方法、アクセス認可の範囲及び方法について実現方法を検討すること。 |
| 9.2 Repositoryクラスの実装 | MyBatis3を利用する場合の記述である。JPAや他のO/R Mapperを利用する場合、それを利用する場合の記述要領を記載すること。 |
| 9.9.4.3 項目（ラベル） | 項目（ラベル）はJSPのハードコーディングとしているが、プロパティファイルに定義することも可能である。メッセージの変更頻度や国際化対応を考慮し、ハードコーディングにするかどうか決定すること。 |
| 9.9.4.5 ガイダンスメッセージ | ガイダンスメッセージはJSPのハードコーディングとしているが、プロパティファイルに定義することも可能である。メッセージの変更頻度や国際化対応を考慮し、ハードコーディングにするかどうか決定すること。 |
| 9.10 プロパティファイルの実装 | 共通バリデーションのメッセージとしてBean Validationのデフォルトメッセージを上書きして使用する場合は、ValidationMessages.propertiesを追加すること。  ValidationMessages.propertiesはクラスパス直下のものを読み込むため、「9.10.2 読み込み設定」は不要になる。 |
| 9.10.3 プロパティファイルの記述 | 画面のタイトル、ボタンの名称等をプロパティファイルに定義することも可能である。国際化対応を考慮し、ハードコーディングにするかどうか決定すること。 |

[1. はじめに 10](#_Toc427326761)

[1.1. 目的 10](#_Toc427326762)

[1.2. 留意事項 10](#_Toc427326763)

[1.3. 章構成 10](#_Toc427326764)

[表 1 内部設計／製造 作業要領・規約の章構成 10](#_Toc427326765)

[2. ソフトウェア概要 11](#_Toc427326766)

[2.1. ソフトウェア概要 11](#_Toc427326767)

[図 1　ソフトウェア概要図 11](#_Toc427326768)

[表 2 使用フレームワーク一覧 11](#_Toc427326769)

[2.2. アーキテクチャ概要 12](#_Toc427326770)

[図 2　アーキテクチャ概要図 12](#_Toc427326771)

[表 3　開発者が開発する要素の一覧 12](#_Toc427326772)

[2.2.1 処理フロー 13](#_Toc427326773)

[3. 開発環境 14](#_Toc427326774)

[3.1. 開発者のローカル環境 14](#_Toc427326775)

[表 4 開発環境一覧 14](#_Toc427326776)

[4. 内部設計・製造の流れ 15](#_Toc427326777)

[4.1. 内部設計の作業の流れ 15](#_Toc427326778)

[図 3　内部設計・製造の作業の流れ 15](#_Toc427326779)

[4.2. 作業概要と成果物 16](#_Toc427326780)

[5. 命名付与基準 17](#_Toc427326781)

[5.1. 設計工程における成果物の単位及び命名 17](#_Toc427326782)

[5.1.1 基本方針 17](#_Toc427326783)

[5.1.1.1 システムで一部作成する成果物 17](#_Toc427326784)

[5.1.1.2 ユースケースグループ単位で作成する成果物 17](#_Toc427326785)

[5.1.1.3 ユースケース単位で作成する成果物 17](#_Toc427326786)

[5.1.1.4 その他の単位で作成する成果物 17](#_Toc427326787)

[5.1.2 成果物ファイル名 18](#_Toc427326788)

[5.2. 製造工程における成果物の単位及び命名 19](#_Toc427326789)

[5.2.1 URI関連の命名規約 19](#_Toc427326790)

[5.2.1.1 URLの命名規約 19](#_Toc427326791)

[5.2.1.2 HTTPパラメータの命名規約 19](#_Toc427326792)

[表 7　HTTPパラメータ一覧 19](#_Toc427326793)

[5.2.2 クラス関連の命名規約 20](#_Toc427326794)

[5.2.2.1 Controllerクラスの命名規約 20](#_Toc427326795)

[5.2.2.2 Formクラスの命名規約 20](#_Toc427326796)

[5.2.2.3 Serviceインターフェース、実装クラスの命名規約 21](#_Toc427326797)

[5.2.2.4 SharedServiceインターフェース、実装クラスの命名規約 21](#_Toc427326798)

[5.2.2.5 Repositoryインターフェースの命名規約 21](#_Toc427326799)

[5.2.2.6 Entityクラスの命名規約 21](#_Toc427326800)

[5.2.2.7 DTOクラスの命名規約 22](#_Toc427326801)

[5.2.2.8 例外クラスの命名規約 22](#_Toc427326802)

[5.2.3 SQL関連の命名規約 23](#_Toc427326803)

[5.2.3.1 SQL\_IDの命名規約 23](#_Toc427326804)

[5.2.3.2 SQL MAPPERファイルの命名規約 24](#_Toc427326805)

[5.2.3.3 SQL MAPPERファイルのnamespace命名規約 24](#_Toc427326806)

[6. 開発リソースの構成 25](#_Toc427326807)

[6.1. 全体構成 25](#_Toc427326808)

[6.1.1 プロジェクト構成 25](#_Toc427326809)

[表 9　STSプロジェクト一覧 25](#_Toc427326810)

[6.1.2 ディレクトリ構成 26](#_Toc427326811)

[表 10 ディレクトリ構成 26](#_Toc427326812)

[6.1.3 Javaパッケージ構成 27](#_Toc427326813)

[6.1.3.1 パッケージのプレフィックス 27](#_Toc427326814)

[6.1.3.2 パッケージとプロジェクトの関係 27](#_Toc427326815)

[表 11 パッケージとプロジェクトの対応一覧 27](#_Toc427326816)

[6.1.4 JSPファイルの配置先 27](#_Toc427326817)

[6.1.5 外部ファイルの配置先 27](#_Toc427326818)

[表 12 外部ファイルの格納先一覧 28](#_Toc427326819)

[6.1.6 プロパティファイルの配置先 28](#_Toc427326820)

[表 13 プロパティファイルの配置先一覧 28](#_Toc427326821)

[7. 内部設計・製造の基本方針 29](#_Toc427326822)

[7.1. 画面遷移 29](#_Toc427326823)

[7.1.1 RDBMSの更新処理（C/U/D）を伴う画面遷移 29](#_Toc427326824)

[7.1.2 検索画面における画面遷移 29](#_Toc427326825)

[7.1.3 ユーザログイン・ユーザログアウトにおける画面遷移 29](#_Toc427326826)

[7.2. 共通領域 30](#_Toc427326827)

[7.2.1 共通領域の定義 30](#_Toc427326828)

[7.3. トランザクション管理 31](#_Toc427326829)

[7.3.1 DBトランザクション 31](#_Toc427326830)

[7.4. 入力チェック 32](#_Toc427326831)

[7.4.1 入力チェックの種類 32](#_Toc427326832)

[7.4.1.1 単項目チェック 32](#_Toc427326833)

[表 14　単項目チェック一覧 32](#_Toc427326834)

[7.4.1.2 相関項目チェック 33](#_Toc427326835)

[7.4.2 入力チェックにおける考慮ポイント 34](#_Toc427326836)

[7.4.2.1 入力チェックの実施タイミング 34](#_Toc427326837)

[7.4.2.2 相関項目チェックの実装方法 35](#_Toc427326838)

[表 15　相関項目チェックの実装方針 35](#_Toc427326839)

[7.4.2.3 エラー時の画面遷移 36](#_Toc427326840)

[7.4.2.4 エラー時に業務処理が必要な場合 37](#_Toc427326841)

[表 16　対象画面と業務処理の実施可否の一覧 37](#_Toc427326842)

[7.5. ログ出力 38](#_Toc427326843)

[7.5.1 ログ出力 38](#_Toc427326844)

[表 17　ログレベル一覧 38](#_Toc427326845)

[図 8　ログ出力の例 39](#_Toc427326846)

[7.5.2 デバッグログについて 39](#_Toc427326847)

[7.6. エラー・例外処理 40](#_Toc427326848)

[7.6.1 エラー・例外の種類 40](#_Toc427326849)

[表 18 エラー・例外一覧 40](#_Toc427326850)

[7.6.2 業務開発者が設計・考慮すべき例外 41](#_Toc427326851)

[7.7. セッション管理 42](#_Toc427326852)

[7.7.1 基本方針 42](#_Toc427326853)

[7.7.2 Controller内で持ち回る場合 42](#_Toc427326854)

[7.7.3 Controllerを跨いで持ち回る場合 42](#_Toc427326855)

[7.7.4 ユーザ情報について 42](#_Toc427326856)

[7.8. メッセージ管理 43](#_Toc427326857)

[7.9. プロパティ管理 44](#_Toc427326858)

[表 19 プロパティファイル一覧 44](#_Toc427326859)

[7.10. ページネーション 45](#_Toc427326860)

[7.11. 二重送信防止、不正画面遷移チェック 46](#_Toc427326861)

[7.11.1 二重送信防止 46](#_Toc427326862)

[7.11.2 不正画面遷移チェック 46](#_Toc427326863)

[7.12. 国際化対応 47](#_Toc427326864)

[7.13. コードリスト 48](#_Toc427326865)

[表 20 コードリスト一覧 48](#_Toc427326866)

[7.14. 画面生成・表示制御 49](#_Toc427326867)

[7.14.1 画面レイアウト 49](#_Toc427326868)

[7.14.2 画面コンテンツ 49](#_Toc427326869)

[7.14.3 CSS 49](#_Toc427326870)

[7.14.4 JavaScript 49](#_Toc427326871)

[7.15. システム時刻 50](#_Toc427326872)

[7.16. 日付操作 51](#_Toc427326873)

[7.17. 認証・認可 52](#_Toc427326874)

[7.17.1 基本方針 52](#_Toc427326875)

[7.17.2 認証処理 52](#_Toc427326876)

[7.17.3 認可処理 52](#_Toc427326877)

[7.17.4 Spring Securityでは実現できないこと 53](#_Toc427326878)

[7.18. セキュリティ対策 54](#_Toc427326879)

[7.18.1 DBアクセス 54](#_Toc427326880)

[7.18.2 パスワードハッシュ 54](#_Toc427326881)

[7.18.3 XSS対策 54](#_Toc427326882)

[7.18.4 CSRF対策 55](#_Toc427326883)

[7.18.5 外部コマンドの実行 55](#_Toc427326884)

[7.19. DBアクセス 56](#_Toc427326885)

[7.19.1 DBアクセス 56](#_Toc427326886)

[7.19.2 Like検索時のエスケープ 56](#_Toc427326887)

[7.19.3 トランザクションについて 56](#_Toc427326888)

[7.19.4 セキュリティ対策について 56](#_Toc427326889)

[7.20. 定数 57](#_Toc427326890)

[7.21. 文字コード 58](#_Toc427326891)

[表 21 文字コード一覧 58](#_Toc427326892)

[8. 設計書の記述要領 59](#_Toc427326893)

[8.1. アクセスパターン一覧 59](#_Toc427326894)

[8.1.1 記述単位 59](#_Toc427326895)

[8.1.2 記述項目 59](#_Toc427326896)

[8.2. リポジトリ一覧 60](#_Toc427326897)

[8.2.1 記述単位 60](#_Toc427326898)

[8.2.2 記述項目 60](#_Toc427326899)

[8.2.3 アクセスパターン一覧とリポジトリの関係 61](#_Toc427326900)

[図 9　「アクセスパターン一覧」とリポジトリの関係 61](#_Toc427326901)

[8.2.4 リポジトリの記述単位 61](#_Toc427326902)

[8.3. クラス仕様書（リポジトリ） 62](#_Toc427326903)

[8.3.1 記述単位 62](#_Toc427326904)

[8.3.2 記述項目 62](#_Toc427326905)

[8.3.3 アクセスパターン一覧とリポジトリメソッドの関係 63](#_Toc427326906)

[図 10　「アクセスパターン一覧」とリポジトリのメソッドの関係 63](#_Toc427326907)

[8.3.4 リポジトリのメソッドの抽出方法 64](#_Toc427326908)

[図 11　関連エンティティを操作するメソッド 64](#_Toc427326909)

[図 12　全カラムの取得 65](#_Toc427326910)

[8.4. イベントURLマッピング一覧 66](#_Toc427326911)

[8.4.1 記述単位 66](#_Toc427326912)

[8.4.2 記述項目 66](#_Toc427326913)

[8.4.3 URLの記述方法 67](#_Toc427326914)

[図 13　遷移先の画面が別のユースケースの場合 67](#_Toc427326915)

[8.4.4 HTTPパラメータの記述方法 68](#_Toc427326916)

[図 14　イベントが実質的に等しい場合 68](#_Toc427326917)

[8.4.5 トランザクショントークンチェックの記述方法 68](#_Toc427326918)

[8.4.6 Spring Securityを利用する際の記述方法 68](#_Toc427326919)

[8.5. イベントフロー図 69](#_Toc427326920)

[8.5.1 記述単位 69](#_Toc427326921)

[8.5.2 記述項目 69](#_Toc427326922)

[8.6. クラス仕様書（サービス） 70](#_Toc427326923)

[8.6.1 記述単位 70](#_Toc427326924)

[8.6.2 記述項目 70](#_Toc427326925)

[8.7. クラス図 72](#_Toc427326926)

[8.7.1 記述単位 72](#_Toc427326927)

[8.7.2 記述フォーマット 72](#_Toc427326928)

[8.7.3 クラスのプロパティが多い場合 72](#_Toc427326929)

[9. コーディングガイド 73](#_Toc427326930)

[9.1. Entityクラスの実装 73](#_Toc427326931)

[9.1.1 Entityクラスの作成 73](#_Toc427326932)

[9.1.2 属性の追加 73](#_Toc427326933)

[9.1.3 属性の型について 73](#_Toc427326934)

[9.2. Repositoryクラスの実装 74](#_Toc427326935)

[9.2.1 Repositoryインターフェースの作成 74](#_Toc427326936)

[9.2.2 インターフェースメソッドの作成 74](#_Toc427326937)

[9.2.3 SQL MAPPERファイルの作成 74](#_Toc427326938)

[9.2.4 SQL MAPPERファイルへのSQLステートメントの追加 74](#_Toc427326939)

[9.3. DTOクラスの実装 77](#_Toc427326940)

[9.3.1 DTOクラスの作成 77](#_Toc427326941)

[9.4. Serviceクラスの実装 78](#_Toc427326942)

[9.4.1 Serviceインターフェースの作成 78](#_Toc427326943)

[9.4.2 インターフェースメソッドの作成 78](#_Toc427326944)

[9.4.3 Serviceクラスの作成 78](#_Toc427326945)

[9.4.4 クラスメソッドの作成 79](#_Toc427326946)

[9.5. Formクラスの実装 80](#_Toc427326947)

[9.5.1 Formクラスの作成 80](#_Toc427326948)

[9.5.2 属性の追加 81](#_Toc427326949)

[9.5.2.1 属性の対象 81](#_Toc427326950)

[図 15　Formクラスの属性 81](#_Toc427326951)

[9.5.2.2 属性の名称及び型 82](#_Toc427326952)

[9.5.3 入力チェックの対応 82](#_Toc427326953)

[9.5.4 入れ子構造（ネスト構造）のFormクラスについて 83](#_Toc427326954)

[9.5.5 属性の型変換について 83](#_Toc427326955)

[9.6. Controllerクラスの実装 84](#_Toc427326956)

[9.6.1 Controllerクラスの作成 84](#_Toc427326957)

[9.6.2 Form初期化のメソッド 84](#_Toc427326958)

[9.6.3 リクエスト処理のメソッドの作成 85](#_Toc427326959)

[9.6.4 リクエスト処理のマッピング 85](#_Toc427326960)

[9.6.5 遷移先の設定 86](#_Toc427326961)

[9.6.5.1 通常の画面遷移 86](#_Toc427326962)

[9.6.5.2 リダイレクトによる画面遷移 87](#_Toc427326963)

[9.6.5.3 フォワードによる画面遷移 87](#_Toc427326964)

[9.6.6 入力チェックの対応 88](#_Toc427326965)

[9.6.7 通常の入力チェック 88](#_Toc427326966)

[9.6.8 グループを設定した入力チェック 88](#_Toc427326967)

[9.6.9 トランザクショントークンの設定 89](#_Toc427326968)

[9.6.10 トランザクショントークンのネームスペース 89](#_Toc427326969)

[9.6.11 トランザクションの開始 89](#_Toc427326970)

[9.6.12 トランザクションのチェック 90](#_Toc427326971)

[9.6.13 その他のメソッド 90](#_Toc427326972)

[9.6.14 Formクラス－ドメイン層のJavaBeanの変換 90](#_Toc427326973)

[9.6.15 メッセージ出力 90](#_Toc427326974)

[9.7. Helperクラスの実装 91](#_Toc427326975)

[9.7.1 Helperクラスの作成 91](#_Toc427326976)

[9.8. Validatorクラスの実装 92](#_Toc427326977)

[9.8.1 Spring Validatorクラスの作成 92](#_Toc427326978)

[9.9. JSPの実装 94](#_Toc427326979)

[9.9.1 外部ファイル（CSS、JavaScript、画像ファイル）の読み込み方法 94](#_Toc427326980)

[9.9.1.1 ファイル読み込みの記述方法 94](#_Toc427326981)

[9.9.2 入力項目の記述方法 95](#_Toc427326982)

[9.9.2.1 テキストフィールド 95](#_Toc427326983)

[9.9.2.2 入力チェックのエラーメッセージ表示 95](#_Toc427326984)

[9.9.2.3 テキストエリア 95](#_Toc427326985)

[9.9.2.4 ラジオボタン 95](#_Toc427326986)

[9.9.2.5 チェックボックス 96](#_Toc427326987)

[9.9.2.6 プルダウン 96](#_Toc427326988)

[9.9.2.7 hidden項目 96](#_Toc427326989)

[9.9.3 出力項目の記述方法 97](#_Toc427326990)

[9.9.3.1 通常の出力項目 97](#_Toc427326991)

[9.9.3.2 入力項目を出力する場合 97](#_Toc427326992)

[9.9.3.3 項目（ラベル） 97](#_Toc427326993)

[9.9.3.4 メッセージ 97](#_Toc427326994)

[9.9.3.5 ガイダンスメッセージ 98](#_Toc427326995)

[9.9.3.6 フォーマットを指定した出力 98](#_Toc427326996)

[9.9.4 繰り返し項目の記述方法 99](#_Toc427326997)

[9.10. プロパティファイルの実装 100](#_Toc427326998)

[9.10.1 ファイル一覧 100](#_Toc427326999)

[9.10.2 読み込み設定 100](#_Toc427327000)

[9.10.2.1 修正対象ファイル（メッセージリソースの読み込み） 100](#_Toc427327001)

[9.10.2.2 修正内容（メッセージリソースの読み込み） 101](#_Toc427327002)

[9.10.2.3 修正対象ファイル（プレイスホルダーの設定） 101](#_Toc427327003)

[9.10.2.4 修正内容（プレイスホルダーの設定） 101](#_Toc427327004)

[9.10.3 プロパティファイルの記述 102](#_Toc427327005)

[9.10.3.1 入力チェックエラーメッセージプロパティ 102](#_Toc427327006)

[9.11. CSSファイルの実装 103](#_Toc427327007)

[9.11.1 CSSファイルの実装 103](#_Toc427327008)

[9.12. JavaScriptファイルの実装 104](#_Toc427327009)

[9.12.1 JavaScriptファイルの実装 104](#_Toc427327010)

# はじめに

## 目的

内部設計／製造作業要領・規約は、システム開発に携わる業務開発者に対し、内部設計／製造の一連の作業手順、設計・実装の観点、遵守すべき規約を示すことで、作業の効率化を図るとともに、多人数での開発における作業者間の品質のばらつきを防止することを目的としている。

なお、本書ではサンプルの前提として架空の「航空チケット予約システム」を題材にしている。「航空チケット予約システム」のシステム概要については、TERASOLUNA Server Framework for Java 5.x 開発手順で別途提供している「サンプル設計書」を参照されたい。本書の本文中に登場する「本システム」とは、特に断りのない限りこの「航空チケット予約システム」を指す。

## 留意事項

本書は内部設計／製造作業要領・規約のサンプルであり、プロジェクト特性に合わせてテーラリングされることを前提としている。本書を利用する際は、「カスタマイズポイント一覧」で提示している情報を参考にして、記述内容の追加、変更、削除を行うこと。

本書は対象システムの特性を踏まえ、TERASOLUNA Server Framework for Java 5.x を用いた設計・実装方針を選択・決定した内容を記したものである。設計・実装における詳細説明はTERASOLUNA Server Framework for Java 5.xのガイドラインを参照しており、本書はガイドラインとあわせて参照するものである。

## 章構成

本書の章構成をに示す。

表 1 内部設計／製造 作業要領・規約の章構成

|  |  |  |
| --- | --- | --- |
| **章番** | **章名** | **概要** |
| 1 | はじめに | 本章  内部設計／製造 作業要領・規約の目的、留意事項、および章構成について記載 |
| 2 | ソフトウェア概要 | 使用するソフトウェアおよびアーキテクチャの概要について記載 |
| 3 | 開発環境 | 内部設計・製造工程の成果物作成に使用するツールや製造時の開発環境について記載 |
| 4 | 内部設計・製造の流れ | 内部設計・製造工程の作業の流れ、前提となる入力情報、各作業で作成する成果物とその作成単位について記載 |
| 5 | 命名付与基準 | 「」で記載した成果物を作成する上で遵守する必要のある、ID体系や成果物の命名付与基準について記載 |
| 6 | 開発リソースの構成 | プロジェクト構成、ディレクトリ構成、パッケージ構成などについて記載 |
| 7 | 内部設計・製造の基本方針 | 内部設計・製造を行う上で考慮する必要があるポイントについて記載 |
| 8 | 設計書の記述要領 | 「」で記載した成果物である設計書の記述単位および記述項目について記載 |
| 9 | コーディングガイド | 製造を行う上で遵守する必要のあるコーディングルールについて記載 |

# ソフトウェア概要

本章では、本システムのソフトウェア概要について説明する。

## ソフトウェア概要

本システムで使用するソフトウェアの概要をに示す。

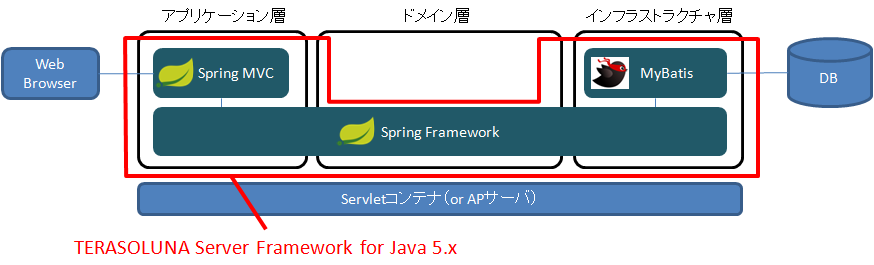


図 1　ソフトウェア概要図

本システムで採用するTERASOLUNA Server Framework for Java 5.x で使用するフレームワークの一覧[[1]](#footnote-1)をに示す。

表 2 使用フレームワーク一覧

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **名称** | **Ver** | **説明** | **ライセンス** | **URL** |
| Spring Framework | 4.1.4 | DIコンテナ、AOPなどの機能を持つオープンソースフレームワーク。 | Apache License 2.0. | http://spring.io/ |
| Spring MVC | 4.1.4 | 画面遷移、入力チェックなどのプレゼンテーション層の処理を行なうオープンソースフレームワーク。 | Apache License 2.0. | http://spring.io/ |
| MyBatis | 3.2.8 | SQLをベースとした、O/Rマッピングフレームワーク。 | Apache License 2.0. | http://mybatis.github.io/mybatis-3/ |

## アーキテクチャ概要

本システムのアーキテクチャの概要図をに、開発者が開発する要素の一覧をにそれぞれ示す。

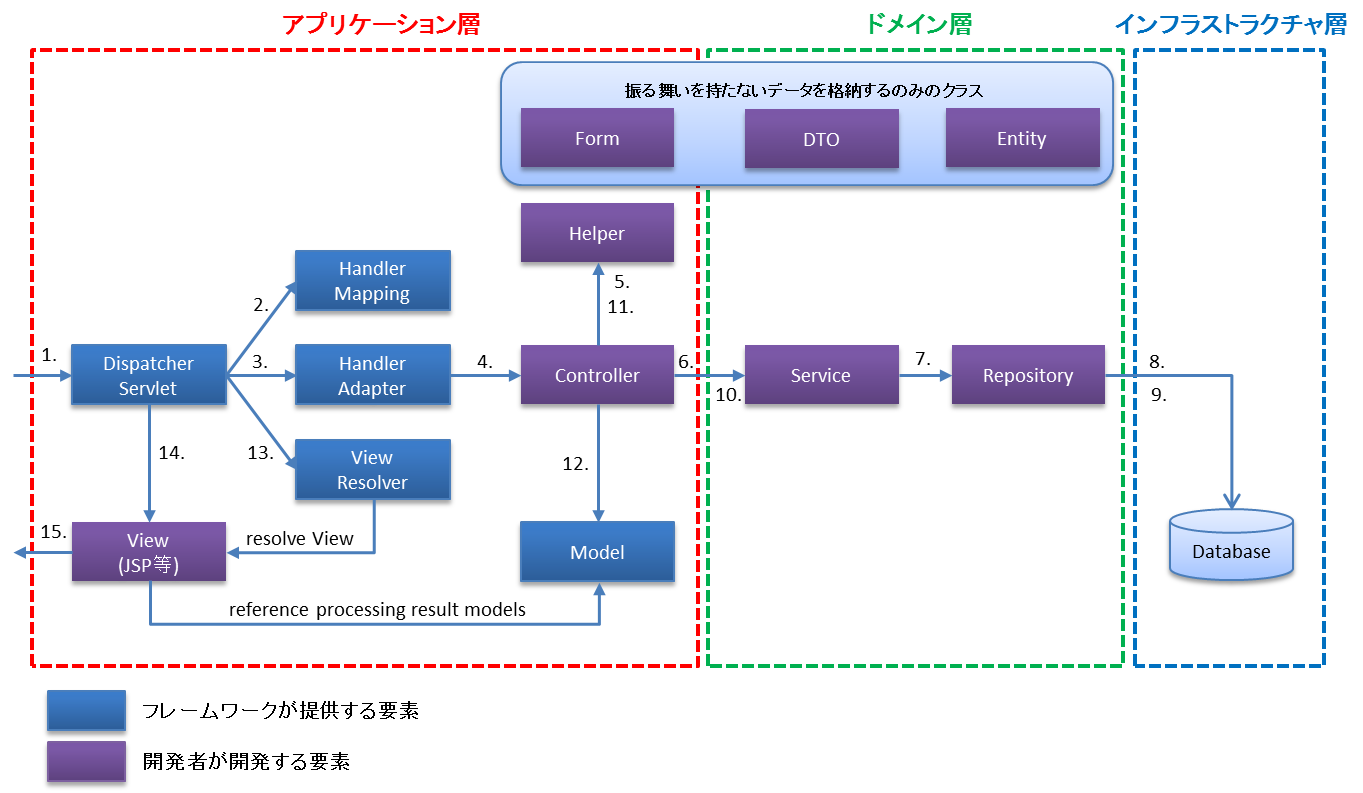


図 2　アーキテクチャ概要図

表 3　開発者が開発する要素の一覧

| **要素** | **概要** |
| --- | --- |
| Controller | リクエストを処理にマッピングし、結果をViewに渡すという画面遷移とセッション管理を担う。 |
| Helper | Controllerを補助する役割を担い、アプリケーション層とドメイン層のモデル相互変換など、Controller本来の処理以外の処理を行う。  Helperは必須ではなく必要に応じて作成する。 |
| Form | 画面のフォームを表現する。フォームの情報をControllerに渡したり、Controllerからフォームに出力する際に用いる。 |
| Entity | ビジネスを行う上で必要な資源や、ビジネスを行っていく過程で発生するものを表現するモデル。  本システムでは、振る舞いを持たないデータを格納するのみのクラス。 |
| DTO | 振る舞いを持たないデータを格納するのみのクラス。業務処理の入出力や画面のデータ構成の単位となる。 |
| Service | 業務処理を提供する。トランザクション境界となる。  インターフェースクラスと実装クラスから構成される。 |
| Repository | Entityの問い合わせや作成、更新、削除のようなCRUD処理を担う。インターフェースクラスと実装クラスから構成されるが、本システムではMyBatis3を利用するため、インターフェースクラスのみ開発する。 |
| View（JSP等） | クライアントへの出力を担う。Spring MVC のViewクラスが該当するが、本システムではHTMLを出力するため、JSPを実装する。 |

### 処理フロー

APサーバが処理を受け取ってからクライアントに処理結果を返却するまでの処理フローを以下に示す。処理番号は中の矢印上の数字を表している。

1．DispatcherServletが、リクエストを受け取る。

2．DispatcherServletは、リクエスト処理を行うControllerの選択を、HandlerMappingに委譲する。

HandlerMappingは、リクエストURLにマッピングされているControllerを選定し、ControllerをDispatcherServletへ返却する。

3．DispatcherServlet は、HandlerAdapterに対してControllerのビジネスロジック処理の実行を委譲する。

4．HandlerAdapter は、Controllerのビジネスロジック処理を呼び出す。

5．Controllerは、Formの情報をEntityまたはDTOに変換する。

（オプション）Helperを呼び出して変換する。

6．Controllerは、EntityまたはDTOを用いて、Serviceを呼び出す。

7．Serviceは、Repositoryを呼び出す。

8．Repostitoryは、O/R Mapperを呼び出し、EntityまたはDTOを永続化する。

9．（実装依存）O/R Mapperは、DBにEntityまたはDTOの情報を保存する。

10．Serviceは、業務処理結果のEntityまたはDTOを、Controllerに返却する。

11．Controllerは、EntityまたはDTOをFormに変換する、

　　（オプション）Helperを呼び出して変換する。

12．Controllerは、処理結果をModelに設定し、遷移先のビュー名を返却する.

13．DispatcherServletは、ビュー名に対応するViewの解決を、ViewResolverに委譲する。

ViewResolverは、ビュー名にマッピングされているViewを返却する。

14．DispatcherServletは、返却されたViewに、レンダリング処理を委譲する。

15．Viewは、Modelが持つ情報をレンダリングして、レスポンスを返却する。

# 開発環境

本章では、内部設計・製造の成果物作成に使用するツール等の開発環境について示す。

## 開発者のローカル環境

開発者のローカル環境をに示す。

表 4 開発環境一覧

|  |  |
| --- | --- |
| **名称** | **説明** |
| Microsoft Office Excel 2010 | 表計算・文章作成ツール。内部設計の成果物は基本的にExcel形式で作成する。 |
| Astah\* community 6.7 | UMLモデリングツール。クラス図を作成する際はUMLのアクティビティ図で作成し、Excel形式の設計書に貼付する。 |
| Oracle JDK 7.0 Update45 | アプリケーション実行環境。 |
| Spring Tool Suite 3.4（STS） | 統合開発ツール。 |
| Apache Tomcat 7.0 | アプリケーションを実行するサーバ機能（Servletコンテナ）。 |
| PostgreSQL 9.2.4 | 本システムで利用するRDBMS。 |
| Firefox ESR 24.6 | 本システムの利用対象とするWebブラウザ。 |

# 内部設計・製造の流れ

本章では、内部設計・製造の作業の流れ、各作業で作成する成果物とその作成単位について示す。

## 内部設計の作業の流れ

内部設計・製造で業務開発者が実施する作業の流れをに示す。



図 3　内部設計・製造の作業の流れ

## 作業概要と成果物

内部設計で実施する作業の概要と成果物、各成果物の作成単位をに示す。

表 5　作業概要と成果物一覧

|  |  |  |  |
| --- | --- | --- | --- |
| **作業** | **概要** | **成果物** | **作成単位** |
| Entityの抽出 | イベント単位、テーブル単位におけるアクセスパターンを整理し、主体となるEntityを抽出する。 | アクセスパターン一覧 | システムで一部 |
| クラス図（エンティティ） | システムで一部 |
| Repositoryの抽出 | 抽出したEntityから必要となるRepositoryを抽出する。 | リポジトリ一覧 | システムで一部 |
| Repository毎の設計 | 抽出したRepositoryが保持すべき処理を明確にし、Repositoryの機能を設計する。 | クラス仕様書（リポジトリ） | リポジトリ単位でファイルを作成し、処理単位で記述 |
| Serviceの抽出 | 外部設計の成果物をもとにビジネスロジックを担当するServiceを抽出する。  抽出したServiceの処理を精査し、Serviceの機能を設計する。 | クラス仕様書（サービス） | クラス単位でファイルを作成し、メソッド単位で記述 |
| Service毎の設計 |
| URLの抽出 | 外部設計の成果物をもとにControllerの設計単位となるURLのグループ単位を抽出する。  抽出したURLに対してパラメータの付与やControllerのクラス、メソッド名の抽出及び設計を行う。 | イベントURLマッピング一覧  ※URLの構成（URI、パラメータ等）を一覧化したもの | システムで一部 |
| URL毎の設計 |
| イベントフロー毎の設計 | 本設計工程において抽出・設計した各機能・モデルの抽出単位の妥当性およびその整合性を確認するため、イベント単位における処理フローを整理し、設計書に反映する。 | イベントフロー図  ※jsp、Form、Controller、セッション、Service、Repository、Entity、DTO等のモデルを記載したもの | イベント単位 |
| 共通機能の抽出 | Repository、Service、Controller等の各機能・モデルの設計と並行して、共通機能の抽出と設計を行う。 | 共通機能の処理に応じて作成  ※本手順では規定しない | 共通機能単位 |
| 共通機能毎の設計 |

# 命名付与基準

本章では、内部設計・製造を行う上で遵守すべき命名付与基準を示す。

## 設計工程における成果物の単位及び命名

### 基本方針

#### システムで一部作成する成果物

* 命名

＜成果物名＞

* 例）

イベントURLマッピング一覧

#### ユースケースグループ単位で作成する成果物

* 命名

＜成果物名＞\_＜ユースケースグループ名＞

* 例）

現状、対応する成果物なし

#### ユースケース単位で作成する成果物

* 命名

＜成果物名＞\_＜ユースケースID＞\_＜ユースケースグループ名＞\_＜ユースケース名＞

* 例）

イベントフロー図\_B2\_チケット予約サービス\_チケットを予約する

#### その他の単位で作成する成果物

* 命名

＜成果物名＞\_＜個別成果物名＞

* 例）

クラス仕様書(リポジトリ)\_ReservationRepository

### 成果物ファイル名

本システムにおける内部設計工程の成果物ファイル名をに示す。

表 6　成果物ファイル名一覧

|  |  |  |  |
| --- | --- | --- | --- |
| **項番** | **成果物** | **形式** | **例** |
| 1 | アクセスパターン一覧 | ＜成果物名＞ | アクセスパターン一覧 |
| 2 | リポジトリ一覧 | リポジトリ一覧 |
| 3 | クラス図 | クラス図(エンティティ) |
| 4 | イベントURLマッピング一覧 | イベントURLマッピング一覧 |
| 5 | イベントフロー図 | ＜成果物名＞\_＜ユースケースID＞\_＜ユースケースグループ名＞\_＜ユースケース名＞ | イベントフロー図\_B2\_チケット予約サービス\_チケットを予約する |
| 6 | クラス仕様書（リポジトリ） | ＜成果物名＞\_＜クラス名＞ | クラス仕様書(リポジトリ)\_MemberRepository |
| 7 | クラス仕様書（サービス） | ＜成果物名＞\_＜ユースケースID＞\_＜クラス名＞ | クラス仕様書(サービス)\_B1\_TicketSearchServiceImpl |

## 製造工程における成果物の単位及び命名

### URI関連の命名規約

#### URLの命名規約

URLからユースケースが判断できるように、業務的な意味の英語表記を名称として付与する。

* 命名

URL　：　/＜名称付与１＞/＜名称付与２＞

名称付与１　：　「ユースケースグループ」を特定できる名詞を指定。

名称付与２　：　「ユースケース」を特定できる動詞を指定。

* 例）

/Ticket/search、/Ticket/reserve

#### HTTPパラメータの命名規約

基本的な画面遷移におけるリクエストについては統一性を持たせるため、に示す特定のHTTPパラメータを用いることとする。なお、リクエストを精査し、類似する処理が複数存在する場合、一覧への追加を検討すること。

表 7　HTTPパラメータ一覧

|  |  |
| --- | --- |
| **HTTPパラメータの例** | **内容** |
| form | 入力画面を表示するためのリクエスト |
| confirm | 入力画面から確認画面に遷移するためのリクエスト |
| complete | ・処理を確定させるための処理における結果を表示するためのリクエスト  ・一般的に確認画面から処理結果画面に遷移する際のPRG（Post-Redirect-GET）パターンにおけるGETリクエスト |
| redo | 確認画面から入力画面に戻るためのリクエスト |

### クラス関連の命名規約

#### Controllerクラスの命名規約

（１）　クラス名

Controllerが担当するユースケースが判断できるような名称を付与する。基本的には担当ユースケースを英語に置き換えたものとする。

* 命名

＜ユースケースを表す英語の名称＞＋Controller

* 例）

MemberRegisterController

（２）　メソッド名

Controllerのメソッド名はマッピングされるURL及びパラメータが判断できるような名称とする。基本的にURLにHTTPパラメータを付与したものとする。

* 命名

＜URLの名称付与２＞＋＜HTTPパラメータ＞

* 例）

URL　：　reserve、HTTPパラメータ　：　Complete

reserveComplete

#### Formクラスの命名規約

Formを利用するControllerとの紐付けが可能な名称を付与する。FormとControllerが一対一の関係の場合、Controllerクラスのサフィックスの“Controller”を“Form”に置き換えた名称とする。Controllerが複数のFormを利用する場合、Formが扱うデータが判断できるような名称を付与する。

* 命名

＜Controllerと紐付け可能な名称＞＋Form　※一対一の場合

＜扱うデータが判断可能な名称＞＋Form　※複数のFormを利用する場合

* 例）

MemberRegisterForm　※一対一の場合

TicketReserveSelectFlightForm　※複数のFormを利用する場合

#### Serviceインターフェース、実装クラスの命名規約

Serviceが担当するビジネスロジックが判断できるような名称を付与する。

* 命名

＜ビジネスロジックの内容を表す名称＞＋Service　※インターフェース

＜ビジネスロジックの内容を表す名称＞＋ServiceImpl ※実装クラス

* 例）

TicketReserveService　※インターフェース

TicketReserveServiceImpl ※実装クラス

#### SharedServiceインターフェース、実装クラスの命名規約

SharedServiceが担当する共通的なビジネスロジック（複数のServiceに関わるビジネスルール等）が判断できるような名称を付与する。

* 命名

＜ビジネスロジックの内容を表す名称＞＋SharedService　※インターフェース

＜ビジネスロジックの内容を表す名称＞＋SharedServiceImpl ※実装クラス

* 例）

TicketSharedService　※インターフェース

TicketSharedServiceImpl ※実装クラス

#### Repositoryインターフェースの命名規約

Repositoryの対象となるエンティティが判断できるような名称を付与する。

* 命名

＜対象のエンティティクラス＞＋Repository

* 例）

MemberRepository

#### Entityクラスの命名規約

クラス名からエンティティを特定可能な名称を付与する。エンティティの対象がテーブルの場合、テーブルの名称とする。ただし、テーブル名称が“\_”で区切られた英単語の場合、各単語の一文字目を大文字に変換して連結した名称とする。

* 命名

＜エンティティを特定可能な名称＞

* 例）

FareType　※テーブル名は「FARE\_TYPE」

#### DTOクラスの命名規約

クラス名からDTOの内容が判断できるような名称を付与する。振る舞いを持たないデータのみのクラスであるFormクラスやEntityクラスの名称を参考に、DTOの内容が分かる名称を付与すること。また、入出力での利用が明確な場合、末尾にInputDtoやOutputDtoを付与すること。

* 命名

＜DTOの内容を表す名称＞＋Dto

＜DTOの内容を表す名称＞＋InputDto　※引数として利用するDtoの場合

＜DTOの内容を表す名称＞＋OutputDto　※戻り値として利用するDtoの場合

* 例）

SelectFlightDto

#### 例外クラスの命名規約

クラス名から例外の内容が判断できるような名称を付与する。

* 命名

＜例外の内容を表す名称＞＋Exception

* 例）

AtrsBusinessException

### SQL関連の命名規約

#### SQL\_IDの命名規約

基本的なCRUDに関する名称はリポジトリとの関連性を明確にするため、SpringのPagingAndSortingRepositoryインターフェースのメソッド名を基本とした名称を付与することとする。

* SQL\_IDのプレフィックス1

SQL\_IDのプレフィックス1の一覧をに示す。

表 8 SQL\_IDのプレフィックス1一覧

|  |  |  |
| --- | --- | --- |
| **区分** | **SQL\_IDのプレフィックス１** | **内容** |
| データ取得 | findOne | 主キーによる１件のデータを取得する場合 |
| findAllSort | 全てのデータをソートした状態で取得する場合 |
| findAllPage | ページング対象に該当するデータを取得する場合 |
| fingList | 複数件のデータを取得する場合 |
| 件数取得 | count | データ件数を取得する場合 |
| 存在チェック | exists | 主キーによるデータの存在チェックを行う場合 |
| データ削除 | delete | 主キーによる１件のデータを削除する場合 |
| deleteAll | 全てのデータを削除する場合 |
| deleteEntity | エンティティに該当するデータを削除する場合 |
| deleteList | エンティティのリストに該当するデータを削除する場合 |
| データ挿入 | insert | データを挿入する場合 |
| データ更新 | update | データを更新する場合 |
| プロシージャ呼び出し | call＜プロシージャ名＞ | プロシージャを呼び出す場合 |

* 関連エンティティを操作する場合（オプション）

プレフィックス２として関連エンティティが分かる名称を付与する。

* 条件を付与する場合（オプション）

プレフィックス３として「By＜条件＞」を付与する。

* 排他ロックのために取得する場合（オプション）

サフィックスとして「ForUpdate」を付与する。

* 例）

findOne

findOneByXxx

findOneByYyyForUpdate

#### SQL MAPPERファイルの命名規約

SQL MAPPERファイルはリポジトリ単位で作成する。そのためSQL MAPPERファイルはRepositoryインターフェース名に“.xml”を付与したものとする。

#### SQL MAPPERファイルのnamespace命名規約

SQL MAPPERファイルにはnamespaceを付与することとし、namespaceはリポジトリインターフェースの完全修飾名とする。

* 命名

リポジトリインターフェースの完全修飾名

* 例）

リポジトリインターフェース　：　jp.co.ntt.atrs.domain.repository.member.MemberRepository

namespace　：　jp.co.ntt.atrs.repository.member.MemberRepository

# 開発リソースの構成

## 全体構成

### プロジェクト構成

本システムにおけるSTSのプロジェクト構成をおよびに示す。

|  |
| --- |
|  |

図 4　プロジェクト構成図

表 9　STSプロジェクト一覧

|  |  |  |
| --- | --- | --- |
| **プロジェクト名** | **表記** | **内容** |
| atrs-domain | ${domain} | ドメイン層の資材を格納するプロジェクト |
| atrs-env | ${env} | 環境依存の資材（設定ファイル等）を格納するプロジェクト |
| atrs-initdb | ${initdb} | DB構築、DB初期ロードデータ等を格納するプロジェクト |
| atrs-parent | ${parent} | Mavenの親pomを格納するプロジェクト  必要に応じてcheckstyleやfindbugsのルール等を格納する |
| atrs-web | ${web} | アプリケーション層の資材を格納するプロジェクト |

### ディレクトリ構成

STSプロジェクトのディレクトリ構成をに示す。

表 10 ディレクトリ構成

|  |  |
| --- | --- |
| **${common}、${domain}、${env}** | **${web}** |
|  |  |

|  |  |  |
| --- | --- | --- |
| **ディレクトリ名** | **用途** | **内容** |
| src/main/java | 開発対象 | Javaのソースコードを格納するディレクトリ  クラスパスに設定される |
| src/main/resources | 開発対象 | リソースファイルを格納するディレクトリ  クラスパスに設定される |
| src/test/java | テスト用 | Javaのソースコードを格納するディレクトリ  クラスパスに設定される |
| src/test/resources | テスト用 | リソースファイルを格納するディレクトリ  クラスパスに設定される |
| src/main/webapp | 開発対象 | Webアプリケーション固有の資材（JSP、CSS等）を格納するディレクトリ  ${web}プロジェクトにのみ存在 |
| libsrc | ― | ライブラリ（jar）のソースコード（jar）を格納するディレクトリ |
| target | ― | Mavenのビルド用ディレクトリ |

### Javaパッケージ構成

#### パッケージのプレフィックス

パッケージのプレフィックス（ルートパッケージ）について以下に示す。

プレフィックス　：　jp.co.ntt.atrs

以下、${ルート}と表記する

#### パッケージとプロジェクトの関係

パッケージと格納するプロジェクトの関係をに示す。

表 11 パッケージとプロジェクトの対応一覧

|  |  |  |
| --- | --- | --- |
| **パッケージ** | **プロジェクト** | **内容** |
| ${ルート}.app.${ユースケースID} | ${web} | アプリケーション層のControllerクラス、Formクラスのソースコードを格納する  appの配下にユースケース単位でサブパッケージを設ける |
| ${ルート}.domain.service.分類 | ${domain} | ドメイン層のServiceクラス、DTOクラスのソースコードを格納する  分類はServiceの名称でサブパッケージを設ける |
| ${ルート}.domain.model | ${domain} | ドメイン層のEntityクラスのソースコードを格納する。 |
| ${ルート}.domain.repository.分類 | ${domain} | ドメイン層のRepositoryインターフェースのソースコートを格納する  分類はrepositoryの名称でサブパッケージを設ける |
| ${ルート}.common.分類 | ${common} | 共通機能のソースコードを格納する  機能を特定可能な名称で分類のサブパッケージを設ける |

### JSPファイルの配置先

JSPファイルは ${web}/src/main/webapp/WEB-INF/views/ 配下にユースケースIDのディレクトリを作成し、その中に配置することとする。

例）　「チケットを予約する」の場合

${web}/src/main/webapp /WEB-INF/view/B2/＜「イベントマッピング一覧」の「JSP名」＞.jsp

### 外部ファイルの配置先

CSS、JavaScript、画像ファイル等の外部ファイルはに示すパスに格納する

表 12 外部ファイルの格納先一覧

|  |  |  |
| --- | --- | --- |
| **項番** | **パス** | **内容** |
| 1 | ${web}/src/main/webapp/resources/css | CSSファイルを格納する |
| 2 | ${web}/src/main/webapp/resources/image | 画像ファイルを格納する |
| 3 | ${web}/src/main/webapp/resources/js | JavaScriptファイルを格納する |

### プロパティファイルの配置先

本システムで利用するプロパティファイルの配置先をに示す。

表 13 プロパティファイルの配置先一覧

|  |  |  |  |
| --- | --- | --- | --- |
| **項番** | **ディレクトリ名** | **用途** | **備考** |
| 1 | ${domain}/src/main/resources  /META-INF/spring | 業務プロパティを保持するプロパティファイル | 「初期出発空港コード」等 |
| 2 | ${web}/src/main/resources/i18n | GUIコンポーネントの項目名や、画面メッセージ、ログメッセージを設定するファイル | 入力フォーム名称など |

# 内部設計・製造の基本方針

本章では、内部設計及び製造を行う上での基本方針を示す。

## 画面遷移

### RDBMSの更新処理（C/U/D）を伴う画面遷移

RDBMSの更新処理（Create、Update、Delete）を伴う画面遷移の場合、二重送信防止対策としてPRG（Post-Redirect-Get）パターンを適用する。このため、画面遷移としてはA画面からB画面への遷移のように一つの遷移に見えるが、実際はPostのリクエストとGetのリクエストの二つの処理を実装することとなる。PRGパターンについては「 」を参照すること。

### 検索画面における画面遷移

本システムの検索画面では、ページネーション機能を使用して検索画面を実装する。ページネーションについては「 」を参照すること。

### ユーザログイン・ユーザログアウトにおける画面遷移

ユーザのログイン処理及びログオフ処理は通常の画面遷移とは異なり、ユーザ認証やユーザ情報の保持方法について特別な検討が必要となる。ログイン・ログオフをSpring Securityを利用して実現する場合、Spring Securityで定められたインターフェースで認証を行うため、通常、Controllerの実装は不要となる。そのため、まずはSpring Securityの利用有無を決定する必要がある。Spring Securityについては「 」を参照すること。

## 共通領域

### 共通領域の定義

共通領域定義とは具体的にはJavaEEのアプリケーションスコープやセッションスコープで管理するオブジェクト及び、Spring Security のユーザ情報（UserDetails）にて保持する情報についての定義である。各スコープに定義するオブジェクトは以下の観点で抽出する。

* アプリケーションスコープ

システムが稼働中に値が不変で、複数の処理や画面から参照するもの。ただし、各業務や画面に認めるのは参照だけであり、アプリケーションスコープ中のオブジェクトを変更、削除することは禁止とする。

なお、基本的にこのスコープの実装はAP基盤開発者等が実施し、個々の業務開発者が実施することはない。

* セッションスコープ

ログイン中の状態でステートフルとして管理しなければならない情報に関するオブジェクトで、複数の業務処理から参照されるもの。ただし、ユーザに関する情報はSpring Securityのユーザ情報（UserDetails）にて定義する。また、単純な画面間の値の受け渡しに使用するオブジェクトでセッションスコープに保存するものに関しては、共通領域定義としては定義しない。

* Spring Securityのユーザ情報（UserDetails）

Spring Securityを利用した認証時（ログイン時）に、ユーザ名やパスワード等デフォルトで用意されている情報以外に併せて取得・保持するユーザの情報を定義する。

なお、基本的にこのスコープの実装はAP基盤開発者等が実施し、個々の業務開発者が実施することはない。

## トランザクション管理

### DBトランザクション

DBトランザクションとは、RDBMSのトランザクションのことである。本システムにおけるDBトランザクションの方針を以下に示す。

* トランザクション境界

ドメイン層のServiceクラスのメソッドをトランザクション境界とする。

* Controllerメソッドにおける注意点

Controllerクラスの一つのメソッド内で複数のServiceクラスのメソッドを呼び出す場合、別々のDBトランザクションになることを考慮して設計を行うこと。

概要をに示す。



図 5　Controllerメソッドとトランザクションの関係

* 排他制御

DBトランザクションは「悲観的ロックによる排他制御」により、データの整合性を保障する方針とする。

## 入力チェック

### 入力チェックの種類

#### 単項目チェック

単項目チェックはBean ValidationおよびHibernate Validatorのアノテーションを利用し、Formクラスの属性にアノテーションとして設定する。本システムで利用する単項目チェックをに示す。なお、その他の単項目チェックについても独自のアノテーションチェックとして扱う要望が出た場合、その都度、独自チェックの作成可否を検討すること。

独自ルールのアノテーションの作成方法についてはガイドライン（「5.5 入力チェック」の「How to extend」）を参照すること。

表 14　単項目チェック一覧

| **ルール名** | | **アノテーション** | **区分** | **備考** |
| --- | --- | --- | --- | --- |
| 必須 | | @NotNull |  | 本システムでは文字列型の未入力を空文字ではなくNullと判断することとする |
| 桁数範囲 | | @Size |  | 属性の型に応じて選択  プリミティブ型の使用は禁止する |
| @Digits |  |
| @DecimalMax |  |
| 文字種別 | 半角文字 | @HalfWidth | 独自実装 |  |
| 半角数字 | @HalfWidthNumber | 独自実装 |  |
| 全角文字 | @FullWidth | 独自実装 |  |
| 全角カタカナ | @FullWidthKatakana | 独自実装 |  |
| 文字列指定サイズ | @FixedLength | 独自実装 |  |
| クレジットカード番号形式 | | @CreditCardNumber |  |  |
| Emailアドレス形式 | | @Email |  |  |
| URL形式 | | @URL |  |  |
| 未来日付 | | @Future |  |  |
| 過去日付 | | @Past |  |  |
| 最小値 | | @Min |  | 属性の型に応じて選択 |
| @DecimalMin |  |
| 最大値 | | @Max |  | 属性の型に応じて選択 |
| @DecimalMax |  |
| コードリスト存在 | | @ExistInCodeList |  | 利用可能なコードリストについては「 」を参照 |
| 正規表現 | | @Pattern |  | 正規表現にて入力を許容するルールを指定 |

#### 相関項目チェック

項目間の整合性をチェックする相関項目チェックはアプリケーション層にて行うこととする。チェックロジックはControllerに実装するのではなく、チェックロジック専用のValidatorに実装し、そのValidatorをControllerにインジェクションして利用すること。

### 入力チェックにおける考慮ポイント

#### 入力チェックの実施タイミング

更新処理のリクエストでは必ず入力チェックを実施すること。入力チェックの実施タイミングの概要をに示す。

* 全ての入力値に対して入力チェックを再度実施すること。
* ValidatorにてSharedServiceやServiceを利用してDBチェックを行っていた場合についても、更新処理時に同一トランザクション内にて再度実施すること。
* 確認画面から処理結果画面への更新処理で入力チェックエラーとなった場合の画面遷移は、後述の「」に従うこと。



図 6　更新処理における入力チェックの実施タイミング

#### 相関項目チェックの実装方法

相関項目チェックにはSpring Validator（org.springframework.validation.Validatorインターフェースを実装した独自Validator）、またはBean Validation（アノテーションで設定する入力チェック）を利用する方法がある。本システムにおける実装方針についてに示す。

表 15　相関項目チェックの実装方針

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | **特徴** | **用途** | **本システムでの利用** |
| Spring Validator | 特定のクラスに対する入力チェックの作成が容易である。  Controllerでの利用が不便。 | 特定のフォームに依存した業務要件固有の入力チェック実装 | 業務処理固有の独自ValidatorはSpring Validatorで実装する。  基本的に業務開発者が実装する。 |
| Bean Validation | 入力チェックの作成はSpring Validatorほど容易ではない。  Controllerでの利用が容易。 | 特定のフォームに依存しない、開発プロジェクト共通の入力チェック実装 | 共通機能として定義するValidatorはBean Validationとして実装する。  基本的にAP基盤開発者や業務共通開発者が実装し、業務開発者に提供する。 |

#### エラー時の画面遷移

入力チェックでエラーとなった場合、通常は遷移元の画面を再表示する。しかし、確認画面から処理結果画面への更新処理における入力チェック等、入力画面でない画面からの遷移も存在する。

通常の利用においてこのケースが発生するのは開発中のバグであり、それ以外では入力データの改竄等の不正なリクエストの場合である。本システムにおける対応を以下に示す。

* 共通的なエラー画面に遷移する

不正なリクエストである点を考慮し、業務を継続させないため、共通的なエラー画面に遷移し、エラーメッセージを表示する。

#### エラー時に業務処理が必要な場合

リクエストスコープの前提で開発している場合、ステートレスであるため、場合によっては業務処理を再度実施する必要がある。入力チェックの対象画面に入力フォームとして定義されていない項目が存在する場合がこれに該当する。

エラー時における業務処理の再実施の概要および実施可否をそれぞれ、に示す。



図 7　エラー時おける業務処理の再実施

表 16　対象画面と業務処理の実施可否の一覧

|  |  |  |
| --- | --- | --- |
| **対象画面** | **業務処理** | **概要** |
| 入力フォームの項目のみ | 不要 | 入力フォームの項目のみの画面では、入力データがリクエストに存在するため、データが欠落することなく画面表示が可能である。 |
| 入力フォームの項目　＋  表示項目を hidden で定義 | 不要 | 表示項目がhiddenで定義されているため、データが欠落することなく画面表示が可能である。 |
| 入力フォームの項目　＋  表示項目をセッションに格納 | 不要 | 表示項目がセッションに格納されているため、データが欠落することなく画面表示が可能である。  ※ 本システムではリクエストスコープが前提であるため、原則としては利用しないが、画面遷移を繰り返す場合やControllerを跨ぐ場合、検索条件を保持する場合等、hiddenでの実装が複雑・困難な場合に利用する。 |
| 入力フォームの項目以外も存在 | 必要 | ステートレスであるため表示項目のデータが欠落した状態で画面表示されてしまう。そのため、データの再取得を行う必要がある。 |

## ログ出力

### ログ出力

業務開発者はデバッグログについて設計・実装を担当する。その他のログレベルについては、外部設計にてメッセージ設計が完了している、もしくはログ出力の実装をAP基盤等で実施するものである。

ログレベルと出力有無及び担当をに、ログ出力の例をにそれぞれ示す。

表 17　ログレベル一覧

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ログレベル | カテゴリ | 出力目的 | 運用時の  出力有無 | 業務開発者  の担当 |
| TRACE | 性能ログ | リクエストの処理時間の測定 | × | × |
| DEBUG | デバッグログ | 開発時のデバッグ | × | ○ |
| INFO | アクセスログ | 業務量の把握 | ○ | × |
| 業務ログ | ログイン・ログアウトの記録 | ○ | ○ |
| WARN | 業務エラーログ | 業務エラーの記録 | ○ | × |
| ERROR | システムエラーログ | システム運用の継続が困難な事象の記録 | ○ | × |

※「TRACE」の「性能ログ」とは、Controllerの処理開始、終了をログ出力したものであり、Controllerが返却したView名とModelに追加した属性、およびControllerの処理に要した時間が出力される。詳細についてはガイドライン（5.6.3.2.2 TraceLoggingInterceptor）を参照すること。

|  |
| --- |
| ●ログの共通フォーマット  date:yyyy-MM-dd HH:mm:ss△thread:${THREAD}△user:${USER}△X-Track:${X-TRACK}△level:${LEVEL}△logger:${LOGGER}△message:${MESSAGE}  △: タブを表す  ${THREAD} : スレッド名  ${USER} : ログイン時はログイン中のお客様番号。未ログイン時は "anonymousUser"。  ${X-TRACK}: リクエストごとに一意となるID。  ${LEVEL} : ログレベル(5桁)。  ${LOGGER} : ログを出力したクラスのFQCN(48桁)。  ${MESSAGE}: メッセージ本文。  ●アクセスログのメッセージ本文  ・アクセス開始時:  ACCESS START [RequestURL:${REQUEST\_URL}], [SessionID:${SESSION\_ID}], [RemoteAddress:${REMOTE\_ADDRESS}], [RemoteHost:${REMOTE\_HOST}]  ・アクセス終了時:  ACCESS FINISH [RequestURL:${REQUEST\_URL}], [SessionID:${SESSION\_ID}], [RemoteAddress:${REMOTE\_ADDRESS}], [RemoteHost:${REMOTE\_HOST}]  ${REQUEST\_URL} : リクエストURL  ${SESSION\_ID} : セッションID  ${REMOTE\_ADDREDSS} : クライアントのIPアドレス  ${REMOTE\_HOST} : クライアントのホスト名  ＜ログ出力の例＞  date:2014-01-09 17:09:17 thread:http-bio-8080-exec-7 user:anonymousUser X-Track:7957d62e669e498f911a576365d5d135 level:INFO logger:jp.co.ntt.atrs.app.common.logging.AccessLogFilter message:ACCESS START [RequestURL:http://10.167.89.149:8080/atrs/Auth/logout?complete], [SessionID:51108AAC5994772BA66BF3F82035DECE], [RemoteAddress:10.133.1.245], [RemoteHost:10.133.1.245]  date:2014-01-09 17:09:17 thread:http-bio-8080-exec-7 user:anonymousUser X-Track:7957d62e669e498f911a576365d5d135 level:INFO logger:jp.co.ntt.atrs.app.common.logging.AccessLogFilter message:ACCESS FINISH [RequestURL:http://10.167.89.149:8080/atrs/Auth/logout?complete], [SessionID:51108AAC5994772BA66BF3F82035DECE], [RemoteAddress:10.133.1.245], [RemoteHost:10.133.1.245]  ●業務処理中に出すINFOログのメッセージ本文(具体的にはログイン・ログアウト)  [${LOG\_ID}］${MESSAGE}  ${LOG\_ID} : ログメッセージID  ${MESSAGE} : メッセージ本文  ＜ログ出力の例＞  date:2014-01-09 17:09:08 thread:http-bio-8080-exec-8 user: X-Track:b1bc2424adb8486885945d01c223d933 level:INFO logger:j.c.n.a.domain.service.a1.AuthLoginServiceImpl message:[i.ar.a1.L0001] お客様番号[0000000001] ログイン |

図 8　ログ出力の例

### デバッグログについて

デバッグログについては出力内容等に特に制限を設けない。業務開発者が開発時に必要と思われる情報を出力して構わない。ただし、デバッグログのメッセージはハードコーディングとし、メッセージのプロパティに定義してはならない。

デバッグログはシステムの商用運用時は出力しない。そのため、商用運用時も障害解析等で必要となる情報はINFO以上のログにて出力するため、メッセージ設計に反映する必要がある。

## エラー・例外処理

### エラー・例外の種類

エラー・例外の種類をに示す。

表 18 エラー・例外一覧

|  |  |
| --- | --- |
| エラー・例外の種類 | 概要 |
| ビジネス例外 | ビジネスルールの違反を検知したことを通知する例外 |
| 正常稼働時に発生するライブラリ例外 | フレームワーク、及びライブラリ内で発生する例外のうち、システムが正常稼働している時に発生する可能性のある例外 |
| システム例外 | システムが、正常稼働している時に、発生してはいけない状態を検知したことを通知する例外 |
| 予期しないシステム例外 | システムが、正常稼働している時には発生しない非検査例外 |
| 致命的なエラー | システム（アプリケーション）全体に影響を及ぼす、致命的な問題が発生していることを通知するエラー |
| リクエスト不正時に発生するフレームワーク例外 | フレームワークが、リクエスト内容の不正を検知したことを通知する例外 |

### 業務開発者が設計・考慮すべき例外

業務開発者が設計・考慮すべき例外は以下の２つである。

* ビジネス例外

この例外はドメイン層でロジックにて、ビジネスルールを違反した場合に、アプリケーションにて明示的にスローする例外である。アプリケーションとして想定される状態であり、システム運用者による対処は不要となる。そのため、システム運用者による対処が必要なものをこの例外に含めてはならない。

フレームワークが提供する例外（BusinessException）もしくはこの例外を継承した例外クラスが該当するが、本システムではビジネスルールに応じた例外クラスを作成することとする。

* 正常稼働時に発生するライブラリ例外

この例外の例として、複数ユーザにて同一データを同時に更新しようとした際に発生する排他例外や、複数ユーザにて同一データを同時に登録しようとした際に発生する一意制約違反例外等がある。

業務開発者が考慮すべき例外として挙げているのは、これらの例外が発生した際、ユーザにどのように通知するのかがシステム毎に異なり、業務仕様に関係するためである。

ビジネス例外のように独自の例外クラスを用意する必要はなく、例外が発生した場合の振る舞いや表示メッセージ等について設計すればよい。

## セッション管理

### 基本方針

Spring-Frameworkのデフォルトはリクエストスコープである。本システムでもこの基本に従い、可能な限りリクエストスコープで設計を行うこととする。業務処理上、リクエストスコープでは実現が難しくセッションスコープを利用することになった場合、以下に示すセッションの特性を考慮して設計を行うこと。

セッションスコープを利用する場合、メモリリークが発生しないように適切に削除すること。また、セッションに格納するオブジェクトのライフサイクルをイベントフロー図に記載すること。

セッション管理の詳細についてはガイドライン（5.8 セッション管理）を参照すること。

### Controller内で持ち回る場合

Controller内で行われる画面遷移においてデータを持ち回る場合、Controllerに@SessionAttributesアノテーションを付与してセッションスコープに設定することとする。これによりセッションに格納したオブジェクトはController単位でライフサイクルが管理されるようになる。

### Controllerを跨いで持ち回る場合

複数のControllerを跨いだ画面遷移においてデータを持ち回る場合、セッションスコープのBeanを定義することとする。

セッションスコープのBean定義とは、セッションスコープに格納したいオブジェクトを@Componentと@Scope(“session”)のクラスアノテーションを付与してクラス定義することである。このオブジェクトを利用するControllerにて、@Injectと@ModelAttributeすることでControllerを跨いでセッションスコープに格納することが可能となる。

### ユーザ情報について

一般のWebシステムでは、ユーザのログイン状況とセッションは深く関係するが、本システムではログイン状況及びユーザ情報についてはセッションではなく、認証・認可にて設計することとする。

詳細については「 」を参照すること。

## メッセージ管理

本システムでは画面に表示するユーザメッセージをプロパティファイルにて管理することとする。詳細については「 」を参照すること。

## プロパティ管理

本システムではに示すデータをプロパティファイルにて管理することとする。なお、プロパティファイルの暗号化については実施しない。その他のデータについてもプロパティファイルにて扱う要望が出た場合、その都度、プロパティファイルへの適用可否を検討すること。

表 19 プロパティファイル一覧

| **対象データ** | **プロパティファイル** | **概要** |
| --- | --- | --- |
| 業務初期データ | atrs.properties | 「初期出発空港コード」等の各種初期データ  「予約代表者に必要な最小年齢」等の各種しきい値データ |
| 入力フォーム名称 | atrs-fields\_ja.properties | 「depAirport=出発空港」等の各種入力フォームの表示項目名称 |
| ユーザメッセージ | atrs-messages\_ja.properties | 共通バリデーションのエラーメッセージやユーザメッセージ |

## ページネーション

ページネーションとは、検索条件に一致するデータをページ分割して表示する方法である。ページネーションにはページを移動するためのリンク表示である「Pagination Links」と、ページネーションに関連する情報（合計件数、合計ページ数、表示ページ数等）を表示する「Pagination Information」の機能がある。ページネーションの詳細についてはガイドライン（5.11 ページネーション）を参照すること。

本システムでは「Pagination Links」は利用するが「Pagination Information」は利用せず、表示機能は独自に実装することとする。

## 二重送信防止、不正画面遷移チェック

### 二重送信防止

* サーバサイド（PRG：Post-Redirect-Getパターンによる防止）

外部設計の画面遷移図では１つのイベントとして扱っていたものが、PRGパターンを適用すると２つのリクエスト（POST＋GET）にマッピングされる。イベントURLマッピング一覧を作成する際にはPRGパターンを考慮して適切にURLを抽出すること。

* クライアントサイド（JavaScriptによるボタンの２度押し防止）

本システムでは二重送信を防止すべきイベントにJavaScriptによるAlertを設けることでクライアントサイドの２度押し防止対策とする。

### 不正画面遷移チェック

不正画面遷移のチェックとは、業務処理が一連の流れで実行される前提で設計されたシステムにおいて、この一連の流れを無視した画面遷移をチェックすることである。例えば、更新処理を行う画面遷移を直接呼び出す、更新処理後の処理結果画面からブラウザの戻るボタン等で直前の画面に戻り、再度更新処理を実行する等の画面遷移のことである。

本システムでは不正画面遷移チェックの対策として、トランザクショントークンチェック機能を利用することとする。トランザクショントークンに関する詳細についてはガイドライン（5.12.1.2.3 トランザクショントークンチェックについて）を参照すること。

内部設計では「トランザクションの開始タイミング」と「トークンチェック実施タイミング」を設計し、イベントURLマッピング一覧に記述する。基本的ポリシーを以下に示す。設計書への具体的な記載方法については後述の記述要領を参照のこと。

* トランザクションの開始タイミング

「入力画面」から「確認画面」に遷移する画面遷移のイベント

* トークンチェック実施タイミング

「確認画面」から「処理結果画面」に遷移する画面遷移のイベント

## 国際化対応

国際化対応とは、ユーザメッセージやGUIコンポーネントのラベル名、テキスト要素等をユーザのlocaleに応じた言語で表示することである。

本システムでは、利用者が確定しており、特定のlocaleでのみ使用することが決まっているため、開発コストを考慮し、国際化の対応は実施しないこととする。

## コードリスト

コードリストとは、アプリケーション内で変更される頻度が極めて少ない「名前＝値」の集まり、もしくは画面のセレクトボックスなどで利用される「名前＝値」の集まりのことである。共通ライブラリを利用することで、コードリストを容易に扱うことが出来る。コードリストの代表的な抽出元は外部設計書「コード値一覧」である。コードリストとして扱うべきものを選択し、実装すること。

本システムではコードリストとしてに示すデータを扱うこととする。なお、その他のデータについてもコードリストとして扱う要望が出た場合、その都度、コードリストへの適用可否を検討すること。

表 20 コードリスト一覧

| **コードリスト** | **ID** | **キー** | **値** |
| --- | --- | --- | --- |
| 空港 | CL\_AIRPORT | airport\_cd | airport\_name |
| 搭乗クラス | CL\_BOARDINGCLASS | boarding\_class\_cd | boarding\_class\_name |
| 運賃種別 | CL\_FARETYPE | fare\_type\_cd | fare\_type\_name |
| 搭乗月 | CL\_DEPMONTH | M | M月 |
| 搭乗日 | CL\_DEPDAY | D | D日 |
| 出発時刻 | CL\_DEPTIME | hhmm | hh:mm |
| 誕生年 | CL\_BIRTHYEAR | YYYY | YYYY |
| 誕生月 | CL\_BIRTHMONTH | MM | MM |
| 誕生日 | CL\_BIRTHDAY | DD | DD |
| クレジットカード有効期限年 | CL\_CREDITYEAR | YY | YY |
| クレジットカード有効期限月 | CL\_CREDITMONTH | MM | MM |
| 性別 | CL\_GENDER | M／F | 男性／女性 |
| クレジットカードタイプ | CL\_CREDITTYPE | credit\_type\_cd | credit\_firm |

## 画面生成・表示制御

### 画面レイアウト

画面レイアウトの定義および生成については、テンプレートエンジンであるApache Tilesを利用することとする。レイアウトは全て画面に適用するため、業務開発者がレイアウトを個別に設計する必要はない。

### 画面コンテンツ

テンプレートにて読み込まれる画面コンテンツはJSPとして実装する。なお、JSPを実装する際は「 」および「 」についても確認すること。

### CSS

画面のスタイル（色彩、サイズ等）にはカスケードスタイルシート（CSS）を利用する。そのため、画面のスタイルに関する設定をJSPに記述することは禁止とする。

### JavaScript

JavaScriptはユーザの操作性向上のために利用することとし、入力チェック等のロジックをJavaScriptで実現することは禁止とする。可読性・メンテナンス性の観点から、JSPにJavaScriptの処理を直接記述することは禁止とし、必ずJSファイルとしてJSPから切り出すこと。

## システム時刻

アプリケーションにてシステム時刻を取得する場合、試験における実時刻の影響を抑えるため、フレームワークで提供する以下のインターフェースを利用すること。

* DateFactoryインターフェース

インターフェースの利用方法についてはガイドライン（5.20 システム時刻）を参照すること。

インターフェースの設定を行うBean定義は、試験及び環境に応じて切り替え可能とするため、

${env}/src/main/resources/META-INF/spring/atrs-env.xmlに定義すること。

## 日付操作

日付データの操作を行う場合、Java標準のDate型ではなく、日付操作処理の豊富なjoda timeを利用すること。

joda timeの利用方法についてはガイドライン（5.21.2 日付操作（Joda Time））を参照すること。

## 認証・認可

### 基本方針

認証及び認可処理はSpring Securityを利用することとする。

### 認証処理

ユーザ認証はSpring Securityを利用するため、ログイン機能及びログアウト機能のControllerの実装は基本的には不要である。

「 」にてデフォルトの情報以外にユーザ情報を保持することになった場合、UserDetails及びUserDetailsServiceを拡張した独自クラスを実装する必要がある。認証処理の詳細についてはガイドライン（6.3 認証）を参照すること。

### 認可処理

認可処理はSpring Securityの認証処理にてログインしたユーザのリクエストに対して、ユーザの権限（ログイン有無）に応じたアクセス認可を行うものである。

認可処理では以下の３つの機能を提供する。各機能の詳細についてはガイドライン（6.5 認可）を参照すること。

* アクセス認可（Web）

クライアントからのリクエストに対して、Spring Security設定ファイルに設定したURLパターンをチェックし、マッチした場合にアクセス認可を行う機能である。

* JSPにおける認可処理

Spring Securityが提供しているカスタムタグ<sec:authorize>を利用することで、画面表示項目を制御することができる。

カスタムタグによる認可処理は、あくまで表示の制御でしかない。特定の権限でリンクが表示されなくても、リンクを改ざんすることで直接リンク先のURLにアクセスすることが可能である。そのため、必ず「アクセス認可（Web）」もしくは「アクセス認可（メソッド）」と併用し、リクエスト改ざんに対応すること。

* アクセス認可（メソッド）

Springのコンテナで管理されているBeanを対象に、メソッドに対してロールベースでのアクセス認可を行う機能である。基本的に、アクセス認可（メソッド）はドメイン層のメソッドに対して付与することとする。

### Spring Securityでは実現できないこと

Spring SecurityではWeb（URLパターン）やメソッドにおける認可（利用制限）は可能であるが、データそのものに対するアクセス制限を行うことはできない。

例えば、「/item?id=A001」へのアクセスは許可するが、「/item?id=B002」へのアクセスは防ぐといった内容である。つまり、認可情報としては「/item?id」へのアクセス許可を保持しているが、データに対するアクセス制限を実現したいという要件の場合である。このような処理についてはデータアクセス専用の共通機能を設けるか、ビジネスロジックにて個別に対応する必要がある。

## セキュリティ対策

### DBアクセス

本システムではO/R MapperのMyBatis3を利用してDBアクセスを行う。MyBatis3のDBアクセスについてセキュリティ上考慮すべきポイントを以下に示す。

* バインド変数

MyBatis3のSQL MAPPERでは#によるバインドを原則とする。

$によるバインドも可能であるが、$ではMyBatis3によるSQLエスケープが実行されないため、特別な要件が無い限り利用を禁止する。

特別な要件がある場合は、Mapなどでフィルタリングすることにより、動的に埋め込まれる値が安全であることを保障すること。

### パスワードハッシュ

本システムではユーザのログインパスワードは平文ではなく、ハッシュ値をDBに格納することとする。なお、ハッシュ値に関する処理は独自に実装するのではなく、Spring Frameworkが有する機能を利用することとする。詳細についてはガイドライン（6.4 パスワードハッシュ化）を参照すること。

### XSS対策

クロスサイトスクリプティング（XSS）とは、アプリケーションのセキュリティ上の不備を意図的に利用し、悪意のあるスクリプトを混入させることである。

ユーザの入力をそのまま出力している場合、XSSの脆弱性にさらされることになる。従って、HTMLのマークアップ言語で、特定の意味を持つ文字をエスケープすることがXSSの脆弱性に対する対抗措置となる。エスケープの種類とその対策を以下に示す。詳細についてはガイドライン（6.6 XSS対策）を参照すること。

* Output Escaping

HTML特殊文字（&、<、>、”、’）をエスケープする。

＜対策＞

文字列として出力する全ての表示項目をf:h()を利用して出力する。

* JavaScript Escaping

JavaScript特殊文字（"、’、\、/、<、>、0x0D（復帰）、0x0A（改行））をエスケープする。外部からの入力をもとにJavaScriptの文字列リテラルを動的に生成する場合にエスケープが必要となる。

＜対策＞

JavaScriptの文字列リテラルを動的に生成する場合はf:js()を利用して出力する。

* Event handler Escaping

JavaScriptのイベントハンドラの値をエスケープする。

＜対策＞

JavaScriptのイベントハンドラの値をエスケープする場合はf:hjs()を利用して出力すること。

### CSRF対策

クロスサイトリクエストフォージェリ（CSRF）とは、Webサイトにスクリプトや自動転送（HTTPリダイレクト）を不正に組み込むことにより、ユーザがログイン済みの別のWebサイトで意図しない何らかの操作を行わせる攻撃手法とのことである。

本システムではCSRF対策として秘密情報（トークン）を利用することとし、トークンに関する処理はSpring Frameworkが提供するライブラリを利用することとする。

CSRF対策の詳細についてはガイドライン（6.7 CSRF対策）を参照すること。

### 外部コマンドの実行

本システムでは外部コマンドを実行する要件がないため省略。

## DBアクセス

### DBアクセス

本システムではO/R MapperのMyBatis3を利用してDBアクセスを行う。そのため、必要となる実装はRepositoryインターフェースとSQL MAPPERファイルの二種類である。

### Like検索時のエスケープ

Like検索を行う場合、検索条件として使用する値をLike検索用にエスケープする必要がある。エスケープ処理については、独自に実装するのではなく、共通ライブラリが提供するQueryEscapeUtilsクラスを利用することとする。

### トランザクションについて

トランザクションについては「 」を参照すること。

### セキュリティ対策について

DBアクセスにおけるセキュリティ対策については「 」を参照すること。

## 定数

* 定数として定義する値が、アプリケーション内で変更される頻度が極めて少ないこと。
* 画面の遷移先を定数として定義しないこと。

画面の遷移先（Controllerの@RequestMappingのメソッドの戻り値）は、Controllerでのみ利用される値である。フレームワークではController自体が設定ファイルとの考えから、設定ファイルで定数を利用するという考えは適さない。そのため、画面の遷移先は定数として定義すべきではないとの認識である。

* 分類可能な定数については、enumとして定義すること。ロジックの判定（条件分岐）で利用する定数については特に注意すること。

## 文字コード

本システムで利用する文字コードについてに示す。個別のファイルにおける文字コードについては別途定義する。

表 21 文字コード一覧

| **対象** | **文字コード** | **備考** |
| --- | --- | --- |
| サーバOSの文字コード | UTF-8 |  |
| 画面（JSP、HTML等）の文字コード | UTF-8 | charset、pageEncodingの指定 |
| DBの文字コード | UTF-8 |  |
| ソースコードの文字コード | UTF-8 |  |
| JVMのデフォルトエンコーディング | UTF-8 | デフォルトエンコーディング  JVM起動オプションの-Dfile.encoding で指定 |

# 設計書の記述要領

## アクセスパターン一覧

### 記述単位

「アクセスパターン一覧」はシステムで1部作成する。アクセスパターンは主体となるエンティティ単位でExcelシートを作成する。

### 記述項目

記述項目をに示す。

表 22 「アクセスパターン一覧」の記述項目一覧

| ****記述項目**** | | ****記述説明**** |
| --- | --- | --- |
| No. | | 連番を記述する。 |
| ユースケースグループ | | ユースケースグループを記述する。 |
| ユースケース | | ユースケースを記述する。 |
| イベント | | イベントを記述する。 |
| アクセスパターン | | アクセスパターンを記述する。 |
| 取得件数 | | 取得件数を記述する。  　１件：１件のデータを取得する場合  　リスト：複数件のデータを取得する場合  　ページング：ページングにおける検索の場合  　件数：count関数で件数を取得する場合 |
| 絞り込み条件 | | |
|  | 左欄 | 検索条件を記述する。  　主キー：主キーで検索する場合  　任意（静）：任意のカラムで検索かつ、WHERE句が静的（固定）の場合  例）ステータスがXXXのものを検索  　任意（動）：任意のカラムで検察かつ、WHERE句が動的に変わる場合  例）入力された検索条件で検索 |
|  | 右欄 | 検索条件の具体的な項目名を記述する。 |
| その他条件 | | ソート条件やページ検索の条件を記述する。 |
| テーブル | | テーブルにおけるCRUDを記述する。 |

## リポジトリ一覧

### 記述単位

「リポジトリ一覧」はシステムで1部作成する。

### 記述項目

記述項目をに示す。

表 23 「リポジトリ一覧」の記述項目一覧

| ****記述項目**** | ****記述説明**** |
| --- | --- |
| 項番 | 連番を記述する。 |
| リポジトリパッケージ | リポジトリクラスのJavaパッケージを記述する。 |
| リポジトリクラス名 | リポジトリクラスの名称を記述する。 |
| エンティティ | エンティティ名を記述する。 |
| テーブル | テーブル名を記述する。 |

### アクセスパターン一覧とリポジトリの関係

「アクセスパターン一覧」とリポジトリの関係をに示す。



* + 関連エンティティとは主体となるエンティティと一対多（多は一を含む）の関係になる多に該当するエンティティである

図 9　「アクセスパターン一覧」とリポジトリの関係

### リポジトリの記述単位

「アクセスパターン一覧」を参考に以下のルールに則り、リポジトリを抽出・記述する。

* 基本的にリポジトリはエンティティと一対一とする。
* 「アクセスパターン一覧」の一シート内（エンティティ内）のテーブルにて、関連エンティティを直接操作するアクセスパターンがある場合、主体となるエンティティと関連エンティティのリポジトリを分けることを検討する。
* マスタメンテナンスではその操作対象がテーブル単位であるため、エンティティがテーブルと一対一になり、結果としてリポジトリがテーブル単位となるが、リポジトリの対象はあくまでエンティティである。

## クラス仕様書（リポジトリ）

### 記述単位

「クラス仕様書（リポジトリ）」はリポジトリ単位でファイルを作成する。

1つの「クラス仕様書（リポジトリ）」は、「クラス概要」シートと「メソッド概要」シートで構成される。

「クラス概要」シートは1ファイルで1つ作成する。対して「メソッド概要」シートは処理（メソッド）単位でExcelシートを作成する。「メソッド概要」シートのシート名には、メソッド名を記述すること。

### 記述項目

記述項目を及び、に示す。

表 24 「クラス仕様書（リポジトリ）　クラス概要シート」の記述項目一覧

| ****記述項目**** | | ****記述説明**** |
| --- | --- | --- |
| クラス概要 | | クラスの概要を記述する。 |
| パッケージ名 | | Javaパッケージを記述する。 |
| クラス名 | | クラスの名称を記述する。 |
| クラス修飾子 | | クラスの修飾子を記述する。 |
| extends | | 継承する親クラスを記述する。 |
| implements | | 実装するインターフェースを記述する。 |
| メソッド | | |
|  | 項番 | 通番を記述する。 |
|  | API概要 | メソッドの概要を記述する。 |
|  | メソッド名 | メソッドの名称を記述する。 |
|  | アクセステーブル | リポジトリがアクセスするテーブル名を記述する。  複数のテーブルにアクセスする場合、一行一テーブルとなるように適宜行を追加する。 |
|  | 引数 | メソッドの引数を記述する。 |
|  | 戻り値 | メソッドの戻り値を記述する。 |

表 25 「クラス仕様書（リポジトリ）　メソッド概要シート」の記述項目一覧

| ****記述項目**** | | ****記述説明**** |
| --- | --- | --- |
| API概要 | | メソッドの概要を記述する。 |
| メソッド名 | | メソッドの名称を記述する。 |
| メソッド修飾子 | | メソッドの修飾子を記述する。 |
| 戻り値 | | 戻り値の型を記述する。 |
| 説明 | | 戻り値の説明を記述する。 |
| メソッド（引数が複数ある場合、行を追加する） | | |
|  | 引数名 | 引数の名称を記述する。 |
|  | 型 | 引数の型を記述する。 |
|  | 説明 | 引数の説明を記述する。 |
| 詳細 | | テーブルアクセスの詳細を記述する。  フォーマットは特に規定しないが以下の内容を含めること。  ・アクセスするテーブル  ・検索・更新の際の絞り込み条件  ・SQL関数を利用する場合はその内容  ・テーブルを結合する場合はその内容 |

### アクセスパターン一覧とリポジトリメソッドの関係

「アクセスパターン一覧」とリポジトリのメソッドの関係をに示す。



図 10　「アクセスパターン一覧」とリポジトリのメソッドの関係

### リポジトリのメソッドの抽出方法

原則としてリポジトリのメソッドは以下のルールに則り、設計・記述することとする。

* 原則としてエンティティに関するCRUDのメソッドは主体となるエンティティのリポジトリに持たせることとする。
* 関連エンティティのみを直接操作する場合、関連エンティティのリポジトリが切り出されるため、その関連エンティティのリポジトリに当該メソッドを持たせることとする。

概要をに示す。

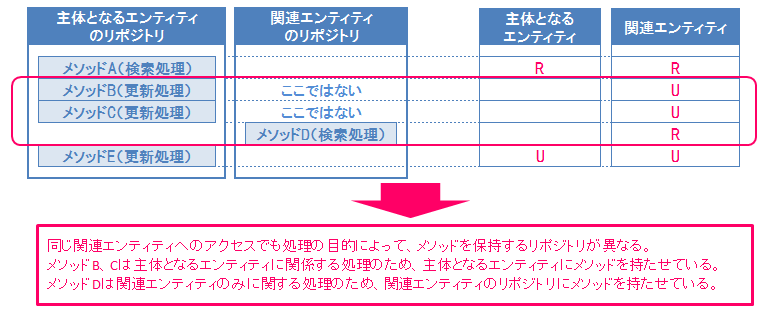


図 11　関連エンティティを操作するメソッド

* 性能的な制限や特別な要件がない限り、Rのアクセスパターンではテーブルの全カラムを取得することとする。これにより、取得するカラムの差異を吸収することが可能となる。

概要をに示す。

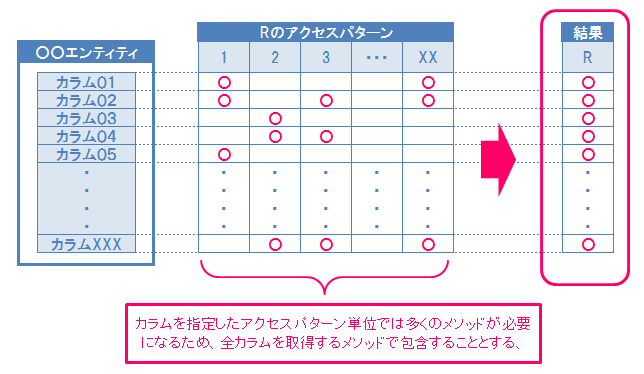


図 12　全カラムの取得

* LOB型のカラムについては、特殊な操作が必要なことや性能的な観点を考慮し、全カラム取得の対象から除外することとする。つまり、LOB型のカラムがある場合は、LOB型のカラムを取得するメソッドとそうでないメソッドの二種類を用意することとなる。
* Rの取得件数（１件、リスト、ページング、件数）に応じて、それぞれ別のメソッドを用意することとする。
* 主キーにおける検索、任意（静）、任意（動）における検索は、それぞれ別のメソッドを用意することとする。
* Uのアクセスパターンでは想定していないカラムの更新を防ぐため、カラムを指定した個別の更新処理とし、それぞれ別のメソッドとして定義することとする。
* Dのアクセスパターンでは想定していない条件での削除を防ぐため、カラムを指定して個別の削除処理とし、それぞれ別のメソッドとして定義することとする。

## イベントURLマッピング一覧

### 記述単位

「イベントURLマッピング一覧」はシステムで1部作成する。

### 記述項目

記述項目をに示す。

表 26 「イベントURLマッピング一覧」の記述項目一覧

| ****記述項目**** | ****記述説明**** |
| --- | --- |
| No. | 連番を記述する。 |
| ユースケースグループID | 「外部設計 イベント一覧」の該当項目を参照。 |
| ユースケースグループ名 | 「外部設計 イベント一覧」の該当項目を参照。 |
| ユースケースID | 「外部設計 イベント一覧」の該当項目を参照。 |
| ユースケース名 | 「外部設計 イベント一覧」の該当項目を参照。 |
| 画面名 | 画面名を記述する。 |
| JSP名 | JSPファイル名を記述する。 |
| イベントID | 「外部設計 イベント一覧」の該当項目を参照。 |
| イベント名 | 「外部設計 イベント一覧」の該当項目を参照。 |
| URL | リクエストを受け付けるURLのパスを記述する。  ※コンテキスト名は不要 |
| HTTPメソッド | リクエストを受け付けるHTTPメソッドを記述する。  　GET：GETメソッドによるリクエストを許可  　POST：POSTメソッドによるリクエストを許可  　－：全てのリクエストを許可  ※GET、POST以外のHTTPメソッドは原則として利用を禁止とする。 |
| HTTPパラメータ | リクエストを受け付けるHTTPパラメータを記述する。  ※必要に応じてパラメータの値まで記述 |
| Controllerクラス | Controllerメソッドの名称を記述する。  ※Javaパッケージ名は不要 |
| Controllerメソッド | Controllerメソッドの名称を記述する。 |
| トランザクショントークンチェック | トランザクショントークンチェックの指定を記述する。  BEGIN：トークンチェックの開始  IN：トークンをチェックする  －：トークンをチェックしない |
| 備考 | 特記すべき備考があれば記述する。  画面遷移に関係しないHTTPパラメータがある場合、その用途について概要を記述する。 |

### URLの記述方法

URLは以下のルールに則り、設計・記述することとする。

* 「名称付与１」は「ユースケースグループ」とする。
* 「名称付与２」は「ユースケース」とする。
* 「 」に従い、URLの値を決定する。
* 遷移先の画面が別の「ユースケース」の場合、「正常系遷移」と「ユースケース外遷移」の２つのURLを抽出すること。

例をに示す。

例）

①と②、③と④、⑤と⑥、⑦と⑧がそれぞれ１つの画面遷移であるが、「正常系遷移」と「ユースケース外遷移」の２つのURLが抽出される。



図 13　遷移先の画面が別のユースケースの場合

### HTTPパラメータの記述方法

HTTPパラメータは以下のルールに則り、設計・記述することとする。

* 「 」に従い、HTTPパラメータの値を決定する。
* イベントの内容を確認し、内容が実質的に等しいものは同じパラメータとする。

例をに示す。

例）

②、④、⑥、⑧の「ユースケース外遷移」は実質的に等しいため、等しいパラメータ②とする。



図 14　イベントが実質的に等しい場合

### トランザクショントークンチェックの記述方法

原則としてトランザクショントークンチェックは以下のルールに則り、設計・記述することとする。

* 「入力画面」から「確認画面」に遷移する画面遷移のイベントは「BEGIN」とする。
* 「確認画面」から「処理結果画面」に遷移する画面遷移のイベントは「IN」とする。
* 「 」を参照し、必要に応じて「トランザクショントークンチェック」を指定する。
* 上記のルールに従って一律にトランザクショントークンチェックを設定するのではなく、ユーザビリティを考慮して設定すること。

### Spring Securityを利用する際の記述方法

ユーザ認証をSpring Securityで実現する場合、URL等はSpring Securityの仕様に準拠することとなる。

「イベントURLマッピング一覧」では「備考」にSpring Securityを利用する旨を記述する。

## イベントフロー図

### 記述単位

「イベントフロー図」はユースケース単位でファイルを作成する。

1つの「イベントフロー図」は、「概要」シートと「イベント」シートで構成される。

「概要」シートは1ファイルで1つ作成する。対して「イベント」シートは、ユースケースには複数のイベントが含まれるため、イベント単位でExcelシートを作成する。「イベント」シートのシート名には、イベントIDを記述すること。

### 記述項目

記述項目を、に示す。

表 27 「イベントフロー図　概要シート」の記述項目一覧

| ****記述項目**** | ****記述説明**** |
| --- | --- |
| オブジェクト | セッションに格納するオブジェクトを記述する。 |
| ライフサイクル | オブジェクトのライフサイクルを記述する。 |
| 概要 | 概要を記述する。 |

表 28 「イベントフロー図　イベントシート」の記述項目一覧

| ****記述項目**** | ****記述説明**** |
| --- | --- |
| イベントID | イベントIDを記述する。 |
| イベント名 | イベント名を記述する。 |
| セッション情報事前 | イベント開始時にセッションに必要な情報がある場合に記述する。  例）  List<B2ReserveFlight>：選択フライト情報 |
| 画面 | 利用する画面を記述する。 |
| アプリケーション層 | アプリケーション層で利用するオブジェクトを記述する。  ・Controllerクラス  ・Formクラス  ・Validatorクラス  ・Helperクラス・・・等 |
| ドメイン層 | ドメイン層で利用するオブジェクトを記述する。  ・Serviceクラス  ・SharedServiceクラス  ・Utilクラス・・・等 |
| インフラストラクチャ層 | インフラストラクチャ層で利用するオブジェクトを記述する。  ・Repositoryクラス・・・等 |
| セッション情報事後 | イベント終了時にセッションに格納する情報がある場合に記述する。 |

## クラス仕様書（サービス）

### 記述単位

「クラス仕様書（サービス）」はService、SharedServiceクラス単位でファイルを作成する。

1つの「クラス仕様書（サービス）」は、「クラス概要」シートと「メソッド概要」シートで構成される。

「クラス概要」シートは1ファイルで1つ作成する。対して「メソッド概要」シートは、メソッド単位でExcelシートを作成する。「メソッド概要」シートのシート名には、メソッド名を記述すること。

### 記述項目

記述項目を、に示す。

表 29 「クラス仕様書（サービス）　クラス概要シート」の記述項目一覧

| ****記述項目**** | | ****記述説明**** |
| --- | --- | --- |
| クラス概要 | | クラスの概要を記述する。 |
| パッケージ名 | | Javaパッケージを記述する。 |
| クラス名 | | クラスの名称を記述する。 |
| クラス修飾子 | | クラスの修飾子を記述する。 |
| extends | | 継承する親クラスを記述する。 |
| implements | | 実装するインターフェースを記述する。 |
| フィールド情報 | | |
|  | フィールド名 | フィールドの名称を記述する。 |
|  | 型 | フィールドの型を記述する。 |
|  | 修飾子 | フィールドの修飾子を記述する。 |
|  | 初期値 | フィールドの初期値を記述する。  インジェクションにて値を設定する場合は「インジェクション」と記述する。 |
|  | 説明 | フィールドの説明を記述する。 |

表 30 「クラス仕様書（サービス）　メソッド概要シート」の記述項目一覧

| ****記述項目**** | | ****記述説明**** |
| --- | --- | --- |
| API概要 | | メソッドの概要を記述する。 |
| メソッド名 | | メソッドの名称を記述する。 |
| メソッド修飾子 | | メソッドの修飾子を記述する。 |
| 戻り値 | | 戻り値の型を記述する。 |
| 説明 | | 戻り値の説明を記述する。 |
| 引数（引数が複数ある場合、行を追加する） | | |
|  | 引数名 | 引数の名称を記述する。 |
|  | 型 | 引数の型を記述する。 |
|  | 説明 | 引数の説明を記述する。 |
| 例外（スローする例外が複数ある場合、行を追加する） | | |
|  | 例外 | 例外を記述する。 |
|  | 説明 | 例外がスローされるケースを記述する。 |
| トランザクション情報 | | |
|  | 属性 | トランザクション情報として明示的に指定する必要があるものについて記述する。属性の種類としては  伝搬性： デフォルト値=REQUIRED  読み取り専用： デフォルト値= false  などがある。 |
|  | 設定値 | 各属性の設定値を記述する。 |
| 処理 | | |
|  | 入力 | 引数の状態を記述する。 |
|  | 処理 | 処理フローを記述する。 |
|  | 出力 | 戻り値の状態を記述する。  例外をスローする場合、例外の状態を記述する。 |

## クラス図

### 記述単位

エンティティの「クラス図」をシステムで1部作成する。

### 記述フォーマット

UMLのクラス図で作成し、Excel形式の設計書に貼付する。

### クラスのプロパティが多い場合

クラス図の目的はエンティティ間の関連を図示化することである。クラスのプロパティが非常に多く、全てのプロパティを表示させると1シートに収まりきらない場合、関連（依存）線が表示された図を1シート目に貼付することとする。

各エンティティのクラス図については、2シート目以降にプロパティを表示した形式で貼付することとする。

# コーディングガイド

## Entityクラスの実装

### Entityクラスの作成

「クラス図（エンティティ）」をもとにEntityクラスを作成する。Entityクラスは「java.io.Serializable」を実装したPOJO（Plain Old Java Object）のクラスとする。 配置及びパッケージについては「 」を参照すること。

Entityクラスの記述例を以下に示す。

|  |
| --- |
| **package** jp.co.ntt.atrs.domain.model;  **import** java.io.Serializable;  **public** **class Member** **implements** Serializable {  } |

### 属性の追加

「クラス図（エンティティ）」をもとにEntityクラスに属性を追加する。EntityクラスはPOJOであるため基本的にはJavaの一般的な作法に従うだけであるが、一対多の関連については配列やコレクション型等、実装方法が多数存在する。実装方法を統一するため、一対多の関連はjava.util.Listの実装に揃えることとする。

### 属性の型について

属性の型として基本的にint、double等のプリミティブ型の利用は禁止し、そのラッパークラス（java.lang.Integer、java.lang.Double等）を利用する方針とする。プリミティブ型はそれぞれデフォルト値を持つため、リクエストパラメータ等が送信された場合とそうでない場合を判断できないためである。ラッパークラスを利用することで送信されていない場合はNullとして明確に判断することが可能となる。

## Repositoryクラスの実装

### Repositoryインターフェースの作成

「クラス仕様書（リポジトリ）」の「クラス概要」シートをもとにRepositoryインターフェースを作成する。本システムではO/R MapperとしてMyBatis3を利用するため、基本的に実装クラスは不要である。 配置及びパッケージについては「 」を参照すること。

### インターフェースメソッドの作成

「クラス仕様書（リポジトリ）」の「メソッド概要」シートの「メソッド名」、「メソッド修飾子」、「戻り値」、「引数名」、「型」を参照し、インターフェースのメソッドを作成する。Repositoryインターフェースの記述例を以下に示す。

|  |
| --- |
| **package** jp.co.ntt.atrs.domain.repository.member;  **import** jp.co.ntt.atrs.domain.model.Member;  **public** **interface** MemberRepository {    Member findOneForLogin(String customerNo);  **int** updateToLoginStatus(Member member);  // 省略  } |

### SQL MAPPERファイルの作成

SQL MAPPERファイルはリポジトリ単位で作成する。SQL MAPPERファイルのnamespace属性は重要な値であり必ず設定すること。設定値については「 」を参照すること。

### SQL MAPPERファイルへのSQLステートメントの追加

「クラス仕様書（リポジトリ）」の「メソッド概要」シートの「戻り値」、「引数名」、「型」、「詳細」を参照し、メソッドに対応するSQLステートメントを追加する。SQL\_IDについては「 」を参照すること。

SQL MAPPERの戻り値の記述方法として、resultTypeを利用する方法とresultMapを利用する方法があるが、本システムではresultMapを利用する方法で統一することとする。ただし、java.lang.String型等の１カラムのみ返す場合はresultTypeを利用しても構わない。

SQL MAPPERファイルを記述する際はバインド変数の扱いに注意すること。詳細については「 」を参照すること。

SQL MAPPERファイルの記述例を以下に示す。

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?><!--Converted at: Wed Sep 18 14:11:03 JST 2013-->  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace=*"jp.co.ntt.atrs.domain.repository.member.MemberRepository"*>  <resultMap id=*"findOneForLogin-member-map"* type=*"Member"*>  <id property=*"customerNo"* column=*"customer\_no"*/>  <result property=*"kanjiFamilyName"* column=*"kanji\_family\_name"*/>  <result property=*"kanjiGivenName"* column=*"kanji\_given\_name"*/>  <association property=*"memberLogin"* resultMap=*"memberLogin-map"*/>  </resultMap>    <resultMap id=*"member-map"* type=*"Member"*>  <id property=*"customerNo"* column=*"customer\_no"*/>  <result property=*"kanjiFamilyName"* column=*"kanji\_family\_name"*/>  <result property=*"kanjiGivenName"* column=*"kanji\_given\_name"*/>  <result property=*"kanaFamilyName"* column=*"kana\_family\_name"*/>  <result property=*"kanaGivenName"* column=*"kana\_given\_name"*/>  <result property=*"birthday"* column=*"birthday"*/>  <result property=*"gender"* column=*"gender"*/>  <result property=*"tel"* column=*"tel"*/>  <result property=*"zipCode"* column=*"zip\_code"*/>  <result property=*"address"* column=*"address"*/>  <result property=*"mail"* column=*"mail"*/>  <result property=*"creditNo"* column=*"credit\_no"*/>  <result property=*"creditTerm"* column=*"credit\_term"*/>  <association property=*"creditType"* resultMap=*"creditType-map"* />  <association property=*"memberLogin"* resultMap=*"memberLogin-map"*/>  </resultMap>  <resultMap id=*"creditType-map"* type=*"CreditType"*>  <id property=*"creditTypeCd"* column=*"credit\_type\_cd"*/>  <result property=*"creditFirm"* column=*"credit\_firm"*/>  </resultMap>    <resultMap id=*"memberLogin-map"* type=*"MemberLogin"*>  <result property=*"password"* column=*"password"*/>  <result property=*"lastPassword"* column=*"last\_password"*/>  <result property=*"loginDateTime"* column=*"login\_date\_time"*/>  <result property=*"loginFlg"* column=*"login\_flg"*/>  </resultMap>  <!-- ログイン時の会員情報取得 -->  <select id=*"findOneForLogin"* parameterType=*"String"* resultMap=*"findOneForLogin-member-map"*>  SELECT  m.customer\_no,  m.kanji\_family\_name,  m.kanji\_given\_name,  m\_l.password,  m\_l.last\_password,  m\_l.login\_date\_time,  m\_l.login\_flg  FROM  member m  JOIN member\_login m\_l ON m.customer\_no = m\_l.customer\_no  WHERE  m.customer\_no = #{customerNo}  </select>  <!-- 省略 -->  </mapper> |

## DTOクラスの実装

### DTOクラスの作成

DTOクラスは「java.io.Serializable」を実装したPOJO（Plain Old Java Object）のクラスとする。 「クラス仕様書（サービス）」の「メソッド概要」シートの「戻り値」、「説明」、「引数名」、「型」、「入力」、「出力」を参照し、必要な属性を追加する。配置及びパッケージについては「 」を参照すること。

## Serviceクラスの実装

### Serviceインターフェースの作成

「クラス仕様書（サービス）」の「クラス概要」シートの「implements」で指定されているServiceインターフェースを作成する。配置及びパッケージについては「 」を参照すること。

### インターフェースメソッドの作成

「クラス仕様書（サービス）」の「メソッド概要」シートを参照し、「メソッド修飾子」が「public」のメソッドをServiceインターフェースに作成する。

Serviceインターフェースの記述例を以下に示す。

|  |
| --- |
| **package** jp.co.ntt.atrs.domain.service.c1;  **import** jp.co.ntt.atrs.domain.model.Member;  **public** **interface** MemberRegisterService {  Member register(Member member);  } |

### Serviceクラスの作成

「クラス仕様書（サービス）」の「クラス概要」シートを参照し、Serviceクラスを作成する。基本的には「 」で作成したServiceインターフェースを実装するクラスとなる。

Serviceのクラスとして認識させるため、クラスアノテーションとして「@Service」を付与する。また、トランザクションが発生するメソッドに対して、メソッドアノテーションとして「@Transactional」を付与する。

### クラスメソッドの作成

「クラス仕様書（サービス）」の「メソッド概要」シートを参照し、Serviceクラスのメソッドを作成する。

Serviceクラスの記述例を以下に示す。

|  |
| --- |
| **package** jp.co.ntt.atrs.domain.service.c1;  // 省略  **import** javax.inject.Inject;  @Service  **public** **class** MemberRegisterServiceImpl **implements** MemberRegisterService{  @Inject  MemberRepository memberRepository;  @Inject  PasswordEncoder passwordEncoder;  **public** Member register(Member member) {  Assert.*notNull*(member);  MemberLogin memberLogin = member.getMemberLogin();  Assert.*notNull*(memberLogin);  // パスワードをエンコードする。  String hashedPassword = passwordEncoder.encode(member.getMemberLogin()  .getPassword());  memberLogin.setPassword(hashedPassword);  memberLogin.setLastPassword(hashedPassword);  memberLogin.setLoginFlg(**false**);  // 入力されたユーザ情報を基に、データベースへの登録を行う。  // MyBatis3の機能(SelectKey)によりmemberにはcustomerNoが格納される。  **int** insertMemberCount = memberRepository.insert(member);  **if** (insertMemberCount != 1) {  **throw** **new** SystemException(LogMessages.***E\_AR\_A0\_L9002***.getCode(),  LogMessages.***E\_AR\_A0\_L9002***.getMessage(insertMemberCount, 1));  }  **int** insertMemberLoginCount = memberRepository.insertMemberLogin(member);  **if** (insertMemberLoginCount != 1) {  **throw** **new** SystemException(LogMessages.***E\_AR\_A0\_L9002***.getCode(),  LogMessages.***E\_AR\_A0\_L9002***.getMessage(insertMemberLoginCount, 1));  }  **return** member;  }  } |

## Formクラスの実装

### Formクラスの作成

「 」をもとにFormクラスを作成する。Formクラスは「java.io.Serializable」を実装したPOJO（Plain Old Java Object）のクラスとする。 配置及びパッケージについては「 」を参照すること。

Formクラスの記述例を以下に示す。

|  |
| --- |
| **package** jp.co.ntt.atrs.app.b2;  **import** java.io.Serializable;  **public** **class** ReservationForm **implements** Serializable {  } |

### 属性の追加

#### 属性の対象

Formクラスの属性の対象となるのは、画面仕様の「論理項目種別」が「テキストボックス」、「パスワードボックス」、「テキストエリア」、「ラジオボタン」、「リストボックス」、「チェックボックス」、「ファイルフィールド」及び設計として必要と思われる「隠しフィールド」の項目である。なお、「隠しフィールド」は画面仕様には存在しないため、画面上のデータの持ち回り方を考慮し、Formクラスを実装する際に組み込むこと。

Controllerの単位がユースケースの単位であるため、このユースケースで利用する全ての画面仕様が対象範囲となる。

例をに示す。



※画面仕様A,B,Cから対象となるtypeの属性を抽出する。

図 15　Formクラスの属性

#### 属性の名称及び型

Formクラスの属性の名称及び型については、業務ロジック（Service等）とのデータのやり取りを考慮し、DTOクラスの属性の名称及び型と整合性を取ることとする。

Controllerクラスの実装で説明するが、FormクラスとDTOクラスとの値の詰め替え処理を「org.dozer.Mapper」を用いて実施するため、名称及び型が一致している必要がある。

### 入力チェックの対応

画面仕様の「画面項目」シートの「入力」、「必須」、「入力桁数」、「文字種別」、「その他入力条件」等を参考に、入力チェックの内容に従いFormクラスの属性に該当するアノテーションを設定する。

なお、本設定と合わせて該当のControllerクラスに入力チェックの設定がされている必要がある。Controllerクラスの入力チェックについては「 」を参照すること。

入力チェックの記述例を以下に示す。

|  |
| --- |
| **public** **class** ReserveTicketForm **implements** Serializable {  /\*\*  \* 予約代表者姓  \*/  @NotNull  @Size(min = 1, max = 10)  @FullWidthKatakana  **private** String repFamilyName;  /\*\*  \* 予約代表者名  \*/  @NotNull  @Size(min = 1, max = 10)  @FullWidthKatakana  **private** String repGivenName;  // 省略  } |

### 入れ子構造（ネスト構造）のFormクラスについて

HTMLのフォームを入れ子構造にして子フォームを定義する場合、Formクラスの実装で注意すべき点を以下に示す。

* 入力チェック

入れ子構造のFormクラスの場合、子フォームの入力チェックを有効にするため、親フォームのFormクラスにて、子フォームのFormクラスを示す属性に「@Valid」アノテーションを付与する必要がある。

* f-query関数

JSPのf:query()関数（EL式）は、引数で指定されたオブジェクト（FormオブジェクトやMapオブジェクト）からリクエストパラメータ（プロパティ名やキー名がパラメータ名となる）を生成する機能であり、検索条件の保持等で利用する。

入れ子構造のFormクラスの場合、このf:query関数を利用することができないので注意すること。

### 属性の型変換について

Spring Frameworkでは、HTML formから送信されたリクエストパラメータ（文字列）を、Formオブジェクトに定義されている型に変換してからバインドする機能を提供している。そのため、Formオブジェクトに定義する属性の型はjava.lang.String型だけでなく、任意の型を定義することができる。

定義する際は型変換のためのアノテーション（@NumberFormat、@DateTimeFormat）を付与する必要がある。型変換の記述例を以下に示す。

|  |
| --- |
| **public** **class** SampleForm implements Serializable {  @NumberFormat (pattern = "#,#")  private Integer price = null;  @DateTimeFormat(pattern = "yyyyMMdd")  private Date birthDate = null;  // 省略  } |

## Controllerクラスの実装

### Controllerクラスの作成

「イベントURLマッピング一覧」の「Controllerクラス」の名称でControllerクラスを作成する。Controllerのクラスとして認識させるため、クラスのアノテーションとして「@Controller」を付与する。「イベントURLマッピング一覧」の「URL」とControllerクラスをマッピングさせるため、クラスのアノテーションとして「@RequestMapping」を追加する。アノテーションの値として「URL」の値を設定する。配置及びパッケージについては「 」を参照すること。

Controllerクラスの記述例を以下に示す。

|  |
| --- |
| **package** jp.co.ntt.atrs.app.c1;  **import** org.springframework.stereotype.Controller;  @Controller  @RequestMapping("Member/register")  **public** **class** MemberRegisterController {  } |

### Form初期化のメソッド

Formを初期化するためのメソッドを追加する。メソッドの名称は「setUpForm」とし、戻り値はFormクラスのオブジェクトである。メソッドのアノテーションとして「@ModelAttribute」を付与する。詳細についてはガイドライン（4.4.2.2 フォームオブジェクトの初期化方法）を参照すること。

Form初期化のメソッドの記述例を以下に示す。

|  |
| --- |
| @ModelAttribute  **public** MemberRegisterForm setUpForm() {  MemberRegisterForm memberRegisterForm = **new** MemberRegisterForm();  **return** memberRegisterForm;  } |

### リクエスト処理のメソッドの作成

「イベントURLマッピング一覧」の「Controllerメソッド」の名称で、リクエストを処理するためのメソッドを追加する。引数は処理によって異なるが、基本的に戻り値は全てのメソッドでString型である。なお、戻り値のStringは画面遷移先（JSP）となる。戻り値の詳細については「 」を参照すること。

リクエスト処理用のメソッドの記述例を以下に示す。

|  |
| --- |
| @Controller  @RequestMapping("Member/register")  **public** **class** MemberRegisterController {    **public** String registerForm() {  **return** "C1/memberRegisterForm";  }  } |

### リクエスト処理のマッピング

「イベントURLマッピング一覧」の「HTTPメソッド」、「HTTPパラメータ」の組み合わせとControllerのメソッドをマッピングさせるため、メソッドのアノテーションとして「@RequestMapping」を追加する。アノテーションの値として「HTTPメソッド」、「HTTPパラメータ」を設定する。

メソッドのマッピングの記述例を以下に示す。

|  |
| --- |
| @RequestMapping(method = RequestMethod.*GET*, params = "form")  **public** String registerForm() {  **return** "C1/memberRegisterForm";  } |

※「HTTP MethodがGET」かつ「HTTP Parameterがform」の場合

### 遷移先の設定

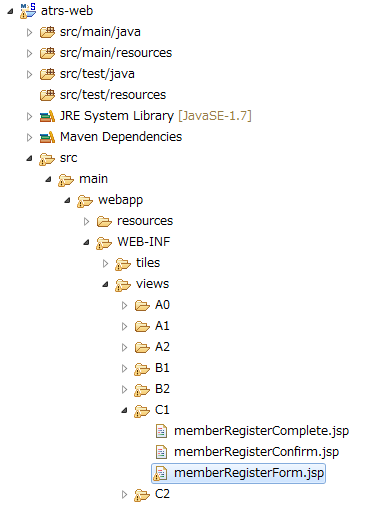
#### 通常の画面遷移

「イベントURLマッピング一覧」の「Controllerメソッド」で定義されたメソッド（リクエストを処理するメソッド）の戻り値により、画面の遷移先（表示するJSP）が決定する。

JSPの指定は ${web}/src/main/webapp/WEB-INF/views からの相対パスとなる。また、その際、拡張子の“.jsp”は記述しない。

通常の画面遷移の記述例を以下に示す。

このJSPに遷移させる



|  |
| --- |
| @RequestMapping(method = RequestMethod.*GET*, params = "form")  **public** String registerForm() {    **return** "C1/memberRegisterForm";  } |

* 参考情報（viewsからの相対パス及び“.jsp”を除く理由）

Spring-mvc.xmlのViewResolverの設定にて、プレフィックスを以下のように定義しているためである。以下の定義を行うとサフィックスは、デフォルトで“.jspとなる。

|  |
| --- |
| <mvc:view-resolvers>  <mvc:tiles />  <mvc:jsp prefix=*"/WEB-INF/views/"* />  </mvc:view-resolvers> |

* 遷移先のパスの定数化について

画面遷移が増えるに連れて、遷移先のパス（例では“C1/memberRegisterForm”）の定数化の議論が出てくると想定される。TERASOLUNA Server Framework for Java 5.x ではControllerクラス自体が設定ファイルも包含するとの考え方から、遷移先のパスの定数化は推奨していない。

#### リダイレクトによる画面遷移

PRG（Post-Redirect-GET）パターンやControllerを跨ぐ画面遷移を実施する場合、リダイレクトによる画面遷移を行う必要がある。リダイレクトによる画面遷移は“redirect:”プレフィックスの後に“/”＋遷移先の「URL」を指定する。

リダイレクトによる画面遷移の記述例を以下に示す。

|  |
| --- |
| @TransactionTokenCheck  @RequestMapping(method = RequestMethod.***POST***)  **public** String register(@Validated MemberRegisterForm memberRegisterForm,  BindingResult result, Model model, RedirectAttributes redirectAttributes) {  // 省略  // Post-Redirect-Get  **return** "redirect:/Member/register?complete";  } |

※「/コンテキスト名/Member/register?complete」にリダイレクトさせる場合

#### フォワードによる画面遷移

入力された情報（リクエストデータ）を引き継いだまま他の処理を実行したい場合等、リダイレクトではなくフォワードによる画面遷移を行う場合、”forward:”プレフィックスの後に“/”＋遷移先の「URL」を指定する。

フォワードによる画面遷移の記述例を以下に示す。

|  |
| --- |
| @RequestMapping(method = RequestMethod.*POST*, params = "forward")  **public** String forwardTest () {  // 省略  // Forward Transfer  **return** "forward:/Member/register";  } |

※「/コンテキスト名/Member/register」にフォワードさせる場合

### 入力チェックの対応

### 通常の入力チェック

画面仕様を参照し、入力チェックを実施するイベントの場合、対応するリクエストの処理メソッドに以下の定義及び実装を追加する。

ポイントはFormクラスの引数アノテーションとして「@Validated」を設定し、入力チェックの結果のオブジェクトであるBindingResultクラスをメソッドの引数に追加することである。また、BindingResultは、Formクラスの直後に定義する必要があるので注意すること。

なお、本設定と合わせて該当のFormクラスに入力チェックの設定がされている必要がある。Formクラスの入力チェックについては「 」を参照すること。

入力チェックの記述例を以下に示す。

|  |
| --- |
| @TransactionTokenCheck(type = TransactionTokenType.*BEGIN*)  @RequestMapping(method = RequestMethod.*POST*, params = "confirm")  public String registerConfirm(@Validated MemberRegisterForm memberRegisterForm,  BindingResult result, Model model) {  if (result.hasErrors()) {  return registerRedo(memberRegisterForm, model);  }  **return** "C1/memberRegisterConfirm";  // 省略  } |

※BindingResult#hasErrors()メソッドで入力チェックの結果（OK、NG）を判断する。

### グループを設定した入力チェック

画面仕様の「チェックタイミング」として複数のイベントが設定されている場合、バリデーションのグループ化を実施する必要がある。

グループ化のポイントを以下に示す。詳細についてはガイドライン（5.5.2.2.3 バリデーションのグループ化）を参照すること。

* Formクラスにグループのインターフェースを定義

チェックタイミング単位にインターフェースを定義する。仮に「チェックタイミング」が３つ定義されている場合、インターフェースを３つ定義する。

* バリデーションをグループ単位に設定

バリデーションの指定時に、グループを指定するのため、group属性に対象のグループインターフェースを指定する。

* Controllerの@Validatedに対象のグループを設定

@Validatedの指定時にFormクラスだけでなく、リクエストにて実施すべき入力チェックのグループを指定する。

### トランザクショントークンの設定

### トランザクショントークンのネームスペース

「イベントURLマッピング一覧」の「トランザクショントークンチェック」欄を「Controller クラス」単位で確認し、“－”以外の記載がある場合、トランザクショントークンのネームスペースの設定を行う。

Controllerクラスにクラスのアノテーションとして、「@TransactionTokenCheck」を付与する。このアノテーションの値がネームスペースとなる。ネームスペースはControllerのURLマッピングと同じとするため、「@RequestMapping」と同じ値を設定する。

ネームスペースの記述例を以下に示す。

|  |
| --- |
| @Controller  @RequestMapping("Member/register")  @TransactionTokenCheck("Member/register")  **public** **class** MemberRegisterController {  // 省略  } |

### トランザクションの開始

「イベントURLマッピング一覧」の「トランザクショントークンチェック」が「BEGIN」となっているメソッドに、トランザクションの開始を示す「@TransactionTokenCheck(type=TransactionTokenType.BEGIN)」のアノテーションを付与する。

トランザクション開始の記述例を以下に示す。

|  |
| --- |
| @RequestMapping(method = RequestMethod.*POST*, params = "confirm")  @TransactionTokenCheck(type = TransactionTokenType.*BEGIN*)  **public** String registerConfirm(@Validated MemberRegisterForm memberRegisterForm,  BindingResult result, Model model) {  // 省略  } |

### トランザクションのチェック

「イベントURLマッピング一覧」の「トランザクショントークンチェック」が「IN」となっているメソッドに、トランザクションのチェックを実施する「@TransactionTokenCheck」のアノテーションを付与する。

TransactionTokenType.INがデフォルトのため、INの場合はtypeでの指定を省略する。詳細についてはガイドライン（5.12.2.3.7 トランザクショントークンチェックのControllerでの利用方法）を参照すること。

トランザクションチェック実施の記述例を以下に示す。

|  |
| --- |
| @RequestMapping(method = RequestMethod.*POST*)  @TransactionTokenCheck  **public** String register(@Validated MemberRegisterForm memberRegisterForm,  BindingResult result, Model model, RedirectAttributes redirectAttributes) {  // 省略  } |

### その他のメソッド

Controllerクラスとして必要なメソッドは前述の通りである。その他、通常のJavaクラスとして必要な処理がある場合、適宜必要なprivateメソッドを追加すること。

### Formクラス－ドメイン層のJavaBeanの変換

ドメイン層がアプリケーション層に依存しないようにするため、Formクラスをドメイン層に渡すことはできない。通常、ドメイン層はDTO、Entity等のJavaBeanを利用するため、FormクラスとこれらのJavaBean間でのデータの詰め替えが必要となる。

属性単位にgetter/setterを利用することで値のコピーが可能であるが、非常に煩雑であるためBeanマッパーである「Dozer」の利用を強く推奨する。

### メッセージ出力

業務エラー等で画面にメッセージを出力する場合、エラーレベル（success、info、warn、error、danger）に応じたResultMessagesを取得し、addメソッドにてメッセージキーと引数を指定する。

メッセージ出力の記述例を以下に示す。

|  |
| --- |
| // errorレベル  ResultMessages messages = ResultMessages.*error*();  // 引数の作成  String[] param = **new** String[]{"引数1", "引数2", "引数3"};  // メッセージキーと引数を追加  messages.add("メッセージのプロパティキー", param);  // ResultMessagesをModelに追加  model.addAttribute(messages); |

## Helperクラスの実装

### Helperクラスの作成

HelperクラスとはControllerクラスを補助する役割のものであり、アプリケーション層とドメイン層のモデル相互変換など、Controller本来の処理以外の処理を行うものである。

Helperの位置づけとしてはControllerの一部とみなして構わない。また、Helperクラスの作成はオプションであり、必ずしも作成する必要はない。作成する場合はPOJOとして作成すること。

## Validatorクラスの実装

### Spring Validatorクラスの作成

入力チェックを担当する独自ValidatorクラスにはSpring ValidatorとBean Validationの二種類があるが、ここでは業務開発者が実装すべきSpring Validatorクラスについて説明する。Validatorクラスの違いについては「 」を参照すること。

Spring Validatorの独自Validatorクラスは「org.springframework.validation.Validator」インターフェースを実施し、コンポーネントスキャンの対象とするため「@Component」をクラスアノテーションとして付与すること。

Validatorクラスの記述例を以下に示す。

|  |
| --- |
| **package** jp.co.ntt.atrs.app.c1;  **import** jp.co.ntt.atrs.app.c0.MemberForm;  **import** jp.co.ntt.atrs.app.c0.MemberValidator;  **import** jp.co.ntt.atrs.domain.service.c1.MemberRegisterErrorCode;  **import** org.springframework.stereotype.Component;  **import** org.springframework.util.StringUtils;  **import** org.springframework.validation.Errors;  **import** org.springframework.validation.ValidationUtils;  **import** org.springframework.validation.Validator;  **import** javax.inject.Inject;  /\*\*  \* 会員情報登録フォームのバリデータ。  \*  \* @author NTT 電電花子  \*/  @Component  **public class** MemberRegisterValidator **implements** Validator {  /\*\*  \* 会員情報バリデータ。  \*/  @Inject  MemberValidator memberValidator;  /\*\*  \* {@inheritDoc}  \*/  @Override  **public boolean** supports(Class<?> clazz) {  **return** MemberRegisterForm.**class**.isAssignableFrom(clazz);  }  /\*\*  \* {@inheritDoc}  \*/  @Override  **public void** validate(Object target, Errors errors) {  MemberRegisterForm form = (MemberRegisterForm) target;  MemberForm memberForm = form.getMemberForm();  // パスワードチェック  String password = form.getPassword();  String reEnterPassword = form.getReEnterPassword();  **if** (StringUtils.*hasLength*(password) && StringUtils.*hasLength*(reEnterPassword)) {  **if** (!password.equals(reEnterPassword)) {  // パスワードと再入力したパスワードが違う場合エラー  errors.reject(MemberRegisterErrorCode.*E\_AR\_C1\_5001*.code());  }  }  **try** {  errors.pushNestedPath("memberForm");  ValidationUtils.*invokeValidator*(memberValidator, memberForm, errors);  } **finally** {  errors.popNestedPath();  }  }  } |

## JSPの実装

### 外部ファイル（CSS、JavaScript、画像ファイル）の読み込み方法

#### ファイル読み込みの記述方法

外部ファイルを読み込む際はWebアプリケーションのコンテキスト名の影響をさけるため、コンテキスト名からの相対パスとして記述することとする。

外部ファイル読み込みの記述例を以下に示す。

* + CSSファイルを読み込む場合

|  |
| --- |
| <link rel=*"stylesheet"*  href=*"*${pageContext.request.contextPath}*/resources/css/CSSファイル名"*> |

* + 画像ファイルを読み込む場合

|  |
| --- |
| <img src=*"*${pageContext.request.contextPath}*/resources/image/画像ファイル名"*  alt=*"画像の説明文" /*> |

* + JavaScriptファイルを読み込む場合

|  |
| --- |
| <script type=*"text/javascript"*  src=*"*${pageContext.request.contextPath}*/resources/js/JavaScriptファイル名"*>  </script> |

### 入力項目の記述方法

#### テキストフィールド

テキストフィールドとは画面仕様の「論理項目種別」で「テキストボックス」として定義している項目である。テキストフィールドの場合、<form:input>タグを利用する。

テキストフィールドの記述例を以下に示す。

|  |
| --- |
| <form:input path=*"Formクラスの属性名"*  cssErrorClass=*"エラー時に適用するCSSのクラス"* maxlength=*"xxx"* /> |

※maxlengthはFormの属性に応じて適宜設定する。なお、maxlengthはHTML5からサポートされた属性のため、HTML5をサポートしていないブラウザ（バージョン）では有効ではないので注意すること。

size属性についてはテキストフィールドの見た目のサイズであるためCSSにて指定すること。

#### 入力チェックのエラーメッセージ表示

単項目の入力チェックでエラーが発生した場合、エラーメッセージを入力項目の近く（上下または左右）に表示する。

入力チェックのエラーメッセージ表示の記述例を以下に示す。

|  |
| --- |
| <form:errors path=*"Formクラスの属性名"* cssClass=*"適用するCSSのクラス"* /> |

#### テキストエリア

テキストエリアとは、画面仕様の「論理項目種別」で「テキストエリア」として定義している項目である。テキストフィールドの場合、<form:textarea>タグを利用する。

テキストエリアの記述例を以下に示す。

|  |
| --- |
| <form:textarea path=*"Formクラスの属性名"*  cssErrorClass=*"エラー時に適用するCSSのクラス"*  cols=*"xxx"* rows=*"xxx"* /> |

※cols、rowsはFormの属性に応じて適宜設定する。

#### ラジオボタン

ラジオボタンとは、画面仕様の「論理項目種別」で「ラジオボタン」として定義している項目である。ラジオボタンの場合、<form:radiobuttons>タグを利用する。

ラジオボタンの記述例を以下に示す。

|  |
| --- |
| <form:radiobuttons path=*"Formクラスの属性名"*  cssErrorClass=*"エラー時に適用するCSSのクラス"* items=*"選択肢のコレクション"*  delimiter=*"区切り文字"* /> |

※items の選択肢のコレクションは事前にControllerにてModelに設定する必要がある。

※delimiter の区切り文字は選択肢を画面表示する際に利用される。

「*&nbsp;*」（半角スペース）や「*<br>*」（改行）等、画面レイアウトに応じて設定すること。

#### チェックボックス

チェックボックスは、画面仕様の「論理項目種別」で「チェックボックス」として定義している項目である。チェックボックスの場合、<form:checkboxes>タグを利用する。

チェックボックスの記述例を以下に示す。

|  |
| --- |
| <form:checkboxes path=*"Formクラスの属性名"*  cssErrorClass=*"エラー時に適用するCSSのクラス"* items=*"選択肢のコレクション"* /> |

※items の選択肢のコレクションは事前にControllerにてModelに設定する必要がある。

　コードリストを利用する場合items に${コードリストのBeanID}を指定する。

#### プルダウン

プルダウンとは、画面仕様の「論理項目種別」で「リストボックス」として定義している項目である。プルダウンの場合、<form:select>タグおよび<form:options>タグを利用する。

プルダウンの記述例を以下に示す。

|  |
| --- |
| <form:select path=*"Formクラスの属性名"*  cssErrorClass=*"エラー時に適用するCSSのクラス"*>  <form:option value=*""*>*未選択*</form:option>  <form:option**s** items=*"選択肢のコレクション"* />  </form:select> |

※items の選択肢のコレクションは事前にControllerにてModelに設定する必要がある。

コードリストを利用する場合items に${コードリストのBeanID}を指定する。

　必要に応じて未選択を示す「<form:option value=*""*>」をoptionsの前に追加すること。

#### hidden項目

hidden項目とはFormクラスの属性を作成する際、「隠しフィールド」として設計した項目である。画面仕様には存在しないがデータ連携を考慮し、実装工程にて追加される。hidden項目の場合、<form:hidden>タグを利用する。

hidden項目の記述例を以下に示す。

|  |
| --- |
| <form:hidden path=*"Formクラスの属性名"* /> |

### 出力項目の記述方法

#### 通常の出力項目

通常の出力項目とは、画面仕様の「論理項目種別」で「ラベル」として定義している項目である。出力項目の場合、EL式の${f:h(～)}を利用する。

通常の出力項目の記述例を以下に示す。

|  |
| --- |
| ${f:h(dtoクラス.属性名)} |

※上記の「dtoクラス」はModelに格納されている属性名であり、通常、クラス名の先頭を小文字に変換したものである。

#### 入力項目を出力する場合

入力項目はFormクラスの属性として実装される。そのため、入力項目を出力する場合、通常の出力項目とは異なり、ModelではなくFormから値を取得する必要がある。

入力項目を出力する場合の記述例を以下に示す。

|  |
| --- |
| ${f:h(フォームクラス.属性名)} |

※上記の「フォームクラス」はModelに格納されている属性名であり、通常、クラス名の先頭を小文字に変換したものである。

#### 項目（ラベル）

項目（ラベル）とは、入力項目に紐付く項目名である。ラベルについてはHTMLのlabel要素を設定するため、<form:label>タグを利用すること。

項目を表示する際の記述例を以下に示す。

|  |
| --- |
| <form:label path=*"Formクラスの属性名"*>ラベルとして表示する値</form:label> |

#### メッセージ

業務メッセージや相関チェックのエラーメッセージは通常の出力項目とは異なる。メッセージを表示する際は、${f:h(～)}ではなく<t:messagesPanel>タグを利用する。

メッセージを表示する際の記述例を以下に示す。

* メッセージをまとめて表示する場合

|  |
| --- |
| <t:messagesPanel /> |

* 属性名を指定して個別にメッセージを表示する場合

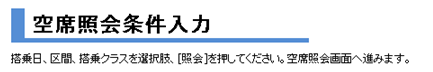
|  |
| --- |
| <t:messagesPanel messagesAttributeName=*"属性の名称"*/> |

※「属性の名称」はControllerでModelに設定した際の名称である。

#### ガイダンスメッセージ

業務メッセージやエラーメッセージではなく画面上に常に表示されるような静的なガイダンスメッセージについては、多言語対応を実施しないためJSPにハードコーディングすることとする。

このようなメッセージ



#### フォーマットを指定した出力

数値型や日付型の属性の場合、画面表示時にフォーマットを指定した出力が必要となる場合がある。

フォーマットを指定して出力する場合の例を以下に示す。

* 数値型をフォーマット指定で出力

数値型の値をフォーマット指定で出力する場合、JSTLのJSPタグライブラリから提供されている<fmt:formatNumber>タグを利用する。

|  |
| --- |
| <fmt:formatNumber value=*"属性の名称"* pattern=*"フォーマット"* /> |

※*"フォーマット"*には”0.00”等のフォーマットを指定する。

* 日時型をフォーマット指定で出力

日時型の値をフォーマット指定で出力する場合、JSTLのJSPタグライブラリから提供されている<fmt:formatDate>タグを利用する。

|  |
| --- |
| <fmt:formatDate value=*"属性の名称"* pattern=*"フォーマット"* /> |

※*"フォーマット"*には”yyyy-MM-dd”等のフォーマットを指定する。

* 日時型をフォーマット指定で出力（joda timeの場合）

日時のオブジェクト型としてorg.joda.time.DateTimeなどを利用する場合、JodaTimeから提供されているJSPタグライブラリを利用する。

|  |
| --- |
| <joda:format value=*"属性の名称"* pattern=*"フォーマット"* /> |

※*"フォーマット"*には”yyyy-MM-dd”等のフォーマットを指定する。

### 繰り返し項目の記述方法

入出力項目の記述方法は前述した通りであるが、繰り返し項目の場合、項番という要素を考慮する必要がある。

繰り返し項目を記述する場合、<c:forEach>タグを利用して繰り返し・ループを実現し、入出力項目の記述自体は前述の方法で各項目を記述することとなる。ポイントはforEachのvarStatusで繰り返しの項番を確認することと、入出力の項目に項番を含めることである。

繰り返し項目の記述例を以下に示す。なお、例は入力項目の場合であるが、繰り返しの実現方法は出力項目の場合も同様である。

|  |
| --- |
| <c:forEach items=*"繰り返し項目のコレクション"* varStatus=*"ステータス変数"*>  <form:label path=*"繰り返し項目のコレクション[${ステータス変数.index}].address"*  cssErrorClass=*"エラー時に適用するCSSのクラス"*>住所</form:label>  <form:input path=*"繰り返し項目のコレクション[${ステータス変数.index}].address"*  cssErrorClass=*"エラー時に適用するCSSのクラス"* maxlength=*"50"* />  <form:errors path=*"繰り返し項目のコレクション[${ステータス変数.index}].address"* />  <form:label path=*"繰り返し項目のコレクション[${ステータス変数.index}].name"*  cssErrorClass=*"エラー時に適用するCSSのクラス"*>氏名</form:label>  <form:input path=*"繰り返し項目のコレクション[${ステータス変数.index}].name"*  cssErrorClass=*"エラー時に適用するCSSのクラス"* maxlength=*"20"* />  <form:errors path=*"繰り返し項目のコレクション[${ステータス変数.index}].name"* />  <form:label path=*"繰り返し項目のコレクション[${ステータス変数.index}].tel"*  cssErrorClass=*"エラー時に適用するCSSのクラス"*>電話番号</form:label>  <form:input path=*"繰り返し項目のコレクション[${ステータス変数.index}].tel"*  cssErrorClass=*"エラー時に適用するCSSのクラス"* maxlength=*"20"* />  <form:errors path=*"繰り返し項目のコレクション[${ステータス変数.index}].tel"* />  </c:forEach> |

※*"繰り返し項目のコレクション"* として住所：address、氏名：name、電話番号:telの属性があるオブジェクトのコレクションと想定した場合の例である。

*${ステータス変数.index}*は0から始まるループのインデックス番号であり、これをコレクションの[添え字]として指定して該当のオブジェクトを指定する。

*ステータス変数*には*index*の他にも以下のような変数があるので必要に応じて利用すること。

* count

1から始まるループのインデックス番号

* first

現在がループの先頭かどうかを判定（true/false）

１行目に装飾等で特別な処理が必要な場合等に利用

* last

現在がループの最後かどうかを判定（true/false）

最終行目に装飾等で特別な処理が必要な場合等に利用

## プロパティファイルの実装

### ファイル一覧

本システムで利用するプロパティファイルの一覧をに示す。なお、記載のファイル名は基底名称である。

表 31　プロパティファイル一覧

|  |  |  |  |
| --- | --- | --- | --- |
| **項番** | **ファイル** | **ファイルパス** | **備考** |
| 1 | 業務初期データ | ${domain}/src/main/resources  /META-INF/spring/atrs | 「初期出発空港コード」等の各種初期データ  「予約代表者に必要な最小年齢」等の各種しきい値データ |
| 2 | 入力フォーム名称 | ${domain}/ src/main/resources  /i18n/atrs-fields | 「depAirport=出発空港」等の各種入力フォームの表示項目名称 |
| 3 | ユーザメッセージ | ${domain}/ src/main/resources  /i18n/atrs-messages | 共通バリデーションのエラーメッセージやユーザメッセージ |

### 読み込み設定

上記で定義したプロパティファイルをメッセージリソースとして取り込むため、Spring Bean定義ファイル（applicationContext.xml）に読み込み設定を追記する。また、プロパティファイルのプレイスホルダーを有効にする設定を追記する。なお、この作業はプロジェクトで一度実施すればよく、各作業者が個別に実施する必要はない。

#### 修正対象ファイル（メッセージリソースの読み込み）

${domain} /src/main/resources/META-INF/spring/atrs-domain.xml

#### 修正内容（メッセージリソースの読み込み）

以下の網掛けの内容（クラスパス上の拡張子を除いたファイルパス）を追記する。

|  |
| --- |
| <bean id=*"messageSource"*  class=*"org.springframework.context.support.ResourceBundleMessageSource"*>  <property name=*"basenames"*>  <list>  <value>i18n/atrs-messages</value>  <value>i18n/atrs-fields</value>  </list>  </property>  </bean> |

#### 修正対象ファイル（プレイスホルダーの設定）

${web} /src/main/resources/META-INF/spring/applicationContext.xml

${web} /src/main/resources/META-INF/spring/spring-mvc.xml

#### 修正内容（プレイスホルダーの設定）

以下の網掛けの内容（クラスパス上及び/META-INF/spring/配下のプロパティファイルに対してプレイスホルダー設定を有効）を追記する。

|  |
| --- |
| <context:property-placeholder  location=*"classpath\*:/META-INF/spring/\*.properties"* /> |

### プロパティファイルの記述

#### 入力チェックエラーメッセージプロパティ

画面の単項目入力チェック（Validation）におけるデフォルトのエラーメッセージが一般的でないため、デフォルトメッセージの変更を行う。なお、相関チェックや業務チェックにおけるエラーメッセージは業務ロジックでのメッセージ出力となるため、業務メッセージプロパティに定義する。

＜プロパティの形式＞

入力チェックのルール名称=入力チェック時のエラーメッセージ

メッセージID=メッセージ本文

例）@NotNull　の場合

|  |
| --- |
| NotNull = This item is required. Please input here. |

例）業務エラーメッセージの場合

|  |
| --- |
| e.ar.c0.5002 =市外局番と市内局番の合計桁数が6～7桁の範囲外です。入力内容をご確認ください。 |

※実際はNative2asciiにて変換されたascii文字列（e.ar.c1.5003=\u5e02\u5916\u5c40\・・・）になっていること。

## CSSファイルの実装

### CSSファイルの実装

CSS（Cascading Style Sheet）ファイルの実装においても一定の品質を担保するため、作成実施要領やコーディング規約等が必要であるが、CSSはその責務上、ユーザインターフェースに強く関係・依存するため、UI規約[[2]](#footnote-2)と合わせて検討する必要がある。そのため、CSSの実装については「UI規約」を参照すること。

## JavaScriptファイルの実装

### JavaScriptファイルの実装

JavaScriptファイルの実装においても一定の品質を担保するため、作成実施要領やコーディング規約等が必要であるが、JavaScriptはユーザの操作性向上のために用いるとの方針からユーザインターフェースに強く関係・依存するため、UI規約と合わせて検討する必要がある。そのため、JavaScriptの実装については「UI規約」を参照すること。

1. TERASOLUNA Server Framework for JavaにおけるOSSスタックの詳細はガイドライン「2.1 TERASOLUNA Server Framework for Java (5.x)のスタック」を参照のこと。 [↑](#footnote-ref-1)
2. 本システムの開発では作業する人数が少数であり、作業内容の統一のためにUI規約を作成する必要がないため、作成対象外としている。 [↑](#footnote-ref-2)