目次

改訂情報

変更年月日	変更内容
2012-12-21	初版
2013-04-01	第２版　下記を追加・変更しました ■ 「*リソース拡張ガイド* 」に「*リソース表示可否判断クラスの設定* 」「*リソース情報をキャッシュする* 」を追加
2013-10-01	第３版　下記を変更しました ■ 「*リソース表示可否判断クラスの設定* 」に認可管理者による権限判断についての備考を追加 ■ 「*imartタグ、カスタムタグによる認可要求* 」の説明を修正
2014-01-01	第４版　下記を変更しました ■ 「*サンプルの資材一覧* 」で提供されていないサンプルに関する記述を削除 ■ 「*ChangeableNameSubjectTypeとは* 」の説明を追加 ■ 「*検索画面* 」のプラグイン実装で `action` 属性に渡される引数について説明を追加
2014-04-01	第５版　下記を変更しました ■ パブリックストレージのルートを示すパスの表記を「%PUBLIC_STORAGE_PATH%」に統一しました。 ■ アプリケーションサーバ側のルートを示すパスの表記を「%CONTEXT_PATH%」に統一しました。

はじめに

本書の目的

本書では認可機構の仕組の詳細について説明します。

説明範囲は以下のとおりです。

- 認可機構に独自のリソースを定義し、連携する方法
- 認可機構で独自のサブジェクトを使用する方法

対象読者

本書では次の利用者を対象としています。

- intra-mart Accel Platformを理解している
- Java を理解しており、認可機構を利用しようとする開発者

本書の構成

- *サンプルについて*
  本書では例としてサンプルアプリケーションをもとに、コーディングの例などを説明しています。ここではそのサンプルの仕様や、サンプルを動作させるために必要な手順について説明します。
  本書を読むことで実現できることの概要を把握したい場合もこちらを参照してください。
- *サブジェクト拡張ガイド*

  intra-mart Accel Platform で提供しているサブジェクト以外のサブジェクトを追加したい開発者向けです。

  この追加を行うことでIM共通マスタだけでなくアプリケーション独自のユーザ情報を使った認可設定を行うことができるようになります。

- *リソース拡張ガイド*

  アプリケーションの認可処理を認可機構と連携して行おうとする開発者向けです。

  アプリケーションの認可対象をリソースとして追加するために、独自のアクションを定義する方法やリソースの構成の方法、アプリケーションからの認可要求の仕方などについて説明しています。アプリケーションの管理機能内で認可設定画面を部分的に呼び出す方法についても解説しています。

サンプルについて

本書では簡単なアプリケーションを例に認可機構の拡張を作成したり連携処理の組み込みを説明しています。このサンプルはIM共通マスタモジュールにサンプルとして含まれており、デプロイするには IM-Juggling で、モジュールIM共通マスタを含め、なおかつサンプルを含める設定で war を作成してください。

ただし war をデプロイしただけではこのサンプルは動作する状態になっていません。後述の手順に従ってサンプルが動作するように設定する必要があります。

本書では解説に必要な部分のソースコードを引用しているためすべてのソースコードを掲載しているわけではありません。必要に応じてサンプルアプリケーションのソースも参照してください。

サンプルアプリケーションの仕様

サンプルアプリケーションの簡単な仕様を紹介します。

このアプリケーションは以下の機能で構成されています。

- 登録されたデータを一覧表示する画面
- データの登録機能
- データの詳細を表示する機能
- データを更新する機能
- データを削除する機能

これに加えて、このアプリケーションで取り扱うデータに対して認可設定を行える認可設定画面部品を使用していますが、 アプリケーションの機能としては単純にレコードを追加・参照・変更・削除できるだけのものです。

データの一覧表示画面

このアプリケーションに対して、認可機構と連携する仕組みが組み込んであり、以下のような権限制御を行えるようになっています。

- 認可設定画面を使う権限がある人の場合、認可設定画面を開いてデータに対する権限を設定できます。 データを登録・削除すると認可機構のリソースも登録・削除されます
- データに対する参照権限がある人のみデータの参照リンクが表示されます。
- データに対する更新権限がある人のみデータの変更ボタンが表示されます。
- データに対する削除権限がある人のみデータの削除ボタンが表示されます。

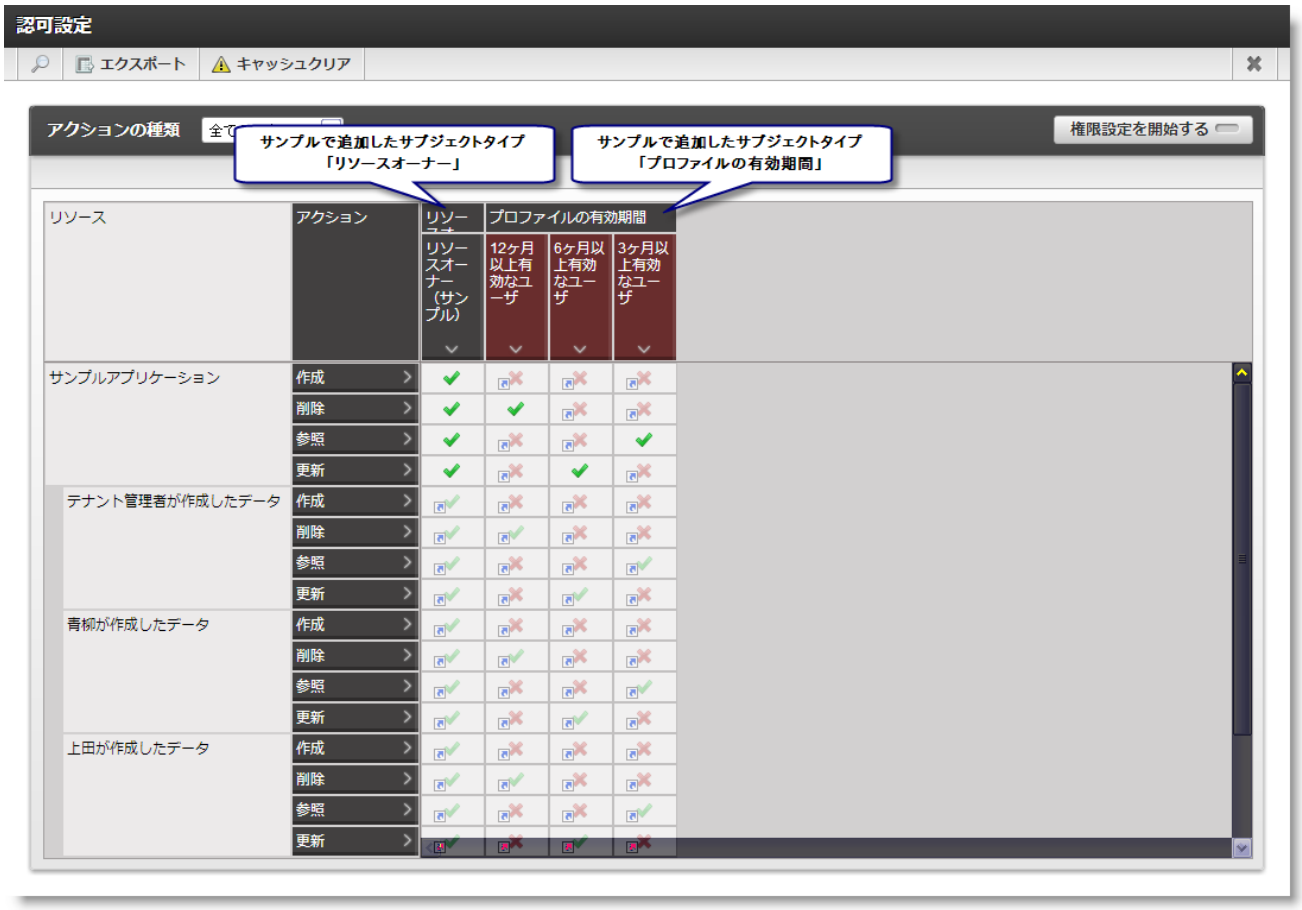
例えば、権限によっては上記の画面は以下のように見えます。

権限が制限されているユーザの場合の表示

このガイドで認可機構に対して拡張する内容

このガイドでは上記のアプリケーションでの権限制御のために、認可機構に以下のような拡張を行うことを例としています。

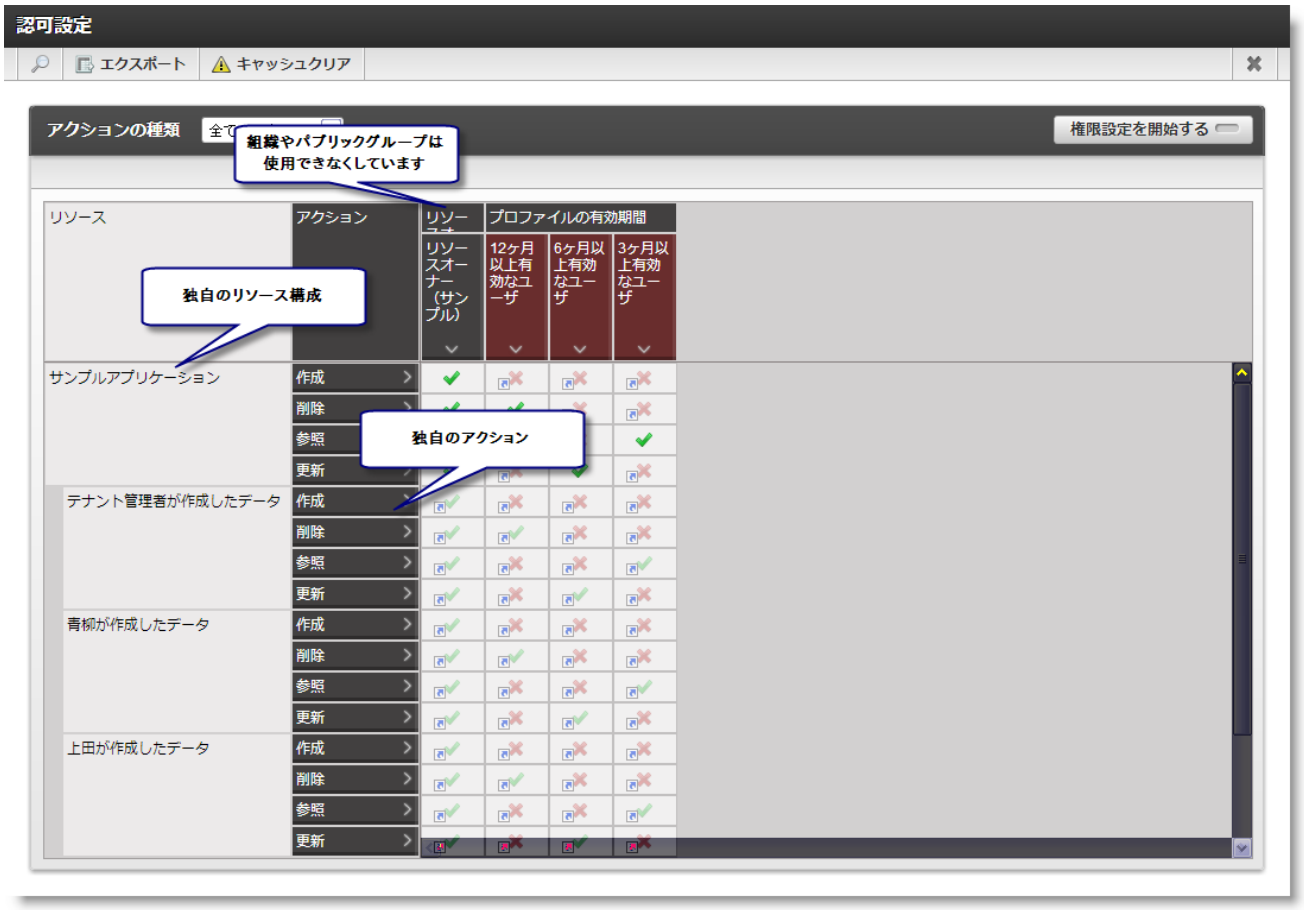
- サブジェクトの拡張
  intra-mart Accel Platform で標準的にサブジェクトとして設定可能なものは「ロール」や「組織」といった標準的に提供されている機能で管理されている情報のみです。サブジェクトの拡張を行う事でこれら以外の情報をサブジェクトとして扱うことができるようになります。
  このガイドでは「ユーザの勤続月数（ユーザのプロファイルの有効期間）」と、サンプルアプリケーションのデータについて「データの所有者（リソースオーナー）」という情報をサブジェクトとして扱う例をもとに解説します。



拡張されたサブジェクトの例

- リソースの拡張
  リソースはリソースタイプに属しており、属しているリソースタイプによって取ることのできるアクションが決まります。intra-mart Accel Platform で標準的に提供されているリソースタイプは限られておりアプリケーションで使用したいアクションにそぐわない場合がありますが、リソースタイプを追加する事で独自のアクションが利用できるようになります。また、独自のリソース構成を定義することでそのアプリケーションの管理するデータに限った認可設定画面を用意することができます。
  このガイドでは、サンプルアプリケーションが作成するデータに対して認可設定を行うために、新しいリソースタイプを作成

し、データに対して「作成」「参照」「更新」「削除」のアクションを定義する例を解説します。また認可設定画面において、サンプルアプリケーションが作成するデータに対して上記サブジェクト拡張で解説する「ユーザの勤続月数（ユーザのプロファイルの有効期間）」および「データの所有者（リソースオーナー）」というサブジェクトのみを認可設定に利用できるようにする例も紹介します。



拡張されたリソース/アクションの例

サンプルのセットアップ手順

このサンプルは動作させるために手動でのセットアップが必要になります。 以下の手順に従ってサンプル資材をセットアップしてください。

1. IM-Juggling で war を作成する。
   warを作成する際、モジュール構成に **「IM共通マスタ 認可連携モジュール」を含め、** Jugglingビルドウィザードの基本設定で **「サンプルを含める」のチェックボックスにチェックを入れてください。**

2. テナント環境セットアップの実行
   intra-mart Accel Platform をインストールして起動したら、まずシステム管理者でログインしてテナント環境セットアップを実行してください。

3. サンプルデータセットアップの実行
   テナント環境セットアップが完了したら、続けてサンプルデータセットアップを実行してください。

4. サンプルで使用するテーブルの作成
   パブリックストレージにある以下のSQLファイルを実行してデータベース上にテーブルを作成してください。

   `%PUBLIC_STORAGE_PATH%/sample/im_master_subjecttypes/sample-authz-ddl.sql`

5. リソースタイプの有効化
   以下の設定ファイルの内容がコメントアウトされていますのでコメントを外します。

   `%CONTEXT_PATH%/WEB-INF/conf/authz-resource-type-config/authz-ext-sample.xml`

6. サブジェクトタイプの有効化
   以下の設定ファイルの内容がコメントアウトされていますのでコメントを外します。

%CONTEXT_PATH%/WEB-INF/conf/authz-subject-type-config/authz-ext-sample.xml

7. サブジェクトリゾルバの有効化
   以下の2つの設定ファイルの内容がコメントアウトされていますのでコメントを外します。

   %CONTEXT_PATH%/WEB-INF/conf/declared-subject-resolvers/authz-ext-sample.xml

   %CONTEXT_PATH%/WEB-INF/conf/ondemand-subject-resolvers/authz-ext-sample.xml

8. 認可サブジェクトコンテキストのデコレータの有効化
   以下の設定ファイルの内容がコメントアウトされていますのでコメントを外します。

   %CONTEXT_PATH%/WEB-INF/conf/context-config/authz-ext-sample.xml

9. ポリシー部分編集定義の有効化
   以下の設定ファイルの内容がコメントアウトされていますのでコメントを外します。

   %CONTEXT_PATH%/WEB-INF/conf/authz-partial-policy-edit-config/authz-ext-sample.xml

10. 認可リソースのインポート
    ジョブネットを使用してパブリックストレージの以下の認可リソースグループのインポートファイルをインポートします。

    %PUBLIC_STORAGE_PATH%/sample/im_master_subjecttypes/sample-authz-resource-group.xml

    ジョブネットの操作については「テナント管理者操作ガイド」を参照してください。

11. システム再起動
    上記までの操作が完了したら、アプリケーションサーバの再起動を行ってください。

アプリケーションサーバが起動したら、Webブラウザからログインして

    http://<HOST>:<PORT>/<CONTEXT_PATH>/sample/authz_ext_sample/list

にアクセスして、サンプルアプリケーションが表示されることを確認してください。

サンプルの資材一覧

サンプルの資材は java のソース以外は war に含まれています。 war を展開した状態では以下の場所に配置されています。

```
%CONTEXT_PATH%/WEB-INF/
 ├─conf
 │  ├─authz-partial-policy-edit-config （サンプルとして作成するポリシー部分編集設定）
 │  │      authz-ext-sample.xml
 │  │
 │  ├─authz-resource-type-config （サンプルとして作成するリソースタイプの設定）
 │  │      authz-ext-sample.xml
 │  │
 │  ├─authz-subject-type-config （サンプルとして作成するサブジェクトタイプの設定）
 │  │      authz-ext-sample.xml
 │  │
 │  ├─context-config （サンプルとして作成するコンテキストデコレータの設定）
 │  │      authz-ext-sample.xml
 │  │
 │  ├─declared-subject-resolvers （サンプルとして作成する declared-subject-resolver の設定）
 │  │      authz-ext-sample.xml
 │  │
 │  ├─message
 │  │  └─sample
 │  │      └─authz_ext （ポリシー部分編集設定のキャプション）
 │  │              caption.properties
 │  │              caption_en.properties
 │  │              caption_ja.properties
 │  │              caption_zh_CN.properties
 │  │
 │  ├─ondemand-subject-resolvers （サンプルとして作成する ondemand-subject-resolver の設定）
 │  │      authz-ext-sample.xml
 │  │
 │  └─routing-jssp-config （サンプルアプリケーションのルーティング設定）
 │          authz-ext-sample.xml
 │
 ├─jssp
 │  └─src
 │      └─sample
 │          └─im_authz_ext_sample （サンプルアプリケーションの実装）
 │                  add.js
 │                  del.js
 │                  get.js
 │                  header.html
 │                  keep_alive_button.html
 │                  list.html
 │                  list.js
 │                  update.js
 │
 ├─lib （サンプルの jar ファイル）
 │  └─im_master_subjecttypes-X.X.X-sample.jar
 │
 ├─plugin （サンプルのサブジェクトタイプを認可設定画面で選択可能にするプラグイン設定）
 │  └─jp.co.intra_mart.sample.authz.subject.search
 │          plugin.xml
 │
 ├─resources
 │  └─sample
 │      └─sql （DAOが使用する外部SQLファイル）
 │              DataDAO_getData.sql
 │              ProfileKeepAliveSIDDAO_getSubjectId.sql
 │              ResourceOwnerSIDDAO_getSubjectId.sql
 │
 └─storage
   └─public ※ ここはパブリックストレージのディレクトリに読み替えてください。
       └─sample
           └─im_master_subjecttypes
                   sample-authz-ddl.sql
                   sample-authz-resource-group.xml
```

サンプルのjavaソースは 最新情報ダウンロードページ(iAP) からダウンロードできます。

http://www.intra-mart.jp/download/product/iap/index.html

```
java
 └─sample
    ├─dao （サンプルアプリケーションやサブジェクトタイプが使用するDAO）
    │      DataDAO.java
    │      Data.java
    │      ProfileKeepAliveSIDDAO.java
    │      ProfileKeepAliveSID.java
    │      ResourceOwnerSIDDAO.java
    │      ResourceOwnerSID.java
    │
    ├─resource
    │  ├─link （サンプルアプリケーションが認可と連携するユーティリティ）
    │  │      Linkage.java
    │  │
    │  └─type （サンプルのリソースタイプ）
    │     │  SampleDataResourceType.java
    │     │  SampleDataModel.java
    │     │
    │     └─action （サンプルのリソースタイプで定義するアクション）
    │           C.java
    │           D.java
    │           R.java
    │           U.java
    │
    └─subject
       ├─decorator （サンプルのコンテキストデコレータ）
       │      ProfileKeepAliveContextDecorator.java
       │      ProfileKeepAliveSubjectIdContainer.java
       │
       ├─resolver （サンプルのサブジェクトリゾルバ２種）
       │      ProfileKeepAliveSubjectResolver.java
       │      ResourceOwnerSubjectResolver.java
       │
       └─type （サンプルのサブジェクトタイプ2種）
              ProfileKeepAliveSubjectType.java
              ProfileKeepAlive.java
              SampleResourceOwnerSubjectType.java
              SampleResourceOwner.java
              SubjectTypeRuntimeException.java
```

サブジェクト拡張ガイド

項目

サブジェクト拡張とは

intra-mart Accel Platform では標準的に以下のようなサブジェクトを提供しています。

- 認証状態
- ロール
- ユーザ
- 組織、役職
- パブリックグループ、役割

これによって特定の組織に所属するユーザに対して権限を与えるといったことが可能になっています。

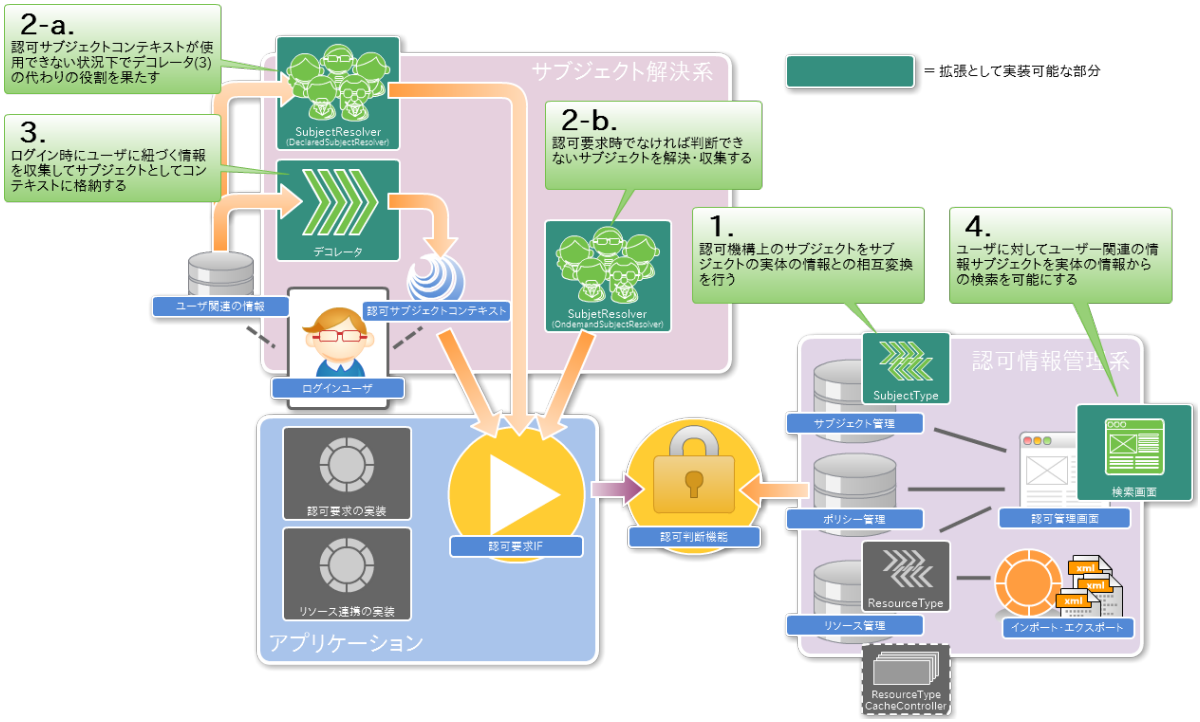
上記以外の解釈でユーザを分類して権限管理をしたい場合にはサブジェクトの拡張機能を作成することで、認可機構に追加して利用できるようになります。

ここではサブジェクト拡張を作成するための手順を解説します。

おおまかな流れ

以下の流れで手順を説明します。

1. *サブジェクトの定義* を行います。
   サブジェクトの特徴を表現するためにサブジェクトの定義を行います。すでにあるマスタデータなどを連携したい場合は単純にそのデータを表すもの（キー情報など）で良いでしょう。そうでない場合、サブジェクトを表すのに必要な要素を定義します。
   この時に、ユーザに対して自明なサブジェクトか、そうでないかを明確にしておきます。
2. 認可機構へのプラグイン実装を追加します。
   最大で以下の4種の実装が必要です。



- *SubjectType* (図中 1)
  サブジェクトのモデルと認可のサブジェクト情報の相互変換、シリアライズを担当します。
- *SubjectResolver* (図中 2-a および 2-b)
  ユーザが何者であるかを判断する処理です。これも新たに追加するサブジェクトモデルの情報を取り扱うための実装を追加します。
  取り扱う情報の特性によって2種の実装の内どちらか (図中 2-a および 2-b) を選択します。
- *認可サブジェクトコンテキストのデコレータ* (図中 3)
  認可サブジェクトコンテキストはユーザが何者であるかをコンテキストとして保持していますが、このコンテキストに新たに追加するサブジェクトモデルの情報を取り扱うためのデコレータを追加します。
  SubjectResolver の実装で 2-b を選択している場合はこれは必要ありません。
- *検索画面* (図中 4)
  認可設定画面でユーザがサブジェクトとして選択できるようにするための画面拡張を追加します。

追加しようとしているのがサブジェクトがユーザに対して自明なサブジェクトではない場合、認可サブジェクトコンテキストのデコレータは必要ありません。

3. *ポリシー部分編集設定* を編集してリソースグループセット（特定のルート配下のリソースツリー）毎に利用するサブジェクトタイプを設定します。
   サブジェクトタイプは追加しても認可設定画面で使えるようにはなりません。サブジェクトタイプをどのリソース群に対して有効にするかの設定を変更する必要があります。

以降、順に説明します。

サブジェクトの定義

サブジェクトの特徴を表現するためにサブジェクトの定義を行います。今回例として扱うサンプルアプリケーションでは以下の２種類の権限の扱いを考えています。

1. ユーザの勤続の長さによって権限を変える
2. データの作成者か否かによって権限を変える

前者はIM共通マスタのユーザプロファイルを見て、プロファイルが有効になってから当日までの期間の長さによって異なるサブジェクトとして解釈することにします。このサブジェクトを追加することによって認可設定画面で「プロファイルがXXカ月以上有効」といったサブジェクトを設定することができる事を目標とします。

後者はサンプルアプリケーションにおいて、データを登録する際にデータベースのレコードに作成者を記録しておき、ログインユーザが作成者と一致する場合そのデータの「オーナー」というサブジェクトとして解釈することとします。 これによって認可設定画面で「リソースオーナー」といったサブジェクトを設定できる事を目標とします。

ここで、ユーザに対して自明なサブジェクトになるかどうかを明確にしておく必要があります。

認可のサブジェクト解決のタイミングは2段階ありユーザのログイン時（厳密にはコンテキストの作成／切替のタイミング）と、認可判断時です。前者はログイン時に明確になっているサブジェクト情報（所属組織など）を解決し、コンテキストに保持します。後者はログイン時にはわからない、状況依存のようなサブジェクトを解決します。

「プロファイルがXXカ月以上有効」かどうかはユーザが誰であるかが判明すれば、ユーザに付随する情報から取得できる情報です。つまりログイン時に判断できます。 これに対して「リソースオーナー」はユーザが誰であるかだけでなく、どのデータについて判断しようとしているかが分からないと判断できません。

つまり

- 「プロファイルがXXカ月以上有効」：ユーザに対して自明。ログイン時に判断可能。
- 「リソースオーナー」：ユーザに対して自明ではない。認可要求時でなければ判断できない。

という事になります。

SubjectType

SubjectTypeとは

アクセス主体の定義実体と認可のサブジェクトの相互変換、およびシリアライズを担当します。

認可機構からサブジェクトに対して付与されるSubjectIDとアクセス主体の定義実体のキー情報とのマッピングを管理する責務があります。認可設定画面で新たなサブジェクトを選択して権限を設定しようとする場合、選択したサブジェクトにSubjectIDが割り当てられます。この時、SubjectTypeは実際に選択されたモデルとSubjectIDの対応を管理しなければなりません。

サブジェクトタイプは認可の持つ「AND」「OR」「NOT」といった複合の条件を考慮する必要はありません。この条件は認可機構側で考慮します。ただし、連携マスタ側でなければ判断できない条件はSubjectType側でサポートしなければなりません。例えば、IM共通マスタの組織情報などにおいて、下位組織といった条件は認可機構側では判断できないため、SubjectTypeがなんらかの情報で判断できるように情報を保持するようにします。

SubjectTypeで使用するモデルクラスの作成

まずサブジェクトタイプで使用するモデルクラスを作成します。既存のクラスで問題なければ使いまわしても構いません。例えば、デフォルトでインストールされるIM共通マスタのユーザサブジェクトの場合、IM共通マスタのAPIが提供している`jp.co.intra_mart.foundation.master.user.model.User` というモデルクラスをそのまま使用しています。同じくIM共通マスタの組織サブジェクトの場合、組織モデルの情報に加えて上位組織や下位組織を含めるか否かの情報も含めてサブジェクトタイプが管理する必要があるため、IM共通マスタのAPIが提供しているクラスと上位下位の条件を合わせて持つモデルクラスを新たに定義しています。

サンプルでは以下の2種類を作成しています。

- 「プロファイルがXXカ月以上有効」であるユーザを表すモデルクラス
- 「リソースオーナー」であることを示すモデルクラス

まず「プロファイルがXXカ月以上有効」なユーザのモデルクラスのソースコードを以下に示します。これは有効な月数だけがわかればよいので、intフィールドを一つを定義したクラスを作成します。

```
/**
 * ProfileKeepAlive クラス
 * @author INTRAMART
 * @version 8.0
 */
public class ProfileKeepAlive {

   int months;

   /**
    * months の Getter
    * @return monthsを返却
    */
   public int getMonths() {
      return months;
   }

   /**
    * months の Setter
    * @param months monthsの設定値
    */
   public void setMonths(final int months) {
      this.months = months;
   }

}
```

「リソースオーナー」であることを示すモデルクラスは情報を持つ必要は特にありません。サブジェクトIDを生成するために必要なIDだけをstaticフィールドとして定義しています。

```
/**
 * SampleResourceOwner クラス
 * @author INTRAMART
 * @version 8.0
 */
public class SampleResourceOwner {

   // リソースオーナーであることを示すID
   public static final String IDENTIFIER = "sample-resource-owner";

}
```

SubjectTypeの実装

SubjectType を実装するには以下のインタフェースを実装しなければなりません。

`jp.co.intra_mart.foundation.authz.model.subjects.SubjectType<T>`

このインタフェースはサブジェクトのモデルと認可のサブジェクト情報の相互変換やシリアライズを行うためのメソッドが定義されています。

すでに述べている通りサンプルでは次の2種のサブジェクトタイプを定義しています。

1. 「プロファイルが有効になってからXX カ月」といった設定ができるサブジェクトタイプ
   - ユーザのプロファイルが有効になってからの期間の長さでサブジェクトを変えるサブジェクトタイプです
   - サブジェクトタイプIDは `sample-prof-keep-alive`
   - この情報はログイン時に判定するものとします。（厳密な時刻までは判定しない）
2. サンプルアプリケーションのデータの作成者を「リソースオーナー」として解釈するサブジェクトタイプ
   - あるデータにアクセスしようとした際に、そのデータの登録ユーザを参照して、ログインユーザと同一であればログインユーザをリソースオーナーというサブジェクトとして解釈します。
   - サブジェクトタイプIDは `sample-resource-owner`
   - この情報はリソースにアクセスする時点にならなければわかりません。

以下は sample-prof-keep-alive の実装クラスです。上記の SubjectType インタフェースを実装しており、モデルクラスとして ProfileKeepAlive を使用しています。

完全修飾クラス名

sample.subject.type.ProfileKeepAliveSubjectType

```
/**
 * プロファイルの有効な期間が何か月かを表すサブジェクトタイプです。
 * @author INTRAMART
 * @version 8.0
 */
public class ProfileKeepAliveSubjectType implements SubjectType<ProfileKeepAlive> {

    private static final long serialVersionUID = -3646347367576852310L;

    public static final String LEADER = "months_for_"; //$NON-NLS-1$

    @Override
    public String createIdentifier(final Object... keys) {

        // keyとして受付可能なのはinteger または数値表現の文字１つとします。
        // そのままキーにしても問題ありませんが、保険としてプリフィックスをつけます。

        if (keys != null && keys.length == 1) {

            if (keys[0] instanceof Integer) {
                return LEADER.concat(String.valueOf(keys[0]));
            }

            if (keys[0] instanceof String && ((String) keys[0]).matches("[0-9]+")) {
                return LEADER.concat(String.valueOf(Integer.parseInt((String) keys[0])));
            }

        }

        throw new SubjectTypeRuntimeException();

    }

    @Override
    public String createIdentifier(final ProfileKeepAlive model) {

        // サブジェクトモデルをハッシュキーを生成するためのIDに変換します。
        if (model != null) {
            return LEADER.concat(String.valueOf(model.getMonths()));
        }

        throw new SubjectTypeRuntimeException();

    }

    @Override
    public I18nValue<String> getDisplayName() {

        // このサブジェクトタイプの名称です。
        final I18nValue<String> name = new I18nValue<String>();
        name.put(Locale.JAPANESE, "プロファイルの有効期間"); //$NON-NLS-1$
        return name;
    }

    @Override
    public String getSubjectTypeId() {
        // サブジェクトタイプのIDです。
        return "sample-prof-keep-alive";
    }

    @Override
    public void onCreateSubject(final Subject subject, final Object... keys) throws SubjectManagingException {
```

```
    // このメソッドは認可機構でサブジェクトを作成した際に呼び出されます。
    // データベースへ保管しておき、以後のマッピング時に参照できるようにしておきます
    final ProfileKeepAliveSID sid = new ProfileKeepAliveSID();
    sid.keepAlive = keys[0] instanceof Integer ? (Integer) keys[0] : Integer.parseInt(String.valueOf(keys[0]));
    sid.subjectId = subject.getSubjectId();

    final ProfileKeepAliveSIDDAO dao = DAOFactory.getTenantDatabaseDAO(ProfileKeepAliveSIDDAO.class);
    dao.insert(sid);

  }

  @Override
  public void onCreateSubject(final Subject subject, final ProfileKeepAlive model) throws SubjectManagingException {

    // このメソッドは認可機構でサブジェクトを作成した際に呼び出されます。
    // データベースへ保管しておき、以後のマッピング時に参照できるようにしておきます
    final ProfileKeepAliveSID sid = new ProfileKeepAliveSID();
    sid.keepAlive = model.getMonths();
    sid.subjectId = subject.getSubjectId();

    final ProfileKeepAliveSIDDAO dao = DAOFactory.getTenantDatabaseDAO(ProfileKeepAliveSIDDAO.class);
    dao.insert(sid);

  }

  @Override
  public void onRemoveSubject(final Subject subject) throws SubjectManagingException {

    // このメソッドは認可機構でサブジェクトを削除した際に呼び出されます。
    // 認可機構から削除されたので、テーブルからも削除します
    final ProfileKeepAliveSID sid = new ProfileKeepAliveSID();
    sid.subjectId = subject.getSubjectId();

    final ProfileKeepAliveSIDDAO dao = DAOFactory.getTenantDatabaseDAO(ProfileKeepAliveSIDDAO.class);
    dao.delete(sid);

  }

  @Override
  public List<Object> parseIdentifier(final String identifier) {

    // createIdentifier で作成する文字列のパース処理です。
    // キーのリストを返します。

    if (identifier == null || !identifier.startsWith(LEADER)) {
      throw new SubjectTypeRuntimeException();
    }

    final List<Object> keys = new ArrayList<Object>();

    // createIdentifier と逆の処理をします。
    keys.add(Integer.parseInt(identifier.substring(LEADER.length())));

    return keys;
  }

  @Override
  public I18nValue<String> resolveDisplayName(final Object... keys) {
    final I18nValue<String> name = new I18nValue<String>();
    name.put(Locale.JAPANESE, ((String) keys[0]).concat(" カ月以上有効なユーザ"));
    return name;
  }

  @Override
  public I18nValue<String> resolveDisplayName(final String subjectId) {

    // サブジェクトIDから表示名を取得して返します。
    final ProfileKeepAliveSIDDAO dao = DAOFactory.getTenantDatabaseDAO(ProfileKeepAliveSIDDAO.class);
    final ProfileKeepAliveSID sid = dao.find(subjectId);
```

```
        if (sid == null) {
            return null;
        }

        final I18nValue<String> name = new I18nValue<String>();
        name.put(Locale.JAPANESE, String.valueOf(sid.keepAlive).concat(" カ月以上有効なユーザ"));
        return name;
    }

    @Override
    public List<Object> resolveKeys(final String subjectId) {

        // サブジェクトIDからキーのリストを取得して返します。
        final ProfileKeepAliveSIDDAO dao = DAOFactory.getTenantDatabaseDAO(ProfileKeepAliveSIDDAO.class);
        final ProfileKeepAliveSID sid = dao.find(subjectId);

        if (sid == null) {
            return null;
        }

        final List<Object> result = new ArrayList<Object>();
        result.add(sid.keepAlive);
        return result;
    }

    @Override
    public ProfileKeepAlive resolveModel(final String subjectId) {

        // サブジェクトIDからサブジェクトモデルを作成して返します。
        final ProfileKeepAliveSIDDAO dao = DAOFactory.getTenantDatabaseDAO(ProfileKeepAliveSIDDAO.class);
        final ProfileKeepAliveSID sid = dao.find(subjectId);

        if (sid == null) {
            return null;
        }

        final ProfileKeepAlive model = new ProfileKeepAlive();
        model.setMonths(sid.keepAlive);
        return model;

    }

}
```

sample-resource-owner の実装につきましてはここでは割愛します。必要であれば下記のクラスのソースコードを参照してください。

完全修飾クラス名
　sample.subject.type.ProfileKeepAliveSubjectType

SubjectTypeの設定

SubjectTypeのクラスを実装したら設定ファイルを記述して認可機構で使えるようにします。 設定ファイルの配置場所は次の場所です。

場所
　%CONTEXT_PATH%/WEB-INF/conf/authz-subject-type-config/<任意の名称>.xml

XMLスキーマの場所
　%CONTEXT_PATH%/WEB-INF/schema/authz-subject-type-config.xsd

この設定ファイルの詳細については「認可仕様書」、または「設定ファイルリファレンス」を参照してください。

実装したサブジェクトタイプの実装クラスと、モデルクラスのセットを直接指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<p:authz-subject-type-config
   xmlns:p="http://www.intra-mart.jp/authz/authz-subject-type-config/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/authz/authz-subject-type-config/ ../../schema/authz-subject-type-config.xsd">

 <!-- サブジェクトタイプ sample-prof-keep-alive の設定 -->
 <subject-type
    type-class="sample.subject.type.ProfileKeepAliveSubjectType"
    model-class="sample.subject.type.ProfileKeepAlive" />

 <!-- サブジェクトタイプ sample-resource-owner の設定 -->
 <subject-type
    type-class="sample.subject.type.SampleResourceOwnerSubjectType"
    model-class="sample.subject.type.SampleResourceOwner" />

</p:authz-subject-type-config>
```

ChangeableNameSubjectTypeとは

参照する基準日によってサブジェクトの名称が変わることがある場合は、SubjectType インタフェースを内包する ChangeableNameSubjectType インタフェースを実装します。
例えば、IM共通マスタのユーザサブジェクトの場合、ユーザ情報は期間情報を複数持ち、基準日によって名称を変えることができますので、ユーザサブジェクトのサブジェクトタイプでは、ChangeableNameSubjectType インタフェースを実装しています。

ChangeableNameSubjectType インタフェースには、基準日を指定して名称を解決するための resolveDisplayNameWithDate メソッドがあります。
このメソッドを追加実装することで、検索基準日に応じた名称を認可設定画面の対象者条件一覧に表示することができます。
SubjectType インタフェースを実装したサブジェクトタイプの場合は、検索基準日を変更しても名称の表示には影響しません。

> **コラム**
> ChangeableNameSubjectType インタフェースは intra-mart Accel Platform 2013 Winter から提供しています。

```
@Override
public I18nValue<String> resolveDisplayName(final Object... keys) {

   // 基準日が指定されなかった場合はテナントのタイムゾーンで現在日時を指定します。
   final TimeZone timeZone = getTenantTimeZone();
   final DateTime baseDate = new DateTime(timeZone);
   return resolveDisplayNameWithDate(baseDate.getDate(), keys);
}

@Override
public I18nValue<String> resolveDisplayName(final String subjectId) {

   // 基準日が指定されなかった場合はテナントのタイムゾーンで現在日時を指定します。
   final TimeZone timeZone = getTenantTimeZone();
   final DateTime baseDate = new DateTime(timeZone);
   return resolveDisplayNameWithDate(baseDate.getDate(), subjectId);
}

@Override
public I18nValue<String> resolveDisplayNameWithDate(Date baseDate, final Object... keys) {
   Calendar calendar = GregorianCalendar.getInstance();
   calendar.setTime(baseDate);
   int year = calendar.get(Calendar.YEAR);

   final I18nValue<String> name = new I18nValue<String>();
   if (year >= 2015) {
      // 基準日が 2015 年以上の場合は名称を変更
      name.put(Locale.JAPANESE, ((String) keys[0]).concat(" カ月以上登録されているユーザ"));
   } else {
      name.put(Locale.JAPANESE, ((String) keys[0]).concat(" カ月以上有効なユーザ"));
   }
   return name;
```

```
}

@Override
public I18nValue<String> resolveDisplayNameWithDate(Date baseDate, final String subjectId) {

   // サブジェクトIDから表示名を取得して返します。
   final ProfileKeepAliveSIDDAO dao = DAOFactory.getTenantDatabaseDAO(ProfileKeepAliveSIDDAO.class);
   final ProfileKeepAliveSID sid = dao.find(subjectId);

   if (sid == null) {
      return null;
   }

   Calendar calendar = GregorianCalendar.getInstance();
   calendar.setTime(baseDate);
   int year = calendar.get(Calendar.YEAR);

   final I18nValue<String> name = new I18nValue<String>();
   if (year >= 2015) {
      // 基準日が 2015 年以上の場合は名称を変更
      name.put(Locale.JAPANESE, String.valueOf(sid.keepAlive).concat(" カ月以上登録されているユーザ"));
   } else {
      name.put(Locale.JAPANESE, String.valueOf(sid.keepAlive).concat(" カ月以上有効なユーザ"));
   }
   return name;
}

private TimeZone getTenantTimeZone() {
   try {
      final TenantInfoManager manager = new TenantInfoManager();
      final TenantInfo tenant = manager.getTenantInfo(true);
      if (tenant == null) {
         throw new RuntimeException();
      }
      return TimeZone.getTimeZone(tenant.getTimeZoneId());
   } catch (final AdminException e) {
     throw new RuntimeException(e);
   }
}
```

SubjectResolver

SubjectResolver はユーザがどのサブジェクトに当たるかという事を判断するためのインタフェースです。SubjectResolver には DeclaredSubjectResolver と OndemandSubjectResolver の2種類のインタフェースがあります。

DeclaredSubjectResolver と OndemandSubjectResolver

サブジェクトリゾルバには2種類のインタフェースが提供されています。追加したいサブジェクトの特性に応じてどちらか選択する必要があります。

DeclaredSubjectResolver

完全修飾クラス名は以下です。
`jp.co.intra_mart.foundation.authz.client.DeclaredSubjectResolver`

ユーザが特定できた時点（例えばログイン時）でサブジェクトが決まる場合 DeclaredSubjectResolver を実装します。つまりサブジェクトにしたい情報が単純にユーザコードがわかれば割り出せる情報であることが必要です。

intra-mart Accel Platform の提供している例で言えばユーザの所属組織がこれにあたります。サンプルアプリケーションのサブジェクトタイプでは `sample-prof-keep-alive` がこちらになります。

OndemandSubjectResolver

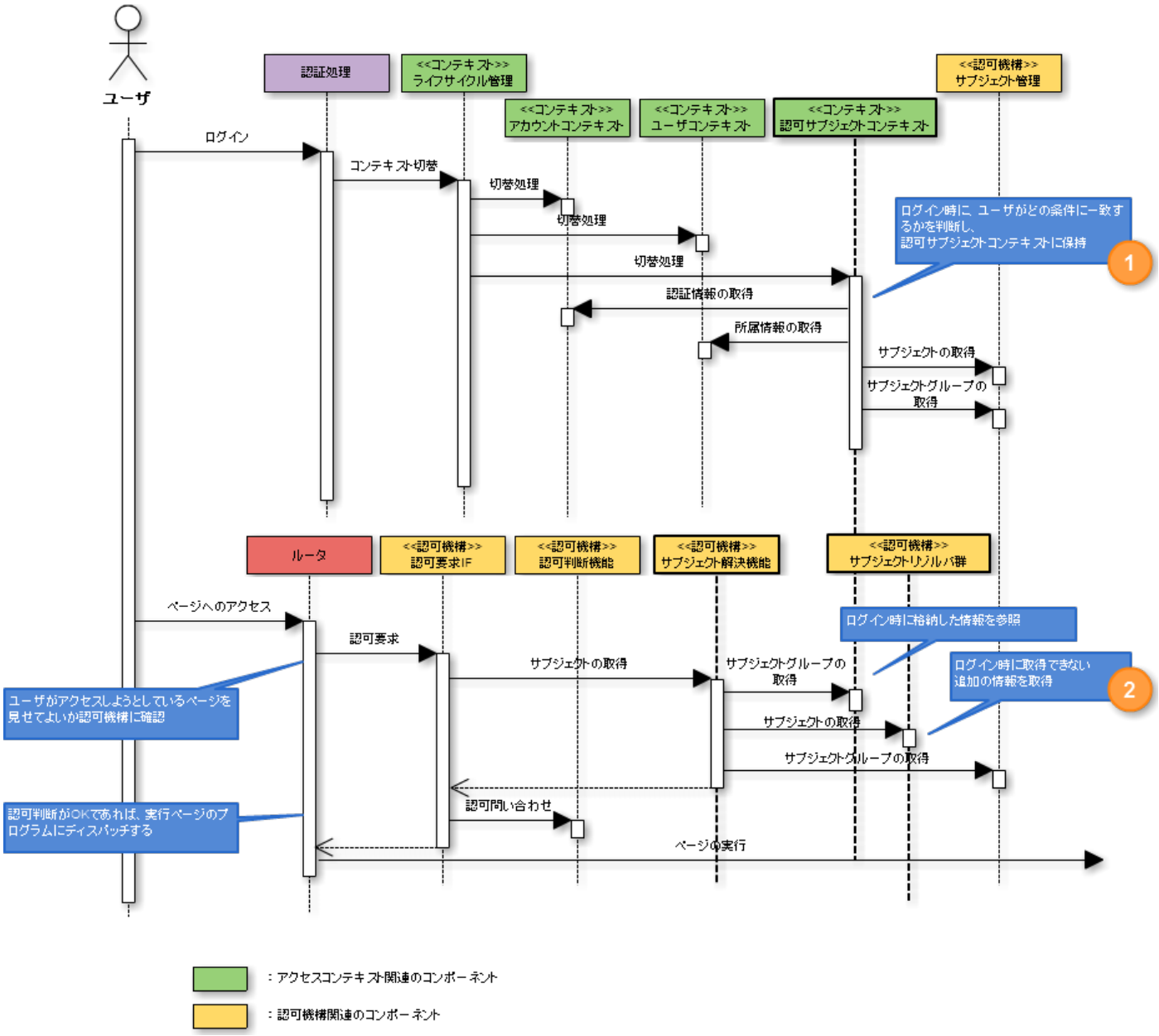
完全修飾クラス名は以下です。
`jp.co.intra_mart.foundation.authz.client.OnDemandSubjectResolver`

認可要求のタイミングにならなければ決定できない情報をサブジェクトにしたい場合 OndemandSubjectResolver を実装します。認可要求時に、認可対象のリソースによってサブジェクトの解釈を変更する場合がこれに当たります。

intra-mart Accel Platform が標準で提供している中にはこの実装はありません。サンプルアプリケーションのサブジェクトタイプでは `sample-resource-owner` がこちらになります。

SubjectResolver の実行タイミング

DeclaredSubjectResolver と OndemandSubjectResolver について紹介しましたが、ここでこれらの実装が呼び出される状況について説明します。



上図は認可判断を行うに当たってサブジェクトの解決を行うタイミングを示したシーケンス図です。認可機構ではユーザが特定できた時点（例えばログイン時）で決まるサブジェクトの場合、ログインと同時にサブジェクトを解決しておき認可サブジェクトコンテキストに保持しておきます(図中 1)。認可要求時には毎回サブジェクト解決を行うのではなく可能であればこの認可サブジェクトコンテキストの情報を使用して認可判断処理を行うことで、処理負荷を軽減しています。

ログイン時にサブジェクトを解決して認可サブジェクトコンテキストに格納する処理は後述する *認可サブジェクトコンテキストのデコレータ* が行っています。

認可要求のタイミングにならなければ決定できないサブジェクトの場合、これでは解決できないので、認可機構では認可要求を受けたタイミングでサブジェクト解決処理を行えるようにしています (図中 2)。OndemandSubjectResolver はこのタイミングで呼び出されます。

通常ログインユーザに対して認可判断を行う場合はサブジェクト解決についてはこれだけで DeclaredSubjectResolver が呼び出されることはありません。

DeclaredSubjectResolverが呼び出されるのは、バッチ処理中などのユーザがログインして利用しているわけではない状況下で認可を要求された時、またはユーザがログインしている場合でも、ログインユーザ以外のユーザに対して認可判断を要求された場合、認可機構は認可サブジェクトコンテキストが利用できないと判断し、OndemandSubjectResolver と同じタイミング (図中 2) で DeclaredSubjectResolver を実行することで、認可サブジェクトコンテキストの持つ情報を補っています。

DeclaredSubjectResolverの実装

すでに述べたとおり「プロファイルが有効になってからXXカ月」は時刻までを厳密に考えなければユーザコードがわかれば割り出せる情報です。これをサブジェクトとする場合、 リゾルバ は DeclaredSubjectResolver を実装します。

以下はサブジェクトタイプ `sample-prof-keep-alive` のリゾルバの例です。ユーザのプロファイルが今日まで何ヶ月間有効かを判定して、それより小さい月数を設定しているサブジェクトを取得して返します。

完全修飾クラス名

`sample.subject.resolver.ProfileKeepAliveSubjectResolver`

```
/**
 * ProfileKeepAliveSubjectResolver クラス <br>
 * プロファイルの現在までの期間の月数に応じてサブジェクトを解決します。
 * @author INTRAMART
 * @version 8.0
 */
public class ProfileKeepAliveSubjectResolver implements DeclaredSubjectResolver {

   /**
    * サブジェクトの解決処理を行います。
    * @param userCd ユーザコード
    * @return サブジェクトIDの一覧
    * @throws BizApiException
    */

   public Set<String> resolveProfileSubjectKeepAlive(final String userCd) throws BizApiException {
      final Set<String> result = new HashSet<String>();

      if (ContextStatus.isAdministrator() || !ContextStatus.isAuthenticated()) {
         // 管理者やゲストユーザの場合には何もしない
         return result;
      }

      final UserManager manager = new UserManager();
      final UserBizKey bizKey = new UserBizKey();
      bizKey.setUserCd(userCd);
      final ITerm term = manager.getUserTerm(bizKey, Env.getSystemDate().getTime());

      if (term != null && !term.isDisable()) {
         // 有効であること

         // 大体の月数を計ります
         final int months = (int) ((System.currentTimeMillis() - term.getStartDate().getTime()) / (30L * 24L * 60L * 60L * 1000L));

         // ※ 本来は現在の期間の前の期間も有効であれば加算すべきですが、ここでは説明のため省略します。

         // サブジェクトIDを保管しているProfileKeepAliveSubjectTypeを参照して
         // 月数より小さい月数を設定しているサブジェクトIDを取得します
         final ProfileKeepAliveSIDDAO dao = DAOFactory.getTenantDatabaseDAO(ProfileKeepAliveSIDDAO.class);
         result.addAll(dao.getSubjectIds(months));

      }

      return result;
   }

   /**
    * サブジェクトの解決処理を行います。
    * @param userCd ユーザコード
    * @return サブジェクトIDのセット
    * @throws Exception 解決処理に問題が発生した場合
    */
   @Override
   public Set<String> resolveSubjectIds(final String userCd) throws Exception {

      return resolveProfileSubjectKeepAlive(userCd);

   }
}
```

上記のとおり、 DeclaredSubjectResolver は引数にユーザコードしか渡されません。これはユーザコードがわかれば割り出せる情報であることに由来しています。

DeclaredSubjectResolverの設定

DeclaredSubjectResolverを有効にするには以下の設定を追加する必要があります。

%CONTEXT_PATH%/WEB-INF/conf/declared-subject-resolvers/<任意の名称>.xml

書式の詳細については「認可仕様書」、または「設定ファイルリファレンス」を参照してください。 以下は ProfileKeepAliveSubjectResolver を有効にするための設定例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declared-subject-resolvers
    xmlns:p="http://www.intra-mart.jp/authz/declared-subject-resolvers/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.jp/authz/declared-subject-resolvers/ ../../schema/declared-subject-resolvers.xsd">

  <p:class-name>sample.subject.resolver.ProfileKeepAliveSubjectResolver</p:class-name>

</p:declared-subject-resolvers>
```

OndemandSubjectResolverの実装

あるリソースについて、リソースの作成者をリソースオーナーとするようなサブジェクトを実現する場合、リゾルバは OndemandSubjectResolver を実装します。

以下はサブジェクトタイプ sample-resource-owner のリゾルバの例です。認可要求されたリソースのURIから対象となるデータを取り出し、その情報の作成者を調べます。作成者が認可要求元のユーザと一致する場合、リソースのオーナーとしてのサブジェクトを解決しています。

完全修飾クラス名
  sample.subject.resolver.ResourceOwnerSubjectResolver

```
/**
 * ResourceOwnerSubjectResolver クラス リソースの登録者をオーナーとして扱います。
 * @author INTRAMART
 * @version 8.0
 */
public class ResourceOwnerSubjectResolver implements OnDemandSubjectResolver {

    private static final String prefix = "sample-data://im-authz-ext-sample/data/";

    /**
     * サブジェクトの解決処理を行います。
     * @param userCd ユーザコード
     * @param resource リソース
     * @param action アクション
     * @return サブジェクトIDのセット
     * @see jp.co.intra_mart.foundation.authz.client.OnDemandSubjectResolver#resolveSubjectIds(java.lang.String,
     *      jp.co.intra_mart.foundation.authz.model.resources.Resource, jp.co.intra_mart.foundation.authz.model.actions.Action)
     */
    @Override
    public Set<String> resolveSubjectIds(final String userCd, final Resource resource, final Action action) {
        return resolveSubjectIds(userCd, resource.getUri(), action.toString());
    }

    /**
     * サブジェクトの解決処理を行います。
     * @param userCd ユーザコード
     * @param resourceUri リソースURI
     * @param action アクション
     * @return サブジェクトIDのセット
     * @see jp.co.intra_mart.foundation.authz.client.OnDemandSubjectResolver#resolveSubjectIds(java.lang.String,
     *      java.lang.String, java.lang.String)
     */
    @Override
    public Set<String> resolveSubjectIds(final String userCd, final String resourceUri, final String action) {

        final Set<String> result = new HashSet<String>();

        if (!resourceUri.startsWith(prefix)) {
            return result;
        }

        final String id = resourceUri.substring(prefix.length());

        // 対象のリソースのレコードを取得して作成者がユーザコードと一致するか確認
        final DataDAO dataDao = DAOFactory.getTenantDatabaseDAO(DataDAO.class);

        final Data data = dataDao.find(id);

        if (data == null || !userCd.equals(data.createUserCd)) {
            // 取得できないか、一致しない場合はサブジェクトなし
            return result;
        }

        // 一致したので、サブジェクトID を取得
        final ResourceOwnerSIDDAO sidDao = DAOFactory.getTenantDatabaseDAO(ResourceOwnerSIDDAO.class);
        final String sid = sidDao.getSid(SampleResourceOwner.IDENTIFIER);

        result.add(sid);

        return result;
    }
}
```

OndemandSubjectResolverの設定

OndemandSubjectResolverを有効にするには以下の設定を追加する必要があります。

```
%CONTEXT_PATH%/WEB-INF/conf/ondemand-subject-resolvers/<任意の名称>.xml
```

書式の詳細については「認可仕様書」、または「設定ファイルリファレンス」を参照してください。 以下は ResourceOwnerSubjectResolver を有効にするための設定例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<p:ondemand-subject-resolvers
    xmlns:p="http://www.intra-mart.jp/authz/ondemand-subject-resolvers/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.jp/authz/ondemand-subject-resolvers/ ../../schema/ondemand-subject-resolvers.xsd">

  <p:class-name>sample.subject.resolver.ResourceOwnerSubjectResolver</p:class-name>

</p:ondemand-subject-resolvers>
```

認可サブジェクトコンテキストのデコレータ

認可サブジェクトコンテキストはユーザがログインした際などにその時点で決まるサブジェクトを保持するコンテキスト情報です。*SubjectResolver* のところで OndemandSubjectResolver の作成を選択した場合、このデコレータを作成する必要はありません。

認可サブジェクトコンテキストはサブジェクトIDの配列と、サブジェクトグループIDの配列を主たる情報として保持していますが、コンテキストデコレータはこのうち、ユーザに対するサブジェクトIDを適切に解決する必要があります。サブジェクトIDはSubjectTypeがアクセス主体の実態とマッピングするIDです。実装するインタフェースは異なりますが、実質行わなければならない処理の内容は DeclaredSubjectResolver と変わりありません。可能なら処理を共有できるよう設計すべきです。

ここでは *DeclaredSubjectResolverの実装* で DeclaredSubjectResolver として実装した「プロファイルが有効になってからXXカ月」のサブジェクトの情報をコンテキストに追加するデコレータの実装です。

```
/**
 * ProfileKeepAliveContextDecorator クラス
 * @author INTRAMART
 * @version 8.0
 */
public class ProfileKeepAliveContextDecorator extends ContextDecoratorSupport {

    /**
     * @param context
     * @param resource
     * @return
     * @see
jp.co.intra_mart.foundation.context.core.ContextDecorator#decorate(jp.co.intra_mart.foundation.context.model.Context,
     *      jp.co.intra_mart.foundation.context.core.Resource)
     */
    @Override
    public Context decorate(final Context context, final Resource resource) {

        if (context == null || !(context instanceof AuthzSubjectContextImpl)) {
            return context;
        }

        final AuthzSubjectContextImpl authzContext = (AuthzSubjectContextImpl) context;

        final AccountContext accContext = Contexts.get(AccountContext.class);

        if (accContext == null || !accContext.isAuthenticated()) {
            // 認証が済んでいなければなにもしない。
            return context;
        }

        try {
            // SubjectResolverの処理を共有します
            final ProfileKeepAliveSubjectResolver resolver = new ProfileKeepAliveSubjectResolver();
            final Set<String> subjectIds = resolver.resolveProfileSubjectKeepAlive(accContext.getUserCd());

            for (final String sid : subjectIds) {
                // SubjectIdContainer に変換して、コンテキストに追加します
                authzContext.addSubjectIdContainer(new ProfileKeepAliveSubjectIdContainer(sid));
            }

            return authzContext;

        } catch (final BizApiException e) {
            throw new LifecycleException(e);
        }
    }

}
```

デコレータの実装を有効にするにはコンテキストの設定も追加する必要があります。

場所

`%CONTEXT_PATH%/WEB-INF/conf/context-config/<任意の名称>.xml`

XML スキーマの場所

`%CONTEXT_PATH%/WEB-INF/schema/context-config.xsd`

上記の実装を有効にするサンプルを以下に示します。

```
<?xml version="1.0"?>
<context-config
   xmlns="http://intra-mart.co.jp/foundation/context/context-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://intra-mart.co.jp/foundation/context/context-config ../../schema/context-config.xsd">

 <!-- 認可サブジェクトコンテキスト -->
 <context name="jp.co.intra_mart.foundation.authz.context.AuthzSubjectContext"
    depends="jp.co.intra_mart.foundation.user_context.model.UserContext">

   <!-- 認可サブジェクトコンテキスト（Webアクセス） -->
   <!-- Target for begin() -->
   <!-- Merge Decorator Settings -->
   <builder target="platform.request" >
    <builder-class>jp.co.intra_mart.system.authz.context.impl.StandardAuthzSubjectContextBuilder</builder-class>
    <decorator>
     <decorator-class>sample.subject.decorator.ProfileKeepAliveContextDecorator</decorator-class>
    </decorator>
   </builder>

   <!-- 認可サブジェクトコンテキスト（Webアクセス切替デフォルト） -->
   <!-- Target for switchTo() -->
   <!-- Merge Decorator Settings -->
   <builder target="platform.request.switch.default" >
    <builder-class>jp.co.intra_mart.system.authz.context.impl.StandardAuthzSubjectContextBuilder</builder-class>
    <decorator>
     <decorator-class>sample.subject.decorator.ProfileKeepAliveContextDecorator</decorator-class>
    </decorator>
   </builder>
 </context>
</context-config>
```

検索画面

認可設定画で、サブジェクトとして追加できるようにするためには、サブジェクトを検索する画面が必要です。この画面は認可機構のプラグインとして実装することができるようになっています。

新しいサブジェクトタイプのための検索画面を追加するために plugin.xmlを用意します。

場所

`%CONTEXT_PATH%/WEB-INF/plugin/jp.co.intra_mart.sample.authz.subject.search/plugin.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension
    point="jp.co.intra_mart.authz.subject.search" >
    <subjectSearchConfig
      name="IM-Master Authz Subject Search Config"
      id="jp.co.intra_mart.sample.authz.subject.search"
      version="8.0.0"
      rank="10"
      enable="true">

      <headerPage>sample/im_authz_ext_sample/header</headerPage>
      <buttons>
        <button
          subjectType="sample-resource-owner"
          displayName="リソースオーナー"
          action="addResourceOwner"
          >
          <options>
          </options>
        </button>
        <button
          subjectType="sample-prof-keep-alive"
          displayName="Keep Alive"
          action="addKeepAlive"
          >
          <viewPage>sample/im_authz_ext_sample/keep_alive_button</viewPage>
          <options>
          </options>
        </button>
      </buttons>

      <conditions>
        <condition id="sample.condition">
          <comparator type="ge"
            displayName="%condition.greater_or_equal" />
        </condition>

      </conditions>

    </subjectSearchConfig>
  </extension>
</plugin>
```

この設定は認可の「条件の新規作成」や「条件の編集」などの操作で使用する「対象者の条件設定画面」に表示されるボタンを追加します。プラグインの設定ファイルには複数のサブジェクトタイプについて同時に設定することができ、上記では `sample-prof-keep-alive` と `sample-resource-owner` のサブジェクトタイプ用のボタンを設定しています。

最初に現れる `<headerPage>` は認可設定画面でヘッダとして読み込むページを指定します。「対象者の条件設定画面」に追加したボタンのイベントハンドラなどをここに記述します。

ここでは次のようなページを読み込んでいます。

```
<script type="text/javascript">

 function addKeepAlive(plugin, baseDate){

  var val = $("#sample-prof-keep-alive-value").val();

  if(!/[0-9]+/.test(val)){
   alert("数字で入力してください。");
   return;
  }

  SubjectControl.update({
     subjectType : "sample-prof-keep-alive" ,
     key         : val ,
     name        : val + " カ月以上有効なユーザ "
  });

 }

 function addResourceOwner(plugin, baseDate){

  SubjectControl.update({
     subjectType : "sample-resource-owner" ,
     key         : "sample-resource-owner" ,
     name        : "リソースオーナー（サンプル）"
  });

 }

</script>
```

SubjectControl.update() は対象者の一覧にサブジェクトを追加するためのメソッドです。このメソッドに対しサンプルのようにサブジェクトタイプ、キーとなる情報、名称を渡すことで対象者に追加することができます。

組織のサブジェクトなどの場合、このイベントハンドラで検索画面をポップアップし、その戻り値を受けて対象者へ追加する仕組みになっています。ここでは簡単にするためそこまでは行わず、ボタンをクリックしたタイミングで対象者にサブジェクトを追加しています。

サブジェクトタイプ sample-resource-owner 用のボタンは普通のボタンとして表示されます。<button> タグの action 属性に指定されているのはボタンがクリックされたときに呼び出す関数の名前です。上記の <headerPage> で指定したページなどに関数を用意しておき、ここに指定します。

この関数に渡される引数は以下の通りです。

引数名	型	説明	導入バージョン
plugin	Object	プラグイン情報が格納されています。 plugin.subjectType: サブジェクトタイプID plugin.displayName: ボタン表示名 plugin.action: アクション名	2012 Autumn
baseDate	Date	認可設定画面で選択された基準日が格納されています。	2013 Winter

サブジェクトタイプ sample-prof-keep-alive では <viewPage> タグを使用してボタンの代わりにjsspページを読み込んでいます。このタグを指定するとボタンの代わりに指定されたjsspページをインクルードして表示します。ここでは次のようなページを読み込んでいます。

```
<imart type="imuiTextbox" id="sample-prof-keep-alive-value" style="text-align:right" size="2" />カ月以上有効なユーザ
<imart type="imuiButton" id="sample-prof-keep-alive" class="imui-small-button" style="width:118px" value="追加" />
```

ボタンの id 属性にサブジェクトタイプと同じ値を指定しておくことで、plugin.xmlの <button> タグに指定した action 属性の関数がクリック時のイベントハンドラとして追加されます。

これらを配備することで 「対象者の条件設定画面」に追加したサブジェクトタイプ用のボタンを追加することができます。ただし、この状態で認可設定画面を開いてもこれらのボタンは表示されません。

次に説明するポリシー部分編集設定で、使用するサブジェクトタイプの設定を変更しなければなりません。

ポリシー部分編集設定

ポリシー部分編集設定はポリシーの編集に当たって使用できるサブジェクトタイプを宣言しています。

認可設定画面部品の使用 でも登場していますがサンプルアプリケーションでは以下のように定義しています。

%CONTEXT_PATH%/WEB-INF/conf/authz-partial-policy-edit-config/authz-ext-sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<authz-partial-policy-edit-config
    xmlns="http://www.intra-mart.jp/authz/authz-partial-policy-edit-config"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.jp/authz/authz-partial-policy-edit-config ../../schema/authz-partial-policy-edit-config.xsd">

  <part-config>
    <part-id>im_authz_ext_sample</part-id>
    <caption-cd>CAP.Z.SAMPLE.AUTHZ.PARTCONFIG.TITLE</caption-cd>
    <resource-groups>
      <resource-group-id>sample-authz-data-crud</resource-group-id>
    </resource-groups>
    <subject-types>
      <subject-type-id>sample-prof-keep-alive</subject-type-id>
      <subject-type-id>sample-resource-owner</subject-type-id>
    </subject-types>
  </part-config>

</authz-partial-policy-edit-config>
```

この設定ファイルで使用するサブジェクトタイプIDとして追加しておかないと、認可設定画面で条件の追加を行おうとしても表示されないようになっています。

ここではもう一つ、例として標準でインストールされる「画面・処理」の認可設定に上記のサブジェクトタイプを追加してみます。

次のファイルを編集します。

%CONTEXT_PATH%/WEB-INF/conf/authz-partial-policy-edit-config/im_authz_impl_router.xml

以下のコメントで示してある行を追加します。

```
<?xml version="1.0" encoding="UTF-8"?>
<authz-partial-policy-edit-config
   xmlns="http://www.intra-mart.jp/authz/authz-partial-policy-edit-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/authz/authz-partial-policy-edit-config ../../schema/authz-partial-policy-edit-
config.xsd">

 <part-config>
  <part-id>im_authz_impl_router</part-id>
  <caption-cd>CAP.Z.IWP.ROUTER.AUTHZ.PARTCONFIG.TITLE</caption-cd>
  <resource-groups>
   <resource-group-id>http-services</resource-group-id>
  </resource-groups>
  <subject-types>
   <subject-type-id>im_authz_meta_subject</subject-type-id>
   <subject-type-id>imm_user</subject-type-id>
   <subject-type-id>imm_company_post</subject-type-id>
   <subject-type-id>imm_department</subject-type-id>
   <subject-type-id>imm_public_grp</subject-type-id>
   <subject-type-id>imm_public_grp_role</subject-type-id>
   <subject-type-id>b_m_role</subject-type-id>
   <subject-type-id>sample-prof-keep-alive</subject-type-id> <!----- この行を追加 ----------->
   <subject-type-id>sample-resource-owner</subject-type-id> <!----- この行を追加 ----------->
  </subject-types>
 </part-config>

</authz-partial-policy-edit-config>
```

確認

実際にサブジェクトが使用できることを確認します。

1. これまでの資材をすべて配置した上でアプリケーションサーバを起動します。
2. テナント管理者でログインし、認可設定画面を開きます。
3. 「リソースの種類」が「画面・処理」となっている事を確認して「権限設定を開始する」をクリックします。
4. 「条件の新規作成」が表示されるのでクリックすると「対象者の条件設定画面」が開き、「XXX カ月以上有効なユーザ」と「リソースオーナー」のボタンが表示されていることを確認できます。
5. 「リソースオーナー」の方はここでは実質使用できませんので、「XXX カ月以上有効なユーザ」の方に `3` と入力して「追加」ボタンをクリックします。
   左側の対象者の欄に「3 カ月以上有効なユーザ」が追加され、名称も自動的に「3 カ月以上有効なユーザ」となっている事を確認します。
6. 「OK」ボタンをクリックし、認可設定画面に戻ります。
   条件として「3 カ月以上有効なユーザ」が追加されていることを確認します。
7. 同様に操作して「6 カ月以上有効なユーザ」を追加します。
8. IM共通マスタのユーザのメンテナンス画面を開き、テナント管理者のプロファイルを開きます。
9. 画面上の期間バーを操作して、現在の期間の開始日を 4 か月前にします。
10. 一度ログアウトし、ログインしなおします。
11. 認可設定画面を開くと、「3 カ月以上有効なユーザ」が赤く表示され、「6 カ月以上有効なユーザ」が暗く表示されていることが確認できます。

> **コラム**
> 認可設定画面ではログイン時に解決されたサブジェクトが赤く表示されるようになっています。ここではテナント管理者のプロファイルの期間を4か月で切ったため、3か月以上有効なユーザとして識別されていますが、6か月以上有効なユーザとしては識別されていないことを示しています。

リソース拡張ガイド

項目

リソース拡張とは

intra-mart Accel Platform では標準的に以下のようなコンポーネントが認可機構と連携しています。

- ルータ
- メニュー
- ポータル
- IM共通マスタ（会社）

それぞれのコンポーネントの持つ特長によって、リソースの階層構造や使用できるアクションが異なっています。

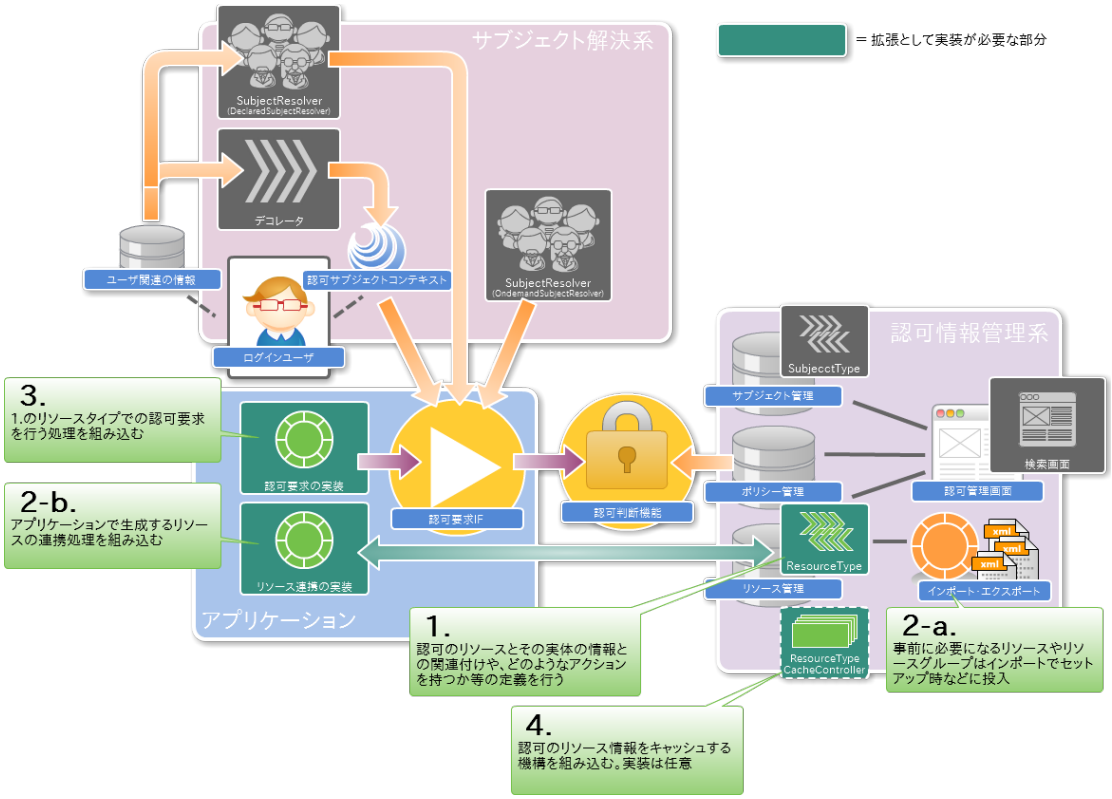
認可機構ではリソースの階層構造によって継承による権限の継承が起こるため、これを利用して管理者の設定の手間を軽減できるようリソースの階層構造を決定すべきです。リソースのアクションについても、リソースの特性と組み合わせて認可設定の際の意味の分かりやすさを鑑みて設計されていなければなりません。

アプリケーションを認可機構と連携させるに当たり、上記に挙げたコンポーネント同様に独自のリソース構造やアクションを独自に定義したい場合、リソースの拡張を作成して実現することができます。

ここではリソースの拡張を行うための手順について説明します。

大まかな流れ

リソースタイプの拡張を行うには下記に示す実装と合わせて、認可機構にリソースに関連する情報の登録が必要になります。



1. アプリケーションのリソースを認可機構でどう表現するかを決めます。
   以下のような事を作業を開始する前に決めておくと作業がスムーズに進められます。個々の詳細については後述します。
   - リソースが使用するアクション
   - リソースのURI書式
   - リソースのグルーピングと階層構造
2. ResourceType クラスを実装します。(図中 1)
   認可機構へのプラグインとしてリソースタイプクラスを実装します。
   リソースタイプは認可対象となる実体の情報から認可機構上のリソースへの関連付けやリソースの持つアクションやURI書式についての責務を持っています。
3. 管理者が認可設定ができるように、アプリケーションのリソースを認可機構へ登録します。
   認可設定を管理する対象はリソースとして予め認可機構に登録しておかなければなりません。稼働前に必要になるリソースグループやリソースの情報がある場合はインポートファイルとして用意しておき、セットアップなどで投入しておきます (図中 2-a) 。
   アプリケーション側で認可対象としたいものが増えたり減ったりした場合、それに合わせて認可機構上のリソースも追加削除が必要になります (図中 2-b) 。
4. 認可処理をアプリケーションに組み込みます。
   ここまでで認可リソースの準備ができているはずなので、アプリケーションに認可機構と連携する実装を組み込みます (図中 3) 。
   - 適切なタイミングでリソースに対して認可要求を行う
   - アプリケーションで登録したリソースの認可設定を認可設定画面部品を呼び出して行う
5. 必要であれば、リソースのキャッシュを実現する ResourceTypeCacheController クラスを実装します。 (図中 4) 。
   リソースが頻繁に利用される場合は、キャッシュを組み込むことでパフォーマンスを改善できます。キャッシュを組み込む際には、リソースのキャッシュコントローラクラスを実装します。
   キャッシュコントローラは、リソースタイプごとに定義することができます。キャッシュコントローラを実装しない場合、リソース情報取得のたびにデータベースへのアクセスが発生します。

リソースを認可機構でどう表現するかを決める

アクションの決定

まずリソースがどんなアクションをとりうるかを明確にしておきます。

サンプルアプリケーションでは単純なデータのCRUD操作を扱うので、以下のアクションを定義することにします。

- Create （作成）
- Read （参照）
- Update （更新）
- Delete （削除）

リソースURI書式の決定

リソースを認可機構上でどう識別するかを定義します。

リソースはURI表記でシステム上一意になる文字列で表現します。

例: `service://tenant/default_home`

認可側では `<リソースタイプID>:` で始まる点と使用可能な文字について制約を設けています(リソースタイプIDについては後述)。

大体の場合において、次のような書式が必要となるでしょう。

`<リソースタイプID>:<アプリケーションの識別子><機能内の識別子1><機能内の識別子2> ... <リソースのID>`

サンプルアプリケーションでは単一のIDで識別可能なデータを扱うので、 割り当てるURIのパターンとして以下のように設計しました。

`sample-data://im-authz-ext-sample/data/<データのID>`

リソースのグループ構造の決定

認可機構内ではリソースは木構造になっていますが、アプリケーションが必要とするリソースをどこに格納するか予め設計しておく必要があります。

単一の根から葉に渡る木構造をひとまとまりとしてリソースグループセットと呼称していますが、機能によってはリソースグループセットを予約しています。例えばメニューグループ用には `im-menu-group` というIDのリソースグループセットが予約されています。

単純に画面を追加して画面に対する表示制御を行いたいのであれば既に `http-services` というリソースグループがありますので そのリソース構造に組み入れればよいでしょう。メニューの場合、`im-menu-group` というIDのリソースグループセットを APIが自動的に管理していますので、他のアプリケーションがこのリソースグループにリソース登録を行うのは適切ではありません。

新しいアプリケーション用に独自のリソースを作りたいのであれば基本的には新たなリソースグループセットを追加する方が良いでしょう。

他の機能の木にリソースを組み入れるか、独自のリソースグループセットを追加するかは、事前に検討してください。

独自のリソースグループセットを作成する場合、他のリソースグループと競合しないよう注意してください。intra-mart Accel Platform で予約されているリソースグループセットについては「認可仕様書」に一覧がありますので、そちらを参照してください。

サンプルアプリケーションでは *サンプルのセットアップ手順* でインポート資材から `sample-authz-data-crud` というIDのリソースグループセットを追加しています。

`sample-authz-resource-group.xml` （リソースグループ追加資材）

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://www.intra-mart.jp/authz/imex/resource-group">
  <authz-resource-group id="sample-authz-data-crud">
    <display-name>
      <name locale="ja">サンプルアプリケーション</name>
    </display-name>
  </authz-resource-group>
</root>
```

リソース構造の設計

権限設定は木構造に従って継承が起こります。このためツリー上の一番末端のノードに対して一つ一つ権限を設定しなくても、 ある程度上位のノードに設定するだけで末端のノードに対して設定を適用することができます。この特性を考慮して管理ユーザの利便性を図りつつリソースの構造を設計しておきます。リソースグループにはそれぞれIDが必要になりますので、リソースの構造と合わせてどのよ

うなID体系を使用するかについても考えておきます。

サンプルでは単純に以下のような構造を考えます。

- sample-authz-data-crud （アプリケーションのルートのリソースグループ）
  - sample-authz-data-crud-data-<データのID> （アプリケーションで操作するデータに対応するリソース）

sample-authz-data-crud はアプリケーション用のリソースグループセットのルートになります。アプリケーション側でデータが追加されるとそれと連動して sample-authz-data-crud の配下にリソースを追加していきます。リソースを作成すると対となるグループが生成されますので、そのグループのIDの採番方式も決めておきます。ここでは連動するデータのIDを含めて以下のような形にしておきます。

```
sample-authz-data-crud-data-<データのID>
```

ResourceType の実装

ResourceTypeとは、リソースの特性を表す定義です。

- リソースがどのようなアクションを持つかを定義します
- リソースのモデルクラスを決定します
- リソースがどのようなURI体系を持つかを定義します

新たにResourceTypeを追加するにはResourceTypeインタフェースを実装するクラスとその関連クラスが必要になります。

ResourceType インタフェース

リソースタイプは以下のインタフェースを実装する必要があります。

jp.co.intra_mart.foundation.authz.model.resources.ResourceType<T>

これを実装するためには、あわせて以下のクラスの実装が必要になります

- ResourceTypeが返すActionクラス群
- ResourceTypeの型引数になっているクラスT （リソースのモデルクラス）

このインタフェース定義から、このクラスで実装しなければいけないポイントは以下の5つです。

1. **リソースタイプIDを決定する**
   - リソースタイプIDはシステム内で重複してはいけません。
   - 255文字までの英数、ハイフンで構成されていなければなりません
2. **このリソースタイプに関連付けられるリソースモデルの型を決定する**
   - リソースモデルはリソースタイプに対して1対1で紐づけられます。この紐づけはシステム上で重複してはいけません。
   - 上記に反しないのであれば、このモデルのために新たなクラスを作成する必要はありません。
3. **このリソースタイプを持つリソースがとることのできる操作（Action）を明確にする**
   - このリソースに関連するアクションがすべてわかる必要があります。
   - アクションはリソースタイプに紐づいており、認可機構上はリソースタイプ＋アクションで一意に識別されます。
4. **リソースモデルのURI表現をサポートする**
   - リソースモデルをURI表現に変換できる必要があります。
5. **Actionの文字列表現をサポートする**
   - 文字列からActionクラスへパースできる必要があります。
   - アクションの文字列表現は100文字以内の英数、ハイフン、アンダースコアで構成されていなければなりません。

サンプルアプリケーションではCRUD(Create, Read, Update, Delete) の操作を持つリソースタイプを定義しています。以下がその実装例です。

```
/**
 * SampleDataResourceType クラス
 * @author INTRAMART
 * @version 8.0
 */
public class SampleDataResourceType implements ResourceType<SampleDataModel> {

    public static final String TYPE_ID = "sample-data";

    public static final String ROOT_ID = "sample-authz-data-crud";

    public static final String IDPREFIX = "sample-authz-data-crud-data-";

    private static final long serialVersionUID = -1025455140074239366L;

    @Override
    public String createResourceUri(final SampleDataModel model) throws UnexpectedResourceModelReceivedException {
        return TYPE_ID.concat("://im-authz-ext-sample/data/").concat(model.getDataId());
    }

    @Override
    public List<Action> getAllActions() {

        final List<Action> actions = new ArrayList<Action>();
        actions.add(new C(this));
        actions.add(new R(this));
        actions.add(new U(this));
        actions.add(new D(this));

        return actions;
    }

    @Override
    public String getResourceTypeId() {
        return TYPE_ID;
    }

    @Override
    public Action parseAction(final String action) {

        if ("C".equals(action)) {
            return new C(this);
        }

        if ("R".equals(action)) {
            return new R(this);
        }

        if ("U".equals(action)) {
            return new U(this);
        }

        if ("D".equals(action)) {
            return new D(this);
        }

        throw new RuntimeException("invalid action :" + action);

    }
}
```

以下がモデルクラスの実装例です。

この例のモデルクラスはデータのIDだけを格納するものになっています。実際に実装する際にはリソースを一意に特定するために必要な情報を収めたクラスにしてください。既述のとおり、すでに存在するクラスで共用が可能ならばそれでもかまいません。

```
/**
 * SampleDataModel クラス
 * @author INTRAMART
 * @version 8.0
 */
public class SampleDataModel {

    private String dataId;

    public SampleDataModel(final String id) {
        dataId = id;
    }

    public String getDataId() {
        return dataId;
    }

    public void setDataId(final String dataId) {
        this.dataId = dataId;
    }

}
```

リソースタイプを認可機構に登録するための設定

%CONTEXT_PATH%/WEB-INF/conf/authz-resource-type-config/ に、設定ファイルを追加します。

この詳細については「設定ファイルリファレンス」、または「認可仕様書」を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<p:authz-resource-type-config
    xmlns:p="http://www.intra-mart.jp/authz/authz-resource-type-config/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.jp/authz/authz-resource-type-config/
../../../../../im_authz_impl/src/main/schema/authz-resource-type-config.xsd">

  <resource-type
      type-class="sample.resource.type.SampleDataResourceType"
      model-class="sample.resource.type.SampleDataModel" />

</p:authz-resource-type-config>
```

- 要素 <authz-resource-type-config>
  - この設定ファイルのルートノードです。
- 要素 <resource-type>
  - リソースタイプ一つ分の定義です。複数定義可能です。
    - 属性 type-class
      - このリソースタイプのクラスを完全修飾クラス名で設定します。
    - 属性 model-class
      - このリソースタイプで使用するモデルクラスを完全修飾クラス名で設定します。
      - type-class で指定しているクラスの型引数に指定されているクラスと一致していなければなりません。

この設定と、設定に書かれたクラスを配置し、再起動すれば認可機構でこのリソースタイプIDから始まるリソースURIを取り扱うことが可能になります。

アプリケーションと認可処理を連携するには、アプリケーション側で連携のための実装を行う必要があります。具体的には以下の2つの点です。

- アプリケーションの必要なポイントで認可要求を発行する実装を組み込むこと
- 認可要求が発生する前に認可機構へリソースを登録しておくこと

以降の節ではAPIやインポートを使用したリソースの登録方法と、アプリケーションへの認可要求の組み込みについて説明します。

リソースの連携

管理者が認可設定を行うためには、あらかじめリソースとして登録されていなければ認可対象が存在することがわかりません。運用中にリソースをリアルタイムに増減させたい場合などは、必要なタイミングで認可機構のAPIを呼び出してリソース登録を行う必要があります。

APIから認可リソース登録を行う場合は ResourceManager クラスを使用します。

今回のサンプルアプリケーションでは、以下のようなタイミングで認可リソースを変更します。

- データを登録されたタイミングで、対応する認可リソースも登録する。
- データのタイトルが更新されたタイミングで、認可リソースの名称を更新する。
- データを削除されたタイミングで、対応する認可リソースも削除する。

登録する内容はドキュメントの前節までに挙げてきた例から、以下のような内容とします：

- リソースグループセットの先頭のリソースグループ (id: `sample-authz-data-crud`)があるものとします。
- 上記のリソースグループ配下にアプリケーションのデータに対応するリソースを追加します。
- リソースのURI書式は `sample-data://im-authz-ext-sample/data/<データのID>` 。（ただし、リソースタイプがリソースモデルからURIを生成するため、このコード例には現れません）
- リソースとセットで作成されるリソースグループのIDは `sample-authz-data-crud-data-<データのID>`

サンプルでは上記の処理をユーティリティクラスとして実装しています。以下では該当箇所のみ解説します。ソースコードは以下のクラスを参照してください。

`sample.resource.link.Linkage`

まずリソースの登録処理です。

```
/**
 * データに対応するリソースを作成します。
 * @param id データのID
 * @param title データのタイトル
 * @throws UnexpectedResourceModelReceivedException
 */
public static void create(final String id, final String title) throws UnexpectedResourceModelReceivedException {

    // リソースを登録する際には ResourceManagerが必要です。
    // ResourceManagerはファクトリクラスからインスタンスを取得します。
    final ResourceManager manager = ResourceManagerFactory.getInstance().getResourceManager();

    // データのIDを使用してモデルクラスを作成します。
    final SampleDataModel model = new SampleDataModel(id);

    // リソースを登録する先のリソースグループ（ルートのグループ）を取得します。
    // SampleDataResourceType.ROOT_ID は im-authz-sample-crud
    final ResourceGroup parent = manager.getResourceGroup(SampleDataResourceType.ROOT_ID);

    // 名称を作成します。本来ならテナントでサポートされている分の名称を登録すべきです。
    final AccountContext context = Contexts.get(AccountContext.class);
    final I18nValue<String> name = new I18nValue<String>(context.getLocale(), title);

    // ルートのグループを親に、リソースの登録を行います。
    manager.registerAsResource(model, name, SampleDataResourceType.IDPREFIX.concat(id), parent);

}
```

データのIDとタイトルを引数に受け、IDからリソースのURIとグループのIDを生成し、タイトルをリソースの名称として使用しています。

次はデータが削除された場合にリソースを削除する処理です。

```
/**
 * リソースの削除を行います。
 * @param id データのID
 */
public static void remove(final String id) {

    final ResourceManager manager = ResourceManagerFactory.getInstance().getResourceManager();

    // リソースグループを削除すると、対応するリソースも削除されます。
    // ここではリソースグループのIDを直接指定して、リソースを削除しています。
    manager.removeResourceGroup(SampleDataResourceType.IDPREFIX.concat(id));

}
```

コメントにもありますが、リソースを消す場合はリソースに1対1で紐づいているリソースグループを削除します。 これでリソースも削除されます。

次はデータのタイトルが変更された際、リソースの名称に適用する更新処理です。

```
/**
 * リソースの名称を更新します。
 * @param id データのID
 * @param title 更新する名称
 */
public static void update(final String id, final String title) {

    final ResourceManager manager = ResourceManagerFactory.getInstance().getResourceManager();

    // 名称を作成します。本来ならテナントでサポートされている分の名称を登録すべきです。
    final AccountContext context = Contexts.get(AccountContext.class);
    final I18nValue<String> name = new I18nValue<String>(context.getLocale(), title);

    // 名称はリソースではなく、リソースに対応するリソースグループが保持しています。
    // このため、リソースグループに対して 名称の更新を行います。
    manager.setResourceGroupNames(SampleDataResourceType.IDPREFIX.concat(id), name);

}
```

サンプルアプリケーションでは画面でデータの追加・更新・削除操作をされるたびにスクリプト開発モデルのファンクションコンテナから上記のユーティリティを呼び出すことでリソースの連携を行っています。

コラム

サンプルアプリケーションはサンプルとしてシンプルにするために画面系の実装から直接リソースの連携を行っていますが、 設計としてはアプリケーションのモデルを扱うAPIをまず設計し、データベースへアプリケーションのデータを追加／削除すると同時に 認可リソースへの連携を行う等、アプリケーションデータと認可リソースの間で齟齬を起こさない設計であることが理想的です。

認可処理を組み込む

APIによる認可要求

アプリケーション固有のデータに対する認可処理を行う場合、APIなどでアクセスしようとするデータをURIに変換し、実行しようとするアクションとともに認可要求を行わなければなりません。

通常のWeb環境では認可サブジェクトコンテキストが存在するので、サブジェクトに関しては認可クライアント側で自動的に解決されます。リソースのURIについてはリソースタイプが登録されていればリソースモデルから変換が行われますので、リソースモデルとアクションがわかれば認可クライアントを介して認可要求を実行できます。

サンプルではリソースの連携用のユーティリティで認可要求の実装をしています。以下では該当箇所のみ解説します。ソースコードは以下のクラスを参照してください。

`sample.resource.link.Linkage`

```
/**
 * データのIDとアクションから認可要求を行います。 <br>
 * このリソースタイプのアクションは
 * <ul>
 * <li>{@link C} 登録</li>
 * <li>{@link R} 参照</li>
 * <li>{@link U} 更新</li>
 * <li>{@link D} 削除</li>
 * </ul>
 * の4種があります。
 * @param id データのID
 * @param action アクション("C","R","U","D")
 * @return 権限があればtrue, 無ければfalse
 * @throws UnexpectedResourceModelReceivedException
 */
public static boolean hasPermission(final String id, final String action) throws
UnexpectedResourceModelReceivedException {

   // idを使用してリソースモデルを作成
   final SampleDataModel model = new SampleDataModel(id);

   // 認可クライアントをファクトリから取得
   final AuthorizationClient client = AuthorizationClientFactory.getInstance().getAuthorizationClient();

   // 認可要求を実行
   final AuthorizeResult result = client.authorize(model, action);

   // 結果を返します。
   return AuthorizeResult.Permit.equals(result);

}
```

上記のように認可クライアントをファクトリから取得して、認可要求を実行します。リソースモデルがリソースタイプとともに認可機構に登録されていれば、リソースモデルを使用して認可要求をすることができます。結果はAuthorizationResult型のenumが返されますので、Permitであるかをみて処理を続行するかどうかを決めます。

imartタグ、カスタムタグによる認可要求

タグを使用して権限によって画面の表示可能領域を制御することができます。

Java EE 開発モデル（ JSP ）の例

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.intra-mart.co.jp/taglib/im-tenant" prefix="imt" %>


<h1>認可要求のサンプル</h1>

<imt:imAuthz uri="service://authz/settings/basic" action="execute" >
  <div>認可されていればここが表示されます。</div>
</imt:imAuthz>

<imt:imAuthz uri="service://authz/settings/basic" action="execute" negative="<%= Boolean.TRUE %>">
  <div>認可されていなければここが表示されます。</div>
</imt:imAuthz>
```

> **注意**
> 属性 negative は intra-mart Accel Platform 2013 Autumn 以降のバージョンで利用可能です。

スクリプト開発モデル（プレゼンテーションページ）の例

```
<h1>認可要求のサンプル</h1>

<imart type="imAuthz" uri="service://authz/settings/basic" action="execute" >
  <div>認可されていればここが表示されます。</div>
</imart>

<imart type="imAuthz" uri="service://authz/settings/basic" action="execute" negative>
  <div>認可されていなければここが表示されます。</div>
</imart>
```

> **注意**
> 属性 negative は intra-mart Accel Platform 2013 Autumn 以降のバージョンで利用可能です。

これらのタグを使用する場合にはリソースのURIとアクションがわかっている必要があります。サンプルアプリケーションでは、各データのuriをファンクションコンテナで求めておき、各アクション毎の権限によって「削除」や「変更」のボタンの表示非表示を切り替えています。

```
<table class="imui-table">
 <!-- アプリケーションのデータを一覧で表示 -->
 <imart type="repeat" list=data item="item">
  <tr>
   <td>
    <!-- 削除アクションの権限がない場合は削除ボタンを表示しない -->
    <imart type="imAuthz" uri=item.uri action="D">
     <imart type="imuiButton" value="削除"  class="imui-button right sample-delete" data-id=item.data_id />
    </imart>
    <!-- 更新アクションの権限がない場合は変更ボタンを表示しない -->
    <imart type="imAuthz" uri=item.uri action="U">
     <imart type="imuiButton" value="変更"  class="imui-button right sample-update" data-id=item.data_id />
    </imart>
    <!-- 参照アクションの権限がある場合は参照リンクを表示 -->
    <imart type="imAuthz" uri=item.uri action="R">
     <h1><a id='<imart type="string" value=item.data_id />' href="javascript:void(0)" class="sample-data"><imart
type="string" value=item.title /></a></h1>
    </imart>
    <!-- 参照アクションの権限がない場合は参照リンクを表示しない -->
    <imart type="imAuthz" uri=item.uri action="R" negative>
     <h1><imart type="string" value=item.title /></h1>
    </imart>
    <div>(<imart type="string" value=item.uri />)</div>
   </td>
  </tr>
 </imart>
</table>
```

認可設定画面部品の使用

アプリケーション用にリソースグループセットを新たに追加した場合、アプリケーション側の管理画面として認可設定画面を表示することができます。このためにポリシー部分編集定義の設定を追加します。

ポリシー部分編集定義はリソースグループセットに対して権限設定に使用するサブジェクトタイプを選択して、部分的な認可設定を可能にするために定義する設定ファイルです。

認可設定画面はこのポリシー部分編集定義を集めて表示されますので、この設定がないと認可設定画面にも表示されません。

この設定の詳細については「設定ファイルリファレンス」、または「認可仕様書」を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<authz-partial-policy-edit-config
   xmlns="http://www.intra-mart.jp/authz/authz-partial-policy-edit-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/authz/authz-partial-policy-edit-config ../../schema/authz-partial-policy-edit-config.xsd">

 <part-config>
  <part-id>im_authz_ext_sample</part-id>
  <caption-cd>CAP.Z.SAMPLE.AUTHZ.PARTCONFIG.TITLE</caption-cd>
  <resource-groups>
   <resource-group-id>sample-authz-data-crud</resource-group-id>
  </resource-groups>
  <subject-types>
   <subject-type-id>sample-prof-keep-alive</subject-type-id>
   <subject-type-id>sample-resource-owner</subject-type-id>
  </subject-types>
 </part-config>

</authz-partial-policy-edit-config>
```

今回例示してきた内容では リソースグループセットを追加していますので、使用するリソースグループIDにそのルートとなるリソースグループのID(id: sample-authz-data-crud) を指定し、この配下のリソースをアプリケーションの管理機能の一部として認可設定できるようにしています。

またこの編集定義で使用できるサブジェクトタイプを sample-prof-keep-alive と sample-resource-owner の2種に設定しています。これらのサブジェクトタイプは *サブジェクト拡張ガイド* でサンプルとして実装しているものです。sample-prof-keep-alive は「プロファイルの有効期間がXXカ月以上」 というサブジェクトを追加するためのサブジェクトタイプでsample-resource-owner アプリケーションデータの作成者を「リソースオーナー」として扱うサブジェクトタイプです。

この設定を追加すると認可設定画面の左上「リソースの種類」のセレクトボックスに、この設定の選択肢が追加されます。

さらにアプリケーションの管理画面などで上記で設定した部分の認可設定画面を部品として呼び出すことができます。
imAuthzPolicyEditorタグを使用して以下のように指定すると、画面にボタンが表示され、ポップアップ画面として認可設定画面を呼び出せます。

```
<imart type="imAuthzPolicyEditor" displayType="button" partId="im_authz_ext_sample">認可設定を開く</imart>
```

partId には 上記で作成した設定ファイルで指定した part-id を指定します。そのほかのパラメータや、使い方の詳細についてはAPIリストを参照してください。

デフォルトでは、認可設定画面を部品として利用できるユーザは、認可設定の基本画面を開くことができる権限を持つユーザに限られます。それ以外の権限状態で認可設定画面を部品として利用可能にする方法の詳細は「*リソース表示可否判断クラスの設定* 」を参照してください。

リソース表示可否判断クラスの設定

通常、imAuthzPolicyEditorタグで認可設定画面を部品として利用できるユーザは、認可設定の基本画面を開くことができる権限を持つユーザに限られます。これは、通常認可設定ができないユーザに認可設定画面を利用させないようにするために、認可機構で制限しています。

しかし、認可設定画面を部品として利用する場合に、特定のリソースに対して呼び出し側の機能の権限に基づいて設定可否を判断したい場合があります。例えば、メニュー設定画面を開くことができるユーザが、登録したメニューグループに対して認可権限を設定するケースです。

認可機構側では個々のリソースグループに対して表示可否を判断できないため、このような細かなリソースグループの表示制御を行いたい場合に、各機能に問い合わせを行って表示可否を判断してもらうコールバック機能があります。

imAuthzPolicyEditorタグではこのコールバック機能を使用して、imAuthzPolicyEditorタグを使用する側の機能へ問い合わせを行います。コールバックの設定はポリシー部分編集定義の設定で行います。

コラム

intra-mart Accel Platform 2013 Autumn 以降のバージョンを利用する場合、認可設定の基本画面を開くことができる権限を持つユーザは、コールバックの設定に関わらずすべてのリソースグループが表示されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<authz-partial-policy-edit-config
   xmlns="http://www.intra-mart.jp/authz/authz-partial-policy-edit-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/authz/authz-partial-policy-edit-config ../../schema/authz-partial-policy-edit-config.xsd">

 <part-config>
  <part-id>im_authz_ext_sample</part-id>
  <caption-cd>CAP.Z.SAMPLE.AUTHZ.PARTCONFIG.TITLE</caption-cd>
  <resource-groups>
   <resource-group-id>sample-authz-data-crud</resource-group-id>
  </resource-groups>
  <subject-types>
   <subject-type-id>sample-prof-keep-alive</subject-type-id>
   <subject-type-id>sample-resource-owner</subject-type-id>
  </subject-types>
  <callbacks> <!----- この行を追加 ----------->
    <resource-group-authorizer>sample.resource.type.SampleDataResourceGroupAuthorizer</resource-group-authorizer> <!----- この行を追加 ----------->
  </callbacks> <!----- この行を追加 ----------->
 </part-config>

</authz-partial-policy-edit-config>
```

resource-group-authorizer で指定したクラスには、認可設定画面を部品として開く際に表示対象のリソース情報が渡されます。この表示可否判断クラスはimAuthzPolicyEditorタグを使用する各機能側で用意します。

表示可否判断クラスを作成するためには、 AuthzPartialResourceGroupAuthorizer インタフェースを実装するか、AuthzPartialResourceGroupAuthorizerHelper クラスを継承するクラスが必要になります。

- AuthzPartialResourceGroupAuthorizer インタフェースを実装する場合

  AuthzPartialResourceGroupAuthorizer インタフェースでは、リソースグループIDのセットを受け取ります。表示対象のリソースグループが複数ある場合でも、認可機構によって一度だけ呼ばれます。

  表示対象のリソースグループにかかわらず一律に表示可否の判断ができる場合や、リソースグループIDから判断可能な場合は、こちらの方法を採用した方が高速に動作します。

- AuthzPartialResourceGroupAuthorizerHelper クラスを継承する場合

  AuthzPartialResourceGroupAuthorizerHelper クラスでは、リソースグループIDが認可のリソース情報に変換され、それを受け取ります。表示対象のリソースグループ配下に存在するリソースの数分だけ、認可機構によって呼ばれます。

  表示対象のリソースで表示可否を判断する場合は、こちらの方法を採用した方が便利です。

サンプルアプリケーションでは、AuthzPartialResourceGroupAuthorizerHelper クラスを継承して、サンプルデータの読み取り権限がある場合のみ表示する実装を定義しています。以下がその実装例です。

```
/**
 * サンプルのリソースグループ表示可否判断クラス
 * @author INTRAMART
 * @version 8.0.3
 */
public class SampleDataResourceGroupAuthorizer extends AuthzPartialResourceGroupAuthorizerHelper {

    @Override
    public AuthorizeResult authorize(final AuthorizationClient client, final Resource resource) throws
UnexpectedResourceModelReceivedException {
        // サンプルのリソースタイプの場合は判断不可
        final ResourceType<?> resourceType = resource.getType();
        if (!SampleDataResourceType.TYPE_ID.equals(resourceType.getResourceTypeId())) {
            return AuthorizeResult.Deny;
        }

        // 読み取り権限があるかどうか確認
        final Action action = resourceType.parseAction("R");
        return client.authorize(resource, action);
    }

}
```

リソース情報をキャッシュする

リソース情報をキャッシュするためには、 ResourceTypeCacheController インタフェースを実装するクラスが必要になります。この実装クラスはリソースタイプごとに定義することができ、複数のリソースタイプで共通の実装を利用することもできます。

リソースのキャッシュコントローラは、リソースURIとリソースグループIDの関係をキャッシュする責務を持っています。キャッシュするサイズやキャッシュをクリアするタイミングなどは、実装や設定を変更することでリソースタイプごとに変化させることができ、リソースタイプにおける特性などからキャッシュの戦略を細かく制御することができます。

ResourceTypeCacheController インタフェース

リソースのキャッシュコントローラは以下のインタフェースを実装する必要があります。

`jp.co.intra_mart.foundation.authz.services.admin.impl.cache.ResourceTypeCacheController`

このインタフェース定義から、このクラスで実装しなければいけないポイントは以下の4つです。

1. 全てのキャッシュをクリアする
2. 一部のキャッシュをクリアする (任意)
3. リソースをキャッシュする
4. キャッシュからリソースを取得する

サンプルアプリケーションでは、アプリサーバのメモリに直接リソースの情報を記憶する実装を定義しています。以下がその実装例です。

```
/**
 * SampleDataResourceTypeCacheController クラス
 * @author INTRAMART
 * @version 8.0.3
 */
public class SampleDataResourceTypeCacheController implements ResourceTypeCacheController {

    /** キャッシュ */
    private static final ConcurrentMap<String, String> cache = new ConcurrentHashMap<String, String>();

    @Override
    public void clearCache() throws ResourceGroupCacheUpdateException {
        // 全てのキャッシュをクリア
        cache.clear();
    }

    @Override
    public void clearCache(final ResourceGroupChangedEvent e) throws ResourceGroupCacheUpdateException {
        // リソースグループが削除された場合のみ
        if (ResourceGroupChangedEvent.EventType.DELETE.equals(e.getEventType())) {
            // 指定されたリソースグループIDに該当するキャッシュをクリア
            for (final Entry<String, String> entry : cache.entrySet()) {
                if (e.getResourceGroupIds().contains(entry.getValue())) {
                    cache.remove(entry.getKey());
                }
            }
        }
    }

    @Override
    public String getResourceGroupId(final String resourceURI) {
        // キャッシュから、リソースURIに対応するリソースグループIDを取得
        return cache.get(resourceURI);
    }

    @Override
    public void setResourceGroupId(final String resourceURI, final String resourceGroupId) {
        // キャッシュに対して、リソースURIに対応するリソースグループIDを設定
        cache.put(resourceURI, resourceGroupId);
    }

}
```

注意

このサンプルではアプリサーバのメモリに直接記憶していますが、アプリサーバを複数台構築した分散環境の場合、全てのサーバに対してキャッシュクリアの要求を行い、その要求を受け取る実装を追加する必要があります。

キャッシュクリアの要求を行わない場合、サーバによってキャッシュの内容に不整合が発生してしまい、接続されたサーバによって認可の判断結果が異なる事象が発生する可能性があります。

キャッシュコントローラを使用するための設定

`%CONTEXT_PATH%/WEB-INF/conf/authz-resource-type-config/` の設定ファイルに、 `cache-class` 属性を追加します。

この詳細については「設定ファイルリファレンス」、または「認可仕様書」も参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<p:authz-resource-type-config
    xmlns:p="http://www.intra-mart.jp/authz/authz-resource-type-config/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.intra-mart.jp/authz/authz-resource-type-config/
../../../../../im_authz_impl/src/main/schema/authz-resource-type-config.xsd">

  <resource-type
      type-class="sample.resource.type.SampleDataResourceType"
      model-class="sample.resource.type.SampleDataModel"
      cache-class="sample.resource.type.SampleDataResourceTypeCacheController" /> <!----- この行を追加 ----------->

</p:authz-resource-type-config>
```

- 属性 cache-class
  - このリソースタイプで使用するキャッシュコントローラクラスを完全修飾クラス名で設定します。

この設定と、設定に書かれたクラスを配置し、再起動すれば認可機構でこのリソースタイプが使用される際にキャッシュ機構を使用することが可能になります。