Springfox Reference Documentation

Dilip Krishnan · Adrian Kelly – Version 2.8.1-SNAPSHOT

Table of Contents

1. Introduction

The Springfox suite of java libraries are all about automating the generation of machine and human readable specifications for JSON APIs written using the spring family of projects (http://projects.spring.io/spring-framework). Springfox works by examining an application, once, at runtime to infer API semantics based on spring configurations, class structure and various compile time java Annotations.

1.1. History

Springfox has evolved from a project originally created by Marty Pitt (https://github.com/martypitt) and was named *swagger-springmvc*. Much kudos goes to Marty.

1.2. Goals

- To extend support for a number of the evolving standards targeted at JSON API specification and documentation such as: swagger (http://swagger.io/), RAML (http://raml.org/) and jsonapi (http://jsonapi.org/).
- To extend support for spring technologies other than spring webmvc (http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html)
- Philosophically, we want to discourage using (swagger-core) annotations that are not material to the service description at runtime. For e.g. the jackson annotations should always trump or have more weight than `@ApiModelProperty` or for e.g. `@NotNull` or specifying @RequestParam#required should always win. Annotations are to to be used only to supplement documentation or override/tweak the resulting spec in cases where its not possible to infer service/schema characteristics.

1.3. What it's not

Endorsed or approved by the Spring Framework Contributors

1.4. Development Environment

- File >> open >> build.gradle
- Make sure to check the 'use the default gradle wrapper' option.
- First time build

```bash
./gradlew cleanIdea idea
```

- To get more output from any gradle commands/tasks append a `-i` (info) or `-d` (debug) e.g.

```bash
./gradlew build -i
```

- To publish to local maven repository

```bash
./gradlew clean build publishToMavenLocal -i
```

> This build is optimized for releasing software to bintray/sonatype. In order for gradle to figure out the version the gradle plugin relies on local folder being a cloned git repository. Downloading the source archive and building will NOT work!

1.4.1. Pre-Commit Build

- Code quality (code coverage, checkstyle)

```bash
./gradlew check
```

1.4.2. Building reference documentation

To build all the current documentation (builds hand written docs and javadocs):

```bash
./gradlew allDocs
```

The docs are generated in the `build/all-docs` folder. To publish the current documentation (snapshot)

```bash
./gradlew publishDocs
```

1.4.3. CI Environment

Circle CI (https://circleci.com/gh/springfox/springfox)

1.5. Releasing

To release a non-snapshot version of Springfox:

- Execute the the release commands: The below properties are required to run a release:
- `GITHUB_TOKEN`
- `BINTRAY_USERNAME`
- `BINTRAY_PASSWORD`

Recommend using [autoenv](https://github.com/kennethreitz/autoenv) with a `.env` file at the root of the repo.

```
    ./gradlew release publishDocs -PbintrayUsername=$BINTRAY_USERNAME -PbintrayPassword=$BINTRAY_PASSWORD
    -PreleaseType=<MAJOR|MINOR|PATCH> -i
```

The release steps are as follows: - check that the git workspace is clean - check that the local git branch is master - check that the local git branch is the same as origin - gradle test - gradle check - upload (publish) all artifacts to Bintray - Bumps the project version in `version.properties` - Git tag the release - Git push

1.5.1. Snapshot

This is normally done by the CI server

```
./gradlew snapshot -PbintrayUsername=<bintrayUsername> -PbintrayPassword=<bintrayPassword>
```

1.5.2. Override deploy

To bypass the standard release flow and upload directly to bintray use the following task - manually set the version in version.properties

```
./gradlew clean build bintrayUpload -PbintrayUsername=$BINTRAY_USERNAME -PbintrayPassword=$BINTRAY_PASSWORD -
PreleaseType=<MAJOR|MINOR|PATCH>
 --stacktrace
```

1.5.3. Releasing documentation

To update the docs for an existing release pass the `updateMode` switch

```
./gradlew releaseDocs
```

1.5.4. Contributing

Please see the wiki (https://github.com/springfox/springfox/wiki) for some guidelines

1.6. Support

If you find issues or bugs please submit them via the Springfox Github project (https://github.com/springfox/springfox/issues)

2. Getting Started

2.1. Dependencies

The Springfox libraries are hosted on bintray (https://bintray.com/springfox/maven-repo/springfox/view) and jcenter. The artifacts can be viewed accessed at the following locations:

- Release:
  - https://jcenter.bintray.com/io/springfox/
  - http://jcenter.bintray.com/io/springfox/
- Snapshot
  - http://oss.jfrog.org/simple/oss-snapshot-local/io/springfox/
  - http://oss.jfrog.org/oss-snapshot-local/io/springfox/

Springfox has multiple modules and the dependencies will vary depending on the desired API specification standard. Below outlines how to include the springfox-swagger2 module which produces Swagger 2.0 API documentation.

2.1.1. Gradle

Release

GROOVY
```
repositories {
  jcenter()
}

dependencies {
    compile "io.springfox:springfox-swagger2:2.8.0"
}
```

Snapshot

GROOVY
```
repositories {
   maven { url 'http://oss.jfrog.org/artifactory/oss-snapshot-local/' }
}

dependencies {
    compile "io.springfox:springfox-swagger2:2.8.1-SNAPSHOT"
}
```

2.1.2. Maven

Release

XML
```xml
<repositories>
    <repository>
      <id>jcenter-snapshots</id>
      <name>jcenter</name>
      <url>https://jcenter.bintray.com/</url>
    </repository>
</repositories>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.8.0</version>
</dependency>
```

Snapshot

XML
```xml
<repositories>
    <repository>
      <id>jcenter-snapshots</id>
      <name>jcenter</name>
      <url>http://oss.jfrog.org/artifactory/oss-snapshot-local/</url>
    </repository>
</repositories>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.8.1-SNAPSHOT</version>
</dependency>
```

3. Quick start guides

3.1. Springfox Spring MVC and Spring Boot

JAVA

```java
/*
 *
 *  Copyright 2015-2018 the original author or authors.
 *
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *         http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 *
 *
 */

package springfox.springconfig;

import com.fasterxml.classmate.TypeResolver;
import org.joda.time.LocalDate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.context.request.async.DeferredResult;
import springfox.documentation.builders.ParameterBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.builders.ResponseMessageBuilder;
import springfox.documentation.schema.ModelRef;
import springfox.documentation.schema.WildcardType;
import springfox.documentation.service.ApiKey;
import springfox.documentation.service.AuthorizationScope;
import springfox.documentation.service.SecurityReference;
import springfox.documentation.service.Tag;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spi.service.contexts.SecurityContext;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger.web.DocExpansion;
import springfox.documentation.swagger.web.ModelRendering;
import springfox.documentation.swagger.web.OperationsSorter;
import springfox.documentation.swagger.web.SecurityConfiguration;
import springfox.documentation.swagger.web.SecurityConfigurationBuilder;
import springfox.documentation.swagger.web.TagsSorter;
import springfox.documentation.swagger.web.UiConfiguration;
import springfox.documentation.swagger.web.UiConfigurationBuilder;
import springfox.documentation.swagger2.annotations.EnableSwagger2;
import springfox.petstore.controller.PetController;

import java.util.List;

import static com.google.common.collect.Lists.*;
import static springfox.documentation.schema.AlternateTypeRules.*;

@SpringBootApplication
@EnableSwagger2 1
@ComponentScan(basePackageClasses = {
    PetController.class
}) 2
public class Swagger2SpringBoot {

  public static void main(String[] args) {
    ApplicationContext ctx = SpringApplication.run(Swagger2SpringBoot.class, args);
  }


  @Bean
  public Docket petApi() {
    return new Docket(DocumentationType.SWAGGER_2) 3
        .select() 4
          .apis(RequestHandlerSelectors.any()) 5
          .paths(PathSelectors.any()) 6
          .build() 7
        .pathMapping("/") 8
        .directModelSubstitute(LocalDate.class,
            String.class) 9
        .genericModelSubstitutes(ResponseEntity.class)
        .alternateTypeRules(
            newRule(typeResolver.resolve(DeferredResult.class,
                typeResolver.resolve(ResponseEntity.class, WildcardType.class)),
                typeResolver.resolve(WildcardType.class))) 10
        .useDefaultResponseMessages(false) 11
        .globalResponseMessage(RequestMethod.GET, 12
            newArrayList(new ResponseMessageBuilder()
                .code(500)
                .message("500 message")
                .responseModel(new ModelRef("Error")) 13
                .build()))
        .securitySchemes(newArrayList(apiKey())) 14
        .securityContexts(newArrayList(securityContext())) 15
        .enableUrlTemplating(true) 21
        .globalOperationParameters( 22
            newArrayList(new ParameterBuilder()
                .name("someGlobalParameter")
                .description("Description of someGlobalParameter")
                .modelRef(new ModelRef("string"))
                .parameterType("query")
                .required(true)
                .build()))
        .tags(new Tag("Pet Service", "All apis relating to pets")) 25
        .additionalModels(typeResolver.resolve(AdditionalModel.class)) 26
        ;
  }

  @Autowired
  private TypeResolver typeResolver;

  private ApiKey apiKey() {
    return new ApiKey("mykey", "api_key", "header"); 16
  }

  private SecurityContext securityContext() {
    return SecurityContext.builder()
        .securityReferences(defaultAuth())
        .forPaths(PathSelectors.regex("/anyPath.*")) 17
        .build();
  }

  List<SecurityReference> defaultAuth() {
    AuthorizationScope authorizationScope
        = new AuthorizationScope("global", "accessEverything");
    AuthorizationScope[] authorizationScopes = new AuthorizationScope[1];
    authorizationScopes[0] = authorizationScope;
    return newArrayList(
        new SecurityReference("mykey", authorizationScopes)); 18
  }

  @Bean
  SecurityConfiguration security() {
    return SecurityConfigurationBuilder.builder() 19
        .clientId("test-app-client-id")
        .clientSecret("test-app-client-secret")
        .realm("test-app-realm")
        .appName("test-app")
        .scopeSeparator(",")
        .additionalQueryStringParams(null)
        .useBasicAuthenticationWithAccessCodeGrant(false)
        .build();
  }

  @Bean
  UiConfiguration uiConfig() {
    return UiConfigurationBuilder.builder() 20
        .deepLinking(true)
        .displayOperationId(false)
        .defaultModelsExpandDepth(1)
        .defaultModelExpandDepth(1)
        .defaultModelRendering(ModelRendering.EXAMPLE)
        .displayRequestDuration(false)
        .docExpansion(DocExpansion.NONE)
        .filter(false)
        .maxDisplayedTags(null)
        .operationsSorter(OperationsSorter.ALPHA)
        .showExtensions(false)
        .tagsSorter(TagsSorter.ALPHA)
        .validatorUrl(null)
        .build();
  }

}
```

3.2. Configuration explained

This library extensively uses googles guava library (https://github.com/google/guava). For e.g. when you see `newArrayList(…)` its actually the guava equivalent of creating an normal array list and adding item(s) to it.

GROOVY
```
//This guava code snippet
List<Something> guavaList = newArrayList(new Something);

//... is equivalent to
List<Something> list = new ArrayList<Something>();
list.add(new Something());
```

1 Enables Springfox swagger 2

2 Instructs spring where to scan for API controllers

3 `Docket` , Springfox’s, primary api configuration mechanism is initialized for swagger specification 2.0

4 `select()` returns an instance of `ApiSelectorBuilder` to give fine grained control over the endpoints exposed via swagger.

5 `apis()` allows selection of `RequestHandler` 's using a predicate. The example here uses an `any` predicate (default). Out of the box predicates provided are `any` , `none` , `withClassAnnotation` , `withMethodAnnotation` and `basePackage` .

6 `paths()` allows selection of `Path` 's using a predicate. The example here uses an `any` predicate (default). Out of the box we provide predicates for `regex` , `ant` , `any` , `none` .

7 The selector requires to be built after configuring the api and path selectors.

8 Adds a servlet path mapping, when the servlet has a path mapping. this prefixes paths with the provided path mapping

9 Convenience rule builder substitutes `LocalDate` with `String` when rendering model properties

10 Convenience rule builder that substitutes a generic type with one type parameter with the type parameter. In this example `ResponseEntity<T>` with `T` . `alternateTypeRules` allows custom rules that are a bit more involved. The example substitutes `DeferredResult<ResponseEntity<T>>` with `T` generically

11 Flag to indicate if default http response codes need to be used or not

12 Allows globally overriding response messages for different http methods. In this example we override the 500 error code for all \`GET\`s ...

13 ...and indicate that it will use the response model `Error` (which will be defined elsewhere)

14 Sets up the security schemes used to protect the apis. Supported schemes are ApiKey, BasicAuth and OAuth

15 Provides a way to globally set up security contexts for operation. The idea here is that we provide a way to select operations to be protected by one of the specified security schemes.

16 Here we use ApiKey as the security schema that is identified by the name `mykey`

17 Selector for the paths this security context applies to.

18 Here we same key defined in the security scheme `mykey`

19 Optional swagger-ui security configuration for oauth and apiKey settings

20 Optional swagger-ui ui configuration currently only supports the validation url

21 * *Incubating* * setting this flag signals to the processor that the paths generated should try and use form style query expansion (https://tools.ietf.org/html/rfc6570#section-3.2.8). As a result we could distinguish paths that have the same path stem but different query string combinations. An example of this would be two apis:

22 Allows globally configuration of default path-/request-/headerparameters which are common for every rest operation of the api, but aren\`t needed in spring controller method signature (for example authenticaton information). Parameters added here will be part of every API Operation in the generated swagger specification.

23 How do you want to transport the api key via a HEADER (header) or QUERY_PARAM (query)?

24 What header key needs to be used to send the api key. By default this value is set to *api_key*. Depending on how the security is setup the name of the header used may need to be different. Overriding this value is a way to override the default behavior.

25 Adding tags is a way to define all the available tags that services/operations can opt into. Currently this only has name and description.

26 Are there models in the application that are not "reachable"? Not reachable is when we have models that we would like to be described but aren’t explicitly used in any operation. An example of this is an operation that returns a model serialized as a string. We do want to communicate the expectation of the schema for the string. This is a way to do exactly that. There are plenty of more options to configure the `Docket` . This should provide a good start.

3.3. Springfox Spring Data Rest

In version greater than 2.6.0, support for spring data rest was added.

This is still in **incubation**.

In order to use it

1. add the `springfox-data-rest` dependency.

3.3.1. Gradle

GROOVY
```groovy
dependencies {
    compile "io.springfox:springfox-data-rest:2.8.0"
}
```

3.3.2. Maven

XML
```xml
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-data-rest</artifactId>
    <version>2.8.0</version>
</dependency>
```

1. Import the configuration from the `springfox-data-rest` module as shown below

3.3.3. java config

JAVA
```java
//For java config
@Import({ ... springfox.documentation.spring.data.rest.configuration.SpringDataRestConfiguration.class, ...})
```

3.3.4. xml config

Import the bean in your xml configuration by defining a bean of the following type

XML
```xml
<bean class="springfox.documentation.spring.data.rest.configuration.SpringDataRestConfiguration.class" />
```

3.4. Springfox Support for JSR-303

In version greater than 2.3.2, support for bean validation annotations was added, specifically for @NotNull, @Min, @Max, and @Size.

In order to use it

- add the `springfox-bean-validators` dependency.

3.4.1. Gradle

GROOVY
```groovy
dependencies {
    compile "io.springfox:springfox-bean-validators:2.8.0"
}
```

3.4.2. Maven

XML
```xml
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-bean-validators</artifactId>
    <version>2.8.0</version>
</dependency>
```

- Import the configuration from the `springfox-bean-validators` module as shown below

3.4.3. java config

JAVA
```java
//For java config
@Import({ ... springfox.bean.validators.configuration.BeanValidatorPluginsConfiguration.class, ...})
```

3.4.4. xml config

Import the bean in your xml configuration by defining a bean of the following type

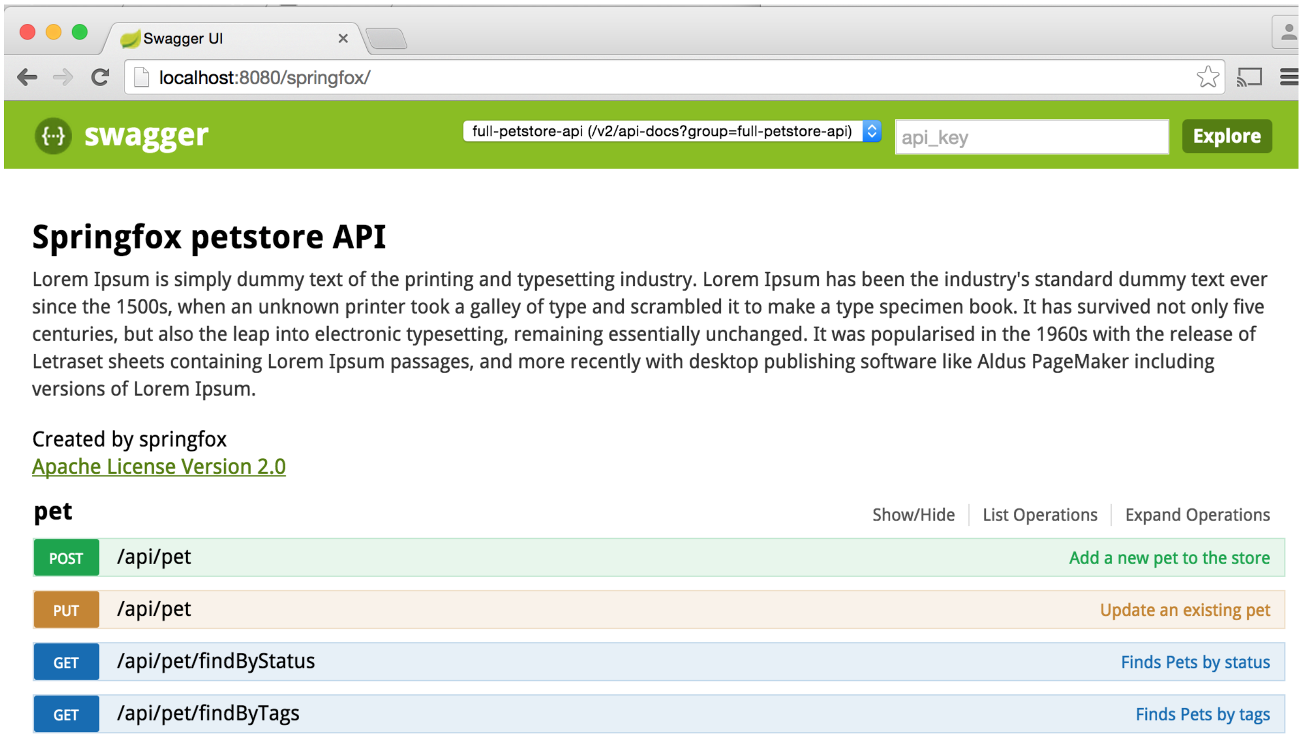
```xml
<bean class="springfox.bean.validators.configuration.BeanValidatorPluginsConfiguration" />
```

3.5. Springfox Swagger UI

The `springfox-swagger-ui` web jar (http://www.webjars.org/) ships with Swagger UI (https://github.com/swagger-api/swagger-ui). To include it in a standard Spring Boot application you can add the dependency as follows:

```groovy
dependencies {
    compile 'io.springfox:springfox-swagger-ui:2.8.0'
}
```

Pulling in the dependency creates a webjar containing the swagger-ui static content. It adds a JSON endpoint `/swagger-resources` which lists all of the swagger resources and versions configured for a given application. The Swagger UI page should then be available at http://localhost:8080/swagger-ui.html


Swagger UI
localhost:8080/springfox/
swagger
full-petstore-api (/v2/api-docs?group=full-petstore-api)
api_key
Explore

Springfox petstore API

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Created by springfox
Apache License Version 2.0

pet — Show/Hide | List Operations | Expand Operations

POST /api/pet — Add a new pet to the store
PUT /api/pet — Update an existing pet
GET /api/pet/findByStatus — Finds Pets by status
GET /api/pet/findByTags — Finds Pets by tags

The swagger ui version is specified in ./build.gradle where `swaggerUiVersion` is a git tag on the https:// github.com/swagger-api/swagger-ui[swagger-ui repo].

All content is served from a webjar convention, relative url taking the following form: `webjars/springfox-swagger-ui/2.8.0/swagger-ui.html`

By default Spring Boot has sensible defaults for serving content from webjars. To configure vanilla spring web mvc apps to serve webjar content see the webjar documentation (http://www.webjars.org/documentation#springmvc)

Swagger-Ui that comes bundled with springfox uses *meta-urls* to configure itself and discover documented endpoints. The urls for the discovery are as shown below.

Url	New Url in 2.5.+	Purpose
/configuration/security	/swagger-resources/configuration/security	Configuring swagger-ui security
/configuration/ui	/swagger-resources/configuration/ui	Configuring swagger-ui options

Since swagger ui is a static resource it needs to rely on **known** endpoints to configure itself at runtime. So these ☝ are ***cool uris** that cannot change. There is some customization that is possible, but swagger-ui needs to be available at the root of the *webcontext*.

Regarding [where swagger-ui itself is served](http://springfox.github.io/springfox/docs/current/#q13) and [where the api docs are served](http://springfox.github.io/springfox/docs/current/#customizing-the-swagger-endpoints) those are totally configurable.

3.6. Springfox RFC6570 support incubating

> *Keep in mind this is experimental*! In order to use this feature

1. Add `springfox-swagger-ui-rfc6570` instead of `springfox-swagger-ui` as a dependency experimental swagger-ui (http://mvnrepository.com/artifact/io.springfox.ui/springfox-swagger-ui-rfc6570/1.0.0).

For gradle:

```groovy
dependencies {
    compile 'io.springfox.ui:springfox-swagger-ui-rfc6570:1.0.0'
}
```

For maven:

```xml
<dependency>
    <groupId>io.springfox.ui</groupId>
    <artifactId>springfox-swagger-ui-rfc6570</artifactId>
    <version>1.0.0</version>
</dependency>
```

> The newer version has changed the group id from *`io.springfox`* to *`io.springfox.ui`*!

- Enable url templating; (see #21 (http://springfox.github.io/springfox/docs/current/#springfox-swagger2-with-spring-mvc-and-spring-boot))

3.7. Springfox samples

The springfox-demos (https://github.com/springfox/springfox-demos) repository contains a number of samples.

4. Architecture

4.1. Background

When we started work on 2.0 swagger specification we realized that we're rewriting the logic to infer the service models and the schema. So we decided to take a step back and break it out into a two step process. First infer the service model into an internal representation. Second create a mapping layer that can map the internal models to different specification formats. Out of the box we will support swagger 1.2 and swagger 2.0, but this leads us to the possibility of supporting other formats and other scenarios as well e.g. RAML, ALPS and hypermedia formats.

4.2. Component Model

The different SpringFox modules are split up as shown below.

ASCII

```
+------------------------------------------+----------------------------------------------+
|                                  springfox-core                                         |
|                                                                                         |
| Contains the internal service and schema description models along with their builders.  |
+------------------------------------------+----------------------------------------------+
                                           ^
+------------------------------------------+----------------------------------------------+
|                                  springfox-spi                                          |
|                                                                                         |
| Contains the service provider interfaces that can be used to extend and enrich the      |
| service models e.g. swagger specific annotation processors.                             |
+------------------------------------------+----------------------------------------------+
                                           |
                                           |
                +--------------------------|------------------------+
                |                          |                        |
+----------------------------------+       |       +--------------------------------------+
|        springfox-schema          |       |       |         springfox-spring-web         |
|                                  |       |       |                                      |
| Schema inference extensions that |       |       | spring web specific extensions that  |
| help build up the schema for the |       |       | can build the service models based   |
|  the parameters, models and      |       |       | on RequestMapping information.       |
|  responses                       |       |       | This is the heart library that       |
+----------------------------------+       |       | infers the service model.            |
                                           |       +--------------------------------------+
                                           |
      +------------------------------------+-------------------------------------+
      |                         springfox-swagger-common                         |
      |                                                                          |
      | Common swagger specific extensions that are aware of the different       |
      | swagger annotations.                                                     |
      +----------+---------------------------------------------------------------+
                 ^                          ^                        ^
      +----------+---------+     +----------+---------+     +-----...
      |                    |     |                    |     |
      | springfox-swagger1 |     | springfox-swagger2 |     |
      |                    |     |                    |     |
      +--------------------+     +--------------------+     +-----...

       Configurations, and mapping layer that know how to convert the service models
       to swagger 1.2 and swagger 2.0 specification documents.
       Also contains the controller for each of the specific formats.
```

5. Swagger

Springfox supports both version 1.2 (https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md) and version 2.0 (https://github.com/swagger-api/swagger-spec/blob/master/versions/2.0.md) of the Swagger (http://swagger.io/) specification. Where possible, the Swagger 2.0 specification is preferable.

The swagger-core annotations (https://github.com/swagger-api/swagger-core/wiki/Annotations), as provided by swagger-core (https://github.com/swagger-api/swagger-core), are typically used to decorate the java source code of an API which is bing 'swaggered'. Springfox is aware of the Swagger-Core Annotations and will favor those annotations over inferred defaults.

5.1. Swagger 1.2 vs Swagger 2.0

One major difference between the two swagger specification is the composition of the generated swagger documentation.

With Swagger 1.2 an applications API is represented as a `Resource Listing` and multiple `API Declarations` which has the implication of producing multiple JSON files (https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#42-file-structure)

With Swagger 2.0 things are much simpler and an application's API can be represented in a single JSON file.

5.2. Moving from swagger-springmvc?

Here is a guide (https://github.com/springfox/springfox/blob/master/docs/transitioning-to-v2.md) to help with the transition from 1.0.2 to 2.0.

Legacy documentation is available here (https://github.com/springfox/springfox/blob/v1.0.2/README.md).

5.3. Springfox configuration and demo applications

The springfox-demos (https://github.com/springfox/springfox-demos) repository contains a number of sample Spring application which can be used a reference.

6. Configuring Springfox

To enable support for swagger specification 1.2 use the `@EnableSwagger` annotation

To enable support for swagger specification 2.0 use the `@EnableSwagger2` annotation

To document the service we use a `Docket`. This is changed to be more inline with the fact that expressing the contents of the documentation is agnostic of the format the documentation is rendered.

Docket stands for (https://www.wordnik.com/words/docket) **A summary or other brief statement of the contents of a document; an abstract.**

`Docket` helps configure a subset of the services to be documented and groups them by name. Significant changes to this is the ability to provide an expressive predicate based for api selection.

JAVA
```java
import static springfox.documentation.builders.PathSelectors.*;
import static com.google.common.base.Predicates.*;

@Bean
public Docket swaggerSpringMvcPlugin() {
  return new Docket(DocumentationType.SWAGGER_2)
          .groupName("business-api")
          .select()
             //Ignores controllers annotated with @CustomIgnore
            .apis(not(withClassAnnotation(CustomIgnore.class)) //Selection by RequestHandler
            .paths(paths()) // and by paths
            .build()
          .apiInfo(apiInfo())
          .securitySchemes(securitySchemes())
          .securityContext(securityContext());
}

//Here is an example where we select any api that matches one of these paths
private Predicate<String> paths() {
  return or(
      regex("/business.*"),
      regex("/some.*"),
      regex("/contacts.*"),
      regex("/pet.*"),
      regex("/springsRestController.*"),
      regex("/test.*"));
}
```

For a list of handy predicates Look at RequestHandlerSelectors (https://github.com/springfox/springfox/blob/master/springfox-core/src/main/java/springfox/documentation/builders/RequestHandlerSelectors.java) and PathSelectors (https://github.com/springfox/springfox/blob/master/springfox-core/src/main/java/springfox/documentation/builders/PathSelectors.java).

6.1. Configuring the ObjectMapper

A simple way to configure the object mapper is to listen for the `ObjectMapperConfigured` event. Regardless of whether there is a customized ObjectMapper in play with a corresponding MappingJackson2HttpMessageConverter, the library always has a configured ObjectMapper that is customized to serialize swagger 1.2 and swagger 2.0 types.

In order to do this implement the `ApplicationListener<ObjectMapperConfigured>` interface. The event has a handle to the ObjectMapper that was configured. Configuring application specific ObjectMapper customizations in this application event handler guarantees that application specific customizations will be applied to each and every ObjectMapper that is in play.

If you encounter a NullPointerException during application startup like this issue (https://github.com/springfox/springfox/issues/635). Its because most likely the `WebMvcConfigurerAdapter` isn't working. These adapter especially in a non-spring-boot scenarios will only get loaded if the @EnableWebMvc annotation is present (http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/config/annotation/WebMvcConfigurer.html).

If using Spring Boot Web MVC, there is no need to use the @EnableWebMvc annotation, as the framework automatically detects Web MVC usage and configures itself as appropriate. In this scenario, Springfox will not correctly generate and expose the Swagger UI endpoint (`/swagger-ui.html`) if @EnableWebMvc is present in the application.

Caveat to using the library is that it depends on Jackson for serialization, more importantly the `ObjectMapper` . A good example of where this breaks down is the following issue when using Gson serialization (http://stackoverflow.com/a/30220562/19219)

6.2. Customizing the swagger endpoints.

By default the swagger service descriptions are generated at the following urls

Swagger version	Documentation Url	Group
1.2	/api-docs	implicit **default** group
1.2	/api-docs?group=external	**external** group via docket.groupName()
2.0	/v2/api-docs	implicit **default** group
2.0	/v2/api-docs?group=external	**external** group via docket.groupName()

To customize these endpoints, loading a property source (http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/context/annotation/PropertySource.html) with the following properties allows the properties to be overridden

Swagger version	Override property
1.2	springfox.documentation.swagger.v1.path
2.0	springfox.documentation.swagger.v2.path

6.3. Overriding property datatypes

Using the `ApiModelProperty#dataType` we can override the inferred data types. However it is restricted to only allow data types to be specified with a fully qualified class name. For e.g. if we have the following definition

JAVA
```java
// if com.qualified.ReplaceWith is not a Class that can be created using Class.forName(...)
// Original will be replaced with the new class
@ApiModelProperty(dataType = "com.qualified.ReplacedWith")
public Original getOriginal() { ... }

// if ReplaceWith is not a Class that can be created using Class.forName(...) Original will be preserved
@ApiModelProperty(dataType = "ReplaceWith")
public Original getAnotherOriginal() { ... }

```

In the case of `ApiImplicitParam#dataType` , since the type itself is usually a scalar type (string, int) use one of the base types specified in the Types class ⇒ `springfox-schema/src/main/java/springfox/documentation/schema/Types.java`

GROOVY
```groovy
  private static final Set<String> baseTypes = newHashSet(
      "int",
      "date",
      "string",
      "double",
      "float",
      "boolean",
      "byte",
      "object",
      "long",
      "date-time",
      "__file",
      "biginteger",
      "bigdecimal");
```

6.4. Docket XML Configuration

To use the plugin you must create a spring java configuration class which uses spring's `@Configuration`. This config class must then be defined in your xml application context.

```xml
<!-- Required so springfox can access spring's RequestMappingHandlerMapping  -->
<mvc:annotation-driven/>

<!-- Required to enable Spring post processing on @Configuration classes. -->
<context:annotation-config/>

<bean class="com.yourapp.configuration.MySwaggerConfig"/>
```

```java
@Configuration
@EnableSwagger //Loads the spring beans required by the framework
public class MySwaggerConfig {

   /**
    * Every Docket bean is picked up by the swagger-mvc framework - allowing for multiple
    * swagger groups i.e. same code base multiple swagger resource listings.
    */
   @Bean
   public Docket customDocket(){
      return new Docket(); //some customization goes here
   }

}
```

6.5. Docket Spring Java Configuration

- Use the `@EnableSwagger` or `@EnableSwagger2` annotation.
- Define one or more Docket instances using springs `@Bean` annotation.

```java
@Configuration
@EnableWebMvc //NOTE: Only needed in a non-springboot application
@EnableSwagger2
@ComponentScan("com.myapp.controllers")
public class CustomJavaPluginConfig {


   @Bean //Don't forget the @Bean annotation
   public Docket customImplementation(){
      return new Docket()
            .apiInfo(apiInfo());
            //... more options available

   }

   //...
}
```

6.6. Support for documentation from property file lookup

Starting with `2.7.0` we support looking up description from the following annotations given a property just like property place holders resolve a value annotation `@Value(${key})`. The following annotations attributes support description resolution.

- `@ApiParam#value()`
- `@ApiImplicitParam#value()`
- `@ApiModelProperty#value()`
- `@ApiOperation#value()`
- `@ApiOperation#notes()`
- `@RequestParam#defaultValue()`
- `@RequestHeader#defaultValue()`

Below are examples of how it would work

Controller Example

SomeController.java

```java
@ApiOperation(value = "Find pet by Status",
    notes = "${SomeController.findPetsByStatus.notes}"...) 1
@RequestMapping(value = "/findByStatus", method = RequestMethod.GET, params = {"status"})
public Pet findPetsByStatus(
    @ApiParam(value = "${SomeController.findPetsByStatus.status}", 2
         required = true,...)
    @RequestParam("status",
        defaultValue="${SomeController.findPetsByStatus.status.default}") String status) { 3
    //...
}

@ApiOperation(notes = "Operation 2", value = "${SomeController.operation2.value}"...) 4
@ApiImplicitParams(
    @ApiImplicitParam(name="header1", value="${SomeController.operation2.header1}", ...) 5
)
@RequestMapping(value = "operation2", method = RequestMethod.POST)
public ResponseEntity<String> operation2() {
  return ResponseEntity.ok("");
}
```

1 Example of `@ApiOperation#notes()`

2 Example of `@ApiParam#value()

3 Example of `@RequestParam#defaultValue()`

4 Example of `@ApiOperation#value()`

5 Example of `@ApiImplicitParams#value()`

Model Example

SomeModel.java

```java
public class SomeModel {
  @ApiModelProperty(value = "${SomeModel.someProperty}", ...) 1
  private long someProperty;
}
```

1 Example of `@ApiModelProperty#value()`

To provide these properties via external properties just add it to your application property file or any property source configured by the application as shown below. When a property place holder cannot be found the default behavior is to echo the expression as-is.

application.properties

```properties
SomeController.findPetsByStatus.notes=Finds pets by status
SomeController.findPetsByStatus.status=Status could be one of ...
SomeController.operation2.header1=Header for bla bla...
SomeController.operation2.value=Operation 2 do something...
SomeModel.someProperty=Some property description
```

6.6.1. Swagger group

A swagger group is a concept introduced by this library which is simply a unique identifier for a Swagger Resource Listing within your application. The reason this concept was introduced was to support applications which require more than one Resource Listing. Why would you need more than one Resource Listing? - A single Spring Web MVC application serves more than one API e.g. publicly facing and internally facing. - A single Spring Web MVC application serves multiple versions of the same API. e.g. v1 and v2

In most cases an application will not need more than one Resource Listing and the concept of swagger groups can be ignored.

6.6.2. Configuring the output of *operationId* in a Swagger 2.0 spec

As defined `operationId` was introduced (https://github.com/swagger-api/swagger-spec/blob/master/versions/2.0.md#fixed-fields-5) in the Swagger 2.0 spec, the `operationId` parameter, which was referred to as `nickname` in pre-2.0 versions of the Swagger spec, provides the author a means by which to describe an API operation with a friendly name . This field is often used by consumers of a Swagger 2.0 spec in order to name functions in generated clients. An example of this can be seen in the swagger-codegen project (https://github.com/swagger-api/swagger-codegen).

The default value of `operationId` according to Springfox

By default, when using Springfox in Swagger 2.0 mode, the value of `operationID` will be rendered using the following structure: "[java_method_name_here]Using[HTTP_verb_here]". For example, if one has a method `getPets()` connected to an HTTP GET verb, Springfox will render `getPetsUsingGET` for the operationId.

Given this annotated method …

```java
@ApiOperation(value = "")
@RequestMapping(value = "/pets", method = RequestMethod.GET)
public Model getAllThePets() {
    ...
}
```

the default `operationId` will render looking like this:

```json
"paths": {
  "/pets": {
    "get": {
            ...
      "operationId":"getAllThePetsUsingGET"
      ...
    }
  }
}
```

Customizing the value of *operationId*

In the event you wish to override the default `operationId` which Springfox renders, you may do so by providing the `nickname` element in an `@ApiOperation` annotation.

Given this annotated method …

```java
@ApiOperation(value = "", nickname = "getMeAllThePetsPlease")
@RequestMapping(value = "/pets", method = RequestMethod.GET)
public Model getAllThePets() {
    ...
}
```

… the customized **operationId** will render looking like this:

```json
"paths": {
  "/pets": {
    "get": {
            ...
      "operationId":"getMeAllThePetsPlease"
      ...
    }
  }
}
```

6.6.3. Changing how Generic Types are Named

By default, types with generics will be labeled with '\u00ab'(<<), '\u00bb'(>>), and commas. This can be problematic with things like swagger-codegen. You can override this behavior by implementing your own `GenericTypeNamingStrategy`. For example, if you wanted `List<String>` to be encoded as 'ListOfString' and `Map<String, Object>` to be encoded as 'MapOfStringAndObject' you could set the `forCodeGeneration` customization option to `true` during plugin customization:

```java
 docket.forCodeGeneration(true|false);
```

6.7. Caching

The caching feature that was introduced in 2.1.0 has been removed. Springfox no longer uses the cache abstraction to improve performance the api scanners and readers. It has been rolled into the library as an internal implementation detail as of 2.1.2. This is a runtime breaking change, however, since its not really breaking api compatibility change other than the introduction of configuration change in consuming applications, we're not incrementing the minor version.

6.8. Configuring Security Schemes and Contexts an Overview

The security provisions in SpringFox at a high level, without getting into the code, has different pieces that all work together in concert

- The API itself needs to be protected. This is achieved by using, for simplicity sake, spring security and may also use a combination of servlet container and tomcat/jersey etc.
- The security scheme which describes the techniques you've used to protect the api. Spring fox supports whatever schemes swagger specification supports (ApiKey, BasicAuth and OAuth2 (certain profiles))
- Finally the security contexts which actually provides information on which api's are protected by which schemes. I think in your example, you're missing the last piece of the puzzle, the security context see 15.

6.9. Example application

For an examples for spring-boot, vanilla spring applications take a look examples (https://github.com/springfox/springfox-demos) in the demo application.

7. Configuring springfox-staticdocs

Support for this module has been deprecated in 2.7.0. Since swagger2markup doesnt support jdk6 anymore it is difficult for build to co-exist with the newer version of swagger2markup. Please use the *latest* instructions provided in the awesome Swagger2Markup Library (https://github.com/Swagger2Markup/swagger2markup).

8. Security

Thanks to Javed Mohammed (https://github.com/mojaiq) we now have an example oauth demo (https://github.com/springfox/springfox-oath2-demo).

this is based on swagger-ui pre 3.x

9. Plugins

9.1. Introduction

Any plugin or extensibility hook that is available is available in the SPI module (https://github.com/springfox/springfox/tree/master/springfox-spi/src/main/java/springfox/documentation/spi). In the `spi` module, anything that ends in `*Plugin` is generally an extensibility point that is meant for library consumers to consume.

The bean validation (JSR-303) is a great example of a contribution to support bean validations (https://github.com/springfox/springfox/tree/master/springfox-bean-validators). Its fairly simple and small in scope and should give an idea how one goes about creating a plugin. Its a set of plugins are act on `ModelProperty`, hence they are implementations of `ModelPropertyBuilderPlugin`.

9.2. Plugins Available For Extensibility

To explicitly state the extensibility points that are available: * At the schema level .Schema Plugins

Name	Description
`ModelBuilderPlugin`	for enriching models
`ModelPropertyBuilderPlugin`	for enriching model properties

Name	Description
`TypeNameProviderPlugin`	these are for overriding names for models

Table 1. Service Plugins

Name	Description
`ApiListingScannerPlugin`	for adding custom api descriptions (see example).
`ApiListingBuilderPlugin`	for enriching api listings
`DefaultsProviderPlugin`	for providing your own defaults
`DocumentationPlugin`	for enriching the documentation context
`ExpandedParameterBuilderPlugin`	for parameter expansion used in the context of `@ModelAttribute`
`OperationBuilderPlugin`	for enriching operations
`OperationModelsProviderPlugin`	for providing additional models that you might load a different way
`ParameterBuilderPlugin`	for enriching parameters (see example)

9.3. Steps To Create A Plugin

The *same* patterns apply to all of the extensibility mechanisms

1. Implement one of the above plugin interfaces
2. Give the plugin an order for e.g. ApiParamParameterBuilder (https://github.com/springfox/springfox/blob/master/springfox-swagger-common/src/main/java/springfox/documentation/swagger/readers/parameter/ApiParamParameterBuilder.java#L42) has an order specified in the bean. In general spring plugins get the highest priority, The swagger plugins (the ones that process all the `@Api…` annotations) layer information on top. So the order that you'd write will need to layer information at the end.
3. Each plugin has
   - a *context (https://github.com/springfox/springfox/blob/master/springfox-swagger-common/src/main/java/springfox/documentation/swagger/readers/parameter/ApiParamParameterBuilder.java#L47) and provides access to any information that the plugin might need to do its job
   - a *builder (https://github.com/springfox/springfox/blob/master/springfox-swagger-common/src/main/java/springfox/documentation/swagger/readers/parameter/ApiParamParameterBuilder.java#L49) for the type of object that the plugin is intended to support for e.g. a `ModelPropertyBuilderPlugin` will have access to a `ModelPropertyBuilder`. This builder is what is used to build the model *after* all the plugins have had access to contribute/enrich the underlying object.
4. Update any builder properties your plugin cares about
5. Register the plugin as a `@bean`, so that the plugin registry can pick it up.

That is it!

9.3.1. Example ParameterBuilderPlugin

Here is an example of how to add parameters by hand.

Consider this controller, VersionedController.java (https://github.com/springfox/springfox/blob/master/springfox-spring-config/src/main/java/springfox/springconfig/VersionedController.java)

```java
 @GetMapping("/user")
 public ResponseEntity<User> getUser(
     @VersionApi int version,  1
     @RequestParam("id") String id) {
   throw new UnsupportedOperationException();
 }
}
```

1 Parameter annotated with `@VersionApi` (https://github.com/springfox/springfox/blob/master/springfox-spring-config/src/main/java/springfox/springconfig/VersionApi.java)

We then create a plugin VersionApiReader.java (https://github.com/springfox/springfox/blob/master/springfox-spring-config/src/main/java/springfox/springconfig/VersionApiReader.java) to provide custom parameter information.

```java
@Component
@Order(SwaggerPluginSupport.SWAGGER_PLUGIN_ORDER + 1000)  1
public class VersionApiReader implements ParameterBuilderPlugin {
  private TypeResolver resolver;

  public VersionApiReader(TypeResolver resolver) {
    this.resolver = resolver;
  }

  @Override
  public void apply(ParameterContext parameterContext) {
    ResolvedMethodParameter methodParameter = parameterContext.resolvedMethodParameter();
    Optional<VersionApi> requestParam = methodParameter.findAnnotation(VersionApi.class);
    if (requestParam.isPresent()) {  2
      parameterContext.parameterBuilder()
          .parameterType("header")
          .name("v")
          .type(resolver.resolve(String.class));  3
    }
  }

  @Override
  public boolean supports(DocumentationType documentationType) {
    return true;  4
  }
}
```

1 Specify an order for the plugin to execute. Higher the number, later the plugin is applied.

2 Check if the VersionApi (https://github.com/springfox/springfox/blob/master/springfox-spring-config/src/main/java/springfox/springconfig/VersionApi.java) is applied to the parameter.

3 Build the parameter with the necessary information using the builder methods.

4 Return true if we want this plugin to apply to all documentation types.

9.3.2. Example ApiListingScannerPlugin

Here is an example of how to add Api Descriptions by hand.

Bug1767ListingScanner.java (https://github.com/springfox/springfox/blob/master/swagger-contract-tests/src/main/java/springfox/test/contract/swagger/Bug1767ListingScanner.java)

```java
  @Override
  public List<ApiDescription> apply(DocumentationContext context) {
    return new ArrayList<ApiDescription>(
        Arrays.asList(  1
            new ApiDescription(
                "/bugs/1767",
                "This is a bug",
                Arrays.asList(  2
                    new OperationBuilder(
                        new CachingOperationNameGenerator())
                        .authorizations(new ArrayList())
                        .codegenMethodNameStem("bug1767GET")  3
                        .method(HttpMethod.GET)
                        .notes("This is a test method")
                        .parameters(
                            Arrays.asList(  4
                                new ParameterBuilder()
                                    .description("search by description")
                                    .type(new TypeResolver().resolve(String.class))
                                    .name("description")
                                    .parameterType("query")
                                    .parameterAccess("access")
                                    .required(true)
                                    .modelRef(new ModelRef("string"))  5
                                    .build()))
                        .build()),
                false)));
  }

  @Override
  public boolean supports(DocumentationType delimiter) {
    return DocumentationType.SWAGGER_2.equals(delimiter);
  }

}
```

1 Add a list of custom `ApiDescription` s

2 Add a list of `Operation` s for each description

3 NOTE: the code generated names are not guaranteed to be unique. For these custom endpoints it is the responsibility of the service author to ensure.

4 NOTE: It is important to also ensure we pass in a model reference even for primitive types

10. Additional extensibility options

Table 2. Other Extensibility

Name	Description
`RequestHandlerCombiner` [1]	for combining apis that react to the same API endpoint given the same input criteria but produces differe provide a `DefaultRequestHandlerCombiner` but this is an extensibility point added to customize it.
`AlternateTypeRuleConvention`	To provide a convention based type rules. Ideally we use these when its cumbersome to define individua because there are just too many, or even in some cases where it involves creating mixin types just for the documentation. `JacksonSerializerConvention` (https://github.com/springfox/springfox/blob/ef1721afc4c910675d9032bee59aea8e75e06d27/springfox-spring-web/src/main/java/springfox/documentation/spring/web/plugins/JacksonSerializerConvention.java) is an example of how we apply rules to types annotated with `JsonSerialize` / `JsonDeserialize` .
`SwaggerResourcesProvider`	Extensibility which allows overriding how the swagger resources are served. By default we serve APIs ho application. This can be used to aggregate APIs as well.

1. This has a shortcoming currently in that, currently the response still hides one of the response representations. This is a limitation of the OAI Spec 2.0.

10.1. Examples of alternate type rule convention

10.1.1. JacksonSerializerConvention

Here is how we configure the `JacksonSerializerConvention` .

FunctionContractSpec.java
(https://github.com/springfox/springfox/blob/c48a8bb019371cb1e3c79d24c3db3791ac5025cc/swagger-contract-tests/src/test/groovy/springfox/test/contract/swaggertests/FunctionContractSpec.groovy#L201-L204)

JAVA

```java
    @Bean
    AlternateTypeRuleConvention jacksonSerializerConvention(TypeResolver resolver) {
      new JacksonSerializerConvention(resolver, "springfox.documentation.spring.web.dummy.models")
    }
  }
}
```

10.1.2. Creating a convention

Below is an example for creating a rule that automatically provides a convention for configuring `Pageable` type.

SpringDataRestConfiguration.java
(https://github.com/springfox/springfox/blob/ef1721afc4c910675d9032bee59aea8e75e06d27/springfox-data-rest/src/main/java/springfox/documentation/spring/data/rest/configuration/SpringDataRestConfiguration.java#L46-L64)

JAVA

```java
  @Bean
  public AlternateTypeRuleConvention pageableConvention(
      final TypeResolver resolver,
      final RepositoryRestConfiguration restConfiguration) {
    return new AlternateTypeRuleConvention() {

      @Override
      public int getOrder() {
        return Ordered.HIGHEST_PRECEDENCE;
      }

      @Override
      public List<AlternateTypeRule> rules() {
        return newArrayList(
            newRule(resolver.resolve(Pageable.class), resolver.resolve(pageableMixin(restConfiguration)))
        );
      }
    };
  }

  private Type pageableMixin(RepositoryRestConfiguration restConfiguration) {
    return new AlternateTypeBuilder()
        .fullyQualifiedClassName(
            String.format("%s.generated.%s",
                Pageable.class.getPackage().getName(),
                Pageable.class.getSimpleName()))
        .withProperties(newArrayList(
            property(Integer.class, restConfiguration.getPageParamName()),
            property(Integer.class, restConfiguration.getLimitParamName()),
            property(String.class, restConfiguration.getSortParamName())
        ))
        .build();
  }

  private AlternateTypePropertyBuilder property(Class<?> type, String name) {
    return new AlternateTypePropertyBuilder()
        .withName(name)
        .withType(type)
        .withCanRead(true)
        .withCanWrite(true);
  }
}
```

It involves creating a generated in-memory type that allows the springfox inference engine to use the new type instead of `Pageable`.

SpringDataRestConfiguration.java
(https://github.com/springfox/springfox/blob/ef1721afc4c910675d9032bee59aea8e75e06d27/springfox-data-rest/src/main/java/springfox/documentation/spring/data/rest/configuration/SpringDataRestConfiguration.java#L68-L88)

```java
  private Type pageableMixin(RepositoryRestConfiguration restConfiguration) {
    return new AlternateTypeBuilder()
        .fullyQualifiedClassName(
            String.format("%s.generated.%s",
                Pageable.class.getPackage().getName(),
                Pageable.class.getSimpleName()))
        .withProperties(newArrayList(
            property(Integer.class, restConfiguration.getPageParamName()),
            property(Integer.class, restConfiguration.getLimitParamName()),
            property(String.class, restConfiguration.getSortParamName())
        ))
        .build();
  }

  private AlternateTypePropertyBuilder property(Class<?> type, String name) {
    return new AlternateTypePropertyBuilder()
        .withName(name)
        .withType(type)
        .withCanRead(true)
        .withCanWrite(true);
  }
}
```

10.2. Aggregating multiple swagger specifications in the same swagger-ui

You need to create a bean that implements the https://github .com/springfox/springfox/blob/cef36c0b0a3e20ef2cb0c23a76ee34866abaf490/springfox-swagger-common/src/main/java/springfox/documentation/swagger/web/SwaggerResourcesProvider.java#L25-L26[`SwaggerResourcesProvider`] interface. Typically you'd have to make a composite bean that uses multiple InMemorySwaggerResourcesProvider (https://github.com/springfox/springfox/blob/cef36c0b0a3e20ef2cb0c23a76ee34866abaf490/springfox-swagger-common/src/main/java/springfox/documentation/swagger/web/InMemorySwaggerResourcesProvider.java#L38) and adds your own json file to it as well.

You need to make this new bean a @Primary bean. otherwise the you'd get an exception about ambiguous beans.

```java
@Configuration
public class SwaggerWsEndpointsConfig {

    @Primary
    @Bean
    public SwaggerResourcesProvider swaggerResourcesProvider(InMemorySwaggerResourcesProvider defaultResourcesProvider) {
        return () -> {
            SwaggerResource wsResource = new SwaggerResource();
            wsResource.setName("ws endpoints");
            wsResource.setSwaggerVersion("2.0");
            wsResource.setLocation("/v2/websockets.json");

            List<SwaggerResource> resources = new ArrayList<>(defaultResourcesProvider.get());
            resources.add(wsResource);
            return resources;
        };
    }
}
```

11. Spring Data Rest extensibility

These are incubating features and may change.

Table 3. Spring Data REST Extensibility

Name	Description	Since

`EntityOperationsExtractor`	This is an extensibility point to extract entity specific operations. This extractor is to allow creation of `RequestHandler`s that are based on Spring-Data-Rest(SDR) endpoints. The SDR configuration information is available via `EntityAssociationContext` and can be used to infer the properties needed to produce the operation descriptions.	2.7.0
`EntityAssociationOperationsExtractor`	This is an extensibility point to extract operations related to entity associations. This extractor is to allow creation of `RequestHandler`s that are based on Spring-Data-Rest(SDR). The SDR configuration information is available via `EntityContext` and can be used to infer the properties needed to produce the operation descriptions.	2.7.0
`RequestHandlerExtractorConfiguration`	Additionally the glue that makes all this possible is the `RequestHandlerExtractorConfiguration`. When not customized, the library uses the `DefaultExtractorConfiguration`.	2.7.0

12. Answers to common questions and problems

Q. Why does springfox ignore the http status code in the return value of a controller method?

A. Reference #822 (https://github.com/springfox/springfox/issues/822#issuecomment-117372109)

Q. What is the relationship between swagger-ui and springfox-swagger-ui?

A. It can be a little confusing:

- Swagger Spec is a specification.
- Swagger Api - an implementation of that specification that supports jax-rs, restlet, jersey etc.
- Springfox libraries in general - another implementation of the specification focused on the spring based ecosystem.
- Swagger.js and Swagger-ui - are client libraries in javascript that can consume swagger specification.
- springfox-swagger-ui - the one that you're referring to, is just packaging swagger-ui in a convenient way so that spring services can serve it up.

Q. I use GSON and don't use Jackson, What should I do?

A. Thanks to @chrishuttonch (https://github.com/chrishuttonch) for describing the solution to this issue (https://github.com/springfox/springfox/issues/867)

> I switched on the excludeFieldsWithoutExposeAnnotation() which meant that none of the objects would produce any data. To get around this I created several serializers for the following classes:

JAVA

```
.registerTypeAdapter(springfox.documentation.service.ApiListing.class, new SwaggerApiListingJsonSerializer())
.registerTypeAdapter(springfox.documentation.spring.web.json.Json.class, new SwaggerJsonSerializer())
.registerTypeAdapter(springfox.documentation.swagger.web.SwaggerResource.class, new SwaggerResourceSerializer())
.registerTypeAdapter(springfox.documentation.service.ResourceListing.class, new SwaggerResourceListingJsonSerializer())
.registerTypeAdapter(springfox.documentation.swagger.web.SwaggerResource.class, new SwaggerResourceSerializer())
.registerTypeAdapter(springfox.documentation.swagger.web.SecurityConfiguration.class, new
SwaggerSecurityConfigurationSerializer())
.registerTypeAdapter(springfox.documentation.swagger.web.UiConfiguration.class, new SwaggerUiConfigurationSerializer());
```

Q. ObjectMapper weirdness in a spring-boot app?

A. It is possible you're experiencing one of the following issues

1. NPE During startup?

> Running in debugger revealed that I had two instances of WebApplicationInitializers in my war. Spring is refreshing context with each one and is resulting in second instance of `OptimizedModelPropertiesProvider` without `onApplicationEvent` call. I was able to fix it by removing second `WebApplicationInitializer` in my code. Seems this is related to spring-boot issue #221 (https://github.com/spring-projects/spring-boot/issues/221) [1]

2. Object Mapper Customizations Not Working?

> Sometimes there are multiple `ObjectMapper` in play and it may result in the customizations not working [2] Spring Boot in `HttpMessageConverters` first adds the Spring Boot configured `MappingJackson2HttpMessageConverter` and then it adds the default `MappingJackson2HttpMessageConverter` from Spring MVC. This causes the `ObjectMapperConfigured` event to fire twice, first for the configured converter (which is actually used) and then for the default converter. So when you f.e. set a custom property naming strategy then in `ObjectMapperBeanPropertyNamingStrategy` this is overwritten by the second event. The following code fixes this:

JAVA

```java
@Configuration
public class MyWebAutoConfiguration extends WebMvcConfigurerAdapter {

    @Override
    public void extendMessageConverters(List<HttpMessageConverter<?>> converters) {
        ObjectMapper objectMapper = null;
        for (HttpMessageConverter converter : converters) {
            if (converter instanceof MappingJackson2HttpMessageConverter) {
                MappingJackson2HttpMessageConverter jacksonConverter =
                        ((MappingJackson2HttpMessageConverter) converter);

                if (objectMapper == null) {
                    objectMapper = jacksonConverter.getObjectMapper();
                } else {
                    jacksonConverter.setObjectMapper(objectMapper);
                }
            }
        }
    }
}
```

Q. How do we use Java 8 types easily. LocalDateTime?

A. The easiest way to to configure dates is via `Docket#directModelSubstitute(LocalDateTime.class, String.class)`. If these are ISO 8601 dates that conform to a string format i.e. `yyyy-MM-dd'T'HH:mm'Z'`. However you won't have any format or validation info.

> Use `java.sql.Date` works great for date precision and `java.util.Date` for date-time precision [3]

The way to correctly map the "Date" and "DateTime" types to their corresponding swagger types:

- Substitute "Date" types (java.util.LocalDate, org.joda.time.LocalDate) by java.sql.Date.

- Substitute "DateTime" types (java.util.ZonedDateTime, org.joda.time.LocalDateTime, ...) by java.util.Date.

```csharp
docket
.directModelSubstitute(LocalDate.class, java.sql.Date.class)
.directModelSubstitute(LocalDateTime.class, java.util.Date.class)
```

Q. How does one use `@ModelAttribute` annotation. It doesn't seem to render the model properties as scalar properties?

A. In order for `@ModelAttribute` annotated types to be inferred the properties need to be bean properties. If the intent is immutability and passing in an object, the preferred approach is to make that a request body, in which case the immutability will follow the rules laid out by jackson to determine what constitutes a request "view" of the request object.

Getters/setters are a clean way to indicate what values can come in to a operation. While it may not be apparent in a trivial model with one level nesting; the design choice will become clear when we realize that model attributes can be arbitrarily nested. Consider (pseudo code in C# for brevity)

```csharp
Person {String firstName {get;set;}
String lastName {get;set;}
Category category {get;set;}
Category {String name {get;set;}
String description {get;}
```

So one could set properties:

- firstName
- lastName
- category.name

Now we don't want category to be able to set description via the operation, how do we control/specify that? It makes it hard to reason about which fields in an object are not intended to be mutated. This is the reason we chose to limit it to objects that expose getters and setters.

I know spring supports fields as well, and it will fall back to fields if setters are not found.

Q. How should we resolve multiple object mappers available as beans especially when using spring-hateoas?

A. The idea is to provide a `@Primary` ObjectMapper. Based on answer provided by @prabhat1790 in issue #890 (https://github.com/springfox/springfox/issues/890)

```java
private static final String SPRING_HATEOAS_OBJECT_MAPPER = "_halObjectMapper";

@Autowired
@Qualifier(SPRING_HATEOAS_OBJECT_MAPPER)
private ObjectMapper springHateoasObjectMapper;

@Primary
@Bean
@Order(value=Ordered.HIGHEST_PRECEDENCE)
@DependsOn(SPRING_HATEOAS_OBJECT_MAPPER)
public ObjectMapper objectMapper() {
  return springHateoasObjectMapper;
}
```

and set the order of the other bean to lowest precedence.

Q. How do I use this library to aggregate swagger-enabled resources from multiple services?

A. Logical explanation of how one might go about doing this is available in the swagger google group (https://groups.google.com/forum/#!searchin/swagger-swaggersocket/multiple/swagger-swaggersocket/g8fgSGUCrYs/A8Ms_lFOoN4J) Additionally this comment (https://github.com/springfox/springfox/issues/1001#issuecomment-147609243) further discusses issues with doing this.

Q. Why are my API params marked as required=false?

A. This is because of how plugins work and how their priority layers information

- `@PathVariables` are always marked as required.
- `@ApiParam` is an optional annotation to describe additional meta-information like description etc.
- `@ApiParam#required()` is defaulted to false, unless you set it to true.

Springfox uses plugins to layer information. There are a set of plugins that are spring specific that apply the inferred values on to the internal service models. The swagger annotation related metadata is layered on top of the spring-mvc descriptions. By definition, plugins don't know and should not know about each other or previously inferred values (in your case required attribute).

So if you choose to augment the definitions with `@ApiParam` then you need to be explicit and set the value to true.

Q. How does one write a plugin to e.g. make default all types required and only some not required?

A. To do this, you'd have to

- add an alternate type rule for `Optional<T>` see `genericModelSubstitutes` in docket
- implement your own ModelPropertyBuilderPlugin (https://github.com/springfox/springfox/blob/master/springfox-spi/src/main/java/springfox/documentation/spi/schema/ModelPropertyBuilderPlugin.java#L26)
- and override the read only property if you find an `Optional` type. See here (https://github.com/springfox/springfox/blob/master/springfox-swagger-common/src/main/java/springfox/documentation/swagger/schema/ApiModelPropertyPropertyBuilder.java#L35) for an example.

Keep in mind that you need the plugin to fire after this plugin… so order it accordingly

Q. Why are all my operations not showing in the UI?

A. This is a known limitation of swagger-spec. There is a work around for it but, swagger-ui won't play nice with it. I have a PR (https://github.com/swagger-api/swagger-js/pull/541) which address that issue. Would be great if you vote up the PR and the underlying issue (https://github.com/swagger-api/swagger-spec/issues/291)

> ⚠ This PR (https://github.com/swagger-api/swagger-js/pull/541) has been closed!

Q. How would one partition apis based on versions?

A. Excerpted from an explanation for issue 963 (https://github.com/springfox/springfox/issues/963)…

(springfox) uses the context path as the starting point.

What you really need to is to define a dynamic servlet registration and create 2 dockets .. one for **api** and one for **api/v2**. This SO post (http://stackoverflow.com/questions/23049736/working-with-multiple-dispatcher-servlets-in-a-spring-application) might help

JAVA
```java
...
  Dynamic servlet = servletContext.addServlet("v1Dispatcher", new DispatcherServlet(ctx1));
        servlet.addMapping("/api");
        servlet.setLoadOnStartup(1);

  Dynamic servlet = servletContext.addServlet("v2Dispatcher", new DispatcherServlet(ctx2));
        servlet.addMapping("/api/v2");
        servlet.setLoadOnStartup(1);
```

Q. How does one configure swagger-ui for non-springboot applications?

A. Excerpted from issue 983 (https://github.com/springfox/springfox/issues/983)…

I was able to get it working by modifying the `dispatcherServlet` to listen on /* , but this prevented `swagger-ui.html` from being served. To fix this to let the `swagger-ui.html` bypass the `dispatcherServlet` i had to create a new servlet mapping:

XML
```xml
<servlet>
  <servlet-name>RestServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value></param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>default</servlet-name>
    <url-pattern>/swagger-ui.html</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>RestServlet</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Also had to let the webjar through the dispatcher servlet:

```
<mvc:resources mapping="/webjars/**" location="classpath:/META-INF/resources/webjars/"/>
```

Tricky to get working, but it works. Perhaps there is a better way to remap swagger-ui.html or let it pass through the dispatcherServlet.

> `swagger-ui.html` is the name of the swagger-ui page. While it cannot be changed one can configure the application such that landing on a particular URL re-directs the browser to the real swagger-ui location. [4]

For e.g. One could move Swagger UI under `/documentation` using this code.

JAVA
```java
@Override
public void addViewControllers(ViewControllerRegistry registry) {

        registry.addRedirectViewController("/documentation/v2/api-docs", "/v2/api-docs?group=restful-api");
        registry.addRedirectViewController("/documentation/swagger-resources/configuration/ui","/swagger-
resources/configuration/ui");
        registry.addRedirectViewController("/documentation/swagger-resources/configuration/security","/swagger-
resources/configuration/security");
        registry.addRedirectViewController("/documentation/swagger-resources", "/swagger-resources");
}

@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
         registry.
                addResourceHandler("/documentation/swagger-ui.html**").addResourceLocations("classpath:/META-
INF/resources/swagger-ui.html");
        registry.
               addResourceHandler("/documentation/webjars/**").addResourceLocations("classpath:/META-
INF/resources/webjars/");
}
```

However, it still requires a redirect to `/documentation/swagger-ui.html` because the path name is hard-coded (https://github.com/springfox/springfox/blob/master/springfox-swagger-ui/src/web/js/springfox.js#L4).

Q. How does one create rules to substitute list and array items?

A. If the following types…

JAVA
```java
ToSubstitute[] array;
List<ToSubstitute> list;
```

Need to look like this over the wire…

```java
Substituted[] array;
List<Substituted> list;
```

This is how the rules need to be configured

```java
rules.add(newRule(resolver.arrayType(ToSubstitute), resolver.arrayType(Substituted)))
rules.add(newRule(resolver.resolve(List, ToSubstitute), resolver.resolve(List, Substituted)))
```

Q. How does one configure a docket with multiple protocols/schemes?

A. Use the `protocols` method to configure the docket to indicate supported schemes.

```java
docket.protocols(newHashSet("http", "https"))
```

Q. How does one use springfox in a project with xml spring configuration?

A. There is a demo application that describes how java-xml (https://github.com/springfox/springfox-demos/tree/master/spring-xml-swagger) configuration needs to be setup.

Q. How does one override the host name?

A. This should be available in v2.3 thanks to this PR (https://github.com/springfox/springfox/pull/1011) by @cbornet. It is still in incubation but host name can be configured per docket

```java
 docket.host("http://maybe-an-api-gateway.host");
```

Q. Infinite loop when springfox tries to determine schema for objects with nested/complex constraints?

A. If you have recursively defined objects, I would try and see if providing an alternate type might work or perhaps even ignoring the offending classes e.g. order using the docket. ignoredParameterTypes(Order.class). This is usually found in Hibernate domain objects that have bidirectional dependencies on other objects.

Q. How are tags implemented in springfox?

Tags which are first class constructs just like operations, models etc. and what you see on operations are references to those Tags. The typical workflow is to register tags in a docket and use the tag definitions on operations(`@ApiOperation`)/controllers(`@Api`) to point to these registered tags (in the docket) by name.

The convenience we have in place just to reduce the amount of boiler plate for the developer is to provide a default description that happens to be the same as the tag name. So in effect we are synthesizing a pseudo Tag by referencing one on the operation.

By defining the Tag on the docket, we are referencing a real tag defined by you.

20. What can I try if configuration non-boot applications do not work as expected?

Thanks to @Pyohwan (https://github.com/Pyohwan)'s suggestion `@Configuration` annotation may not working with @EnableSwagger2. So shouldn't attach @Configration. So if you have a configuration class that pulls in the springfox configuration using the `@EnableSwagger2` like below, try removing the `@Configuration` on this class as shown below.

```java
@EnableSwagger2
public class SwaggerConfig {
...
```

and use `@Import` annotation on `WebMvcConfigurerAdapter` or similar configuration class.

```java
@Configuration
@EnableWebMvc
@ComponentScan(...)
@Import(SwaggerConfig.class)
public class MvcConfig extends WebMvcConfigurerAdapter {
...
```

*21. How to add CORS support? (thanks @gangakrishh (https://github.com/gangakrishh))

Based on the spring guide (https://spring.io/guides/gs/rest-service-cors/), Creating a `WebMvcConfigurer` we can configure a request mapping to allow specific origins.

```java
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurerAdapter() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/some-request-mapping").allowedOrigins("http://localhost:9000");
            }
        };
```

*22. How to configure the docket when using Immutables (https://immutables.github.io)? (thanks https://github.com/kevinm416[@kevinm416])

This is related to #1490 (https://github.com/springfox/springfox/issues/1490).The way to configure this is to create a custom alternateTypeRules in the `Docket` config. For e.g. if you have an immutable `MyClass` that generates `ImmutableMyClass`, then we would add a rule for it as shown below.

```java
@Bean
public Docket docket() {
  return new Docket(DocumentationType.SWAGGER_2)
    .alternateTypeRules(AlternateTypeRules.newRule(MyClass.class,
                ImmutableMyClass.class));
```

If you're using a library for you models, you may need to make the Immutable visible outside the package.

*23. Why does setting `@ApiImplicitParam#paramType="form"` output undefined dataType?

If you change the datatype to "__file" once 2.7.0 is released, it will fix your issue.

The reason we use "__file" is because if a consuming library defines a custom type `File` then that type ends up getting treated just as if it were a `file` data type, even if that wasn't the intent. To distinguish the usage of the custom type from the natively understood `file` primitive we introduced this convention.

*24. How to configure springfox for Vavr/Javaslang Jackson module support?

We need to first create an alternate type rules convention that tells springfox to treat the vavr persistent collections just like the native java collections. The following example only configurers Set, List and Map types.

```java
package springfox.test.contract.swagger;

import com.fasterxml.classmate.TypeResolver;
import io.vavr.collection.Map;
import io.vavr.collection.Set;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import springfox.documentation.schema.AlternateTypeRule;
import springfox.documentation.schema.AlternateTypeRuleConvention;
import springfox.documentation.schema.WildcardType;

import java.util.ArrayList;
import java.util.List;

import static springfox.documentation.schema.AlternateTypeRules.newRule;

@Component
public class VavrDefaultsConvention implements AlternateTypeRuleConvention {
    @Autowired
    private TypeResolver typeResolver;

    @Override
    public List<AlternateTypeRule> rules() {
        ArrayList<AlternateTypeRule> rules = new ArrayList<AlternateTypeRule>();
        rules.add(
                newRule(
                        typeResolver.resolve(io.vavr.collection.List.class, WildcardType.class),
                        typeResolver.resolve(List.class, WildcardType.class)));
        rules.add(
                newRule(
                        typeResolver.resolve(Set.class, WildcardType.class),
                        typeResolver.resolve(java.util.Set.class, WildcardType.class)));
        rules.add(
                newRule(
                        typeResolver.resolve(Map.class, WildcardType.class, WildcardType.class),
                        typeResolver.resolve(java.util.Map.class, WildcardType.class, WildcardType.class)));
        return rules;
    }

    @Override
    public int getOrder() {
        return HIGHEST_PRECEDENCE;
    }
}
```

view raw (https://gist.github.com/dilipkrish/2bf80a0285a04b4037c8994fd8cb8b80/raw/03b1dddc1f318635c92283a2851ec85a64ad77f8/VavrDefaultsConvention.java) VavrDefaultsConvention.java (https://gist.github.com/dilipkrish/2bf80a0285a04b4037c8994fd8cb8b80#file-vavrdefaultsconvention-java) hosted with ❤ by GitHub (https://github.com)

Optionally, to use jackson effectively, for non-collection types we can register an application listener that registers the vavr module with the objectmapper that is being used.

```java
@Component
public class SwaggerJacksonModuleWithVavr implements ApplicationListener<ObjectMapperConfigured> {

  @Override
  public void onApplicationEvent(ObjectMapperConfigured event) {
    event.getObjectMapper().registerModule(new VavrModule());
  }
}
```

*25. Why are properties missing from certain models/types?

A good example of when this might happen is when using Immutables (the library) that are not particularly obvious. Immutables are great, except that, unless u craft your model correctly it might lead to confusing results, as you’ve seen. Also how springfox infers the model is opposite to how the service consumes it. By that I mean that, take for e.g. Model has properties a (read/write), b (write only), c (readonly) we read the writable properties for requests and readable properties for responses.

So if your model is only used for requests, it will have a and b And if your model is only used of responses it will have a and c and if it is present in both request and response it will have a, b and c.

We’re working to make this better thanks to model enhancements work (https://github.com/springfox/springfox/pull/2056) by Maksim (https://github.com/MaksimOrlov)

*26. Why is `http://host:port/swagger-ui.html` blank? Or why do I get a 404 when I navigate to `http://host:port/swagger-ui.html`?

In non-spring boot application ensure that the resource handlers are added for the springfox-swagger-ui webjars

If its a spring boot application:

- check to see if the `spring.resources.add-mappings` property is set to true.
- check to see if spring security is applied that the appropriate resources are permitted.

JAVA

```java
@Configuration
public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {
  @Override
  public void configure(WebSecurity web) throws Exception {
    http
      .csrf().disable()
      .exceptionHandling()
      .authenticationEntryPoint(unauthorizedHandler)
      .accessDeniedHandler(accessDeniedHandler)
      .and()
      .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
      .and()
      .authorizeRequests()
      //.antMatchers("/actuator/**").permitAll()
      .antMatchers("/actuator/**").hasAuthority("ADMIN")
      .antMatchers(
        HttpMethod.GET,
        "/v2/api-docs",
        "/swagger-resources/**",
        "/swagger-ui.html**",
        "/webjars/**",
        "favicon.ico"
      ).permitAll()
      .antMatchers("/auth/**").permitAll()
      .anyRequest().authenticated();

    http
      .addFilterBefore(authenticationTokenFilterBean(), UsernamePasswordAuthenticationFilter.class)
      .headers()
      .cacheControl();
  }
}
```

Or we could simply ignore these resources altogether

JAVA

```java
@Configuration
public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {
  @Override
  public void configure(WebSecurity web) throws Exception {
    web.ignoring()
      .antMatchers(
        "/v2/api-docs",
        "/swagger-resources/**",
        "/swagger-ui.html**",
        "/webjars/**");
  }
}
```

*27. Controller receives parameter as a String but we are expecting a json Type. How can we model this?

FeatureDemonstrationService.class
(https://github.com/springfox/springfox/blob/master/springfox-spring-web/src/test/java/springfox/documentation/spring/web/dummy/controllers/FeatureDemonstrationService.java#L243-267)

```java
@RequestMapping(value = "/2031", method = RequestMethod.POST)
@ResponseBody
@ApiOperation(value = "/2031")
@ApiImplicitParams({
    @ApiImplicitParam(name="contents", dataType = "CustomTypeFor2031") 1
})
public void save(@PathVariable("keyId") String keyId,
                  @PathVariable("id") String id,
                  @RequestBody String contents 2
                ) {
}

public static class CustomTypeFor2031 { 3
  private String property;

  public String getProperty() {
    return property;
  }

  public void setProperty(String property) {
    this.property = property;
  }
}
```

1 Assume we have a controller action that has a `String` parameter (contents). This is how we can override the on-the-wire type (CustomTypeFor2031).

2 The method parameter that comes in as a `String` that is actually interpreted as `CustomTypeFor2031`

3 `CustomTypeFor2031` complex type that we read from json String.

Once we do this we need to configure this change so that we can add the model to the definitions section.

Swagger2TestConfig (https://github.com/springfox/springfox/blob/eedbfb1b2dcf0344e21aefd0731515aa63fc9070/swagger-contract-tests/src/test/groovy/springfox/test/contract/swaggertests/Swagger2TestConfig.groovy#L118)

```java
return new Docket(DocumentationType.SWAGGER_2)
    .groupName("featureService")
    .useDefaultResponseMessages(false)
    .additionalModels(resolver.resolve(FeatureDemonstrationService.CustomTypeFor2031.class)) 1
```

1 Add `CustomTypeFor2031` as a additional model.

---

1. Thanks to @shasti421 (https://github.com/shasti421)
2. thanks Jaap-van-Hengstum (https://github.com/springfox/springfox/issues/1140)
3. thanks @cbornet (https://github.com/springfox/springfox/issues/1161)
4. Thanks @chornyi (https://github.com/springfox/springfox/issues/1080#issuecomment-169185653)

Version 2.8.1-SNAPSHOT
Last updated 2018-01-14 16:26:28 CST