目次

改訂情報

変更年月日	変更内容
2015-08-01	初版
2015-12-01	第2版　下記を追加・変更しました。 ・「*Web API Maker を利用したAPIの作成方法*」の「*サービスクラスの作成<基本>*」にBasic認証に関する説明を追記しました。
2017-08-01	第3版　下記を追加・変更しました。 ・「*Web API Maker を利用したAPIの作成方法*」の「*セキュアトークンによるトークンチェックを行う*」に設定ファイルに関する説明を追記しました。
2018-08-01	第4版　下記を追加・変更しました。 ・「*Web API Maker を利用したAPIの作成方法*」の「*引数・戻り値に指定可能な型*」に任意のレスポンスを返す場合の説明を追記しました。
2019-04-01	第5版　下記を追加・変更しました。 ・「*Web API Maker を利用したAPIの作成方法*」に「*例外発生時に任意のプロパティを返却する*」を追記しました。

はじめに

本書の目的

本書では intra-mart Accel Platform で提供する Web API Maker を用いたプログラミング方法や注意点等について説明します。

説明範囲は以下の通りです。

- Web API Maker の概要・基本仕様
- Web API Maker を利用したAPIの作成方法
- Web API Maker で作成したAPI利用時の注意点

対象読者

本書は、以下の条件を満たす人を対象としています。

- Web API Maker を用いて intra-mart Accel Platform 上で Web API を提供したい開発者

次の内容を理解していることが必須となります。

- Javaを理解している

注意事項

1. Web API Maker を利用するにあたり、いくつかの制限事項が存在します。制限事項についての詳細は 「リリースノート 制限事項」を参照してください。

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。
サンプルコードを参考に開発する場合は、上記について十分に注意してください。

本書の構成

本書は、以下のような内容で構成されています。

- *Web API Maker 概要*
- *Web API Maker を利用したAPIの作成方法*
- *作成したAPIの利用について*
- *サンプルについて*

Web API Maker 概要

Web API Maker とは

Web API とは HTTPベースでデータをやりとりするための API です。
Web API Maker を利用することで intra-mart Accel Platform 上でWebサービスをシンプルに実現できます。

Web API Maker の機能

Web API Maker では以下の機能を提供します。

- Java言語のクラス・メソッドに対してアノテーションを付与することにより、対象のクラスをWebサービス・プロバイダとして利用可能とする機能
- 上記のWebサービス・プロバイダにWebサービスとしての仕様をJSON形式で出力する機能

Web API Maker のモジュール構成

Web API Maker は以下のモジュール構成になります。

- Web API Maker
- Web API Maker OAuth認証モジュール
  - Web API Maker でOAuth2による認証を行う場合に必要となります。

Web API Maker を利用したAPIの作成方法

項目

全体の作業の流れ

Web API Maker を利用したAPIの作成は以下の流れで行います。

- ファクトリクラスの作成
  ファクトリクラスおよびサービスクラスのインスタンスを生成するクラスを作成します。
- サービスクラスの作成
  Webサービス・プロバイダとしてのインタフェースの決定、任意の処理を実装します。
- パッケージの登録
  作成したファクトリクラスを列挙します。

前提条件

Web API Maker を利用しAPIを作成するには以下の前提条件があります。

- 提供するAPIのファクトリクラス、サービスクラス、サービスクラスのインタフェースに含まれるモデルはすべてpublicである必要があります。
- モデルを利用した場合に有効となるプロパティは、対となる getter, setter があるメンバのみです。
- モデルは引数無しのコンストラクタを提供する必要があります。
- データの受け渡しにXMLを利用する場合でインタフェースにモデルが含まれる場合、そのモデルクラスに@XmlRootElementアノテーションを付与する必要があります。
- モデルのプロパティにMapがあり、そのvalueの型がObjectである場合は、値に格納される値が形式毎に異なる可能性があります。
  - データの受け渡しにJSONを利用する場合は Boolean, BigDecimal, String です。
  - データの受け渡しにXMLを利用する場合は String のみです。

> **注意**
> データの受け渡しに関しては、 *利用方法* を参照してください。

ファクトリクラスの作成

ファクトリクラス、および、サービスクラスのインスタンス生成を行うファクトリクラスを作成します。

ファクトリクラスでは以下の **サービスプロバイダアノテーション** を利用します。
Web API Maker ではサービスクラスに付与されているアノテーションを元にWebサービス・プロバイダとしての仕様が決定します。
仕様の決定はファクトリクラスの **@ProvideService** が付与されているメソッドの戻り値のクラスに付与されているアノテーションにて決定します。

サービスプロバイダアノテーションの種類

@WebAPIMaker	Web API Maker機能を利用することを意味するアノテーションです。 ファクトリクラスのクラスに付与します。
@ProvideFactory	ファクトリクラスのインスタンスの取得元であることを意味するアノテーションです。 ファクトリクラスのファクトリインスタンス取得メソッドに付与します。 付与されたメソッドは一度のみ実行され、ファクトリクラスのインスタンスは Web API Maker 内部でキャッシュされます。
@ProvideService	サービスクラスのインスタンスの取得元であることを意味するアノテーションです。 ファクトリクラスのサービスインスタンス取得メソッドに付与します。 このアノテーションを付与されたメソッドは提供するWebサービス・プロバイダを実行する度に呼び出されます。 メソッドを作りこむことでサービスクラスのインスタンスのライフサイクルを管理できます。

ファクトリクラスの実装例は以下の通りです。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import jp.co.intra_mart.foundation.web_api_maker.annotation.ProvideFactory;
import jp.co.intra_mart.foundation.web_api_maker.annotation.ProvideService;
import jp.co.intra_mart.foundation.web_api_maker.annotation.WebAPIMaker;

@WebAPIMaker
public class PetServiceFactory {

   @ProvideFactory
   public static PetServiceFactory getFactory() {
      return new PetServiceFactory();
   }

   @ProvideService
   public PetService getService() {
      return new PetService();
   }

}
```

サービスクラスの作成

サービスクラスでは、アノテーションによるWebサービス・プロバイダとしてのインタフェースの決定、任意の処理を実装します。
提供したいWebサービスをメソッドとして記述します。
ここではサービスクラスの作成方法と各アノテーションに関して説明します。

サービスクラスの作成＜基本＞

サービスクラスは以下の手順で作成します。

1. 認証方式を決定して、対応する **認証アノテーション** をサービスクラスに付与します。
   認証アノテーションは、API実行時の認証方式を指定するアノテーション群です。

認証アノテーション概要

@IMAuthentication	Cookieを利用したアクセスを行う場合に付与するアノテーション。 特別な認証処理を行わず、Cookieに紐づくセッションの認証状態でアクセスします。
@BasicAuthentication	Basic認証による認証行う場合に付与するアノテーション。 Basic認証を利用する場合のエンドポイントは、 **/basic** + 後述の **@Path** のvalueの値です。 **/basic** の文字列はアノテーションの属性で変更する事が可能です。

@OAuth	OAuth2による認証を行う場合に付与するアノテーション。 OAuth認証を利用する場合のエンドポイントは、 **/oauth** + 後述の **@Path** のvalue値です。 **/oauth** の文字列はアノテーションの属性で変更する事が可能です。 別途scopeの定義を行う必要があります。

コラム

intra-mart Accel Platform のOAuth認証については、「OAuth認証モジュール 仕様書 」を参照してください。 OAuthのscope定義 については「設定ファイルリファレンス」-「 クライアントのアクセス範囲設定 」を参照してください。

コラム

Basic認証では、以下のような環境で認証時にテナントを指定できます。

- バーチャルテナントによる複数テナントを利用していて、リクエスト情報を利用したテナント自動解決機能を利用していない環境

上記環境では、認証時にユーザコードとして「テナントID\ユーザコード」を指定すると、指定したテナントに対して認証を行います。
ユーザコードとしてユーザコードのみを指定すると、デフォルトテナントに対して認証を行います。

2. 各メソッドにアクセスするURLを決め、 **パスアノテーション** をメソッドに付与します。
   パスアノテーションはリクエストを送る際にURLにバインドするパスを指定するアノテーションです。

パスアノテーション概要

@Path	**/xxx** 形式でパスを指定します。 パスに **{xxx}** 形式を含めることでxxxの値をパラメータとして利用可能です。

3. 受け付ける HTTP メソッドに対応する **HTTPメソッドアノテーション** をメソッドに付与します。

   パスアノテーションを付与したメソッドにて許可するHTTPメソッドを指定します。
   指定するアノテーションによってメソッドで利用可能な引数の型が異なります。詳しくは後述の *引数・戻り値に指定可能な型* を参照してください。
   パスアノテーションを付与したメソッドには1つ以上付与する必要があります。

HTTPメソッドアノテーション概要

@GET	GETメソッドを受け付けます。 このアノテーションを付与した場合、エンティティボディを利用するパラメータは利用できません。
@PUT	PUTメソッドを受け付けます。
@POST	POSTメソッドを受け付けます。
@DELETE	DELETEメソッドを受け付けます。 このアノテーションを付与した場合、エンティティボディを利用するパラメータは利用できません。

4. メソッドに引数がある場合、値をどのように受け取るかに対応する **パラメータアノテーション** をメソッドに付与します。
   HTTPメソッドやパラメータアノテーションごとに受け取り可能な型が異なります。詳しくは後述の *引数・戻り値に指定可能な型* を参照してください。

パラメータアノテーション概要

@Parameter	クエリパラメータまたはフォームパラメータ（マルチパート含む）として引数情報を受け付けます。 パラメータ名を指定する必要があります。メソッド内で同一のパラメータ名を利用できません。
@Header	リクエストヘッダの値を引数情報として受け付けます。 パラメータ名を指定する必要があります。メソッド内で同一のパラメータ名を利用できません。

@Variable	パラメータの値を引数情報として受け付けます。 @Path アノテーションに指定したパス値にパラメータの形式を含める必要があります。 パラメータ名を指定する必要があります。メソッド内で同一のパラメータ名を利用できません。
@Body	エンティティボディの内容を引数情報として受け付けます。
@Bean	上記のパラメータを一括して 1 つのBeanとして受け取るアノテーション @Beanが指定された型が持つメンバにパラメータアノテーションが付与されている場合、それらをバインドした状態のBeanが利用できます。 詳細は後述の *@Beanについて* を参照してください。
アノテーションなし	各引数にnullまたは初期値（プリミティブbooleanだとfalse）が必ずわたります。 HttpServletRequestおよびHttpServletResponseを指定した場合、リクエスト中のそれらのインスタンスがバインドされるため、 主にリクエストやレスポンスを直接参照したい場合に利用します。
@Required	実行するメソッドに渡す引数の値が必須であることを表すアノテーションです。 上記のアノテーションと組み合わせて利用します。

サービスクラスの実装例は以下の通りです。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import jp.co.intra_mart.foundation.web_api_maker.annotation.Body;
import jp.co.intra_mart.foundation.web_api_maker.annotation.DELETE;
import jp.co.intra_mart.foundation.web_api_maker.annotation.GET;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PUT;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Variable;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.model.Pet;

@IMAuthentication
public class PetService {

    @Path("/sample/pet-shop/pet/{id}")
    @DELETE
    public void delete(@Required @Variable(name = "id") final int id) throws NotFoundException {
        for (final Pet p : ConstantData.PETS) {
            if (p.getId() == id) {
                // delete
                return;
            }
        }
        throw new NotFoundException();
    }

    @Path("/sample/pet-shop/pet/{id}")
    @GET
    public Pet get(@Required @Variable(name = "id") final int id) throws NotFoundException {
        for (final Pet pet : ConstantData.PETS) {
            if (pet.getId() == id) {
                return pet;
            }
        }
        throw new NotFoundException();
    }

    @Path("/sample/pet-shop/pets")
    @POST
    public void register(@Required @Body final Pet pet) {
        // register
    }

    @Path("/sample/pet-shop/pet/{id}")
    @PUT
    public void update(@Required @Variable(name = "id") final int id, @Required @Body final Pet pet) throws
NotFoundException {
        for (final Pet p : ConstantData.PETS) {
            if (p.getId() == id) {
                // update
                return;
            }
        }
        throw new NotFoundException();
    }

}
```

> **注意**
> サービスクラスはインタフェースでもクラスでも動作します。

@Beanについて

@Beanを利用することで、複数のパラメータを１つのBeanでまとめて受け取ることができます。
@Beanを付与された引数のクラスが持つメンバのSetterメソッドに付与されているアノテーションによりメンバの値をリクエストにバインドします。

@Beanを付与するクラスは以下を満たす必要があります。

- publicなクラスである
- publicな引数なしのコンストラクタが存在する

@Beanに付与されたメソッドが以下を満たす場合 1 つのメンバとして扱われます。

- 対となるgetter, setterが存在する
  - メソッド名がgetXxxx(booleanの場合に限りisXxxx)である引数なしでかつ戻り値を持つメソッドが存在する
  - メソッド名がsetXxxxである引数が 1 つであり、戻り値が無いメソッドが存在する
  - getterの引数の型、戻り値の型が同一である
- staticではない
- publicである
- setterメソッドにパラメータアノテーションが付与されている（ただし、@Beanは利用不可です）

実装例は以下の通りです。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import jp.co.intra_mart.foundation.web_api_maker.annotation.Bean;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Body;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PUT;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Variable;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.meta.PetShop;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.meta.PetTag;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.model.Pet;

@IMAuthentication
public class PetService {

   public static class BeanClass {

      public int id;

      public Pet pet;

      public int getId() {
         return id;
      }

      public Pet getPet() {
         return pet;
      }

      @Required
      @Variable(name = "id")
      public void setId(final int id) {
         this.id = id;
      }

      @Required
      @Body
      public void setPet(final Pet pet) {
         this.pet = pet;
      }

   }

   @Path("/sample/pet-shop/pet/{id}")
   @PUT
   public void update(@Bean final BeanClass bean) throws NotFoundException {
      for (final Pet p : ConstantData.PETS) {
         if (p.getId() == bean.id) {
            // update
            return;
         }
      }
      throw new NotFoundException();
   }
}
```

セキュアトークンによるトークンチェックを行う

intra-mart Accel Platform ではセキュアトークンを利用することにより、生成されるトークンに対してサーバ側でトークンチェック処理を行い、CSRF対策を行うことができます。
セキュアチェックアノテーションを利用することで、Web API実行前にセキュアトークンによるトークンチェックを行うかどうかを指定できます。

セキュアチェックアノテーション概要

@Secured	Web API実行前にセキュアトークンによるトークンチェックを行います。 このアノテーションはメソッドに付与可能です。 トークンの発行は SecureTokenManager で行います。 トークンの発行とWeb APIへのリクエストは同一のセッション内で行う必要があります。 OAuthを利用したWeb APIの場合、このアノテーションは無視されます。 このアノテーションを付与されたWeb APIを実行する場合、クライアントはリクエストヘッダ **X-Intramart-Secure-Token** にトークンを含める必要があります。 トークンチェックにより、トークンが妥当ではない場合、HTTPステータスコード403が返却されます。

実装例は以下の通りです。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import jp.co.intra_mart.foundation.web_api_maker.annotation.Body;
import jp.co.intra_mart.foundation.web_api_maker.annotation.DELETE;
import jp.co.intra_mart.foundation.web_api_maker.annotation.GET;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PUT;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Secured;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Variable;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.model.Pet;

@IMAuthentication
public class PetService {

    @Path("/sample/pet-shop/pet/{id}")
    @DELETE
    @Secured
    public void delete(@Required @Variable(name = "id") final int id) throws NotFoundException {
        for (final Pet p : ConstantData.PETS) {
            if (p.getId() == id) {
                // delete
                return;
            }
        }
        throw new NotFoundException();
    }

    @Path("/sample/pet-shop/pet/{id}")
    @GET
    public Pet get(@Required @Variable(name = "id") final int id) throws NotFoundException {
        for (final Pet pet : ConstantData.PETS) {
            if (pet.getId() == id) {
                return pet;
            }
        }
        throw new NotFoundException();
    }

    @Path("/sample/pet-shop/pets")
    @POST
    @Secured
    public void register(@Required @Body final Pet pet) {
        // register
    }

    @Path("/sample/pet-shop/pet/{id}")
    @PUT
    @Secured
    public void update(@Required @Variable(name = "id") final int id, @Required @Body final Pet pet) throws
NotFoundException {
        for (final Pet p : ConstantData.PETS) {
            if (p.getId() == id) {
                // update
                return;
            }
        }
        throw new NotFoundException();
    }

}
```

コラム

2017 Summer(Quadra)から、設定ファイルによりセキュアトークンチェックを無効化できるようになりました。
設定に関しては、「設定ファイルリファレンス」-「Web API Makerセキュアトークンフィルタ設定」を参照してください。

認可チェックを行う

認可アノテーションを利用することで、Web API 実行前にIM-Authzによる認可チェックを行うことができます。

認可アノテーション概要

@Authz	Web API 実行前にIM-Authzによって、指定した認可リソース（または認可マッパ）にて認可判断を行います。 予め、認可リソースを準備する必要があります。 このアノテーションはクラスまたはメソッドに付与可能です。 クラスに付与した場合、すべてのメソッドに対して認可が有効です。クラス、メソッド両方に付与されている場合、メソッドの指定が優先されます。 認可によりWeb APIの実行する権限がないと判断された場合、HTTPステータスコード401または403が返却されます。 未認証の場合に401、認証された状態で権限がない場合は403が返却されます。

実装例は以下の通りです。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import jp.co.intra_mart.foundation.authz.annotation.Authz;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Body;
import jp.co.intra_mart.foundation.web_api_maker.annotation.DELETE;
import jp.co.intra_mart.foundation.web_api_maker.annotation.GET;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PUT;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Variable;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.model.Pet;

@IMAuthentication
@Authz(uri = "service://sample/pet-shop", action = "execute")
public class PetService {

   @Path("/sample/pet-shop/pet/{id}")
   @DELETE
   public void delete(@Required @Variable(name = "id") final int id) throws NotFoundException {
      for (final Pet p : ConstantData.PETS) {
         if (p.getId() == id) {
            // delete
            return;
         }
      }
      throw new NotFoundException();
   }

   @Path("/sample/pet-shop/pet/{id}")
   @GET
   public Pet get(@Required @Variable(name = "id") final int id) throws NotFoundException {
      for (final Pet pet : ConstantData.PETS) {
         if (pet.getId() == id) {
            return pet;
         }
      }
      throw new NotFoundException();
   }

   @Path("/sample/pet-shop/pets")
   @POST
   public void register(@Required @Body final Pet pet) {
      // register
   }

   @Path("/sample/pet-shop/pet/{id}")
   @PUT
   public void update(@Required @Variable(name = "id") final int id, @Required @Body final Pet pet) throws
NotFoundException {
      for (final Pet p : ConstantData.PETS) {
         if (p.getId() == id) {
            // update
            return;
         }
      }
      throw new NotFoundException();
   }

}
```

コラム

認可 については「認可仕様書」を参照してください。

返却するステータスコードを指定する

Web API Maker では、処理が正常に終了した場合はステータスコード **200** が返却され、例外が投げられ処理が終了した場合はステータス

コード **500** が返却されます。
@Response を付与することで HTTPのレスポンスでクライアントに返却するステータスコードを指定できます。

HTTPレスポンスアノテーション概要

@Response	このアノテーションはサービスクラスのメソッド、またはExceptionクラスに付与が可能です。 このアノテーションを付与する事により、Web API Maker が指定のレスポンスを返却します。 **ただし、セキュアチェック、認証、認可チェック等によるエラー発生時はその限りではありません。**

以下は Exception クラスに付与した例です。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import jp.co.intra_mart.foundation.web_api_maker.annotation.Response;

@Response(code = 404)
public class NotFoundException extends Exception {

   public NotFoundException(final String message) {
      super(message);
   }

}
```

> 注意
> @Responseを指定しない場合にレスポンスされる HTTP ステータスについては *利用方法* を参照してください。

手動レスポンスを返却する

Web API Maker では、HTTPのレスポンスとして、エラーの有無とAPIの戻り値情報を書き込みます。メソッドが返す値は常に無視されます。
@PreventWritingResponse を付与する事により、Web API Maker 側からレスポンスの書き込みが行われません。

手動レスポンスアノテーション概要

@PreventWritingResponse	このアノテーションはメソッドに付与が可能です。 このアノテーションを付与する事により、Web API Maker 側からレスポンスの書き込みを行いません。メソッドが返す値は常に無視されます。 **ただし、セキュアチェック、認証、認可チェック等によるエラー発生時は書き込みが発生します** APIの引数に HttpServletResponse（パラメータアノテーション無し）を指定する事で HttpServletResponse を取得可能なので、そのレスポンスを利用して任意のレスポンスを返す事が可能です。

実装例は以下の通りです。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import java.io.IOException;

import javax.servlet.http.HttpServletResponse;

import jp.co.intra_mart.foundation.web_api_maker.annotation.GET;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PreventWritingResponse;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.meta.PetShop;

@IMAuthentication
public class PetService {

    @Path("/sample/pet-shop/redirect")
    @GET
    @PreventWritingResponse
    public void redirect(final HttpServletResponse response) throws IOException {
        response.sendRedirect("/sample/pet-shop");
    }

}
```

APIドキュメントを登録する

APIドキュメント（Swagger specification）出力用の **APIドキュメントアノテーション** を利用することで、APIドキュメントの出力を行うことができます。
APIドキュメントアノテーションは、主にAPIドキュメントでの各メソッドのカテゴライズに利用します。
APIドキュメントを出力するためには、アノテーション自体を作成し付与する必要があります。
各項目にメッセージコードを指定することで、国際化対応ができます。

サービスクラスに付与するアノテーションの実装例は以下の通りです。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.meta;

@Retention(RetentionPolicy.RUNTIME)
@Category(value = "petshop", name = "ペットショップ", description = "ペットショップAPIです", version = "1.0.0")
public @interface PetShop {
}
```

サービスクラスのメソッドに付与するアノテーションの実装例は以下の通りです。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.meta;

@Retention(RetentionPolicy.RUNTIME)
@Tag(name = "tag", description = "CAP.Z.IWP.WEBAPIMAKER.SAMPLE.PETSHOP.CATEGORY.DESCRIPTION") //メッセージコードを指定することで国際化できます
public @interface PetTag {
}
```

作成したアノテーションを利用し、APIドキュメントを登録する例は以下の通りです。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import jp.co.intra_mart.foundation.web_api_maker.annotation.Body;
import jp.co.intra_mart.foundation.web_api_maker.annotation.DELETE;
import jp.co.intra_mart.foundation.web_api_maker.annotation.GET;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PUT;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Variable;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.meta.PetShop;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.meta.PetTag;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.model.Pet;

@IMAuthentication
@PetShop
public class PetService {

    @Path("/sample/pet-shop/pet/{id}")
    @DELETE(summary = "ペット情報を削除します。", description = "ペット情報を削除します。")
    @PetTag
    public void delete(@Required @Variable(name = "id", description = "ペットID") final int id) throws NotFoundException {
        for (final Pet p : ConstantData.PETS) {
            if (p.getId() == id) {
                // delete
                return;
            }
        }
        throw new NotFoundException();
    }

    @Path("/sample/pet-shop/pet/{id}")
    @GET(summary = "ペット情報を取得します。", description = "ペット情報を取得します。")
    @PetTag
    public Pet get(@Required @Variable(name = "id", description = "ペットID") final int id) throws NotFoundException {
        for (final Pet pet : ConstantData.PETS) {
            if (pet.getId() == id) {
                return pet;
            }
        }
        throw new NotFoundException();
    }

    @Path("/sample/pet-shop/pets")
    @POST(summary = "ペット情報を登録します。", description = "ペット情報を登録します。")
    @PetTag
    public void register(@Required @Body final Pet pet) {
        // register
    }

    @Path("/sample/pet-shop/pet/{id}")
    @PUT(summary = "ペット情報を更新します。", description = "ペット情報を更新します。")
    @PetTag
    public void update(@Required @Variable(name = "id", description = "ペットID") final int id, @Required @Body(description =
"ペット情報")
    final Pet pet) throws NotFoundException {
        for (final Pet p : ConstantData.PETS) {
            if (p.getId() == id) {
                // update
                return;
            }
        }
        throw new NotFoundException();
    }

}
```

注意

APIドキュメントの参照方法は *API仕様を参照する* を参照してください。

例外発生時に任意のプロパティを返却する

Web API Maker では例外発生時に、任意のプロパティを返却できます。
例外発生時にスローする Exception クラスのメソッドに、@ReturnValue が付与することで任意の情報をプロパティに含めて返却します。

リターンバリューアノテーション概要

@ReturnValue	このアノテーションは、**例外発生時にスローする Exception クラスの getter メソッド** に付与できます。 このアノテーションが付与されている getter メソッドから取得した値をプロパティに含めて返却します。

以下は、@ReturnValue の実装方法です。

サービスクラスから Exception クラスのインスタンスをスローします。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import jp.co.intra_mart.foundation.web_api_maker.annotation.Body;
import jp.co.intra_mart.foundation.web_api_maker.annotation.DELETE;
import jp.co.intra_mart.foundation.web_api_maker.annotation.GET;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PUT;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Variable;
import jp.co.intra_mart.sample.web_api_maker.pet_shop.model.Pet;

@IMAuthentication
public class PetService {

   @Path("/sample/pet-shop/pet/{id}")
   @DELETE
   public void delete(@Required @Variable(name = "id") final int id) throws NotFoundException {
      for (final Pet p : ConstantData.PETS) {
         if (p.getId() == id) {
            // delete
            return;
         }
      }
      throw new NotFoundException("例外が発生しました。", "入力された id に関する情報は存在しません。", id, true);
   }

   @Path("/sample/pet-shop/pet/{id}")
   @GET
   public Pet get(@Required @Variable(name = "id") final int id) throws NotFoundException {
      for (final Pet pet : ConstantData.PETS) {
         if (pet.getId() == id) {
            return pet;
         }
      }
      throw new NotFoundException("例外が発生しました。", "入力された id に関する情報は存在しません。", id, true);
   }

   @Path("/sample/pet-shop/pets")
   @POST
   public void register(@Required @Body final Pet pet) {
      // register
   }

   @Path("/sample/pet-shop/pet/{id}")
   @PUT
   public void update(@Required @Variable(name = "id") final int id, @Required @Body final Pet pet) throws
NotFoundException {
      for (final Pet p : ConstantData.PETS) {
         if (p.getId() == id) {
            // update
            return;
         }
      }
      throw new NotFoundException("例外が発生しました。", "入力された id に関する情報は存在しません。", id, true);
   }

}
```

スローする Exception クラスです。
プロパティに含めて返却したいメンバ情報の getter メソッドに、 @ReturnValue を付与します。

```
package jp.co.intra_mart.sample.web_api_maker.pet_shop.service;

import jp.co.intra_mart.foundation.web_api_maker.annotation.Response;
import jp.co.intra_mart.foundation.web_api_maker.annotation.ReturnValue;

public class NotFoundException extends Exception {

   String optionalMessage;

   int parameterValue;

   boolean errorFlag;

   public NotFoundException(final String message, final String optionalMessage, final int parameterValue, final boolean errorFlag)
{
      super(message);
      this.optionalMessage = optionalMessage;
      this.parameterValue = parameterValue;
      this.errorFlag = errorFlag;
   }

   @ReturnValue
   public String getOptionalMessage() {
      return optionalMessage;
   }

   @ReturnValue
   public int getParameterValue() {
      return parameterValue;
   }

   public boolean isErrorFlag() {
      return errorFlag;
   }

}
```

API実行時に上記の例外が発生した場合には、以下のようなレスポンスが返却されます。

```
 {
  "error": true,
  "errorMessage": "例外が発生しました。",
  "data": {
    "optionalMessage": "入力された id に関する情報は存在しません。",
    "parameterValue": 111111
  }
 }
```

コラム

@ReturnValue は、 2019 Spring(Violette) 以降から利用可能です。

引数・戻り値に指定可能な型

Web API Maker ではHTTPメソッドやパラメータアノテーションごとに受け取り可能な型が異なります。
引数・戻り値に指定可能な型は以下の通りです。

引数・戻り値に指定可能な型（パラメータアノテーション別）

型	@Parameter	@Header	@Variable	@Body	なし	戻り値	備考
void						○	
byte[]	○	○	○		○	○	数値変換されず、バイナリを扱います。

型	@Parameter	@Header	@Variable	@Body	なし	戻り値	備考
char[]	○	○	○		○	○	String#toCharArray() と等価。
InputStream	△ ※1	x	x	○	null	x	@Bodyに利用する場合Swagger spec出力に非対応
InputStream[]	△ ※1	x	x	x	null	x	Swagger spec出力に非対応
AttachmentFile	△ ※1	x	x	x	null	x	
AttachmentFile[]	△ ※1	x	x	x	null	x	Swagger spec出力に非対応
boolean	○	○	○		false	○	※2
Boolean	○	○	○		null	○	※2
byte	○	○	○		0	○	※2
Byte	○	○	○		null	○	※2
char	○	○	○		0	○	※2
Character	○	○	○		null	○	※2
short	○	○	○		0	○	※2
Short	○	○	○		null	○	※2
int	○	○	○		0	○	※2
Integer	○	○	○		null	○	※2
long	○	○	○		0	○	※2
Long	○	○	○		null	○	※2
float	○	○	○		0	○	※2
Float	○	○	○		null	○	※2
double	○	○	○		0	○	※2
Double	○	○	○		null	○	※2
CharSequence	○	○	○		null	○	※2
Calendar	○	○	○		null	○	※2
Date	○	○	○		null	○	※2
Locale	○	○	○		null	○	※2
TimeZone	○	○	○		null	○	※2
URI	○	○	○		null	○	※2
URL	○	○	○		null	○	※2
UUID	○	○	○		null	○	※2
*[] ※3	○	○	x		null	○	
List<*> ※3	○	○	x		null	○	
Set<*> ※3	○	○	x		null	○	
ServletRequest	x	x	x	x	request	x	
HttpServletRequest	x	x	x	x	request	x	
ServletResponse	x	x	x	x	response	x	
HttpServletResponse	x	x	x	x	response	x	
（その他）	△ ※1			○	null	○	@Parameterに利用する場合Swagger spec出力に非対応

※1 マルチパートのみ指定可

※3 *[]、List<*>、Set<*> それぞれの * は※2に示した型に限る

コラム

戻り値はJSONまたはXML形式で返却されます。 ファイル等の任意のレスポンスを返したい場合の実装方法については *手動レスポンスを返却する* を参照してください。

注意

Swagger spec出力に関しては *Web上からAPI仕様を参照し実行する* を参照してください。

注意

利用できない型を指定した場合、 intra-mart Accel Platform の起動時に例外が出力されます。

パッケージの登録

クラスパスに **@WebAPIMaker** が付与されたファクトリクラスのパッケージを列挙したファイルを配置します。
複数ある場合は複数行にわけて書き込みます。

例として intra-mart e Builder for Accel Platform を利用する場合は以下のようにファイルを作成してください。

- src/main/resources/META-INF/im_web_api_maker/packages

```
jp.co.intra_mart.sample.web_api_maker.pet_shop.service
```

Web API Maker を利用したAPIの作成手順は以上です。

項目

利用方法

@Path で指定した URL にアクセスすると REST-API が実行可能です。

Web API Maker では、データの受け渡しには標準でJSON、XMLの2種類のフォーマットに対応しています。
クライアントからのデータ送信時に、ひとまとまりのオブジェクト情報を送信する際は、オブジェクトを各フォーマットでリクエストボディに含める事が可能です。
サーバからの戻り値も、このフォーマットです。
利用するフォーマットはリクエストヘッダの値を利用して決定します。

- リクエストヘッダ **Content-Type**
  メッセージボディにオブジェクトを含める際に指定します。
  JSONの場合は、 **application/json** 、XMLの場合は、 **application/xml** と指定します。
- リクエストヘッダ **Accept**
  戻り情報のフォーマットを指定します。
  JSONの場合は、 **application/json** 、XMLの場合は、 **application/xml** と指定します。
  404などのエラー時は指定した形式となる保証はありません。

戻り値の例は以下の通りです。（application/json場合）

```
{
 "error": false,
 "data": {
  "id": 1,
  "name": "Dog",
  "sold": false,
  "attribute": {
   "breed": "golden"
  }
 }
}
```

```
{
 "error": true,
 "errorMessage": "[E.IWP.WEBAPIMAKER.CONVERTER.10001] JSON文字列からの変換に失敗しました。 json:434343"
}
```

コラム
値がnullであるプロパティはJSON、XML共に出力されません。

また、レスポンスされる HTTP ステータスは、以下の通りです。

レスポンスされる HTTP ステータスの種類

200 OK	Accept ヘッダで指定された MIME タイプで処理結果を返却します。
400 Bad Request	リクエストの内容が不正です。 Content-Type が application/json だが JSON 文字列でない 等が考えられます。
401 Unauthorized	未認証状態で、認可が通りません。
403 Forbidden	認証済み状態ですが、認可が通りません。
404 Not Found	URL が定義されていません。

405 Method Not Allowed	そのメソッドではアクセスできません。
406 Not Acceptable	Accept ヘッダに指定されているタイプが全部サポートしておらず、レスポンスを返すことができません。
415 Unsupported Media Type	Content-Type ヘッダに指定されているタイプがサポートしておらず、リクエストを読むことができません。
500 Internal Server Error	内部サーバエラーです。

セッション管理

Web API Maker では、リクエストヘッダ **X-Intramart-Session** を指定することによりセッション管理を行えます。
X-Intramart-Sessionには、 **keep** , **once** , **never** の３つの値が利用できます。
ヘッダを省略した場合は keep を指定した場合の挙動と同じです。
指定した値による挙動は認証方式によって以下のように異なります。

IMAuthenticationの場合

X-Intramart-Session	**動作**
keep	セッション管理を何も行いません。
once	セッション管理を何も行いません。
never	Web API実行後にログイン状態である場合、ログアウトを行います。

BasicAuthenticationの場合

X-Intramart-Session	**動作**
keep	認証済みでない場合は、Web API実行前にログインを行います。実行後は何も行いません。
once	認証済みでない場合は、Web API実行前にログインを行います。Web API実行前に未認証であった場合はログアウトを行います。
never	認証済みでない場合は、Web API実行前にログインを行います。Web API実行後にログイン状態である場合、ログアウトを行います。

OAuthの場合

X-Intramart-Session	**動作**
keep	認証済みでない場合は、Web API実行前にログインを行います。実行後は何も行いません。
once	認証済みでない場合は、Web API実行前にログインを行います。Web API実行前に未認証であった場合はログアウトを行います。
never	認証済みでない場合は、Web API実行前にログインを行います。Web API実行前に未認証であった場合はログアウトを行います。

API仕様を参照する

以下のURLにてAPIの仕様を返すJSONを取得可能です。

- http://<HOST>:<PORT>/<CONTEXT_PATH>/api-docs/${api-category}

${api-category} はサービスクラスで指定している **@Category** アノテーションのvalue値です。

注意

上記のAPI仕様のURLは認可によって閲覧を制限できます。
リソース名は「Swagger specification」です。
認可 については「認可仕様書」を参照してください。

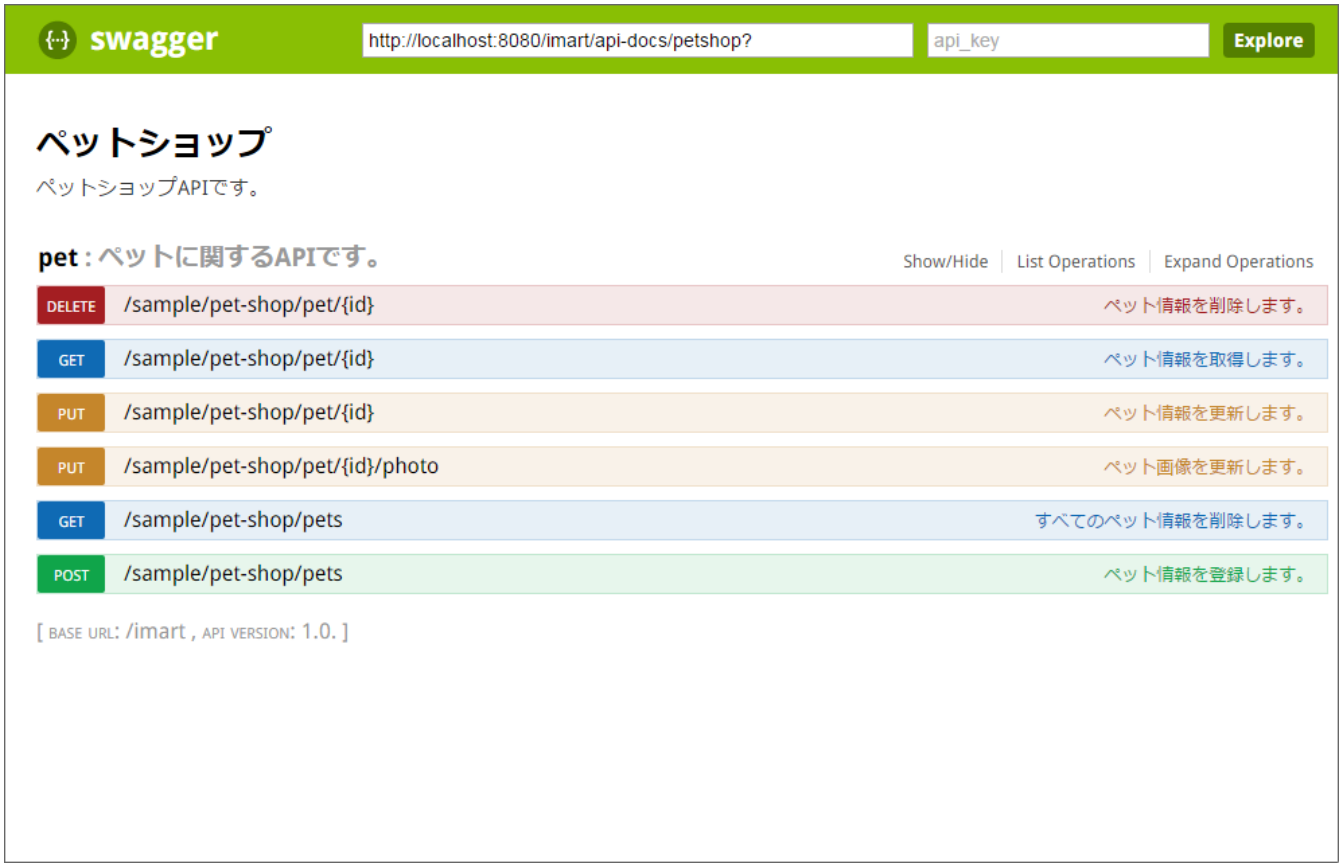
> **注意**
>
> API仕様出力を行えるサービスクラスは、 **@IMAuthentication** が付与されている物のみです。
> @IMAuthentication が付与されておらず、 @OAuth または @BasicAuthentication が付与されているサービスクラスの仕様は出力できません。

Web上からAPI仕様を参照し実行する

以下のURLにてswagger uiを表示できます。

- http://<HOST>:<PORT>/<CONTEXT_PATH>/swagger_ui/

ページ上部のテキストボックスにてAPI仕様のURL（http://<HOST>:<PORT>/<CONTEXT_PATH>/api-docs/${api-category}）を入力することで仕様の表示、実行が可能です。



サンプルについて

Web API Maker には簡単なWeb APIのサンプルが含まれています。
デプロイするには IM-Juggling で Web API Maker 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。

warのデプロイ後、テナント環境セットアップ、サンプルデータセットアップすることで本サンプルを利用できます。

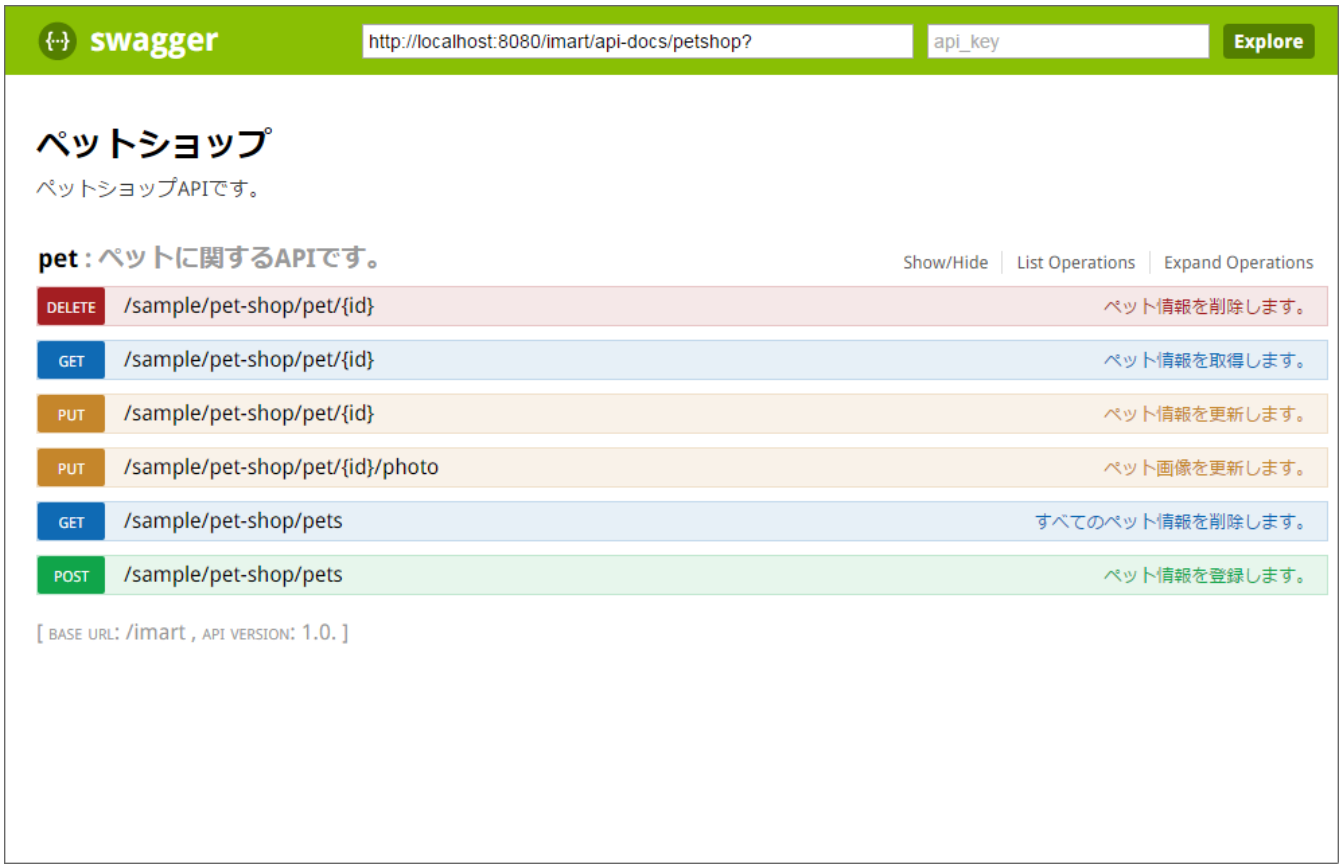
サンプルアプリケーションの仕様

以下のURLにてAPIの仕様を返すJSONを取得可能です。

- http://<HOST>:<PORT>/<CONTEXT_PATH>/api-docs/petshop

また、以下のURLにてswagger uiを表示できます。
ページ上部のテキストボックスにてAPI仕様のURLを入力することで仕様が表示可能です。

- http://<HOST>:<PORT>/<CONTEXT_PATH>/swagger_ui/



サンプルの資材一覧

サンプルのjavaソースは以下からダウンロードできます。ダウンロードには製品のライセンスキーが必要です。

プロダクトファイルダウンロード