アプリケーション開発ガイド

第0．5版

2017年8月30日

改訂履歴

|  |  |  |  |
| --- | --- | --- | --- |
| **バージョン** | **日付** | **内容** | **備考** |
| 0.50 | 2017/8/30 |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 目　　次

[1. 目　　次 3](#_Toc491862322)

[1. はじめに 6](#_Toc491862323)

[(ア) 本ガイドについて 6](#_Toc491862324)

[① 対象読者 6](#_Toc491862325)

[② 前提知識 6](#_Toc491862326)

[2. 全体アーキテクチャ 7](#_Toc491862327)

[(ア) コンセプト 7](#_Toc491862328)

[(イ) サーバー構成 8](#_Toc491862329)

[(ウ) ソフトウェア構成 9](#_Toc491862330)

[(エ) アプリケーションの配置 10](#_Toc491862331)

[3. アプリケーション実行制御 11](#_Toc491862332)

[(ア) トランザクション制御 11](#_Toc491862333)

[① トランザクション管理 11](#_Toc491862334)

[② ユーザトランザクション 11](#_Toc491862335)

[③ マルチデータソース 11](#_Toc491862336)

[(イ) セッション管理 12](#_Toc491862337)

[(ウ) コンテキスト管理 12](#_Toc491862338)

[① 動的オブジェクトの注入 12](#_Toc491862339)

[② EL式からのアクセス 12](#_Toc491862340)

[(エ) 例外処理 13](#_Toc491862341)

[① 業務的な例外処理 13](#_Toc491862342)

[② チェック例外と非チェック例外 13](#_Toc491862343)

[4. アプリケーション共通ライブラリ 14](#_Toc491862344)

[(ア) アプリケーション情報 14](#_Toc491862345)

[① コンテキスト情報機能 14](#_Toc491862346)

[② メッセージリソース機能 15](#_Toc491862347)

[(イ) 画面機能 17](#_Toc491862348)

[① PDF生成機能 17](#_Toc491862349)

[② 国際化 17](#_Toc491862350)

[(ウ) 外部連携 18](#_Toc491862351)

[① REST受信機能 18](#_Toc491862352)

[② メール送信機能 23](#_Toc491862353)

[(エ) 認証機能 25](#_Toc491862354)

[① ログイン機能 25](#_Toc491862355)

[② トークン発行 26](#_Toc491862356)

[③ トークン削除 27](#_Toc491862357)

[④ トークン有効性チェック 27](#_Toc491862358)

[(オ) ユーザー管理機能 28](#_Toc491862359)

[① ユーザー情報 28](#_Toc491862360)

[② ユーザーID登録 29](#_Toc491862361)

[③ ユーザーID本登録 30](#_Toc491862362)

[④ パスワード初期化トークン通知 31](#_Toc491862363)

[⑤ パスワード初期化 31](#_Toc491862364)

[⑥ パスワード変更 31](#_Toc491862365)

[⑦ ロール 32](#_Toc491862366)

[(カ) ロギング機能 32](#_Toc491862367)

[① SQLトレース 33](#_Toc491862368)

[② デバッグログ 34](#_Toc491862369)

[③ 性能トレース 34](#_Toc491862370)

[(キ) ワークフロー機能 35](#_Toc491862371)

[(ク) ライブラリ機能 35](#_Toc491862372)

[5. アプリケーション開発環境 36](#_Toc491862373)

[(ア) 開発環境 36](#_Toc491862374)

# はじめに

## 本ガイドについて

この「アプリケーション開発ガイド」は、Javaフレームワーク『handywedge』（MITライセンス）を用いた Webアプリケーションを開発するためのガイドである。  
アプリケーション開発者は本ガイドに従って開発を行う。

### 対象読者

本ガイドは、アプリケーション開発者を対象とする。

### 前提知識

アプリケーションを開発するにあたり、次の知識が必要である。  
①　JavaによるWebアプリケーションの開発  
②　Java EE8 （JSF、CDI）

③　slf4J（ログ出力）

# 全体アーキテクチャ

## コンセプト

Javaフレームワーク『handywedge』を用いたアプリケーション開発のコンセプトを以下に示す。

**・製品ベンダーに依存しないWebシステム基盤アーキテクチャ**

アプリケーションサーバー、データベースなどオープンな技術を使用することで製品ベンダーに依存しないWebシステム基盤の設計が可能。

信頼性、完全性が求められるデータベースは有償の製品を使用するなど、要件に応じた基盤設計を実現する。

**・システム要件に応じた多彩なアプリケーション・アーキテクチャ**

フレームワーク『handywedge』は、シンプルな構成のフレームワークであり、Webアプリケーションに一般的に必要な機能（認証、ユーザー管理、ワークフローなど）に加え、開発現場で使用頻度の高い様々なライブラリ（ロギング、REST通信、メール送信など）を提供。

システム要件、また開発者のスキルにより多彩なアプリケーション設計が可能。

## サーバー構成

Javaフレームワーク『handywedge』を使用したサーバー論理構成例を以下に示す。



APサーバー

APサーバー

DBサーバー

DBサーバー



アプリケーションが多くなった場合APサーバーをグループ分けし、グループごとに対象のアプリケーションを配備する

APサーバー

ファイアーウォール

ロードバランサー

ロードバランサー



・アプリケーションサーバーは、業務要件に応じシングル構成、また複数台構成と依存することはない。

・アプリケーションの負荷の増大に伴い、アプリケーションサーバーを増やしスケールアウトする。

・アプリケーションサーバーの物理的限界を超えてしまう場合は、アプリケーションサーバーをグループ分けし負荷分散装置で対象のアプリケーションサーバーグループに振り分ける。

・アプリケーションサーバーへの負荷分散はsticky 接続のステートフルな構成とする。（オンメモリーセッションを前提とする場合）

## ソフトウェア構成

ソフトウェア構成を以下に示す。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **アプリケーション** | | | | | | | |
| **UIコンポーネント** |  |  |  | | | |  |
| **CDI** | | **フレームワーク(handywedge)** | | | |  |
| **AP Server (Servlet Container)** | | | | **JDBC** | **PDF Engine** | **Logging Facade** | |
| **Logger** | |
| **Java VM** | | | | | | | |
| **OS** | | | | | | | |

|  |  |  |
| --- | --- | --- |
| **ソフトウェア** | **プロダクト** | **備　考** |
| OS |  |  |
| Java VM | Java 8 |  |
| AP Server |  | Tomcat 8など |
| JDBC |  | PostgreSQL 9、SQLServer、Oracleなど |
| PDF Engine | JasperReport 6.3.0 |  |
| Logger |  | log4j 2.5など |
| Logging Facade | slf4j 1.7.13 |  |
| UI Component |  | PrimeFaces 6など |
| CDI | Weld 2.2.4 Final |  |

各ソフトウェアのバージョンは互換性を確認した上で変更されることがある。

## アプリケーションの配置

アプリケーションのモジュールは次のようなwarファイルの形式で構成する。（構成例）

handywedge-sample.war

├── META-INF

│ └── context.xml　tomcat設定ファイル

├── WEB-INF

│ ├── beans.xml　CDI設定

│ ├── faces-config.xml　JSF設定

│ ├── web.xml　Webアプリケーション設定

│ ├── classes

│ │ ├── handywedge-sample\_en.properties　システムプロパティ（ロケール米国）

│ │ ├── handywedge-sample\_ja.properties　システムプロパティ（ロケール日本）

│ │ ├── handywedge-sample.properties　システムプロパティ

│ │ ├── jasperreport\_extension.properties　帳票設定

│ │ ├── log4j2.xml　ロギング設定

│ │ ├── report

│ │ │ └── \*.jasper　帳票テンプレート

│ │ └── jp

│ │ └── cstudio

│ │ └── handywedge

│ │ └── sample

│ │ ├── \*.class　Javaクラス

│ │ └...

│ └── lib

│ ├── handywedge-api-\*.jar　FWライブラリ

│ ├── handywedgw-core-\*.jar　FWライブラリ

│ └── \*.jar　FWおよびAPの依存ライブラリ

├── index.xhtml　Webリソース

…

# アプリケーション実行制御

## トランザクション制御

トランザクションとは永続性記録装置（データベース）への登録、更新、削除などの処理を行う際、整合性を保持するために分割することができない一連の単位を表す。

### トランザクション管理

トランザクション開始点となるメソッドにフレームワークが提供する@FWTransactionalアノテーションを付加することで、このメソッドをトランザクションの境界としフレームワークがトランザクションの管理を行う。フレームワークでは@FWTransactionalアノテーションが付加されたメソッドが正常に終了すれば当該トランザクションはコミットし、例外またはエラーがスローされればそのトランザクションをロールバックする。

@FWTransactional(dataSourceName = "jdbc/ds\_handywedge")  
**public** **int** insert (Todo todo) {  
　：  
}

トランザクションが開始されている状態で、再度フレームワークが提供する@ FWTransactionalアノテーションを付加したメソッドが呼び出された場合でも、既に開始されているトランザクションに属する。

### ユーザトランザクション

トランザクションはフレームワークで管理するため、アプリケーションによるトランザクション処理（コミット、ロールバック）は原則禁止とする。

### マルチデータソース

トランザクションの単位でデータソースを切り替えることができる。  
@FWTransactionalアノテーション [dataSourceName] にデータソース(JNDI名)を指定する。

@FWTransactional(dataSourceName = "jdbc/ds\_handywedge") **// JNDI名を指定する**  
**public** **int** insert (Todo todo) {  
　：  
}

## セッション管理

セッションとはユーザーがログインしてからログアウトするまでの期間のことを表す。  
本フレームワークではセッション情報の管理は行わない。その為、アプリケーション開発者は業務要件に応じ実装する。

## コンテキスト管理

CDI（Contexts and Dependency Inject）を活用すると、依存するオブジェクトを注入するだけでなく次のようなメリットを享受できる。  
・インスタンスのライフサイクル管理やオブジェクトの生成、値のセットなどが簡単になる  
・コンポーネントやクラスを疎結合にできる  
・インターセプトで自動的にログを出力することができる  
  
CDIを利用するには、warファイルのWEB-INF配下にbeans.xmlファイルを配置する。

## 動的オブジェクトの注入

条件によって注入するオブジェクトを変えたい場合には、@Producesアノテーションを付与したメソッドで注入するオブジェクトを動的に変更することができる。

## EL式からのアクセス

JSFではManaged Beanに@Namedアノテーションを付けることで、JSFのEL式からアクセスすることができる。

また、暗黙オブジェクト[fwContext]を用い、アプリケーション情報、ユーザー情報（ログインユーザー）を取得することができる。

#### ＜使用例＞

<span>あなたのロールは「<h:outputText value="#{fwContext.user.roleName}" />」です</span>

## 例外処理

原則として処理の途中で発生した例外（Exceptionクラスのサブクラス）はアプリケーションが責任を持って例外処理を行う。  
致命的なエラー（Errorクラスのサブクラス）の場合はフレームワークにエラー処理を委譲する。

### 業務的な例外処理

アプリケーションは業務的に独自に実装した例外をスローすることができる。例えば、新規に登録しようとしたIDが既に登録されている場合、業務ロジックで例外をスローして画面側でエラー処理を行いたいような場合には、フレームワークで規定する@FWBusinessExceptionアノテーションを付加した業務例外クラスを実装してスローする。@FWBusinessExceptionアノテーションが付加されていない例外がスローされ、フレームワークがキャッチするとフレームワークでは致命的なエラーとして処理する。

@FWBusinessException  
**public** **class** SampleAppException **extends** Exception {  
　：  
}

フレームワークでは致命的なエラーをキャッチした場合、エラーの内容およびスタックトレース情報をエラーログに出力する。

@FWBusinessExceptionアノテーションが指定されているクラスを継承したサブクラスでも@FWBusinessExceptionアノテーションは有効ではあるが、各クラスに明示的に指定することを推奨する。また、当該クラスをフレームワークでキャッチした場合、警告（WARN）レベルでログ出力する。

### チェック例外と非チェック例外

基本的にはアプリケーションではスローされる例外は呼び出し元でチェックできるようチェック例外として実装する。  
設計上ありえないケースなどを例外とする場合は、非チェック例外（Runtime Exception）でスローし、フレームワークに致命的エラーと処理させる。この場合、エラーの原因となった変数などを例外クラスに持たせておくと障害解析がやりやすくなる。

# アプリケーション共通ライブラリ

## アプリケーション情報

### コンテキスト情報機能

コンテキスト情報としてログインしているユーザーに関する情報やリクエストに関する情報を提供する。

#### ＜コンテキスト情報機能で提供している情報＞

|  |  |
| --- | --- |
| **項目名** | **内容** |
| ユーザー情報 | ログイン中のユーザー情報　※詳細は「(オ) ユーザー管理機能 ①ユーザー情報」参照 |
| リクエストID | リクエスト毎に一意になるID |
| リクエストURL | リクエストURL（クエリーストリングは含まない） |
| リクエスト開始時刻 | リクエストを受け付けた時刻 |
| 最終アクセス時刻 | 前回のリクエストを受け付けた時刻 |
| サーバー名 | リクエストを処理しているサーバー名 |
| アプリケーションID | アプリケーションを特定するID |
| コンテキストパス | アプリケーションのコンテキストパス |
| APIトークン | リクエストで認証されたAPIトークン |

#### ＜使用方法＞

　FWContextインターフェイスの変数を定義し、＠Injectアノテーションを付加してオブジェクトを注入する。

**@Inject**

**plivate** FWContext context  
 ：  
 FWUser **user** = context.getUser();

String **loginId** = **user**.getId ();

### メッセージリソース機能

ログインしているユーザーのロケールに応じたメッセージリソースを提供する。

#### ＜構成例＞

　プロパティファイルの基底名をアプリケーションIDとし、ロケール毎にプロパティを定義することができる。

　例）アプリケーションIDが「handywedge-sample」の場合

handywedge-sample.war

：

├── WEB-INF

│ 　：

│ ├── classes

│ │ ├── handywedge-sample\_en.properties　システムプロパティ（ロケール米国）

│ │ ├── handywedge-sample\_ja.properties　システムプロパティ（ロケール日本）

│ │ ├── handywedge-sample.properties　システムプロパティ

│ 　：

…

※プロパティファイルは、ルート直下に配置

#### ＜FW設定プロパティ＞

「システムプロパティ」には、フレームワーク設定値として指定の項目が存在する。

※フレームワーク設定値は、別紙「FWコンフィグガイド」参照。

#### ＜使用方法＞

FWMessageResourceインターフェイスの変数を定義し、＠Injectアノテーションを付加してオブジェクトを注入する。

@Inject  
 **private** FWMessageResources resources;

FWMessageResourceからメッセージを取得する。

message = resources.get("welcome.message ");

## 画面機能

### PDF生成機能

フレームワークでは、PDF生成エンジンJasperreportを使用してPDFファイルを生成する機能を提供する。

#### ＜テンプレートファイル定義＞

　Jasperreport Studio等を利用して帳票のフォーマットをデザインしたテンプレートファイル（Jasper Reportファイル）を作成する。

　作成したテンプレートファイルは、warファイルの「WEB-INF/classes/report」ディレクトリに配備する。

ディレクトリの配下には任意のディレクトリを作成することができる。

#### ＜使用方法＞

　FWPDFReportWriterインターフェイスの変数を定義し、＠Injectアノテーションを付加してオブジェクトを注入する。

@Inject  
 **private** FWPDFReportWriter **writer**;

テンプレートファイル名を指定してFWPDFReportのインスタンスを生成しする。

アプリケーションID名のディレクトリまではフレームワークが自動的にセットするので、それ以降のパスを指定する。

        FWPDFReport **report** = **new** FWPDFReport(templateFile);

Mapにテンプレートファイルで指定したフィールド名と出力する値のペアを登録し、FWPDFReportのインスタンスにセットする。

        Map<String, Object> **parameter** = **new** HashMap<>();  
      parameter.put("testHeader", paramValue);   
        report.setParameters(parameter);

明細行のように繰り返し出力するものは1行分のデータを１つのオブジェクト（フィールド名と紐付くアクセサを持ったDTO）として、

Listに登録してFWPDFReportのインスタンスにセットする。

        List <TestEntity> **list** = **new** ArrayList<>();  
  
        TestEntity **entity1** = **new** TestEntity();  
        entity1.setId("ID00001");  
        entity1.setName("AXXXX1");  
        list.add(entity1);   
  
        TestEntity **entity2** = **new** TestEntity();  
        entity2.setId("ID00002");  
        entity2.setName("AXXXX2");  
        list.add(entity2);   
  
        report.addAllDetails(list);

FWPDFReportWriterにFWPDFReportのインスタンスを登録する。

        writer.addReport(report);

PDFを出力させるOutputStreamを用意しprintメソッドを実行する。

この例ではメモリを節約するため、HttpResponseのOutputStreamに出力している。

        FacesContext **fc** = FacesContext.**getCurrentInstance**();  
        ExternalContext **ec** = fc.getExternalContext();  
        ec.responseReset();  
        ec.setResponseContentType("application/pdf");  
      ec.setResponseHeader("Content-Disposition", "attachment; filename=\"" + reportFile + "\"");  
        OutputStream **os** = ec.getResponseOutputStream();  
  
        **try** {  
            writer.print(os);   
        } **catch** (Exception **e**) {  
            logger.error("帳票出力でエラーが発生しました。", e);   
            // エラー処理  
        } **finally** {  
            fc.responseComplete();  
            os.close();  
        }

### 国際化

フレームワークでは、コンテキスト情報で持つユーザーのロケールに合わせたメッセージリソースを提供する。

プロパティファイルの基底名をアプリケーションIDとし、ロケール毎にプロパティを定義することができる。

※プロパティの配置（構成例）、及びビジネスロジックからのメッセージリソース取得は「(ア) アプリケーション情報 ②メッセージリソース機能 ＜構成例＞」参照。ここではロケール毎のプロパティ定義例、及び画面からのメッセージリソース取得方法を記載する。

#### ＜プロパティ定義例＞

*アプリケーションID*\_**en**.properties　// プロパティ（ロケール米国）

*アプリケーションID*\_**ja**.properties　// プロパティ（ロケール日本）

#### ＜プロパティ（ロケール米国）＞

welcome.message=hello test app!(en)

#### ＜プロパティ（ロケール日本）＞

welcome.message=テストアプリへようこそ！

#### ＜使用方法＞

暗黙オブジェクト[fwMsgResources]を用い、JSFからメッセージを取得することができる。

<h:outputText value="#{fwMsgResources.get('welcome.message')}" />

## 外部連携

### REST受信機能

REST受信機能を利用し、RESTでリクエストを受け付けてビジネスロジックを実行することができる。

ビジネスロジックをフレームワークが提供するクラスを継承して実装することで、URLで実行クラスを指定し実行する。

リクエストとレスポンスはともにJSONで送受信し、こちらもフレームワークが提供するクラスを継承し、JSONに変換できる形式で実装する。

URLに指定するクラス名はパッケージ付で指定するが、予めフレームワークプロパティにプレフィックスを指定することで短縮することが可能。

URL：

http://APサーバー/コンテキストパス/fw/rest/app/実行クラス名

#### ＜FWプロパティ（設定例）＞

[kvs]をプレフィックス（短縮名）として定義した例。

# RESTパスマッピング  
fw.rest.kvs=jp.cstudio.handywedge.test.app.rest.RESTKeyValueStore

#### ＜実装方法＞

ビジネスロジック、リクエスト、レスポンスともフレームワークが提供するクラスを継承する形で実装する。

##### ビジネスロジック

ビジネスロジッククラスはFWRESTControllerを継承する。メソッドのアノテーションで変換するクラスを指定する。

基本的にアプリケーション例外はスローしないように実装し、エラー応答はエラーコードで表現する。例外がスローされた場合はフレームワークでキャッチし、エラーコード-9000台を設定してクライアントに応答する。

**public** **class** SampleBatch **extends** FWRESTController {

@Inject

**private** Logger logger;

@Inject

**private** SampleBatchService service;

@Override

@FWRESTRequestClass(SampleBatchRequest.**class**)

**public** FWRESTResponse doPost(FWRESTRequest request) {

**try** {

logger.info("サンプルバッチを実行します。");

SampleBatchRequest **parameter** = (SampleBatchRequest) request;

logger.debug("parameter=" + parameter);

List<Data> **targetList**;

logger.info("対象データを取得します。");

**if** (parameter.getFrom() != **null** && parameter.getTo() != **null**) {

targetList = service.selectBetweenDate(parameter.getFrom(), parameter.getTo());

} **else** {

targetList = service.selectAll();

}

logger.debug("target count=" + targetList.size());

・・・

SampleBatchResponse **response** = **new** SampleBatchResponse();

response.setReturn\_cd(0);

response.setReturn\_msg("success");

SampleBatchResponseParameter **para** = **new** SampleBatchResponseParameter();

para.setCount(count);

response.setParameter(para);

logger.info("対象データの処理が終了しました。件数=" + count);

logger.info("サンプルバッチを終了します。");

**return** response;

} **catch** (Exception **e**) {

logger.error("バッチ実行中にエラーが発生しました。", e);

SampleBatchResponse **response** = **new** SampleBatchResponse();

response.setReturn\_cd(-1);

response.setReturn\_msg(e.getMessage());

**return** response;

}

}

}

##### リクエストクラス

リクエストクラスはFWRESTRequestを継承する。

必要があればJacksonライブラリのAPIを参照してアノテーションで装飾する。

https://github.com/FasterXML/jackson

**public** **class** SampleBatchRequest **extends** FWRESTRequest {

**private** String from;

**private** String to;

**public** String getFrom() {

**return** from;

}

**public** **void** setFrom(String from) {

**this**.from = from;

}

**public** String getTo() {

**return** to;

}

**public** **void** setTo(String to) {

**this**.to = to;

}

@Override

**public** String toString() {

StringBuilder **builder** = **new** StringBuilder();

builder.append("SampleBatchRequest [from=").append(from).append(", to=").append(to).append("]");

**return** builder.toString();

}

}

##### レスポンスクラス

レスポンスクラスはFWRESTResponseを継承する。

リターンコードとリターンメッセージの2つのプロパティはフレームワークで規定し、リターンコードは必ず設定する。

負の値はエラーを示し、-9000～-9999についてはフレームワークの予約番号とする。

**public** **abstract** **class** CommonResponse<P> **extends** FWRESTResponse {

@JsonProperty("PARA")

**public** **abstract** P getParameter();

@JsonProperty("PARA")

**public** **abstract** **void** setParameter(P parameter);

}

**public** **class** SampleBatchResponse **extends** CommonResponse<SampleBatchResponseParameter> {

**private** SampleBatchResponseParameter para;

@Override

**public** SampleBatchResponseParameter getParameter() {

**return** para;

}

@Override

**public** **void** setParameter(SampleBatchResponseParameter parameter) {

**this**.para = parameter;

}

@Override

**public** String toString() {

StringBuilder **builder** = **new** StringBuilder();

builder.append("SampleBatchResponse [para=").append(para).append(", getReturn\_cd()=").append(getReturn\_cd())

.append(", getReturn\_msg()=").append(getReturn\_msg()).append("]");

**return** builder.toString();

}

}

**public** **class** SampleBatchResponseParameter {

**private** **int** count;

**public** **int** getCount() {

**return** count;

}

**public** **void** setCount(**int** count) {

**this**.count = count;

}

@Override

**public** String toString() {

StringBuilder **builder** = **new** StringBuilder();

builder.append("SampleBatchResponseParameter [count=").append(count).append("]");

**return** builder.toString();

}

}

上記のようなレスポンスクラスを実装した場合、JSONは以下のようになる。

{"RETURN\_CD":0,"RETURN\_MSG":"success","PARA":{"count":1752}}

#### ＜設定＞

web.xmlにREST用のフィルター設定が必要。

<filter-mapping>

<filter-name>handywedge\_rest\_filter</filter-name>

<url-pattern>/fw/rest/\*</url-pattern>

</filter-mapping>

### メール送信機能

メールを送信する場合は、フレームワークが提供するメール送信機能を使う。

当機能を使用するには、予めフレームワークのプロパティにSMTPに関する設定が必要となる。

#### ＜使用方法＞

FWMailTransportインターフェイスの変数を定義し、＠Injectアノテーションを付加してオブジェクトを注入する。

@Inject  
 **private** FWMailTransport mail;

FWMailMessageのインスタンスを生成して値をセットし、FWMailTransportで送信する。

FWMailMessage mailMessage = **new** FWMailMessage();  
 mailMessage.setToAddress(**new** String[] { "aaa@bbb.co.jp" }); // TO  
 mailMessage.setFromAddress("ccc@ddd.co.jp"); // FROM  
 mailMessage.setCcAddress(**new** String[] { "eee@fff.co.jp" }); // CC  
 mailMessage.setSubject("Is this Test?"); // 件名  
 mailMessage.setCharacterEncoding(FWMailCharacterEncoding.***ISO\_2022\_JP***); // 文字コード。デフォルトでUTF-8に設定  
 mailMessage.setPriority(FWMaiPriority.***HIGH***); // 重要度。デフォルトはNormalで設定済。  
 mailMessage.setHtmlFlg(**true**); // HTML形式で送信。 デフォルトはTextで送信  
 mailMessage.setBody("This is test mail."); // 本文  
 **try** {  
 mail.send(mailMessage);  
 } **catch** (FWException e) {  
 logger.error("mail api error.", e);  
 }

#### ＜FWプロパティ（設定例）＞

フレームワークプロパティに設定が必要。

# SMTPサーバー #必須  
fw.mail.host=smtp.gmail.com  
  
# SMTPポート #def=25  
fw.mail.port=587  
  
# 添付ファイル上限値（合計） #def=0=無制限  
fw.mail.max\_filesize=0  
  
# STARTTLS拡張方式を有効化 #def=false  
fw.mail.starttls.enable=true  
  
# STARTTLSが必須か #def=false  
fw.mail.starttls.required=true  
  
# SMTPユーザー #必須  
fw.mail.user=xxxxx@cstudio.jp  
  
# SMTPパスワード #必須  
fw.mail.password=password

## 認証機能

### ログイン機能

フレームワークは、事前に登録済みのユーザー情報（ID、パスワード）を用い認証を行う機能を提供する。

また、認証後はフレームワークにてユーザー情報を保有し、ログイン中のユーザー情報を提供する。

※ユーザー情報については「(オ)ユーザー管理機能 ①ユーザー情報」参照。

#### ＜使用方法＞

FWLoginManagerインターフェイスの変数を定義し、＠Injectアノテーションを付加してオブジェクトを注入する。

@Inject  
 **private** FWLoginManager loginMgr;

ログイン処理の実装例

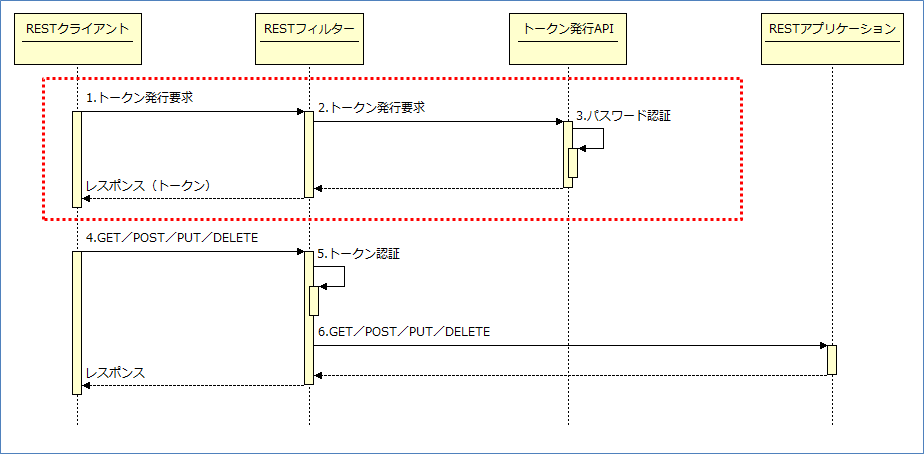
**boolean** **auth** = loginMgr.login(id, password); // 認証された場合はtrueを返し、同時にセッションのユーザー情報に値を設定  
 **if** (auth) {  
 // 認証OK時の処理  
 } **else** {  
 // 認証NG時の処理  
 }

ログアウト処理の実装例

loginMgr.logout(); // セッションからユーザー情報を削除

### トークン発行

フレームワークは、パスワード認証にてREST APIを利用するためのトークンを発行する機能を提供する。（下図赤枠部）



リソースURI（RESTエンドポイント）は以下の通り。JSON形式POSTリクエストにより実行する。

※本機能を使用するためのパラメータ設定は別紙「FWコンフィグガイド」参照。

RESTエンドポイント：

[http://*（APサーバー）*/](http://（APサーバー）/)*＜コンテキストパス＞*/fw/rest/api/token/pub

### トークン削除

フレームワークは、「②トークン発行」で発行したトークンを削除する機能を提供する。

リソースURI（RESTエンドポイント）は以下の通り。JSON形式DELETEリクエストにより実行する。

※本機能を使用するためのパラメータ設定は別紙「FWコンフィグガイド」参照。

RESTエンドポイント：

[http://*（APサーバー）*/](http://（APサーバー）/)*＜コンテキストパス＞*/fw/rest/api/token/delete

### トークン有効性チェック

フレームワークは、「②トークン発行」で発行したトークンの有効性をチェックする機能を提供する。

リソースURI（RESTエンドポイント）は以下の通り。JSON形式GETリクエストにより実行する。

※本機能を使用するためのパラメータ設定は別紙「FWコンフィグガイド」参照。

RESTエンドポイント：

[http://*（APサーバー）*/](http://（APサーバー）/)*＜コンテキストパス＞*/fw/rest/api/token/validate

## ユーザー管理機能

### ユーザー情報

フレームワークでは、基本的なユーザー属性を保有管理する機能を提供する。

以下にフレームワークが管理・提供する情報項目を示す。

※本情報はログイン中のユーザー情報としてコンテキスト情報機能にて取得可能な項目。

　コンテキスト情報機能については「(ア) アプリケーション情報 ①コンテキスト情報機能」参照。

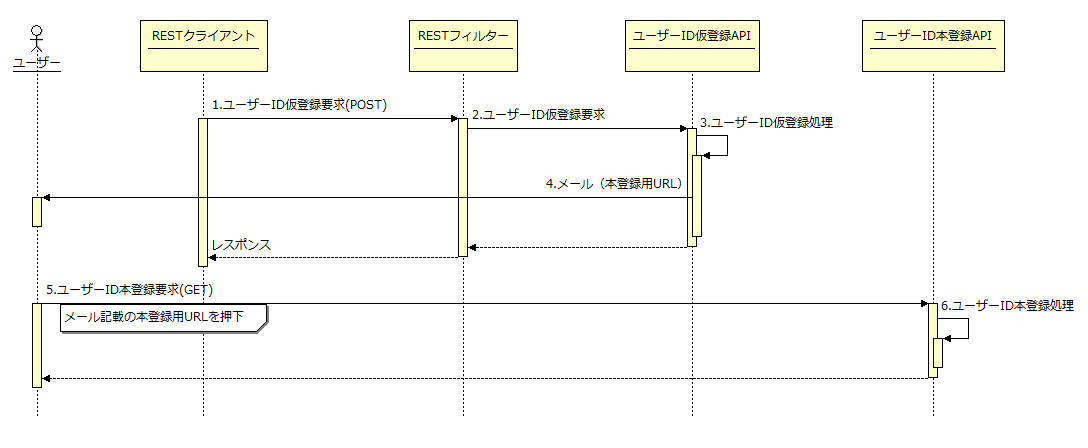
#### ＜ユーザー情報＞

|  |  |
| --- | --- |
| **項目名** | **内容** |
| ユーザーID | システムユーザーを識別する一意なID |
| ユーザー名 | 上記項目の名前情報 |
| ロケール情報 | 通常はDBに登録されているロケール情報が設定される（アプリケーションで上書き可能） |
| 最終ログイン時間 | 最終ログインの時刻（ログイン後に当該項目を取得した場合、今回のログイン時刻が取得可） |
| 前回ログイン時間 | 前回（１つ前）のログインした時刻 |
| ロールコード | ユーザーに設定されているロールを識別する一意なID |
| ロール名 | 上記項目の名前情報 |

### ユーザーID登録

フレームワークは、ユーザーID登録、及びユーザーID仮登録機能を提供する。また、仮登録時はユーザーに本登録用URLをメール通知する。

以下にユーザーID仮登録時のシーケンスを示す。



リソースURI（RESTエンドポイント）は以下の通り。JSON形式POSTリクエストにより実行する。

※本機能を使用するためのパラメータ設定は別紙「FWコンフィグガイド」参照。

RESTエンドポイント：

[http://*（APサーバー）*/](http://（APサーバー）/)*＜コンテキストパス＞*/fw/rest/api/user

### ユーザーID本登録

フレームワークは、「②ユーザーID登録」で仮登録したユーザーIDを本登録する機能を提供する。

リソースURI（RESTエンドポイント）は以下の通り。GETリクエスト（ユーザーID仮登録にて発行されたメール本文に記載の本登録用URLリンク押下）により実行する。

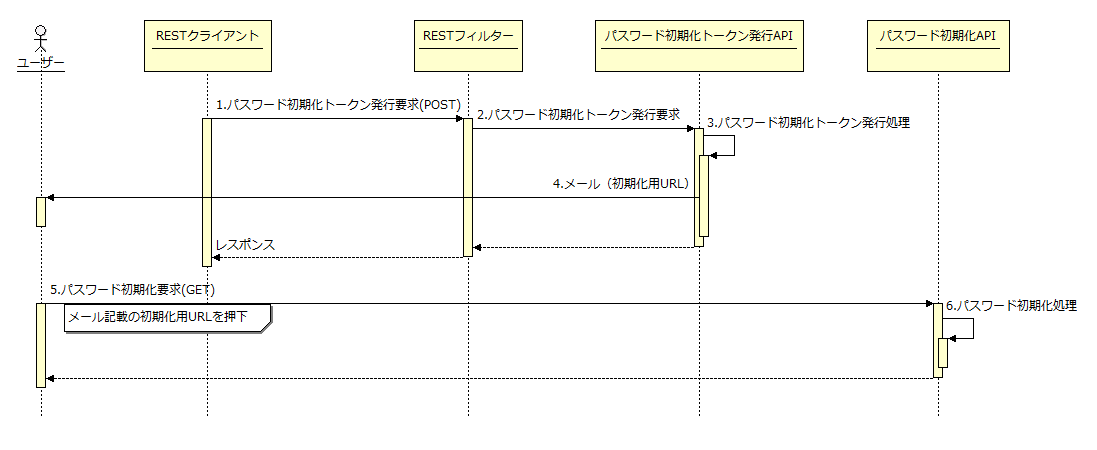
※本機能を使用するためのパラメータ設定は別紙「FWコンフィグガイド」参照。

RESTエンドポイント：

[http://*（APサーバー）*/](http://（APサーバー）/)*＜コンテキストパス＞*/fw/rest/api/user/actual

### パスワード初期化トークン通知

XXXXX



作成中

### パスワード初期化

### パスワード変更

### ロール

作成中

## ロギング機能

ビジネスロジックの入口・出口、SQLの実行、例外の発生など主要な場所ではフレームワークが自動的にログを出力するが、アプリケーションは必要に応じて任意でログを出力することができる。  
ログ出力は拡張性を持たせるためにロギングファサードslf4jを使って出力するため、アプリケーション開発者はロガーの実装クラスを意識せずにログを出力する。  
ログの出力制御はlog4j2.xmlなどで設定を行う。

#### ＜ログレベル＞

出力するログのレベルは以下の通りとする。

|  |  |  |
| --- | --- | --- |
| **レベル** | **内容** | **出力内容の基準** |
| error | エラー | 継続不可能な問題が発生した場合、その内容と問題を解決するために必要な情報を出力する。 |
| warn | 警告 | 問題が発生したが継続可能な場合、その内容と問題を解決するために必要な情報を出力する。 |
| info | 情報 | 設定ファイル等を読み込んだときの内容や、特にログとして出力しておきたい情報を出力する。 リクエストおよびビジネスロジックメソッドの開始、終了の情報はフレームワークで自動的にログ出力する。 |
| debug | デバッグ | アプリケーション開発者がデバッグに必要な情報を出力する。（外部クラスから呼ばれるメソッドの引数の内容や戻り値等） |
| trace | トレース | より詳細なデバッグ情報を出力する。 |

#### ＜ログ出力カテゴリ＞

出力内容により以下のカテゴリに分かれる。

|  |  |
| --- | --- |
| **カテゴリ** | **カテゴリ内容** |
| FW | フレームワークより出力するログ |
| AP | 業務アプリケーションより出力するログ（Logger#info()、debug()など、いずれのメソッドを利用しても自動的に割り当てられる） |
| PERF | 処理時間計測ログ |
| RESP | リクエスト処理時間計測ログ（フレームワークより出力するログ） |
| JDBC | JDBC関連ログ |

### SQLトレース

トランザクションの開始・終了、コミット・ロールバック、例外発生についてはフレームワークが自動的にログ出力。

また、JDBCドライバが発行するSQLは、log4jdbc（JDBCドライバをラップしログ出力するライブラリ）を使用しログ出力する。

以下にSQLトレースについてロガー log4j2を使用した設定例を示す。

#### ＜log4j2.xml（設定例）＞

<Logger name="jdbc.connection" level="warn" additivity="false">  
 <AppenderRef ref="${app\_id}\_jdbc" />  
</Logger>  
  
<Logger name="jdbc.sqltiming" level="debug" additivity="false">  
 <AppenderRef ref="${app\_id}\_jdbc" />  
</Logger>  
  
<Logger name="jdbc.sqlonly" level="error" additivity="false">  
 <AppenderRef ref="${app\_id}\_jdbc" />  
</Logger>  
  
<Logger name="jdbc.audit" level="warn" additivity="false">  
 <AppenderRef ref="${app\_id}\_jdbc" />  
</Logger>  
  
<Logger name="jdbc.resultset" level="warn" additivity="false">  
 <AppenderRef ref="${app\_id}\_jdbc" />  
</Logger>

#### ＜出力イメージ＞

出力フォーマットは同じく[ log4j2.xml ]にて設定する。出力カテゴリは「JDBC」。

2017-08-16 09:56:13.019 DEBUG [*userid*] [JDBC] *requestId* jdbc.sqltiming org.apache.tomcat.dbcp.dbcp2.DelegatingStatement.executeQuery(DelegatingStatement.java:206)   
3. SELECT \* FROM xxxxx ORDER BY id ASC   
 {executed in 2 msec}

### デバッグログ

アプリケーションは必要に応じて任意でログを出力することができる。使用方法を以下に示す。

#### ＜使用方法＞

Loggerインターフェイスの変数を定義し、＠Injectアノテーションを付加してオブジェクトを注入する。

@Inject  
 **private** FWLogger logger;   
  
 **public String** method(String key) {  
 String value = msg.get(key);  
 logger.debug("method() key={}, value={}", key, value);  
 }

#### ＜出力イメージ＞

出力カテゴリは「AP」となる。（Logger#info()、debug()など、いずれのメソッドを利用しても自動的に割り当てられる）

2017-08-16 15:15:08.244 DEBUG [*userid*] [AP] *requestId* *呼出し元クラス* method() key=XXXXX, value=XXXXX

### 性能トレース

アプリケーションは必要に応じて任意で処理に掛かる性能トレースログを出力することができる。使用方法を以下に示す。

#### ＜使用方法＞

ある処理（メソッド）における処理時間を出力する場合の例。

@Inject  
 **private** FWLogger logger;   
  
 **public void** methodA() {  
 **long** **startTime** = logger.perfStart("methodA"); **// 引数にメソッド名を指定。返り値としてlong値を受ける**  
 ：  
 （業務処理）  
 ：  
 logger.perfEnd ("methodA", startTime); **// long値を引数に指定する**  
 }

#### ＜出力イメージ＞

＜使用方法＞で示したコードに応じたログ出力例。Endログに経過時間を出力。出力カテゴリは「PERF」となる。

2017-08-16 15:14:54.490 INFO [*userid*] [PERF] *requestId* *呼出し元クラス* methodA() start.   
：  
2017-08-16 15:14:54.493 INFO [*userid*] [PERF] *requestId* *呼出し元クラス* methodA() end. ElapsedTime[**3**]ms

## ワークフロー機能

作成中

## ライブラリ機能

# アプリケーション開発環境

## 開発環境

※別紙「handywedge開発環境構築ガイド」参照。