为了生成Swagger文档，swagger-core提供了一组注释来声明和操作输出。swagger-core输出符合[Swagger规范](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md)。用户不需要熟悉Swagger规范的全部方面以便使用它，但作为参考，它可以回答有关生成的输出的几个问题。本页介绍了swagger-core提供的注释。它们分为三个 - 用于声明资源的注释，用于声明操作的注释集以及用于声明API模型的注释集。

每个注释的文档都是对其用法的概述。每个注释还包含指向其javadoc的链接（在标题的末尾和概述的末尾）。javadocs为您提供有关每个注释的附加信息，尤其是处理一些边缘情况。

至少，@Api需要声明API资源，@ApiOperation并且需要声明API操作。没有这两者的组合，将不会产生任何输出。Servlet需要@ApiImplicitParam定义方法参数，而基于JAX-RS的应用程序可以使用基本@XxxxParam注释（@QueryParam，@PathParam...）。

**1.3.9中的**新功能：现在是注释@Inherited。这意味着在接口或类上定义它们将影响实现/扩展它们的类。

目录：

* [快速注释概述](https://github.com/swagger-api/swagger-core/wiki/Annotations#quick-annotation-overview)
* [资源API声明](https://github.com/swagger-api/swagger-core/wiki/Annotations#resource-api-declaration)
* [操作声明](https://github.com/swagger-api/swagger-core/wiki/Annotations#operation-declaration)
* [模范宣言](https://github.com/swagger-api/swagger-core/wiki/Annotations#model-declaration)

**为方便起见，还提供了**[**javadoc**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs)**。**

**快速注释概述**

| **名称** | **描述** |
| --- | --- |
| [@Api](https://github.com/swagger-api/swagger-core/wiki/Annotations#api) | 将类标记为Swagger资源。 |
| [@ApiImplicitParam](https://github.com/swagger-api/swagger-core/wiki/Annotations#apiimplicitparam-apiimplicitparams) | 表示API操作中的单个参数。 |
| [@ApiImplicitParams](https://github.com/swagger-api/swagger-core/wiki/Annotations#apiimplicitparam-apiimplicitparams) | 一个包装器，允许列出多个ApiImplicitParam对象。 |
| [@ApiModel](https://github.com/swagger-api/swagger-core/wiki/Annotations#apimodel) | 提供有关Swagger模型的其他信息。 |
| [@ApiModelProperty](https://github.com/swagger-api/swagger-core/wiki/Annotations#apimodelproperty) | 添加和操作模型属性的数据。 |
| [@ApiOperation](https://github.com/swagger-api/swagger-core/wiki/Annotations#apioperation) | 描述针对特定路径的操作或通常是HTTP方法。 |
| [@ApiParam](https://github.com/swagger-api/swagger-core/wiki/Annotations#apiparam) | 为操作参数添加其他元数据。 |
| [@ApiResponse](https://github.com/swagger-api/swagger-core/wiki/Annotations#apiresponses-apiresponse) | 描述操作的可能响应。 |
| [@ApiResponses](https://github.com/swagger-api/swagger-core/wiki/Annotations#apiresponses-apiresponse) | 一个包装器，允许列出多个ApiResponse对象。 |
| [@Authorization](https://github.com/swagger-api/swagger-core/wiki/Annotations#authorization-authorizationscope) | 声明要在资源或操作上使用的授权方案。 |
| [@AuthorizationScope](https://github.com/swagger-api/swagger-core/wiki/Annotations#authorization-authorizationscope) | 描述OAuth2授权范围。 |

**资源API声明**

[**@Api**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/Api.html)

将@Api被用来声明扬鞭资源API。它有双重用途 - 它会影响资源列表*和* API声明。只有带注释的类@Api才会被Swagger扫描。

在资源列表中，注释将转换为[资源对象](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#512-resource-object)。

在API声明中，它基本上将作为[API声明](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#52-api-declaration)本身的基础。

JAX-RS的用法是：

@Path（“ / pet ”）

@ Api（value = “ / pet ”，description = “关于宠物的操作”）

@Produces（{ “ application / json ”，“ application / xml ” }）

公共 类 PetResource {

...

}

这里我们有一个暴露的宠物资源/pet。在@Api这里指出，*文件*这一资源将根据托管/pet以及（记住它可以是任何URL）而且也给这个资源的描述。Swagger会选择@Produces注释，但如果您愿意，可以覆盖此值。

资源列表的输出将是（作为apis数组中的值）：

{

“ path ”： “ / pet ”，

“ description ”： “关于宠物的行动”

}

Servlet示例将是：

@Api（值 = “ /样品/用户”，描述 = “从servlet得到一些数据”，消耗= “应用/ JSON，应用/ XML ”）

的公共 类 SampleServlet 延伸 的HttpServlet {

在这种情况下，*文档*也将托管在/sample/users。请注意，与JAX-RS不同，这*必须*是Servlet的URL映射。我们还添加了一个consumes属性来@Api声明暴露的API接受哪些内容类型。

资源列表的输出将是（作为apis数组中的值）：

{

“ path ”： “ / sample ”，

“ description ”： “关于宠物的操作”

}

注意路径是如何"/sample"而不是"/sample/users"。“/ users”部分将用作操作路径。

@Api也可用于在资源级别声明授权。这些定义适用于此资源下的所有操作，但如果需要，可以在操作级别覆盖。这适用于JAX-RS和Servlet。作为简化示例：

@Api（value = “ / sample ”，

authorizations = {

@Authorization（value = “ sampleoauth ”，scopes = {}）

}

）

在此示例中，我们添加了先前声明的OAuth2授权方案，没有任何范围。有关更多详细信息，请查看[@Authorization](https://github.com/swagger-api/swagger-core/wiki/Annotations#authorization-authorizationscope)注释。

**1.3.7中的**新功能：您现在可以basePath为给定的API 定义特定的API。

**1.3.8中的**新功能：布尔hidden属性已添加到注释中。这可以用来完全隐藏@Api，即使它声明了。在使用子资源删除不需要的工件时，这尤其有用。

**有关此注释，用法和边缘情况的更多详细信息，请查看**[**javadoc**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/Api.html)**。**

**操作声明**

[**@ApiOperation**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiOperation.html)

所述@ApiOperation用于将API资源内声明一个单个操作。操作被认为是路径和HTTP方法的唯一组合。只@ApiOperation扫描带有注释的方法并添加API声明。

注释将影响Swagger输出的两个部分，即每个路径创建一个的[API对象](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#522-api-object)，以及每个@ApiOperation创建一个[操作对象](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#523-operation-object)的[操作对象](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#523-operation-object)。请记住，在使用Servlet时，@Api会在设置路径时影响API对象。

JAX-RS的用法是：

@GET

@Path（“ / findByStatus ”）

@ApiOperation（值 = “通过状态查找宠物”，

注意到 = “多个状态值可以设置有逗号分隔的字符串”，

响应 = 宠物。类，

responseContainer = “列表”）

公 响应 findPetsByStatus（...）{ ... }

该value注释的是对API的简短描述。由于这显示在Swagger-UI的操作列表中且位置大小有限，因此应保持较短（最好短于120个字符）。将notes允许你给有关操作（例如，你可以包括索取样品和响应这里）显著的更多细节。response是方法的返回类型。请注意，实际的方法声明返回Response但是这是一个通用的JAX-RS类，而不是发送给用户的实际响应。如果返回的对象是实际结果，则可以直接使用它，而不是在注释中声明它。由于我们想要返回一份宠物清单，我们声明使用了responseContainer。请记住，Java具有类型擦除功能，因此在返回类型中使用泛型可能无法正确解析，response应该直接使用。的@GETJAX-RS注释将被用作（HTTP）method的操作的字段，并且@Path会告诉我们的动作的路径（操作是相同的路径，一个用于所使用的每个HTTP方法下分组）。

输出将是：

{

“ path ”： “ / pet / findByStatus ”，

“ operations ”： [

{

“方法”： “获取”，

“摘要”： “按状态查找宠物”，

“注释”： “可以使用逗号分隔字符串提供多个状态值”，

“类型”： “数组”，

“项目”： {

“ $ ref “： ” Pet “

}，

“ nickname ”： “ findPetsByStatus ”，

。

。

。

Servlet示例将是：

@ApiOperation（列举HTTPMethod = “ GET ”，

值 = “资源获得用户”，

响应 = 的sampleData 。类，

绰号= “的getUser ”）

公开 空隙的doPost（HttpServletRequest的请求，HttpServletResponse的响应）抛出的ServletException，IOException异常 { ... }

基于JAX-RS样本，我们已经知道了什么value，response这里意味着什么。在这里，我们有两个额外的属性。在httpMethod使用显式声明的HTTP方法在此操作中使用，因为我们没有像JAX-RS的信息。这nickname也是新的，并作为操作的唯一名称。在JAX-RS中，这将默认为方法名称，并且对于Servlet，必须由用户声明其正确的功能。该path操作将从中得到@ApiServlet的本身注解。

输出将是：

{

“ method ”： “ GET ”，

“ summary ”： “获取用户的资源”，

“ type ”： “ SampleData ”，

。

。

。

**有关此注释，用法和边缘情况的更多详细信息，请查看**[**javadoc**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiOperation.html)**。**

[**@ApiResponses**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiResponses.html)**，**[**@ ApiResponse**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiResponse.html)

使用HTTP状态代码返回错误（或其他成功消息）是一种常见做法。虽然在[@ApiOperation中](https://github.com/swagger-api/swagger-core/wiki/Annotations#apioperation)定义了操作的常规返回类型，但应使用这些注释来描述其余的返回代码。

在@ApiResponse介绍具体的可能反应。它不能直接在方法上使用，需要包含在数组值中@ApiResponses（无论是否有一个响应或更多）。

如果响应伴随身体，也可以描述身体模型（每个响应一个模型）。

用法（JAX-RS，Servlet或其他）之间的使用没有区别：

@ApiResponses（value = { @ApiResponse（code = 400，message = “提供的ID无效”），

@ ApiResponse（代码 = 404，消息 = “找不到宠物”）}）

公共 响应 getPetById（...）{ .. 。 }

**有关此注释，用法和边缘情况的更多详细信息，请查看javadocs（**[**@ApiResponses**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiResponses.html)**，**[**@ ApiResponse**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiResponse.html)**）**。

[**@Authorization**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/Authorization.html)**，**[**@ AuthorizationScope**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/AuthorizationScope.html)

这些注释[仅用作@Api](https://github.com/swagger-api/swagger-core/wiki/Annotations#api)和[@ApiOperation的](https://github.com/swagger-api/swagger-core/wiki/Annotations#apioperation)输入，而不是直接用于资源和操作。一旦声明并配置了API中支持的授权方案，就可以使用这些注释来记录资源或特定操作所需的授权方案。这@AuthorizationScope是特定于OAuth2授权方案的情况，您可能希望指定特定的受支持范围。

@Authorization和@AuthorizationScope分别转换为[授权对象](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#5210-authorizations-object)和[范围对象](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#5211-scope-object)。

实现之间的行为（JAX-RS，Servlet或其他）是相同的：

@ApiOperation（value = “向商店添加新宠物”，

授权 = {

@Authorization（

value = “ petoauth ”，

scopes = {

@AuthorizationScope（

scope = “ add：pet ”，

description = “允许添加宠物”））

}

）

}

）

public Response addPet（...）{ ... }

在这种情况下，我们声明addPet操作使用petoauth授权方案（我们假设它是OAuth2授权方案）。然后使用@AuthorizationScope我们微调定义，说它需要add:pet范围。如上所述，您可以看到它@AuthorizationScope用作输入@Authorization，然后用作输入@ApiOperation。请记住，这些注释只能用来作为输入@Api和@ApiOperation。直接在类或方法上使用它们中的任何一个都将被忽略。

输出将是：

“授权”： {

“ petoauth ”： [

{

“范围”： “添加：宠物”，

“描述”： “允许添加宠物”

}

]

}

**有关此注释，用法和边缘情况的更多详细信息，请查看javadoc（**[**@Authorization**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/Authorization.html)**，**[**@ AutoizationScope**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/AuthorizationScope.html)**）**。

[**@ApiParam**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiParam.html)

的@ApiParam仅与JAX-RS参数注释（使用@PathParam，@QueryParam，@HeaderParam，@FormParam和在JAX-RS 2， @BeanParam）。虽然swagger-core默认会扫描这些注释，但@ApiParam可以使用它来添加有关参数的更多详细信息，或者在从代码中读取值时更改这些值。

在Swagger规范中，这转换为[参数对象](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#524-parameter-object)。

Swagger将获取value()这些注释并将它们用作参数名称，并且基于注释，它还将设置参数类型。对于body参数（JAX-RS方法的单个输入参数），名称将自动设置为body（根据Swagger规范的要求）。

@DefaultValue如果存在，Swagger还将使用值作为默认值属性。

@Path（“ / {username} ”）

@ ApiOperation（value = “ Updated user ”，

notes = “这只能由登录用户完成。”）

public Response updateUser（

@ApiParam（value = “需要的名称）更新“，required = true）@PathParam（” username “）字符串用户名，

@ ApaParam（value = “更新的用户对象”，required = true）用户用户）{ ... }

这里我们有两个参数。第一个，username这是路径的一部分。第二个是正文，在本例中是一个User对象。请注意，两个参数都将required属性设置为true。对于@PathParam，这是多余的，因为默认情况下它是强制性的，不能被覆盖。

输出将是：

“参数”： [

{

“ name ”： “ username ”，

“ description ”： “需要更新的名称”，

“ required ”： true，

“ type ”： “ string ”，

“ paramType ”： “ path ”，

“ allowMultiple ”： false

}，

{

“ name ”： “ body ”，

“ description ”： “ Updated user ”，

“ required ”： true，

“ type ”： “ User ”，

“ paramType ”： “ body ”，

“ allowMultiple ”： false

}

]

**有关此注释，用法和边缘情况的更多详细信息，请查看**[**javadoc**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiParam.html)**。**

[**@ApiImplicitParam**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiImplicitParam.html)**，**[**@ ApiImplicitParams**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiImplicitParams.html)

您可能希望手动描述操作参数。这可能有多种原因，例如：

* 使用不使用JAX-RS注释的Servlet。
* 想要隐藏定义的参数，并使用完全不同的定义覆盖它。
* 描述在到达JAX​​-RS实现之前过滤器或其他资源使用的参数。

由于可以包含多个参数，因此@ApiImplicitParams允许多个@ApiImplicitParam定义。

在Swagger规范中，这些转换为[参数对象](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#524-parameter-object)。

当定义隐含参数，它设置很重要name，dataType并paramType为扬鞭的定义是正确的。

@ApiImplicitParams（{

@ApiImplicitParam（name = “ name ”，value = “ User's name ”，required = true，dataType = “ string ”，paramType = “ query ”），

@ ApiImplicitParam（name = “ email ”，value = “ User's电子邮件“，required = false，dataType = “ string ”，paramType = “ query ”），

@ ApiImplicitParam（name = “ id ”，value = “ User ID ”，required = true，dataType = “ long ”，paramType = “ query ”）

}）

public void doPost（HttpServletRequest request，HttpServletResponse response）抛出ServletException，IOException { ... }

在上面的示例中，我们可以看到具有多个参数的Servlet定义。在dataType可以是原始或类名。的paramType可以是任何由扬鞭支持的参数类型（参考JavaDocs或进一步的细节规范）的。

“参数”： [

{

“ name ”： “ name ”，

“ description ”： “ User name ”，

“ required ”： true，

“ type ”： “ string ”，

“ paramType ”： “ query ”，

“ allowMultiple ”： false

}，

{

“ name ”： “ email ”，

“ description ”： “ User's email ”，

“ required ”： false，

“ type ”： “ string ”，

“ paramType ”： “ query ”，

“ allowMultiple ”： false

}，

{

“ name ”： “ id ”，

“ description ”： “ User ID ”，

“ required ”： true，

“ type ”： “ integer ”，

“ format ”： “ int64 ”，

“ paramType ”： “ query ”，

“ allowMultiple ”： 假

}

]

**有关此注释，用法和边缘情况的更多详细信息，请查看javadocs（**[**@ApiImplicitParam**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiImplicitParam.html)**，**[**@ ApiImplicitParams**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiImplicitParams.html)**）**。

**模范宣言**

[**@ApiModel**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiModel.html)

Swagger-core在整个API内省中基于对它们的引用构建模型定义。将@ApiModel允许你从一个简单的描述或名称变更为多态性的定义操纵模型的元数据。

这转换为Swagger规范中的[模型对象](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#527-model-object)。

在其基本功能中，您可以使用@ApiModel更改模型的名称并向其添加说明：

@ApiModel（value = “ DifferentModel ”，description = “文档的示例模型”）

类 OriginalModel { ... }

在这里，我们将模型的名称从OriginalModel更改为DifferentModel。

输出将是：

“ DifferentModel ”： {

“ id ”： “ DifferentModel ”，

“ description ”： “文档的示例模型”，

。

。

}

为了支持多态和继承，我们使用discriminator和subTypes字段。两者都必须用于Swagger输出才有效。

该discriminator字段必须是顶部模型中的字段，该字段将用于确定正在使用哪个子模型。例如，如果您有一个Animal类Cat，Dog并且Chicken作为子类，该animalType字段可以用作鉴别器来确定实际使用哪个动物。

在subTypes必须列出继承模型的类。类本身不必从超类型继承。实际上，Swagger不会自动读取扩展类，您必须手动描述这些类subTypes才能对它们进行解析。

@ApiModel（值= “超模”，鉴别 = “ FOO ”，子类型 = { 子模型。类}）

公共 类 名模 { ... }

@ApiModel（value = “ SubModel ”）

公共 类 SubModel { ... }

上面的代码片段是一个如何描述继承的简单示例。注意SubModel不会扩展SuperModel。以同样的方式，您可以添加多个继承类。可以有任意数量的继承级别。

这个输出将是：

“ SuperModel ”： {

“ id ”： “ SuperModel ”，

“ required ”： [

“ foo ”

]

“ properties ”： {

“ foo ”： {

“ type ”： “ string ”

}

}，

“ subTypes ”： [ “ SubModel ” ]，

“ discriminator ”： “ foo ”

}，

“ SubModel ”： {

“ id ”： “ SubModel ”，

“ properties ”： {

...

}，

}

**有关此注释，用法和边缘情况的更多详细信息，请查看**[**javadoc**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiModel.html)**。**

[**@ApiModelProperty**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiModelProperty.html)

虽然swagger-core将内省字段和setter / getter，但它也会读取和处理JAXB注释。该@ApiModelProperty允许控制特定扬鞭-定义，例如允许的值，以及附加的注释。如果您想在某些情况下隐藏属性，它还提供其他过滤属性。

有关Swagger Spec中的相关信息，请查看[Property Object](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/1.2.md#529-property-object)。

@ApiModelProperty（value = “商店中的宠物状态”，allowableValues = “可用，待定，已售出”）

public String getStatus（）{

return status;

}

这是向模型属性添加简短描述的简单示例。还可以观察到，虽然status是String，但我们将其记录为只有三个可能的值。

它的输出将是：

“ properties ”： {

...，

“ status ”： {

“ type ”： “ string ”，

“ description ”： “商店中的宠物状态”，

“枚举”： [

“可用”，

“待定”，

“已售出”

]

}

}

**有关此注释，用法和边缘情况的更多详细信息，请查看**[**javadoc**](http://docs.swagger.io/swagger-core/v1.3.12/apidocs/index.html?com/wordnik/swagger/annotations/ApiModelProperty.html)**。**