目次

改訂情報

変更年月日	変更内容
2018-04-01	初版
2018-12-01	第2版 下記を追加・変更しました。 ■ 「*クライアント側のバリデーション*」を追加
2019-04-01	第3版 下記を追加しました。 ■ 「*用途を追加する*」を追加 ■ 「*制約を追加する*」を追加 ■ 「*辞書項目検索*」を追加 ■ 「*エンティティAPI*」を追加

はじめに

本書の目的

本書は、IM-Repository for Accel Platform（以下 IM-Repository）におけるそれぞれの機能を拡張する仕組の詳細および、基本的な使用方法も併せて説明します。

説明範囲は以下のとおりです。

- 辞書項目APIの使用方法について
- 列挙型APIの使用方法について

対象読者

本書では以下のユーザを対象としています。

- IM-Repositoryを利用して処理を実装したい
- IM-Repositoryと連携した機能を実装したい

サンプルコードについて

本書に掲載されているサンプルコードは可読性を重視しており、性能面や保守性といった観点において必ずしも適切な実装ではありません。
開発においてサンプルコードを参考にされる場合には、上記について十分に注意してください。

本書の構成

- *辞書項目API*

  辞書項目APIについて説明します。

- *列挙型API*

  列挙型APIについて説明します。

辞書項目API

IM-Repository上で管理される辞書項目は、項目内に用途や制約といった様々な定義が存在します。 また、辞書項目の集合である辞書はバージョン管理されてもいます。

本項ではこの辞書項目の情報を取得するAPIの利用方法を説明します。

最新バージョン

最新バージョンの辞書を取得する

REST-API

メソッド	GET
URI	%ベース URL%/api/repository/dictionary/current

JavaEE開発モデル

```
// 最新バージョンの辞書を取得
Dictionary dictionary = RepositoryServiceProvider.getInstance().getDictionaryService().getCurrent();

// 辞書のバージョンを確認する
dictionary.getVersion();

// 辞書項目を検索する
Criteria criteria = new Criteria();
criteria.setName("%表示名%");
dictionary.search(criteria);
```

辞書項目

辞書項目を取得する

JavaEE開発モデル

```
Dictionary dictionary = RepositoryServiceProvider.getInstance().getDictionaryService().getCurrent();

// 辞書項目を取得する
Item item = dictionary.find(ItemId.of("%辞書項目のID%"));
if (item != null) {
   // 辞書項目名を取得する
   item.getName();
}
```

辞書項目のエイリアスを取得する

JavaEE開発モデル

```
Dictionary dictionary = RepositoryServiceProvider.getInstance().getDictionaryService().getCurrent();

// 指定された辞書項目のエイリアスを取得する
List<Alias> alias = dictionary.findAlias(ItemId.of("%辞書項目のID%"));
```

エイリアス

項目

- エイリアスを取得する

エイリアスを取得する

JavaEE開発モデル

```
Dictionary dictionary = RepositoryServiceProvider.getInstance().getDictionaryService().getCurrent();

// エイリアスを取得する
Alias alias = dictionary.find(AliasId.of("%エイリアスのID%"));
if (alias != null) {
   // エイリアス名を取得する
   alias.getName();
}
```

用途と制約

項目

- 用途の参照
- 制約の確認方法
- 制約用途に指定した制約に適合するかどうかを確認する

辞書項目やエイリアスそのものは、メタデータとして存在するだけで利用価値はありません。 これらに用途を指定することで、単純なメタデータではなく様々なアプリケーションから 利用できる存在に成ります。

現在用意している用途は

- データ
- 制約

の2種類です。

データ用途は、対象のメタデータの形式、例えばデータベースのスキーマを生成するための情報などを 管理できます。

制約用途は、対象のメタデータのバリデーションを行うための情報を管理できます。

用途の参照

REST-API

メソッド	GET
URI	%ベースURL%/api/repository/dictionary/usages

JavaEE開発モデル

```
Dictionary dictionary = RepositoryServiceProvider.getInstance().getDictionaryService().getCurrent();

// 辞書項目、エイリアスの順に用途を取得する
Item item = dictionary.find(ItemId.of("%辞書項目のID%");
Alias alias = dictionary.find(AliasId.of("%エイリアスのID%");

Usages usages = null;
if (item != null) {
   usages = item.getUsages();
} else if (alias != null) {
   usages = alias.getUsages();
}
```

制約の確認方法

REST-API

メソッド	GET
URI	%ベースURL%/api/repository/dictionary/usages

JavaEE開発モデル

```
Dictionary dictionary = RepositoryServiceProvider.getInstance().getDictionaryService().getCurrent();

// 辞書項目、エイリアスの順に用途を取得する
Item item = dictionary.find(ItemId.of("%辞書項目のID%");
Alias alias = dictionary.find(AliasId.of("%エイリアスのID%");

Usages usages = null;
if (item != null) {
   usages = item.getUsages();
} else if (alias != null) {
   usages = alias.getUsages();
}

// 制約の一覧を取得する
RestrictionUsage restrictionUsage = usages.get("制約用途のID文字列");
List<Restriction> list = restrictionUsage.getRestrictions();
```

制約用途に指定した制約に適合するかどうかを確認する

辞書項目の設定

あらかじめ、次のような設定を行います。これら以外の項目は任意に指定してください。

辞書項目ID	sample-dictionary-item	
用途	制約にチェックを付ける	
数値制約	制約を追加で追加しておく	
	最小値	0
	最大値	100
	最小整数桁	空
	最大整数桁	空
	最小小数桁	空
	最大小数桁	空

JavaEE開発モデル

次のような実装を行うことで、上記の制約に基づいたバリデーションを行うことができます。

```
/**
 * 辞書項目IDと値の組み合わせで以下のような結果が返ります。
 */
public void someMethod() throws DictionaryServiceException {
    this.validate("sample-dictionary-item", 0);   // true
    this.validate("sample-dictionary-item", 100); // true
    this.validate("sample-dictionary-item", -1);  // false
    this.validate("sample-dictionary-item", 101); // false
}

/**
 * 指定した辞書項目の制約を用いて、対象の値が適合するかどうかを確認します。
 *
 * @param itemId 辞書項目ID
 * @param target 対象の値。ここでは数値が制約に合致するかどうかを確認することとします。
 * @return 適合する場合は true を、適合しない場合は false を返します。
 * @throws DictionaryServiceException 辞書サービスでエラーが発生した場合にスローされます。
 */
public boolean validate(final String itemId, final Double target) throws DictionaryServiceException {
    final Dictionary dictionary = RepositoryServiceProvider.getInstance().getDictionaryService().getCurrent();

    final Item item = dictionary.find(ItemId.of(itemId));
    if (item != null) {
        final Usages usages = item.getUsages();
        final RestrictionUsage restrictionUsage = usages.get(UsageId.of(RestrictionUsage.ID));
        for (final Restriction restriction : restrictionUsage.getRestrictions()) {
            // validate の第一引数はバリデーションエラーのメッセージに含める名称です。
            // 現在提供している制約のエラーメッセージでは使用されていません。
            final ValidateResult validationResult = restriction.validate("名前", target);
            if (validationResult.getInvalidParams().size() > 0) {
                return false;
            }
        }
    }
    return true;
}
```

用途を追加する

標準で提供されているデータ用途、制約用途ではメタデータとして不足している場合、用途そのものを追加できます。

例として、辞書項目を画面へ表示する際の位置を用途として追加してみます。

用途の項目として

- x座標 （変数名：x）
- y座標 （変数名：y）
- 水平方向への配置位置 （変数名：align）

を持つものとします。

また、水平方向への配置位置はエイリアスが参照している辞書項目とは別の値を持てるものとします。

用途の実装

用途を増やすには、クライアント側とサーバ側の両方の実装が必要です。

- クライアント側
  - Vue.js のコンポーネント
- サーバ側
  - 用途の項目を表現するクラス
  - インポート・エクスポート時の設定、処理を行うクラス
  - 設定ファイル
  - 影響範囲検出のためのクラス

それぞれの実装で共通となる ID が必要です。この ID はシステムで一意とならなければなりません。 未知の用途が追加されることを考慮し、FQDN やモジュールIDなど競合しない文字列を検討してください。

この例では簡単のため sample_usage を用途のIDとします。

クライアント側の実装

クライアント側は、Vue.js のコンポーネントとして実装します。 「Vue.js のコンポーネント」の詳細は Vue.js の Web サイトを参照してください。

用途のコンポーネントとして必要なのは以下の項目です。

- id
  - 上記「用途のID」を指定します。この文字列でサーバ側とクライアント側を紐づけます。
- aliasEditable
  - エイリアスが参照元の辞書項目とは別の値を持てる項目のキーを指定します。
- inject
  - 親コンポーネントから注入される関数を定義します。
- template
  - 画面に表示する HTML のテンプレートを定義します。
  - 辞書項目画面のツリー形式、表形式、差分表示の3つに対応するように定義します。

- props
  - dictionaryItem: { type: Object, default: function() { return {} }
    - 辞書項目がこのプロパティ名で引き渡されます。
  - anotherItem: { type: Object, default: function() { return {} }
    - 辞書項目がこのプロパティ名で引き渡されます。
    - このプロパティは、差分表示画面で使用されます。
  - readonly: { type: Boolean, default: false }
    - このプロパティは、表敬式画面などで使用されます。
- computed
  - 入力データ
    - computed のゲッター、セッターを定義します。
    - セッターでは、このコンポーネントを呼び出している親コンポーネントにデータの変更を伝えます。

これらを踏まえて実装します。

src/main/public/im_repository/sample.js

```
//
// 表示形式用途のコンポーネントを定義します。
//
const SampleUsage = {
 // 「用途のID」を指定します。この文字列でサーバ側とクライアント側を紐づけます。
 id: 'sample_usage',
 // エイリアスが参照元の辞書項目とは別の値を持てる項目のキーを指定します。
 aliasEditable: [
  'align'
 ],
 // 親コンポーネントから注入される関数を定義します。
 inject: ['isEditable', 'getUsageData', 'getUsageDataValue', 'getAnotherUsageDataValue', 'hasDifference'],
 // 画面に表示する HTML のテンプレートを定義します。
 template:
  '<usage usage-id="sample_usage" title="表示形式">' +
   '<usage-table>' +
    '<usage-table-row usage-id="sample_usage" title="x座標" property-key="x">' +
     '<input v-if="isEditable()" name="x" type="text" v-model="x">' +
     // 編集不可の場合は label で表示する
     '<label v-else>{{ x }}</label>' +
    '</usage-table-row>' +
    '<usage-table-row usage-id="sample_usage" title="y座標" property-key="y">' +
     '<input v-if="isEditable()" name="y" type="text" v-model="y">' +
     '<label v-else>{{ y }}</label>' +
    '</usage-table-row>' +
    '<usage-table-row usage-id="sample_usage" title="水平方向への配置位置" property-key="align">' +
     '<input v-if="isEditable()" name="align" type="text" v-model="align">' +
     '<label v-else>{{ align }}</label>' +
    '</usage-table-row>' +
   '</usage-table>' +
  '</usage>',
 computed: {
  x: {
   get() {
    return _.get(this.getUsageData(this.usageId), 'x', null)
   },
   set(value) {
    this.$emit('merge', {
     usageId: this.usageId,
     usageData: { x: value }
    })
   }
  },
  y: {
   get() {
    return _.get(this.getUsageData(this.usageId), 'y', null)
   },
   set(value) {
```

```
      this.$emit('merge', {
       usageId: this.usageId,
       usageData: { y: value }
      })
     }
    },
    align: {
     get() {
      return _.get(this.getUsageData(this.usageId), 'align', null)
     },
     set(value) {
      this.$emit('merge', {
       usageId: this.usageId,
       usageData: { align: value }
      })
     }
    }
   },
   props: {
    // 読み込み専用かどうかのフラグ
    readonly: {
     type: Boolean,
     default: false
    }
   },
   data() {
    return {
     usageId: SampleUsage.id
    }
   }
}

//
// コンポーネントを追加します。
//
if (window.imRepository && imRepository.usageComponents) {
 window.imRepository.usageComponents.push(SampleUsage)
} else if (window.imRepository) {
 window.imRepository.usageComponents = [ SampleUsage ]
} else {
 window.imRepository = {
  usageComponents: [ SampleUsage ]
 }
}
```

親コンポーネントから渡される情報

親コンポーネントから props へ渡されるものは次のものです。 上記例では readonly しか使用していません。

名前	型	説明
dictionaryItem	Object	辞書項目またはエイリアス
anotherItem	Object	差分表示時に使用される辞書項目またはエイリアス
readonly	Boolean	表形式や差分表示など読み込み専用を示すフラグ

親コンポーネントから provide される関数は次のものです。

名前	引数	返り値	説明
isEditable()			この項目が編集可能かどうかを返す関数です。
		Boolean	編集可能な場合 true を、編集不可能な場合 false を返します。
hasDifference(one, another)			指定した2つのオブジェクトに差分があるかどうかを返す関数です。

名前	引数	返り値	説明
	{Object} one		差があるかどうかを確認するオブジェクト
	{Object} another		差があるかどうかを確認するもう一つのオブジェクト
		Boolean	差がある場合 true を、差がない場合 false を返します。
getUsageData(usageId, defaults)			辞書項目から、指定した用途IDの用途のデータを返します。
	{String} usageId		用途ID
	{Object} [defaults]		用途が取得できなかった場合の値。オプショナル
		Object	用途のデータを返します。該当するものがなければ null
getUsageDataValue(usageId, key)			辞書項目から、指定した用途IDの用途のデータから指定したキーに 該当する値を返します。
	{String} usageId		用途ID
	{String} key		用途のデータのキー
		Object	用途のデータの値を返します。 該当するものがなければ null を返します。
getAnotherUsageDataValue(usageId, key)			差分表示の際に他方の辞書項目から、指定した用途IDの用途の データから指定したキーに該当する値を返します。
	{String} usageId		用途ID
	{String} key		用途のデータのキー
		Object	用途のデータの値を返します。 該当するものがなければ null を返します。

また、この他に親コンポーネントから provide されるオブジェクトも存在します。内部で使用しています。

名前	型	説明
dictionaryItem	Object	辞書項目またはエイリアス
anotherItem	Object	差分表示時に使用される辞書項目またはエイリアス
readonly	Boolean	表形式や差分表示など読み込み専用を示すフラグ
eventHub	Object	削除ボタンをクリックしたときのイベントをやりとりするオブジェクト

親コンポーネントで定義されているコンポーネント

usage コンポーネント

このコンポーネントは、標準の用途と同じ見た目を提供するものです。 必ず使わなければならないものではありませんが、見た目の調整などが軽減されるので利用をお勧めします。

div タグとして展開されます。また、内部に form タグを内包しています。

usage コンポーネントでは以下のプロパティを要求します。これらの値を指定してください。

名前	型	説明
id	String	div の id。div の内部に生成される form の id にも使用されます。form の id は、この id に _form をつけたものです。
title	String	この用途の見出し名を指定します。

usage-table コンポーネント

このコンポーネントは、標準の用途と同じ見た目を提供するものです。 必ず使わなければならないものではありませんが、見た目の調整などが軽減されるので利用をお勧めします。 table タグとして展開されます。また、内部に tbody タグを内包しています。

table タグには imui-form クラスが指定されています。 別なクラスを指定したい場合は、 `<usage-table class="some_class">` のように class 属性を指定してください。

usage-table-row コンポーネント

このコンポーネントは、標準の用途と同じ見た目を提供するものです。 必ず使わなければならないものではありませんが、見た目の調整などが軽減されるので利用をお勧めします。 tr タグとして展開されます。また、内部に差分の有無を示すセル、行のヘッダを示すセル、用途のプロパティを表示するセルの3つのセルを内包しています。

usage-table-row コンポーネントでは以下のプロパティを要求します。これらの値を指定してください。

名前	型	説明
usage-id	String	用途の id。
title	String	この用途のプロパティの名前を指定します。
property-key	String	この用途のプロパティのキーを指定します。

サーバ側の実装

サーバ側の実装で必要なのは以下の4つです。

- 用途の項目を表現するクラスの実装
- インポート・エクスポート時の設定、処理を行うクラスの実装
- 設定ファイルの追加
- 影響範囲検出のためのクラス

用途の項目を表現するクラスの実装

このクラスの実装で大事なことは、用途の項目をどのようなデータ構造として扱うのか？を決めることです。
jp.co.intra_mart.foundation.repository.metadata.dictionary.usage.Usage を実装したクラスとして実装します。

クライアント側とは JSON 形式でデータのやり取りをします。 上記のクライアン側の例では

```
return {
  x: '',
  y: '',
  align: ''
}
```

となっているので、サーバ側では Map<String, Object> としてデータを扱います。

また、align は参照先の辞書項目とは別の値を持てるとしたので、AliasEditable アノテーションを付けます。 もしこのアノテーションをつけ忘れると、画面で入力した値が、参照先の辞書項目の値で書き換えられた上で保存されます。

src/main/java/sample/usage/SampleUsage.java

```
public class SampleUsage implements Usage {
    /**
     * この用途のIDです。
     */
    public static final String USAGE_ID = "sample_usage";

    private static final long serialVersionUID = 8444395879883374534L;

    // 用途の項目を格納する Map です。
    private final Map<String, Object> usageData = new HashMap<>();

    // x座標
```

```
    private String x;

    // y座標
    private String y;

    // 水平方向への配置位置
    @AliasEditable
    private String align;

    /**
     * 用途の表示名です。
     * 辞書項目の右側、用途の一覧にチェックボックスとともに表示されます。
     */
    @Override
    public String getName() {
        return "表示位置";
    }

    /**
     * 用途のIDを返します。
     */
    @Override
    public UsageId getUsageId() {
        return UsageId.of(USAGE_ID);
    }

    /**
     *  用途の項目データを返します。
     */
    @Override
    public Object getUsageData() {
        usageData.put("x", this.x);
        usageData.put("y", this.y);
        usageData.put("align", this.align);
        return usageData;
    }

    /**
     * 用途の項目データを設置します。
     * このメソッドは画面からと、永続化層の双方からセットされます。
     */
    @Override
    public void setUsageData(final Object data) {
        if (data != null && data instanceof Map<?, ?>) {
            @SuppressWarnings("unchecked")
            final Map<String, Object> map = (Map<String, Object>) data;

            this.x = Optional.ofNullable(map.get("x")).map(String.class::cast).orElse(StringUtil.EMPTY_STRING);
            this.y = Optional.ofNullable(map.get("y")).map(String.class::cast).orElse(StringUtil.EMPTY_STRING);
            this.align = Optional.ofNullable(map.get("align")).map(String.class::cast).orElse(StringUtil.EMPTY_STRING);
        }
    }

    /**
     * クライアント側の JavaScript 実装のパスを返します。
     */
    @Override
    public String[] getScriptPathList() {
        return new String[] { "im_repository/sample.js" };
    }

    /**
     * x座標を返します。
     * @return x座標
     */
    public String getX() {
        return x;
    }

    /**
```

```
    * y座標を返します。
    * @return y座標
    */
   public String getY() {
      return y;
   }

   /**
    * 水平位置を返します。
    * @return 水平位置
    */
   public String getAlign() {
      return align;
   }
}
```

インポート・エクスポート時の設定、処理を行うクラスの実装

インポート・エクスポートで利用されるクラスです。用途の各フィールドに対してインポート・エクスポートで使用するキーを定義します。 jp.co.intra_mart.system.repository.metadata.dictionary.import_export.StandardUsageSerializer を継承したクラスとして実装します。

```
/** x座標のインポート・エクスポートに使用するキーです。 */
private static final String SAMPLE_USAGE_X = "sample_usage_x";

/** y座標のインポート・エクスポートに使用するキーです。 */
private static final String SAMPLE_USAGE_Y = "sample_usage_y";

/** 水平方向への配置位置のインポート・エクスポートに使用するキーです。 */
private static final String SAMPLE_USAGE_ALIGN = "sample_usage_align";
```

このキーは後述する *インポート・エクスポートで使用する設定ファイルの追加* でも利用し、システムで一意とならなければなりません。 未知の用途、制約が追加されることを考慮し、FQDN やモジュールIDなど競合しない文字列を検討してください。 またシステムで利用している設定ファイルのキーにも競合しないように注意してください。

システムで利用している設定ファイルの配置場所は「WEB-INF/conf/im-repository-dictionary-config-type/im-repository-dictionary-config-type.xml」です。

getKeySettingsメソッドでインポート・エクスポートで使用するキーと用途の各フィールドの対応を設定することにより、エクスポート時の出力、インポート時の取込に対応させることができます。

```
public KeySetting[] getKeySettings() {
   return new KeySetting[] {
      KeySetting.of(SAMPLE_USAGE_X, "x"),
      KeySetting.of(SAMPLE_USAGE_Y, "y"),
      KeySetting.of(SAMPLE_USAGE_ALIGN, "align")};
}
```

validateメソッドはインポート時のインポート確認画面に出力されるエラーメッセージ、ワーニングメッセージを設定できます。 エラーメッセージが含まれる場合はインポートを継続できないので、内容に応じて設定してください。

src/main/java/sample/usage/SampleUsageSerializer.java

```
public class SampleUsageSerializer extends StandardUsageSerializer {

    /** x座標のインポート・エクスポートに使用するキーです。 */
    private static final String SAMPLE_USAGE_X = "sample_usage_x";

    /** y座標のインポート・エクスポートに使用するキーです。 */
    private static final String SAMPLE_USAGE_Y = "sample_usage_y";

    /** 水平方向への配置位置のインポート・エクスポートに使用するキーです。 */
    private static final String SAMPLE_USAGE_ALIGN = "sample_usage_align";

    /**
     * XMLの設定値とフィールド名の組み合わせを返します。
     */
    @Override
    public List<KeySetting> getKeySettings() {
        final List<KeySetting> list = new ArrayList<>();
        list.add(KeySetting.of(SAMPLE_USAGE_X, "x"));
        list.add(KeySetting.of(SAMPLE_USAGE_Y, "y"));
        list.add(KeySetting.of(SAMPLE_USAGE_ALIGN, "align"));
        return list;
    }

    /**
     * 用途タイプを返却します。
     * 用途のclassを指定してください。
     */
    @SuppressWarnings("unchecked")
    @Override
    public Class<Usage> getSupportType() {
        return (Class<Usage>) (Object) SampleUsage.class;
    }

    /**
     * インポート時のバリデーションを行います。
     */
    @Override
    public void validate(final Class<?> element, final ImportDataModel importData) {
        // 入力チェックを行う対象
        final String key = SAMPLE_USAGE_X;

        if (!isEditable(element, key)) {
            // 編集不可の場合は値が設定されていないためバリデーションを行いません。
            return;
        }

        // インポートデータから値を取得
        final String x = getImportDataValue(importData.get(key));

        // バリデーション内容
        if (x != null && x.isEmpty()) {
            // エラーメッセージの追加例
            setErrorMessage("必須入力項目です。");
        } else {
            try {
                // 数値化
                Integer.valueOf(x);
            } catch (final NumberFormatException e) {
                // ワーニングメッセージの追加例
                setWarningMessage("数値ではありません。");
            }
        }
    }
}
```

設定ファイルの追加

実装したクラスをシステムに読み込ませるための設定ファイルの追加

実装したクラスをシステムに読み込ませるための設定ファイルを追加します。 設定ファイルは java.util.ServiceLoader で読み込まれます。

src/main/resources/META-INF/services/jp.co.intra_mart.foundation.repository.metadata.dictionary.usage.Usage

```
sample.usage.SampleUsage
```

src/main/resources/META-INF/services/jp.co.intra_mart.system.repository.metadata.dictionary.import_export.usage.UsageSerializer

```
sample.usage.SampleUsageSerializer
```

インポート・エクスポートで使用する設定ファイルの追加

下記の例は sample_usage と sample_usage の各項目の入出力設定です。

<excelConfig kind="dictionary_usage"> は用途の使用・不使用の欄を表します。 今回のサンプルの用途向けに記述を追加します。

<excelConfig kind="sample_usage"> は今回のサンプルの用途の各項目向けの欄を表します。 今回のサンプルの用途の各項目の記述を追加します。

src/main/conf/im-repository-dictionary-config-type/sample-usage.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<im-repository-dictionary-config-type
   xmlns="http://www.intra-mart.jp/repository/im-repository-dictionary-config-type"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/repository/im-repository-dictionary-config-type im-repository-dictionary-config-type.xsd ">
   <sheetConfig kind="dictionary">
      <excelConfig kind="dictionary_usage">
         <columnData>
            <key>sample_usage</key>
            <index>20200</index>
            <validationAddress>inputlist!$B$1:$B$3</validationAddress>
            <editDisable>Alias</editDisable>
            <line no="1">
               <name></name>
               <color>TAN</color>
               <borderTop>THIN</borderTop>
               <borderBottom>THIN</borderBottom>
               <borderLeft></borderLeft>
               <borderRight></borderRight>
            </line>
            <line no="2">
               <name>サンプル用途</name>
               <color>TAN</color>
               <borderTop>THIN</borderTop>
               <borderBottom>THIN</borderBottom>
               <borderLeft>THIN</borderLeft>
               <borderRight>THIN</borderRight>
            </line>
         </columnData>
      </excelConfig>
      <excelConfig kind="sample_usage">
         <columnData>
            <key>sample_usage_x</key>
            <index>50000</index>
            <validationAddress></validationAddress>
            <editDisable>Alias</editDisable>
            <line no="1">
               <name>表示位置</name>
               <color>GREY_25_PERCENT</color>
               <borderTop>THIN</borderTop>
               <borderBottom>THIN</borderBottom>
               <borderLeft></borderLeft>
               <borderRight></borderRight>
            </line>
```

```
        <line no="2">
          <name>x座標</name>
          <color>GREY_25_PERCENT</color>
          <borderTop>THIN</borderTop>
          <borderBottom>THIN</borderBottom>
          <borderLeft>THIN</borderLeft>
          <borderRight>THIN</borderRight>
        </line>
      </columnData>
      <columnData>
        <key>sample_usage_y</key>
        <index>50100</index>
        <validationAddress></validationAddress>
        <editDisable>Alias</editDisable>
        <line no="1">
          <name></name>
          <color>GREY_25_PERCENT</color>
          <borderTop>THIN</borderTop>
          <borderBottom>THIN</borderBottom>
          <borderLeft></borderLeft>
          <borderRight></borderRight>
        </line>
        <line no="2">
          <name>y座標</name>
          <color>GREY_25_PERCENT</color>
          <borderTop>THIN</borderTop>
          <borderBottom>THIN</borderBottom>
          <borderLeft>THIN</borderLeft>
          <borderRight>THIN</borderRight>
        </line>
      </columnData>
      <columnData>
        <key>sample_usage_align</key>
        <index>50200</index>
        <validationAddress></validationAddress>
        <editDisable></editDisable>
        <line no="1">
          <name></name>
          <color>GREY_25_PERCENT</color>
          <borderTop>THIN</borderTop>
          <borderBottom>THIN</borderBottom>
          <borderLeft></borderLeft>
          <borderRight>THIN</borderRight>
        </line>
        <line no="2">
          <name>水平位置</name>
          <color>GREY_25_PERCENT</color>
          <borderTop>THIN</borderTop>
          <borderBottom>THIN</borderBottom>
          <borderLeft>THIN</borderLeft>
          <borderRight>THIN</borderRight>
        </line>
      </columnData>
    </excelConfig>
  </sheetConfig >
</im-repository-dictionary-config-type>
```

タグの説明

sheetConfigタグ

Excel ファイルのシートを表現します。 kind属性で出力するシート名を指定します。 指定できる値は **category** または **dictionary** のいずれかです。 今回の例ではdictionaryシートに出力したいので、dictionary を指定します。

```
<sheetConfig kind="dictionary">
```

excelConfigタグ

kind 属性を指定することで、列のグループを表現します。 kind属性 dictionary_usage は用途の使用・不使用を表す列のグループです。 今回の例では、追加したsample_usageを出力するための設定を追加します。

```
<excelConfig kind="dictionary_usage">
```

kind属性に用途IDを指定すると、その用途IDが示す用途を表す列のグループとして扱われます。 今回の例では、追加した用途を表す列のグループになり、用途の項目を columnData として追加していきます。

```
<excelConfig kind="sample_usage">
```

columnDataタグ

Excel に対する出力設定を行うタグの親タグです。 属性情報はありません。

keyタグ

追加した用途のID、用途の項目のキーを設定します。

```
<!-- サンプル用途の使用・不使用の列を追加します -->
<excelConfig kind="dictionary_usage">
 <columnData>
  <!-- 用途IDを指定します -->
  <key>sample_usage</key>
  ...
```

用途の各項目のキーは、*インポート・エクスポート時の設定、処理を行うクラスの実装* で定義したインポート・エクスポートで使用するキーを指定します。

```
<!-- サンプル用途の各項目を列を追加します -->
<excelConfig kind="sample_usage">
 <columnData>
  <key>sample_usage_x</key>
  ...
 <columnData>
  <key>sample_usage_y</key>
  ...
 <columnData>
  <key>sample_usage_align</key>
  ...
```

indexタグ

出力位置を示す値です。 excelConfig タグ内での位置を示す値です。

```
<index>20200</index>
```

validationAddressタグ

Excel での入力規則を示す値です。 Excel 上で利用する値を設定してください。

```
<validationAddress>inputlist!$B$1:$B$3</validationAddress>
```

入力規則が必要ない場合は値を空にしてください。

```
<validationAddress></validationAddress>
```

editDisableタグ

Excel での編集不可を示す値です。 編集不可の場合はAliasを指定してください。編集可能な場合は、空のタグを指定するか、タグ自体を書かないでください。

```
<editDisable>Alias</editDisable>
```

lineタグ

Excel でのヘッダ行の出力位置を指定します。ヘッダ行は2行で構成されています。 1行目に出力したい場合は 1 を、2行目に出力したい場合は 2 を指定してください。

```
<line no="1">
```

nameタグ

カラムの名称を指定してください。

```
<name>表示位置</name>
```

多言語化が必要な場合は、多言語対応 を参考にメッセージプロパティを定義した上でメッセージコードを指定してください。

```
<name>I18N.MESSAGE.EXAMPLE</name>
```

colorタグ

セルの背景色を設定できます。 指定できる値は org.apache.poi_v3_9.ss.usermodel.IndexedColors の値です。

```
<color>TAN</color>
```

背景色が必要ない場合は値を空にしてください。

```
<color></color>
```

borderTopタグ、borderBottomタグ、borderLeftタグ、borderRightタグ

セルの罫線を設定できます。 指定できる値は org.apache.poi_v3_9.ss.usermodel.CellStyle の値です。

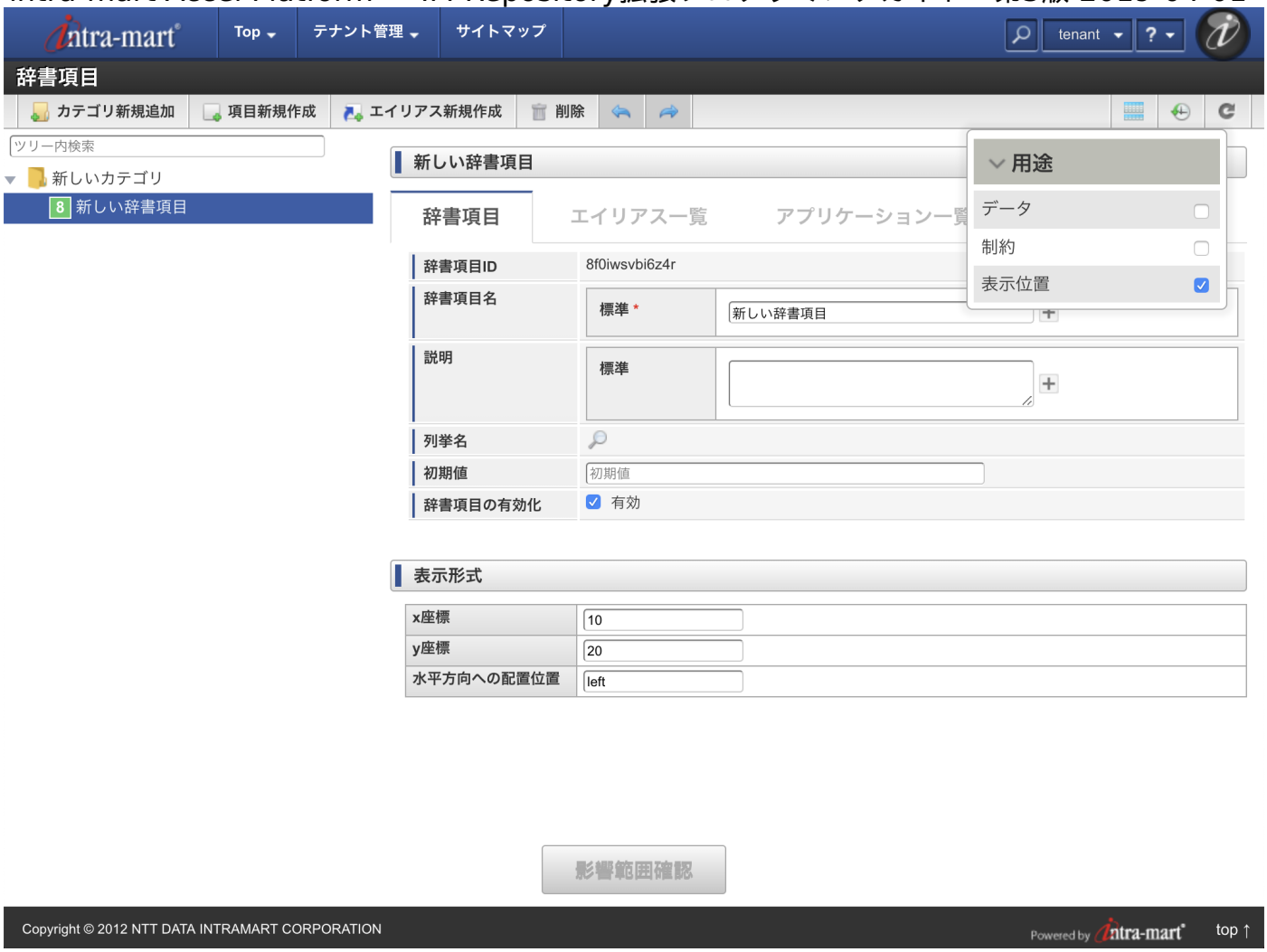
```
<borderTop>THIN</borderTop>
<borderBottom>THIN</borderBottom>
<borderLeft>THIN</borderLeft>
<borderRight>THIN</borderRight>
```

罫線が必要ない場合は値を空にしてください。

```
<borderTop></borderTop>
<borderBottom></borderBottom>
<borderLeft></borderLeft>
<borderRight></borderRight>
```

影響範囲検出のためのクラス

これらの実装を行い、モジュールのビルドや war ファイルのデプロイを行うと、次のような画面が現れ、用途が追加されていることがわかります。



制約を追加する

標準で提供されている制約では不足している場合、制約そのものを追加できます。

例として、日付書式であることを制約として追加してみます。（正規表現制約を使えば同じことができますが、実装例として参考にしてください）

制約の項目として

- 日付書式（変数名：format）

を持つものとします。

制約の実装

制約を増やすには、サーバ側とクライアント側の両方の実装が必要です。

- クライアント側
  - Vue.js のコンポーネント
- サーバ側
  - 制約を表現するクラスをインスタンス化するクラス
  - 制約を表現するクラス
  - インポート・エクスポート時の設定、処理を行うクラス
  - 設定ファイル
  - 影響範囲検出のためのクラス

それぞれの実装で共通となる ID が必要です。この ID はシステムで一意とならなければなりません。 未知の制約が追加されることを考慮し、FQDN やモジュールIDなど競合しない文字列を検討してください。

この例では簡単のため sample_restriction を制約のIDとします。

クライアント側の実装

クライアント側は、Vue.js のコンポーネントとして実装します。 「Vue.js のコンポーネント」の詳細は Vue.js の Web サイトを参照してください。

制約のコンポーネントとして必要なのは以下の項目です。

- id
  - 上記「制約のID」を指定します。この文字列でサーバ側とクライアント側を紐づけます。
- label
  - 制約の選択肢となるセレクトボックスに表示する名前を定義します。
- sortOrder
  - 制約の選択肢となるセレクトボックスに表示するソート順を定義します。
- inject
  - 親コンポーネントから注入される関数、プロパティを定義します。
- template
  - 画面に表示する HTML のテンプレートを定義します。
- computed
  - 入力データ
    - computed のゲッター、セッターを定義します。
    - セッターでは、このコンポーネントを呼び出している親コンポーネントにデータの変更を伝えます。
    - また、制約のデータは id をキーに持つオブジェクトとして定義します。

これらを踏まえて実装します。

src/main/public/im_repository/sample_restriction.js

```
//
// 日付書式制約のコンポーネントを定義します。
//
const SampleRestriction = {
 // 「制約のID」を指定します。この文字列でサーバ側とクライアント側を紐づけます。
 id: 'sample_restriction',
 // 制約の選択肢に表示する文字列
 label: '日付書式',
 // 制約の選択肢のソート番号
 sortOrder: 2000,
 // 親コンポーネントから注入される関数
 inject: ['isEditable', 'getRestriction'],
 // 画面に表示する HTML のテンプレートを定義します。
 template:
  // デフォルトの制約と同じような見た目で制約を表示します。
  '<restriction ' +
   // HTML の id 属性です。 <div id="odd_or_even_restriction"...> のように展開されます。
   'id="sample_restriction" ' +
```

```
    // 種類です。
    ':type="restrictionLabel" ' +
    // 制約IDです。
    ':restrictionId="restrictionId"> ' +

    // 制約のプロパティを表示します。
    '<restriction-property ' +
     // 制約IDです。
     ':restrictionId="restrictionId"' +
     // 制約のプロパティのIDです。 <tr id="option"..> のように展開されます。
     'id="format"' +
     // 制約のプロパティのキーです。
     'property-key="format">' +

     // 制約のプロパティのタイトルです。
     // '<th slot="tableHeader">日付書式</th>' +

     // ここからは <td> の内部として展開されます。
     '<td slot="propertyCell">' +
      '<label>書式：</label>' +
      '<input ' +
       'v-if="isEditable()" ' +
       'v-model="format" ' +
       'name="format" ' +
       'type="text">' +
      '<label v-else>{{ format }}</label>' +
     '</td>' +
    '</restriction-property>' +
  '</restriction>',
 data() {
  return {
   restrictionId: SampleRestriction.id,
   restrictionLabel: SampleRestriction.label
  }
 },
 props: {
  dictionaryItem: {
   type: Object,
   required: true
  }
 },
 computed: {
  // この制約の項目
  format: {
   get() {
    // 辞書項目から制約用途を取得、制約用途の中から this.restrictionId の制約を取得、その成約の中から format プロパティを取得する
    return _.get(this.getRestriction(this.dictionaryItem, this.restrictionId), 'format', '')
   },
   set(value) {
    this.$emit('merge', { id: this.restrictionId, format: value })
   }
  }
 }
}

//
// コンポーネントを追加します
//
if (window.imRepository && window.imRepository.restrictions) {
 window.imRepository.restrictions.push(SampleRestriction)
} else if (window.imRepository) {
 window.imRepository.restrictions = [ SampleRestriction ]
} else {
 window.imRepository = {
  restrictions: [SampleRestriction]
 }
}
```

親コンポーネントから渡される情報

親コンポーネントから props へ渡されるものは次のものです。 上記例では dictionaryItem しか使用していません。

名前	型	説明
dictionaryItem	Object	辞書項目またはエイリアス
anotherItem	Object	差分表示時に使用される辞書項目またはエイリアス
readonly	Boolean	表形式や差分表示など読み込み専用を示すフラグ

親コンポーネントから provide される関数は *親コンポーネントから渡される情報* に加えて次のものがあります。

名前	引数	返り値	説明
hasDifferencePropert(restrictionId, key)			指定した制約IDと制約データのキーに該当するデータに 差分があるかどうかを返す関数です。
	{String} restrictionId |		制約ID
	{String} key		制約データのキー
		Boolean	差がある場合 true を、差がない場合 false を返します。
getRestriction(item, restrictionId)			指定した制約IDの制約を取得する関数です。
	{Object} item		辞書項目またはエイリアス
	{String} restrictionId		制約ID
		Object	制約のオブジェクトを返します。 指定した制約IDの制約が存在しない場合 nullを返します。

また、この他に親コンポーネントから provide されるオブジェクトも存在します。内部で使用しています。

名前	型	説明
dictionaryItem	Object	辞書項目またはエイリアス
anotherItem	Object	差分表示時に使用される辞書項目またはエイリアス
readonly	Boolean	表形式や差分表示など読み込み専用を示すフラグ
eventHub	Object	削除ボタンをクリックしたときのイベントをやりとりするオブジェクト

親コンポーネントで定義されているコンポーネント

restriction コンポーネント

このコンポーネントは、標準の制約と同じ見た目を提供するものです。 div タグとして展開されます。また、内部に form, table タグを内包しています。

restriction コンポーネントでは以下のプロパティを要求します。これらの値を指定してください。

名前	型	説明
id	String	div の id。div の内部に生成される form の id にも使用されます。form の id は、この id に _form をつけたものです。
type	String	この制約の種類の表示名を指定します。
restrictionId	String	制約IDを指定します。

restriction-property コンポーネント

このコンポーネントは、制約のプロパティをテーブルの行として表示するためのものです。 tr タグとして展開されます。また、内部に差分の有無を示すセル、行のヘッダを示すセル、制約のプロパティを表示するセルの3つのセルを内包しています。

restriction コンポーネントでは以下のプロパティを要求します。これらの値を指定してください。

名前	型	説明
id	String	tr の id。
property-key	String	この制約のプロパティのキーを指定します。
restrictionId	String	制約IDを指定します。

サーバ側の実装

サーバ側の実装で必要なのは以下の4つです。

- 制約を表現するクラスをインスタンス化するクラス
- 制約を表現するクラス
- インポート・エクスポート時の設定、処理を行うクラス
- 設定ファイル
- 影響範囲検出のためのクラス

制約を表現するクラスをインスタンス化するクラス

このクラスの役割は、

- システムに前述のクライアント側のスクリプトパスを知らせること
- 制約用途のインスタンス化の際に、制約のインスタンスを生成すること

です。 jp.co.intra_mart.foundation.repository.metadata.dictionary.restriction.RestrictionFactory を実装したクラスとして実装します。

ScriptPath アノテーションを使い、クライアント側のスクリプトを配置するパスを指定します。

この例では im_repository/sample_restriction.js というファイルを使用します。

```
@ScriptPath(value = "im_repository/sample_restriction.js")
```

制約のインスタンスを生成する際、制約用途は制約の情報を制約のコンストラクタに渡します。 上記のクライアン側の例では

```
return {
  id: this.restrictionId,
  format: value
}
```

としました。

クライアント側からサーバ側に渡される情報を、Map<String, Object> として扱います。 逆にサーバ側からクライアント側へ渡す情報は、Restriction として扱います。

src/main/java/sample/restriction/SampleDateFormatRestrictionFactory.java

```
// クライアント側のスクリプトパスを指定します。
@ScriptPath(value = "im_repository/sample_restriction.js")
public class SampleDateFormatRestrictionFactory implements RestrictionFactory {

    @Override
    public String getId() {
        return SampleDateFormatRestriction.ID;
    }

    // クライアント側からサーバ側へ渡す情報を生成します。
    @Override
    public Restriction create(final Map<String, Object> restrictionData) throws DictionaryServiceException {
        final String format = (String) restrictionData.get("format");
        return new SampleDateFormatRestriction(format);
    }

    // サーバ側からクライアント側へ渡す情報を生成します。
    @Override
    public Map<String, Object> create(final Restriction restriction) throws DictionaryServiceException {
        final SampleDateFormatRestriction r = (SampleDateFormatRestriction) restriction;
        final Map<String, Object> map = new HashMap<>();
        map.put("id", this.getId());
        map.put("format", r.getFormat());
        return map;
    }
}
```

制約を表現するクラス

このクラスの役割は、Factory から渡されたパラメータを保持し、そのパラメータで制約の確認を行うことです。

src/main/java/sample/restriction/SampleDateFormatRestriction.java

```
public class SampleDateFormatRestriction extends AbstractRestriction {

    /** 制約ID */
    public static final String ID = "sample_restriction";

    private final String format;

    /**
     * 空のコンストラクタ。
     * ServiceLoader でインスタンス化するのに必須です。
     */
    public SampleDateFormatRestriction() {
        this.format = StringUtil.EMPTY_STRING;
    }

    /**
     * コンストラクタ
     * @param format 日付書式
     */
    public SampleDateFormatRestriction(final String format) {
        this.format = format;
    }

    @Override
    public String getId() {
        return ID;
    }

    @Override
    public String getName() {
        return "日付書式";
    }

    // value がこの制約に適合するかどうかを確認します。
    // 制約は複数個チェックされる可能性があるので、Boolean 等を返さず、適合しないものを ValidateResult に追加する、という方法を採っています。
    @Override
    protected void doValidate(final String name, final Object value, final ValidateResult result) {
        if (!(value instanceof CharSequence)) {
            result.getInvalidParams().add(new InvalidParam(name, MetadataMsg.MSG_E_IWP_MESSAGE_REPOSITORY_METADATA_DICTIONARY_RESTRICTION_NOT_COVERED.get()));
            return;
        }

        final String str = value.toString();
        final Pattern pattern = Pattern.compile(this.format);
        final Matcher matcher = pattern.matcher(str);
        if (matcher.matches()) {
            result.getInvalidParams().add(new InvalidParam(name, "書式に適合しません"));
            return;
        }
        return;
    }

    /**
     * 日付書式を返します。
     * @return 日付書式
     */
    public String getFormat() {
        return format;
    }
}
```

インポート・エクスポート時の設定、処理を行うクラス

インポート、エクスポートで利用されるクラスです。
jp.co.intra_mart.system.repository.metadata.dictionary.import_export.StandardRestrictionSerialize を継承したクラスとして

実装します。

エクスポート処理

制約の内容は 1 つのセルに出力されます。項目が複数ある場合は、カンマなどで結合した文字列として出力してください。 今回の例では制約の項目が 1 つのみで空白入力を許しています。 制約が追加されたことを示すboolean値と、制約の項目を結合した文字列を出力します。

```
public String serialize(final Restriction restriction) {
    final SampleDateFormatRestriction sample = (SampleDateFormatRestriction) restriction;
    final StringBuilder param = new StringBuilder();
    // このメソッドが呼び出されるのは、この制約が有効になっているためです。
    // ですので出力する有効・無効フラグは true です。
    param.append(Boolean.TRUE.toString());
    param.append(CONCAT_STRING);
    param.append(sample.getFormat() == null ? StringUtil.EMPTY_STRING : sample.getFormat());
    return param.toString();
}
```

インポート処理

インポート内容から各フィールドの値を取得し、制約を作成してください。 エクスポート処理で出力された内容を解析する必要がある場合は、それらの処理を行ってください。

```
public Restriction deserialize(final Class<?> element, final Restriction restriction, final ImportDataModel data, final Item
refItem) throws DictionaryServiceException {
    // インポート内容から日付書式のデータを取得します。
    final ImportExcelDataModel importData = data.get(SampleDateFormatRestriction.ID);
    if (importData == null) {
        return null;
    }

    SampleDateFormatRestriction sample = null;

    if (element == Item.class) {
        // 辞書項目の場合はインポート内容から制約を作成します。
        final String value = importData.getValue();
        if (!value.isEmpty()) {
            final String[] splitValue = value.split(CONCAT_STRING);
            if (Boolean.valueOf(splitValue[0].trim()) && splitValue.length == 2) {
                sample = new SampleDateFormatRestriction(splitValue[1].trim());
            } else {
                sample = new SampleDateFormatRestriction();
            }
        }
    } else if (element == Alias.class) {
        // エイリアスの場合は参照先辞書項目の内容を利用します。
        final RestrictionUsage usage = refItem.getUsages().get(UsageId.of(RestrictionUsage.ID));
        sample = usage == null ? null : (SampleDateFormatRestriction) usage.getRestriction(SampleDateFormatRestriction.ID);
        return sample;
    }
    return sample;
}
```

検証処理

validateメソッドはインポートファイルに記載された値を検証します。 検証した結果、処理を続行できない場合はエラーメッセージをセットしてください。 処理は続行できるけれど、何らかの警告を表示したい場合は警告メッセージをセットしてください。

エラーメッセージをセットするには、表示したいエラーメッセージを引数に `setErrorMessage` を呼び出してください。 警告メッセージをセットするには、表示したい警告メッセージを引数に `setWarningMessage` を呼び出してください。

```
// インポート内容から日付書式のデータを取得します。
final ImportExcelDataModel data = importData.get(SampleDateFormatRestriction.ID);
if (data == null) {
    return;
}

if (!data.getValue().isEmpty()) {
    if (element == Item.class) {
        final String[] values = data.getValue().split(CONCAT_STRING, -1);

        if (PARAM_LENGTH != values.length) {
            // ワーニングメッセージの追加例
            setWarningMessage("要素数が2ではありません。 (Sheet : dictionary, Row : " + data.getRow() + ", Col : " + data.getCol()
+ ", 要素数 : " + values.length + ")");
            return;
        }

        if (!values[1].trim().isEmpty()) {
            try {
                new SimpleDateFormat(values[1].trim());
            } catch (final NullPointerException | IllegalArgumentException e) {
                // エラーメッセージの追加例
                setErrorMessage("書式が不正です。 (Sheet : dictionary, Row : " + data.getRow() + ", Col : " + data.getCol() + ", 書式 :
" + values[1].trim() + ")");
            }
        }
    }
}
```

エクスポート処理、インポート処理、検証処理をまとめます。

src/main/java/sample/restriction/SampleDateFormatRestrictionSerializer.java

```
public class SampleDateFormatRestrictionSerializer extends StandardRestrictionSerializer {

    // インポートファイルのパラメータ数
    static final int PARAM_LENGTH = 2;

    /**
     * 制約タイプを返却します。 制約のclassを指定してください。
     */
    @SuppressWarnings("unchecked")
    @Override
    public Class<Restriction> getSupportType() {
        return (Class<Restriction>) (Object) SampleDateFormatRestriction.class;
    }

    /**
     * エクスポート時の処理 制約の内容は1 セルに出力されます。 項目が複数ある場合は項目を連結して出力してください。
     * @param restriction 制約
     */
    @Override
    public String serialize(final Restriction restriction) {
        final SampleDateFormatRestriction sample = (SampleDateFormatRestriction) restriction;
        final StringBuilder param = new StringBuilder();
        param.append(Boolean.TRUE.toString());
        param.append(CONCAT_STRING);
        param.append(sample.getFormat() == null ? StringUtil.EMPTY_STRING : sample.getFormat());
        return param.toString();
    }

    /**
     * インポート時の処理
     * @param element 辞書項目またはエイリアス
     * @param restriction 制約
     * @param data インポート内容
     * @param refItem 参照先辞書項目
     */
    @Override
```

```
    @Override
    public Restriction deserialize(final Class<?> element, final Restriction restriction, final ImportDataModel data, final Item
refItem) throws DictionaryServiceException {
        // インポート内容から日付書式のデータを取得します。
        final ImportExcelDataModel importData = data.get(SampleDateFormatRestriction.ID);
        if (importData == null) {
            return null;
        }

        SampleDateFormatRestriction sample = null;

        if (element == Item.class) {
            // 辞書項目の場合はインポート内容から制約を作成します。
            final String value = importData.getValue();
            if (!value.isEmpty()) {
                final String[] splitValue = value.split(CONCAT_STRING);
                if (Boolean.valueOf(splitValue[0].trim()) && splitValue.length == 2) {
                    sample = new SampleDateFormatRestriction(splitValue[1].trim());
                } else {
                    sample = new SampleDateFormatRestriction();
                }
            }
        } else if (element == Alias.class) {
            // エイリアスの場合は参照先辞書項目の内容を利用します。
            final RestrictionUsage usage = refItem.getUsages().get(UsageId.of(RestrictionUsage.ID));
            sample = usage == null ? null : (SampleDateFormatRestriction)
usage.getRestriction(SampleDateFormatRestriction.ID);
            return sample;
        }
        return sample;
    }

    /**
     * インポート時のバリデーションを行います。
     */
    @Override
    public void validate(final Class<?> element, final ImportDataModel importData) {
        // インポート内容から日付書式のデータを取得します。
        final ImportExcelDataModel data = importData.get(SampleDateFormatRestriction.ID);
        if (data == null) {
            return;
        }

        if (!data.getValue().isEmpty()) {
            if (element == Item.class) {
                final String[] values = data.getValue().split(CONCAT_STRING, -1);

                if (PARAM_LENGTH != values.length) {
                    // ワーニングメッセージの追加例
                    setWarningMessage("要素数が2ではありません。 (Sheet : dictionary, Row : " + data.getRow() + ", Col : " +
data.getCol() + ", 要素数 : " + values.length + ")");
                    return;
                }

                if (!values[1].trim().isEmpty()) {
                    try {
                        new SimpleDateFormat(values[1].trim());
                    } catch (final NullPointerException | IllegalArgumentException e) {
                        // エラーメッセージの追加例
                        setErrorMessage("書式が不正です。 (Sheet : dictionary, Row : " + data.getRow() + ", Col : " + data.getCol() + ",
書式 : " + values[1].trim() + ")");
                    }
                }
            }
        }
    }
}
```

実装したクラスをシステムに読み込ませるための設定ファイルの追加

次に、実装したクラスをシステムに読み込ませるための設定ファイルを追加します。 設定ファイルは java.util.ServiceLoader で読み込まれます。

jp.co.intra_mart.foundation.repository.metadata.dictionary.restriction.RestrictionFactory

```
sample.restriction.SampleDateFormatRestrictionFactory
```

jp.co.intra_mart.foundation.repository.metadata.dictionary.restriction.Restriction

```
sample.restriction.SampleDateFormatRestriction
```

jp.co.intra_mart.system.repository.metadata.dictionary.import_export.restriction.RestrictionSerializer

```
sample.restriction.SampleDateFormatRestrictionSerializer
```

インポート・エクスポートで使用する設定ファイル

下記の例は sample_restriction の入出力設定です。

src/main/conf/im-repository-dictionary-config-type/sample-restriction.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<im-repository-dictionary-config-type
   xmlns="http://www.intra-mart.jp/repository/im-repository-dictionary-config-type"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/repository/im-repository-dictionary-config-type im-repository-dictionary-config-type.xsd ">
   <sheetConfig kind="dictionary">
      <excelConfig kind="restriction_usage" usage="child">
         <columnData>
            <key>sample_restriction</key>
            <index>49000</index>
            <validationAddress></validationAddress>
            <editDisable>Alias</editDisable>
            <line no="1">
               <name></name>
               <color>LIGHT_TURQUOISE</color>
               <borderTop>THIN</borderTop>
               <borderBottom>THIN</borderBottom>
               <borderLeft></borderLeft>
               <borderRight>THIN</borderRight>
            </line>
            <line no="2">
               <name>サンプル</name>
               <color>LIGHT_TURQUOISE</color>
               <borderTop>THIN</borderTop>
               <borderBottom>THIN</borderBottom>
               <borderLeft>THIN</borderLeft>
               <borderRight>THIN</borderRight>
            </line>
         </columnData>
      </excelConfig>
   </sheetConfig >
</im-repository-dictionary-config-type>
```

タグの説明

sheetConfigタグ
Excel ファイルのシートを表現します。 kind属性で出力するシート名を指定します。 指定できる値は **category** または **dictionary** のいずれかです。 今回の例ではdictionaryシートに出力したいので、dictionary を指定します。

```
<sheetConfig kind="dictionary">
```

excelConfigタグ

kind 属性を指定することで、列のグループを表現します。 kind属性 restriction_usage は制約用途を表す列のグループです。 今回の例では、追加したsample_restriction を出力するための設定を追加します。

```
<excelConfig kind="restriction_usage" usage="child">
```

columnDataタグ

Excel に対する出力設定を行うタグの親タグです。 属性情報はありません。

keyタグ

追加した制約のIDを設定します。

```
<key>sample_restriction</key>
```

indexタグ

出力位置を示す値です。 excelConfigタグ内での位置を示す値です。

```
<index>49000</index>
```

validationAddressタグ

Excel での入力規則を示す値です。 Excel 上で利用する値を設定してください。

```
<validationAddress>inputlist!$B$1:$B$3</validationAddress>
```

入力規則が必要ない場合は値を空にしてください。

```
<validationAddress></validationAddress>
```

editDisableタグ

Excel での編集不可を示す値です。 編集不可の場合はAliasを指定してください。編集可能な場合は、空のタグを指定するか、タグ自体を書かないでください。

```
<editDisable>Alias</editDisable>
```

lineタグ

Excel でのヘッダ行の出力位置を指定します。ヘッダ行は2行で構成されています。 1行目に出力したい場合は 1 を、2行目に出力したい場合は 2 を指定してください。

```
<line no="1">
```

nameタグ

カラムの名称を指定してください。

```
<name>サンプル</name>
```

多言語化が必要な場合は、多言語対応 を参考にメッセージプロパティを定義した上でメッセージコードを指定してください。

```
<name>I18N.MESSAGE.EXAMPLE</name>
```

colorタグ

セルの背景色を設定できます。 指定できる値は org.apache.poi_v3_9.ss.usermodel.IndexedColors の値です。

```
<color>LIGHT_TURQUOISE</color>
```

背景色が必要ない場合は値を空にしてください。

```
<color></color>
```

borderTopタグ、borderBottomタグ、borderLeftタグ、borderRightタグ

セルの罫線を設定できます。 指定できる値は org.apache.poi_v3_9.ss.usermodel.CellStyle の値です。

```
<borderTop>THIN</borderTop>
<borderBottom>THIN</borderBottom>
<borderLeft>THIN</borderLeft>
<borderRight>THIN</borderRight>
```

罫線が必要ない場合は値を空にしてください。

```
<borderTop></borderTop>
<borderBottom></borderBottom>
<borderLeft></borderLeft>
<borderRight></borderRight>
```

影響範囲検出のためのクラス

これらの実装を行い、モジュールのビルドや war ファイルのデプロイを行うと、次のような画面が現れ、制約が追加されていることがわかります。

バリデーション

項目

- バリデーションを利用する

バリデーションを利用する

制約用途に指定した制約に適合するかどうかを確認する の設定を行っている場合、次のような実装を行うことでリクエストの パラメータに対してバリデーションを行うことができます。

JavaEE開発モデル

JSR 303 Bean Validationに準拠したカスタムバリデーションとして、辞書項目の制約に基づいたものを提供します。
@Dictionary(id=”xxx”) アノテーションを付与することにより、辞書項目に定義されている制約を適用できます。
この例では SampleForm の foo に対して 0 から 100 までの範囲内であるという制約が適用されます。

```
package sample;

import jp.co.intra_mart.foundation.repository.metadata.dictionary.validator.Dictionary;

public class SampleForm {
   // 辞書項目IDをアノテーションに指定する
   // id = 辞書項目IDまたはエイリアスID、name = 項目名
   @Dictionary(id = "sample-dictionary-item", name = "foo")
   private Long foo;
}
```

スクリプト開発モデル

Jssp Validator のカスタムバリデーションとして、辞書項目の制約に基づいたものを提供します。
バリデーションルールのキーに dictionary を、値に辞書項目IDを指定することにより、辞書項目に定義されている制約を適用できます。
この例ではリクエストパラメータの foo に対して 0 から 100 までの範囲内であるという制約が適用されます。

validation.js

```
var init = {
  'foo': { // リクエストパラメータ名
      caption: "CAP.Z.EXAMPLE.NAME", // エラーメッセージのキャプション
      dictionary: 'sample-dictionary-item'
   }
}
```

sample.js

```
let foo;
/**
 * @validate validation#init
 * @onerror handleError
 */
function init(request) {
  foo = request.foo;
}

function handleError(request, validationErrors) {
  var param = {
    errors: validationErrors.getMessages()
  };
  forward("error", param);
}
```

クライアント側のバリデーション

上述の Jssp Validator の設定、実装を行うとクライアント側のバリデーションも行えます。

クライアント側では UIデザインガイドライン（ PC 版） の JSSP Validation と連携する場合 のような実装をすることで辞書の制約によるバリデーションが行われます。

sample.html （JSSP Validation と連携する場合 をまとめたもの）

```
<imart type="head">
 <script src="ui/libs/jquery-validation-1.9.0/jquery.validate.js"></script>
 <imart type="imuiValidationRule" rule="/sample/validation#init" rulesName="rules" messagesName="messages" />
</imart>

<div id="container" class="imui-form-container-wide">
 <form id="myform" method="POST">
  <input type="text" name="foo" />
  <imart type="imuiButton" id="submit-button" value="送信" class="imui-small-button" />
 </form>
</div>
<script>
 $(function() {
  imuiDisableOnSubmit('#myform');
  $('#submit-button').click(function() {
   if (imuiValidate('#myform', rules, messages)) {
    imuiConfirm('メッセージ', '確認', function() {
     imuiAjaxSend('#myform', 'POST', 'json');
    });
   }
  });
 });
</script>
```

> **注意**
> 辞書のバリデーションはクライアントからサーバにリクエストを送信し、サーバ側でバリデーションを行い、その結果をクライアント側で表示するという仕組みで動作します。 また、項目ごとにリクエストが送信されます。大量の項目がある場合、リクエスト数が相当量になる可能性があります。 バリデーションの仕組みを十分理解した上でご利用ください。

辞書項目検索

項目

- 辞書項目を検索する
- 結果を返却する

辞書項目を検索する

スクリプト開発モデル

sample.js

```
function openSearchDictionaryDialog() {
 // 辞書項目検索画面を呼び出します
 window.open('%ベースURL%/repository/search/metadata/dictionary');
}
```

> **コラム**
> 下記のようにコールバック関数を利用することもできます。
>
> ('%ベースURL%/repository/search/metadata/dictionary#%コールバック関数%')
>
> コールバック関数を指定しない場合、デフォルトのコールバック関数は'onDictionarySelected'です。

結果を返却する

スクリプト開発モデル

sample.js

```
// 辞書項目検索画面で選択した項目は配列で返却され、上から順番に格納されます。
window.onDictionarySelected = function (selected) {
 if (selected.length > 0) {
  // 配列の0番目の辞書項目IDを取得します
  var itemId = selected[0].properties.itemId;
  // 配列の0番目の辞書項目名を取得します
  var itemName = selected[0].localizeName.default;
 }
}
```

コラム

コールバック関数を指定する場合

onDictionarySelectedを指定したコールバック関数に変更してください。

sample.jsの引数の配列オブジェクトの説明をします。

・selected

取得できる情報は、辞書項目とエイリアスの2種類です。

・辞書項目

プロパティ	説明	型
enabledFlag	辞書項目の有効化を示すフラグ	boolean
label	ツリーの表示名	string
localizeDescription（※1）	説明	object
localizeName（※1）	辞書項目名	object
parentId	親のID	string
properties.enumerationId	参照先列挙ID	string
properties.enumerationName	参照先列挙名	string
properties.itemId	辞書項目ID	string
properties.usages（※2）	用途	array
type	item	string

・エイリアス

プロパティ	説明	型
label	ツリーの表示名	string
localizeDescription（※1）	説明	object
localizeName（※1）	エイリアス名	object
parentId	親のID	string
properties.aliasId	エイリアスID	string
properties.referenceItemId	参照先辞書項目ID	string
type	alias	string

（※1）localizeDescription、localizeNameには多言語情報が格納されており、構成は以下です。

プロパティ	説明	型
default	標準	string
ja	日本語	string

プロパティ	説明	型
en	英語	string
zh_CN	中国語（簡体字）	string

（※2）properties.usagesにはデータ用途と制約用途が格納されており、構成は以下です。

プロパティ	説明	型
name	用途名 データ用途の場合はデータ、制約用途の場合は制約です。	string
usageId	用途ID データ用途の場合はdata_usage、制約用途の場合はrestriction_usageです。	string
usageData（※3）	用途	object

（※3）現在の用途で用意されている、usageDataには以下の要素が格納されています。

・データ用途

プロパティ	説明	型
databasePhysicalName	辞書項目、または、エイリアスがデータベースで使用された時の物理名。	string
databaseScale	辞書項目、または、エイリアスがデータベースで使用された時の最大桁数。	number
databaseSize	辞書項目、または、エイリアスがデータベースで使用された時の小数桁数。	number
databaseType	辞書項目、または、エイリアスがデータベースで使用された時のデータ型。	string
logicalName	辞書項目、または、エイリアスがデータベースで使用された時の論理名。 （※1）と同様に多言語情報が格納されています。	object
comment	辞書項目、または、エイリアスがデータベースで使用された時のコメント。 （※1）と同様に多言語情報が格納されています。	object
javaType	辞書項目、または、エイリアスがJavaで使用された時のデータ型。	string
javaVariableName	辞書項目、または、エイリアスがJavaで使用された時の変数名。	string
javaScriptType	辞書項目、または、エイリアスがJavaScriptで使用された時の型。	string
javaScriptVariableName	辞書項目、または、エイリアスがJavaScriptで使用された時の変数名。	string
logicDesignerType	辞書項目、または、エイリアスがIM-LogicDesignerで使用された時の型。	string
logicDesignerVariableName	辞書項目、または、エイリアスがIM-LogicDesignerで使用された時の変数名。	string

・制約用途

桁数

プロパティ	説明	型
id	制約ID digit_restriction	string
max	最大桁数	string
min	最小桁数	string

バイト数

プロパティ	説明	型
id	制約ID byte_restriction	string
max	最大バイト数	string
min	最小バイト数	string

数値

プロパティ	説明	型
id	制約ID numeric_restriction	string
max	最大値	string
min	最小値	string
floatDigitsMax	最大小数桁	string
floatDigitsMin	最小小数桁	string
integerDigitsMax	最大整数桁	string
integerDigitsMin	最小整数桁	string

アルファベット

プロパティ	説明	型
id	制約ID alpha_restriction	string
option	大文字・小文字の判定	string
symbols	入力可能な記号	string

数字

プロパティ	説明	型
id	制約ID number_restriction	string
symbols	入力可能な記号	string

アルファベットと数字

プロパティ	説明	型
id	制約ID alphanumeric_restriction	string
option	大文字・小文字の判定	string
symbols	入力可能な記号	string

ひらがな文字列

プロパティ	説明	型
id	制約ID hiragana_restriction	string
symbols	入力可能な記号	string

カタカナ文字列

プロパティ	説明	型
id	制約ID katakana_restriction	string
option	半角・全角の判定	string
symbols	入力可能な記号	string

メールアドレス

プロパティ	説明	型
id	制約ID email_restriction	string

URL

プロパティ	説明	型
id	制約ID url_restriction	string

正規表現

プロパティ	説明	型
id	制約ID regex_restriction	string
regex	正規表現	string

日付時刻

プロパティ	説明	型
id	制約ID datetime_restriction	string
format	フォーマット	string

ID

プロパティ	説明	型
id	制約ID id_restriction	string

UserCd

プロパティ	説明	型
id	制約ID usercd_restriction	string

注意

制約によってid以外のプロパティ名が変わるので、注意してください。

コラム

それぞれのプロパティを取得する場合のサンプルは以下の通りです。

※selectedの0番目を取得するものとします。

```
// 辞書項目名（日本語）を取得する場合
var itemName = selected[0].localizeName.ja;

// 説明（英語）を取得する場合
var description = selected[0].localizeDescription.en;

// 参照先列挙IDを取得する場合
var enumerationId = selected[0].properties.enumerationId;
```

列挙型API

IM-Repositoryはlabel-value形式の列挙データを統合的に管理する機能を提供します。
また、辞書データと連動させ、辞書項目の値としてこの列挙データからのみ選べるといった連携を行います。
本項ではこの列挙型の情報を取得するAPIの利用方法を説明します。

列挙型

項目
- 列挙型を取得する

列挙型を取得する

REST-API

メソッド	GET
URI	%ベースURL%/api/repository/metadata/enumeration

JavaEE開発モデル

```
// 列挙型を取得する
EnumerationService enumerationService = RepositoryServiceProvider.getInstance().getEnumerationService();
Enumerations enumerations = enumerationService.getEnumerations();
```

列挙型検索

項目
- 列挙型を検索する
- 結果を返却する

列挙型を検索する

スクリプト開発モデル

列挙型検索画面を呼び出す方法は2通りあります。

・列挙を全て呼び出したい場合（※1）

sample.js

```
function openSearchEnumerationDialog() {
 // 列挙型検索画面を呼び出します
 window.open('%ベースURL%/repository/search/metadata/enumeration');
}
```

・列挙が有効化のデータのみを呼び出したい場合（※2）

sample.js

```
function openSearchEnumerationDialog() {
 // 列挙型検索画面を呼び出します
 window.open('%ベースURL%/repository/search/metadata/enumeration?enabled=true');
}
```

コラム

下記のようにコールバック関数を利用することもできます。

（※1）の場合(‘%ベースURL%/repository/search/metadata/enumeration#%コールバック関数%’)

（※2）の場合(‘%ベースURL%/repository/search/metadata/enumeration?enabled=true#%コールバック関数%’)

コールバック関数を指定しない場合、デフォルトのコールバック関数は’onEnumerationSelected’です。

結果を返却する

スクリプト開発モデル

sample.js

```
window.onEnumerationSelected = function (selected) {
 if (selected) {
  // 列挙IDを取得します
  var enumerationId = selected.identify;
  // 列挙名を取得します
  var enumerationName = selected.name.default;
 }
}
```

コラム

コールバック関数を指定する場合

onEnumerationSelectedを指定したコールバック関数に変更してください。

sample.jsの引数のオブジェクトの説明をします。

・selected

取得できる情報は、以下です。

プロパティ	説明	型
identify	型を含まない列挙ID	string
parentId	親のID	string
name（※1）	列挙名	object
description（※1）	説明	object
enumerationItems（※2）	列挙項目	array
enabled	列挙の有効化を示すフラグ	boolean
label	ツリーの表示名	string

（※1）name、descriptionには多言語情報が格納されており、構成は以下です。

プロパティ	説明	型
default	標準	string
ja	日本語	string
en	英語	string
zh_CN	中国語（簡体字）	string

（※2）enumerationItemsには以下の要素が格納されています。

プロパティ	説明	型
label（※3）	列挙項目表示名	object
value	値	string

（※3）labelも多言語情報が格納されています。構成はname、descriptionと同様です。

コラム

それぞれのプロパティを取得する場合のサンプルは以下の通りです。

```
// 列挙名（日本語）を取得する場合
var enumerationName = selected.name.ja;

// 説明（英語）を取得する場合
var description = selected.description.en;

// 列挙項目の1件目の列挙項目表示名（標準）を取得する場合
var enumerationName = selected.enumerationItems[0].label.default;
```

エンティティAPI

IM-Repository上で管理されるエンティティは、バージョン管理および辞書項目と紐づけたエンティティ項目や関連といった定義が存在します。
本項ではこのエンティティの情報を取得するAPIの利用方法を説明します。

エンティティ検索

項目

エンティティを検索する

スクリプト開発モデル

エンティティ検索画面を呼び出す方法は2通りあります。

・エンティティを全て呼び出したい場合（※1）

sample.js

```
function openSearchEntitysetDialog() {
 // エンティティ検索画面を呼び出します
 window.open('%ベースURL%/repository/search/metadata/entityset');
}
```

・エンティティが有効化のデータのみを呼び出したい場合（※2）

sample.js

```
function openSearchEntitysetDialog() {
 // エンティティ検索画面を呼び出します
 window.open('%ベースURL%/repository/search/metadata/entityset?enabled=true');
}
```


コラム

下記のようにコールバック関数を利用することもできます。

（※1）の場合(‘%ベースURL%/repository/search/metadata/entityset#%コールバック関数%’)

（※2）の場合(‘%ベースURL%/repository/search/metadata/entityset?enabled=true#%コールバック関数%’)

コールバック関数を指定しない場合、デフォルトのコールバック関数は’onEntitysetSelected’です。

結果を返却する

スクリプト開発モデル

sample.js

```
// エンティティ検索画面で選択した項目は配列で返却され、上から順番に格納されます。
window.onEntitysetSelected = function (selected) {
 if (selected.length > 0) {
  // 配列の0番目のエンティティIDを取得します
  var entityId = selected[0].properties.entityId;
  // 配列の0番目のエンティティ名を取得します
  var entityName = selected[0].localizeName.default;
 }
}
```

コラム

コールバック関数を指定する場合

onEntitysetSelectedを指定したコールバック関数に変更してください。

sample.jsの引数の配列オブジェクトの説明をします。

・selected

取得できる情報は、以下です。

プロパティ	説明	型
enabledFlag	エンティティの有効化を示すフラグ	boolean
label	ツリーの表示名	string
localizeName（※1）	エンティティ名	object
localizeDescription（※1）	説明	object
parentId	親のID	string
properties.entityId	エンティティID	string
entityItem（※2）	エンティティ項目	array
relations（※3）	関連	array

（※1）localizeName、localizeDescriptionには多言語情報が格納されており、構成は以下です。

プロパティ	説明	型
default	標準	string
ja	日本語	string
en	英語	string
zh_CN	中国語（簡体字）	string

（※2）entityItemには以下の要素が格納されています。

プロパティ	説明	型
entityItemId	辞書項目IDまたはエイリアスID	string
entityItemType	辞書項目種別 ・辞書項目の場合（dictionary-item） ・エイリアスの場合（dictionary-alias）	string
primaryKey	Primary Key制約	boolean
required	Not Null制約	boolean

（※3）relationsには以下の要素が格納されています。

プロパティ	説明	型
relationId	関連ID	string
multiplicity	多重度	string
comment	コメント	string
sourceEntityId	関連元エンティティID	string
targetEntityId	関連先エンティティID	string
relationItems（※4）	関連項目	array

（※4）relationItemsにはentityItemが紐付いており、以下の要素が格納されています。

プロパティ	説明	型
sourceId	関連先エンティティ項目ID entityItemのentityItemIdが紐付きます	string
sourceType	関連先エンティティ項目種別 entityItemのentityItemTypeが紐付きます	string
targetId	関連元エンティティ項目ID entityItemのentityItemIdが紐付きます	string
targetType	関連元エンティティ項目種別 entityItemのentityItemTypeが紐付きます	string

コラム

それぞれのプロパティを取得する場合のサンプルは以下の通りです。

※selectedの0番目を取得するものとします。

```
// エンティティ名（日本語）を取得する場合
var entityName = selected[0].localizeName.ja;

// 説明（英語）を取得する場合
var description = selected[0].localizeDescription.en;

// エンティティ項目を取得する場合
var entityItem = selected[0].entityItem;

// 関連と関連項目を取得する場合
var relations = selected[0].relations;
var relationItems = relations[0].relationItems;
```