Spring Boot Actuator Web API Documentation

Andy Wilkinson
# Table of Contents

1. Overview .................................................................................................................. 2  
   1.1. URLs .................................................................................................................. 2  
   1.2. Timestamps ......................................................................................................... 2  
2. Audit Events (auditevents) ..................................................................................... 3  
   2.1. Retrieving Audit Events .................................................................................... 3  
      2.1.1. Query Parameters ....................................................................................... 3  
      2.1.2. Response Structure .................................................................................. 3  
3. Beans (beans) .......................................................................................................... 5  
   3.1. Retrieving the Beans .......................................................................................... 5  
      3.1.1. Response Structure .................................................................................... 6  
4. Caches (caches) ....................................................................................................... 8  
   4.1. Retrieving All Caches ......................................................................................... 8  
      4.1.1. Response Structure .................................................................................... 8  
   4.2. Retrieving Caches by Name ............................................................................. 9  
      4.2.1. Query Parameters ...................................................................................... 9  
      4.2.2. Response Structure .................................................................................. 9  
   4.3. Evict All Caches ............................................................................................... 10  
   4.4. Evict a Cache by Name .................................................................................... 10  
      4.4.1. Request Structure ...................................................................................... 10  
5. Conditions Evaluation Report (conditions) ........................................................... 11  
   5.1. Retrieving the Report ....................................................................................... 11  
      5.1.1. Response Structure .................................................................................... 12  
6. Configuration Properties (configprops) ................................................................. 14  
   6.1. Retrieving the @ConfigurationProperties Bean ............................................. 14  
      6.1.1. Response Structure .................................................................................... 15  
7. Environment (env) .................................................................................................. 16  
   7.1. Retrieving the Entire Environment ................................................................... 16  
      7.1.1. Response Structure .................................................................................... 17  
   7.2. Retrieving a Single Property ........................................................................... 18  
      7.2.1. Response Structure .................................................................................... 18  
8. Flyway (flyway) ...................................................................................................... 20  
   8.1. Retrieving the Migrations .................................................................................. 20  
      8.1.1. Response Structure .................................................................................... 20  
9. Health (health) ....................................................................................................... 22  
   9.1. Retrieving the Health of the application .......................................................... 22  
      9.1.1. Response Structure .................................................................................... 23  
   9.2. Retrieving the Health of a component ............................................................. 24  
      9.2.1. Response Structure .................................................................................... 24
This API documentation describes Spring Boot Actuators web endpoints.
Chapter 1. Overview

Before you proceed, you should read the following topics:

- URLs
- Timestamps

In order to get the correct JSON responses documented below, Jackson must be available.

1.1. URLs

By default, all web endpoints are available beneath the path `/actuator` with URLs of the form `/actuator/{id}`. The `/actuator` base path can be configured by using the `management.endpoints.web.base-path` property, as shown in the following example:

```properties
management.endpoints.web.base-path=/manage
```

The preceding `application.properties` example changes the form of the endpoint URLs from `/actuator/{id}` to `/manage/{id}`. For example, the URL `info` endpoint would become `/manage/info`.

1.2. Timestamps

All timestamps that are consumed by the endpoints, either as query parameters or in the request body, must be formatted as an offset date and time as specified in ISO 8601.
Chapter 2. Audit Events (**auditevents**)

The **auditevents** endpoint provides information about the application’s audit events.

### 2.1. Retrieving Audit Events

To retrieve the audit events, make a **GET** request to `/actuator/auditevents`, as shown in the following curl-based example:

```
```

The preceding example retrieves **logout** events for the principal, **alice**, that occurred after 09:37 on 7 November 2017 in the UTC timezone. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 121

{
  "events" : [ {
    "timestamp" : "2020-10-29T10:24:42.727Z",
    "principal" : "alice",
    "type" : "logout"
  } ]
}
```

### 2.1.1. Query Parameters

The endpoint uses query parameters to limit the events that it returns. The following table shows the supported query parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>after</td>
<td>Restrictions the events to those that occurred after the given time. Optional.</td>
</tr>
<tr>
<td>principal</td>
<td>Restrictions the events to those with the given principal. Optional.</td>
</tr>
<tr>
<td>type</td>
<td>Restrictions the events to those with the given type. Optional.</td>
</tr>
</tbody>
</table>

### 2.1.2. Response Structure

The response contains details of all of the audit events that matched the query. The following table describes the structure of the response:
<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>events</td>
<td>Array</td>
<td>An array of audit events.</td>
</tr>
<tr>
<td>events[].timestamp</td>
<td>String</td>
<td>The timestamp of when the event occurred.</td>
</tr>
<tr>
<td>events[].principal</td>
<td>String</td>
<td>The principal that triggered the event.</td>
</tr>
<tr>
<td>events[].type</td>
<td>String</td>
<td>The type of the event.</td>
</tr>
</tbody>
</table>
Chapter 3. Beans (beans)

The beans endpoint provides information about the application's beans.

3.1. Retrieving the Beans

To retrieve the beans, make a GET request to /actuator/beans, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/beans' -i -X GET
```

The resulting response is similar to the following:
3.1.1. Response Structure

The response contains details of the application’s beans. The following table describes the structure of the response:
<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contexts</td>
<td>Object</td>
<td>Application contexts keyed by id.</td>
</tr>
<tr>
<td>contexts.*.parentId</td>
<td>String</td>
<td>Id of the parent application context, if any.</td>
</tr>
<tr>
<td>contexts.*.beans</td>
<td>Object</td>
<td>Beans in the application context keyed by name.</td>
</tr>
<tr>
<td>contexts.<em>.beans.</em>.aliases</td>
<td>Array</td>
<td>Names of any aliases.</td>
</tr>
<tr>
<td>contexts.<em>.beans.</em>.scope</td>
<td>String</td>
<td>Scope of the bean.</td>
</tr>
<tr>
<td>contexts.<em>.beans.</em>.type</td>
<td>String</td>
<td>Fully qualified type of the bean.</td>
</tr>
<tr>
<td>contexts.<em>.beans.</em>.resource</td>
<td>String</td>
<td>Resource in which the bean was defined, if any.</td>
</tr>
<tr>
<td>contexts.<em>.beans.</em>.dependencies</td>
<td>Array</td>
<td>Names of any dependencies.</td>
</tr>
</tbody>
</table>
Chapter 4. Caches (caches)

The caches endpoint provides access to the application’s caches.

4.1. Retrieving All Caches

To retrieve the application’s caches, make a GET request to /actuator/caches, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/caches' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 435

{
   "cacheManagers" : {
      "anotherCacheManager" : {
         "caches" : {
            "countries" : {
               "target" : "java.util.concurrent.ConcurrentHashMap"
            }
         }
      },
      "cacheManager" : {
         "caches" : {
            "cities" : {
               "target" : "java.util.concurrent.ConcurrentHashMap"
            },
            "countries" : {
               "target" : "java.util.concurrent.ConcurrentHashMap"
            }
         }
      }
   }
}
```

4.1.1. Response Structure

The response contains details of the application’s caches. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cacheManagers</td>
<td>Object</td>
<td>Cache managers keyed by id.</td>
</tr>
</tbody>
</table>
4.2. Retrieving Caches by Name

To retrieve a cache by name, make a GET request to `/actuator/caches/{name}`, as shown in the following curl-based example:

$ curl 'http://localhost:8080/actuator/caches/cities' -i -X GET

The preceding example retrieves information about the cache named `cities`. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 113

{
   "target" : "java.util.concurrent.ConcurrentHashMap",
   "name" : "cities",
   "cacheManager" : "cacheManager"
}
```

### 4.2.1. Query Parameters

If the requested name is specific enough to identify a single cache, no extra parameter is required. Otherwise, the `cacheManager` must be specified. The following table shows the supported query parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cacheManager</td>
<td>Name of the cacheManager to qualify the cache. May be omitted if the cache name is unique.</td>
</tr>
</tbody>
</table>

### 4.2.2. Response Structure

The response contains details of the requested cache. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Cache name.</td>
</tr>
<tr>
<td>cacheManager</td>
<td>String</td>
<td>Cache manager name.</td>
</tr>
<tr>
<td>Path</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>target</td>
<td>String</td>
<td>Fully qualified name of the native cache.</td>
</tr>
</tbody>
</table>

### 4.3. Evict All Caches

To clear all available caches, make a DELETE request to `/actuator/caches` as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/caches' -i -X DELETE
```

### 4.4. Evict a Cache by Name

To evict a particular cache, make a DELETE request to `/actuator/caches/{name}` as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/caches/countries?cacheManager=anotherCacheManager' -i -X DELETE
```

As there are two caches named `countries`, the `cacheManager` has to be provided to specify which Cache should be cleared.

#### 4.4.1. Request Structure

If the requested name is specific enough to identify a single cache, no extra parameter is required. Otherwise, the `cacheManager` must be specified. The following table shows the supported query parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cacheManager</td>
<td>Name of the cacheManager to qualify the cache. May be omitted if the cache name is unique.</td>
</tr>
</tbody>
</table>
Chapter 5. Conditions Evaluation Report

(conditions)

The conditions endpoint provides information about the evaluation of conditions on configuration and auto-configuration classes.

5.1. Retrieving the Report

To retrieve the report, make a GET request to /actuator/conditions, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/conditions' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 3259

{
    "contexts" : {
        "application" : {
            "positiveMatches" : {
                "EndpointAutoConfiguration#endpointOperationParameterMapper" : [ {
                    "condition" : "OnBeanCondition",
                    "message" : "@ConditionalOnMissingBean (types:
                        org.springframework.boot.actuate.endpoint.invoke.ParameterValueMapper; SearchStrategy: all) did not find any beans"
                } ],
                "EndpointAutoConfiguration#endpointCachingOperationInvokerAdvisor" : [ {
                    "condition" : "OnBeanCondition",
                    "message" : "@ConditionalOnMissingBean (types:
                        org.springframework.boot.actuate.endpoint.invoker.cache.CachingOperationInvokerAdvisor; SearchStrategy: all) did not find any beans"
                } ],
                "WebEndpointAutoConfiguration" : [ {
                    "condition" : "OnWebApplicationCondition",
                    "message" : "@ConditionalOnWebApplication (required) found 'session' scope"
                } ]
            },
            "negativeMatches" : {
                "WebFluxEndpointManagementContextConfiguration" : [ {
                    "notMatched" : [ {
                        "condition" : "OnWebApplicationCondition",
                        "message" : "not a reactive web application"
                    } ],
                    "matched" : [ {
```
5.1.1. Response Structure

The response contains details of the application's condition evaluation. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ConditionalOnClass found required classes</td>
<td>@ConditionalOnClass found required classes</td>
</tr>
<tr>
<td>'org.springframework.web.reactive.DispatcherHandler'</td>
<td>@ConditionalOnClass found required classes</td>
</tr>
<tr>
<td>'org.springframework.http.server.reactive.HttpHandler'</td>
<td>@ConditionalOnClass found required classes</td>
</tr>
<tr>
<td>Path</td>
<td>Type</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>contexts</td>
<td>Object</td>
</tr>
<tr>
<td>contexts.*.positiveMatches</td>
<td>Object</td>
</tr>
<tr>
<td>contexts.<em>.positiveMatches.</em>.condition</td>
<td>String</td>
</tr>
<tr>
<td>contexts.<em>.positiveMatches.</em>.message</td>
<td>String</td>
</tr>
<tr>
<td>contexts.*.negativeMatches</td>
<td>Object</td>
</tr>
<tr>
<td>contexts.<em>.negativeMatches.</em>.notMatched</td>
<td>Array</td>
</tr>
<tr>
<td>contexts.<em>.negativeMatches.</em>.notMatched.*.con</td>
<td>String</td>
</tr>
<tr>
<td>contexts.<em>.negativeMatches.</em>.notMatched.*.messag</td>
<td>String</td>
</tr>
<tr>
<td>contexts.<em>.negativeMatches.</em>.matched</td>
<td>Array</td>
</tr>
<tr>
<td>contexts.<em>.negativeMatches.</em>.matched.*.con</td>
<td>String</td>
</tr>
<tr>
<td>contexts.<em>.negativeMatches.</em>.matched.*.message</td>
<td>String</td>
</tr>
<tr>
<td>contexts.*.unconditionalClasses</td>
<td>Array</td>
</tr>
<tr>
<td>contexts.*.parentId</td>
<td>String</td>
</tr>
</tbody>
</table>
Chapter 6. Configuration Properties (configprops)

The configprops endpoint provides information about the application's @ConfigurationProperties beans.

6.1. Retrieving the @ConfigurationProperties Bean

To retrieve the @ConfigurationProperties beans, make a GET request to /actuator/configprops, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/configprops' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 1806

{
  "contexts": {
    "application": {
      "beans": {
        "management.endpoints.web.cors-
org.springframework.boot.actuate.autoconfigure.endpoint.web.CorsEndpointProperties": {
          "prefix": "management.endpoints.web.cors",
          "properties": {
            "allowedHeaders": [ ],
            "allowedMethods": [ ],
            "allowedOrigins": [ ],
            "maxAge": {
              "units": [ "SECONDS", "NANOS" ]
            },
            "exposedHeaders": [ ]
          }
        },
        "management.endpoints.web-
org.springframework.boot.actuate.autoconfigure.endpoint.web.WebEndpointProperties": {
          "prefix": "management.endpoints.web",
          "properties": {
            "pathMapping": { },
            "exposure": {
              "include": [ "*" ],
              "exclude": [ ]
            }
          },
          "basePath": "/actuator"
        }
      }
    }
  }
}```
6.1.1. Response Structure

The response contains details of the application's `@ConfigurationProperties` beans. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contexts</td>
<td>Object</td>
<td>Application contexts keyed by id.</td>
</tr>
<tr>
<td>contexts.<em>.beans.</em></td>
<td>Object</td>
<td><code>@ConfigurationProperties</code> beans keyed by bean name.</td>
</tr>
<tr>
<td>contexts.<em>.beans.</em>.prefix</td>
<td>String</td>
<td>Prefix applied to the names of the bean's properties.</td>
</tr>
<tr>
<td>contexts.<em>.beans.</em>.properties</td>
<td>Object</td>
<td>Properties of the bean as name-value pairs.</td>
</tr>
<tr>
<td>contexts.*.parentId</td>
<td>String</td>
<td>Id of the parent application context, if any.</td>
</tr>
</tbody>
</table>
Chapter 7. Environment (env)

The `env` endpoint provides information about the application's Environment.

7.1. Retrieving the Entire Environment

To retrieve the entire environment, make a **GET** request to `/actuator/env`, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/env' -i -X GET
```

The resulting response is similar to the following:
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 795

{
    "activeProfiles" : [ ],
    "propertySources" : [ {
        "name" : "systemProperties",
        "properties" : {
            "java.vendor" : { 
                "value" : "AdoptOpenJDK"
            },
            "java.vm.specification.vendor" : { 
                "value" : "Oracle Corporation"
            },
            "java.runtime.version" : { 
                "value" : "1.8.0_272-b10"
            }
        }
    }, {
        "name" : "systemEnvironment",
        "properties" : {
            "JAVA_HOME" : { 
                "value" : "/opt/openjdk",
                "origin" : "System Environment Property \"JAVA_HOME\"
            }
        }
    }, {
        "name" : "applicationConfig: [classpath:/application.properties]",
        "properties" : {
            "com.example.cache.max-size" : { 
                "value" : "1000",
                "origin" : "class path resource [application.properties]:1:29"
            }
        }
    }]
}

7.1.1. Response Structure

The response contains details of the application's Environment. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activeProfiles</td>
<td>Array</td>
<td>Names of the active profiles, if any.</td>
</tr>
<tr>
<td>propertySources</td>
<td>Array</td>
<td>Property sources in order of precedence.</td>
</tr>
<tr>
<td>propertySources.[].name</td>
<td>String</td>
<td>Name of the property source.</td>
</tr>
</tbody>
</table>
### 7.2. Retrieving a Single Property

To retrieve a single property, make a GET request to `/actuator/env/{property.name}`, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/env/com.example.cache.max-size' -i -X GET
```

The preceding example retrieves information about the property named `com.example.cache.max-size`. The resulting response is similar to the following:

```json
HTTP/1.1 200 OK
Content-Disposition: inline;filename=f.txt
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 445

{
  "property": {
    "source": "applicationConfig: [classpath:/application.properties]",
    "value": "1000"
  },
  "activeProfiles": [],
  "propertySources": [
    { "name": "systemProperties" },
    { "name": "systemEnvironment" },
    { "name": "applicationConfig: [classpath:/application.properties]",
      "property": {
        "value": "1000",
        "origin": "class path resource [application.properties]:1:29"
      }
    }
  ]
}
```

#### 7.2.1. Response Structure

The response contains details of the requested property. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertySources.[].properties</td>
<td>Object</td>
<td>Properties in the property source keyed by property name.</td>
</tr>
<tr>
<td>propertySources.[].properties.*.value</td>
<td>String</td>
<td>Value of the property.</td>
</tr>
<tr>
<td>propertySources.[].properties.*.origin</td>
<td>String</td>
<td>Origin of the property, if any.</td>
</tr>
<tr>
<td>Path</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>property</td>
<td>Object</td>
<td>Property from the environment, if found.</td>
</tr>
<tr>
<td>property.source</td>
<td>String</td>
<td>Name of the source of the property.</td>
</tr>
<tr>
<td>property.value</td>
<td>String</td>
<td>Value of the property.</td>
</tr>
<tr>
<td>activeProfiles</td>
<td>Array</td>
<td>Names of the active profiles, if any.</td>
</tr>
<tr>
<td>propertySources</td>
<td>Array</td>
<td>Property sources in order of precedence.</td>
</tr>
<tr>
<td>propertySources[].name</td>
<td>String</td>
<td>Name of the property source.</td>
</tr>
<tr>
<td>propertySources[].property</td>
<td>Object</td>
<td>Property in the property source, if any.</td>
</tr>
<tr>
<td>propertySources[].property.value</td>
<td>Varies</td>
<td>Value of the property.</td>
</tr>
<tr>
<td>propertySources[].property.origin</td>
<td>String</td>
<td>Origin of the property, if any.</td>
</tr>
</tbody>
</table>
Chapter 8. Flyway (flyway)

The flyway endpoint provides information about database migrations performed by Flyway.

8.1. Retrieving the Migrations

To retrieve the migrations, make a GET request to /actuator/flyway, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/flyway' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 515

{
   "contexts": {
      "application": {
         "flywayBeans": {
            "flyway": {
               "migrations": [ {
                  "type": "SQL",
                  "checksum": -156244537,
                  "version": "1",
                  "description": "init",
                  "script": "V1__init.sql",
                  "state": "SUCCESS",
                  "installedBy": "SA",
                  "installedOn": "2020-10-29T10:24:46.503Z",
                  "installedRank": 1,
                  "executionTime": 1
               }
               ]
            }
         }
      }
   }
}
```

8.1.1. Response Structure

The response contains details of the application’s Flyway migrations. The following table describes the structure of the response:
<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contexts</td>
<td>Object</td>
<td>Application contexts keyed by id</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations</td>
<td>Array</td>
<td>Migrations performed by the Flyway instance, keyed by Flyway bean name.</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].checksum</td>
<td>Number</td>
<td>Checksum of the migration, if any.</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].description</td>
<td>String</td>
<td>Description of the migration, if any.</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].executionTime</td>
<td>Number</td>
<td>Execution time in milliseconds of an applied migration.</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].installedBy</td>
<td>String</td>
<td>User that installed the applied migration, if any.</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].installedOn</td>
<td>String</td>
<td>Timestamp of when the applied migration was installed, if any.</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].installedRank</td>
<td>Number</td>
<td>Rank of the applied migration, if any. Later migrations have higher ranks.</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].script</td>
<td>String</td>
<td>Name of the script used to execute the migration, if any.</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].state</td>
<td>String</td>
<td>State of the migration. (PENDING, ABOVE_TARGET, BELOW_BASELINE, BASELINE, IGNORED, MISSING_SUCCESS, MISSING_FAILED, SUCCESS, UNDONE, AVAILABLE, FAILED, OUT_OF_ORDER, FUTURE_SUCCESS, FUTURE_FAILED, OUTDATED, SUPERSEDED)</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].type</td>
<td>String</td>
<td>Type of the migration. (SCHEMA, BASELINE, SQL, UNDO_SQL, JDBC, UNDO_JDBC, SPRING_JDBC, UNDO_SPRING_JDBC, CUSTOM, UNDO_CUSTOM)</td>
</tr>
<tr>
<td>contexts.<em>.flywayBeans.</em>.migrations[].version</td>
<td>String</td>
<td>Version of the database after applying the migration, if any.</td>
</tr>
<tr>
<td>contexts.*.parentId</td>
<td>String</td>
<td>Id of the parent application context, if any.</td>
</tr>
</tbody>
</table>
Chapter 9. Health (health)

The health endpoint provides detailed information about the health of the application.

9.1. Retrieving the Health of the application

To retrieve the health of the application, make a GET request to /actuator/health, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/health' -i -X GET
```

The resulting response is similar to the following:
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 670

{
  "status" : "UP",
  "details" : {
    "diskSpace" : {
      "status" : "UP",
      "details" : {
        "total" : 194687758336,
        "free" : 66235674624,
        "threshold" : 10485760
      }
    }
  },
  "db" : {
    "status" : "UP",
    "details" : {
      "database" : "HSQL Database Engine",
      "hello" : 1
    }
  },
  "broker" : {
    "status" : "UP",
    "details" : {
      "us1" : {
        "status" : "UP",
        "details" : {
          "version" : "1.0.2"
        }
      },
      "us2" : {
        "status" : "UP",
        "details" : {
          "version" : "1.0.4"
        }
      }
    }
  }
}

9.1.1. Response Structure

The response contains details of the health of the application. The following table describes the structure of the response:
9.2. Retrieving the Health of a component

To retrieve the health of a particular component of the application, make a `GET` request to `/actuator/health/{component}`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/health/db' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 99

{
    "status": "UP",
    "details": {
        "database": "HSQL Database Engine",
        "hello": 1
    }
}
```

9.2.1. Response Structure

The response contains details of the health of a particular component of the application. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>String</td>
<td>Overall status of the application.</td>
</tr>
<tr>
<td>details</td>
<td>Object</td>
<td>Details of the health of the application. Presence is controlled by <code>management.endpoint.health.show-details</code></td>
</tr>
<tr>
<td>details.*.status</td>
<td>String</td>
<td>Status of a specific part of the application.</td>
</tr>
<tr>
<td>details.*.details</td>
<td>Object</td>
<td>Details of the health of a specific part of the application.</td>
</tr>
</tbody>
</table>
### 9.3. Retrieving the Health of a component instance

If a particular component consists of multiple instances (as the broker indicator in the example above), the health of a particular instance of that component can be retrieved by issuing a GET request to `/actuator/health/{component}/{instance}`, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/health/broker/us1' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 66

{
    "status": "UP",
    "details": {
        "version": "1.0.2"
    }
}
```

### 9.3.1. Response Structure

The response contains details of the health of an instance of a particular component of the application. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>String</td>
<td>Status of a specific part of the application</td>
</tr>
<tr>
<td>details</td>
<td>Object</td>
<td>Details of the health of a specific part of the application</td>
</tr>
</tbody>
</table>
Chapter 10. Heap Dump \texttt{(heapdump)}

The \texttt{heapdump} endpoint provides a heap dump from the application's JVM.

10.1. Retrieving the Heap Dump

To retrieve the heap dump, make a \texttt{GET} request to \texttt{/actuator/heapdump}. The response is binary data in \texttt{HPROF} format and can be large. Typically, you should save the response to disk for subsequent analysis. When using curl, this can be achieved by using the \texttt{-O} option, as shown in the following example:

\begin{verbatim}
$ curl 'http://localhost:8080/actuator/heapdump' -O
\end{verbatim}

The preceding example results in a file named \texttt{heapdump} being written to the current working directory.
Chapter 11. HTTP Trace (httptrace)

The httptrace endpoint provides information about HTTP request-response exchanges.

11.1. Retrieving the Traces

To retrieve the traces, make a GET request to /actuator/httptrace, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/httptrace' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 503

{
    "traces" : [
        {
            "timestamp" : "2020-10-29T10:24:48.268Z",
            "principal" : {
                "name" : "alice"
            },
            "session" : {
                "id" : "ab8864b4-201d-4330-ad87-373f920900c6"
            },
            "request" : {
                "method" : "GET",
                "uri" : "https://api.example.com",
                "headers" : {
                    "Accept" : [ "application/json" ]
                }
            },
            "response" : {
                "status" : 200,
                "headers" : {
                    "Content-Type" : [ "application/json" ]
                }
            },
            "timeTaken" : 0
        }
    ]
}
```

11.1.1. Response Structure

The response contains details of the traced HTTP request-response exchanges. The following table describes the structure of the response:
<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>traces</td>
<td>Array</td>
<td>An array of traced HTTP request-response exchanges.</td>
</tr>
<tr>
<td>traces[].timestamp</td>
<td>String</td>
<td>Timestamp of when the traced exchange occurred.</td>
</tr>
<tr>
<td>traces[].principal</td>
<td>Object</td>
<td>Principal of the exchange, if any.</td>
</tr>
<tr>
<td>traces[].principal.name</td>
<td>String</td>
<td>Name of the principal.</td>
</tr>
<tr>
<td>traces[].request.method</td>
<td>String</td>
<td>HTTP method of the request.</td>
</tr>
<tr>
<td>traces[].request.remoteAddress</td>
<td>String</td>
<td>Remote address from which the request was received, if known.</td>
</tr>
<tr>
<td>traces[].request.uri</td>
<td>String</td>
<td>URI of the request.</td>
</tr>
<tr>
<td>traces[].request.headers</td>
<td>Object</td>
<td>Headers of the request, keyed by header name.</td>
</tr>
<tr>
<td>traces[].request.headers.*[]</td>
<td>Array</td>
<td>Values of the header</td>
</tr>
<tr>
<td>traces[].response.status</td>
<td>Number</td>
<td>Status of the response</td>
</tr>
<tr>
<td>traces[].response.headers</td>
<td>Object</td>
<td>Headers of the response, keyed by header name.</td>
</tr>
<tr>
<td>traces[].response.headers.*[]</td>
<td>Array</td>
<td>Values of the header</td>
</tr>
<tr>
<td>traces[].session</td>
<td>Object</td>
<td>Session associated with the exchange, if any.</td>
</tr>
<tr>
<td>traces[].session.id</td>
<td>String</td>
<td>ID of the session.</td>
</tr>
<tr>
<td>traces[].timeTaken</td>
<td>Number</td>
<td>Time, in milliseconds, taken to handle the exchange.</td>
</tr>
</tbody>
</table>
Chapter 12. Info (info)

The info endpoint provides general information about the application.

12.1. Retrieving the Info

To retrieve the information about the application, make a GET request to /actuator/info, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/info' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 235

{
  "git": {
    "commit": {
      "time": "+52797-09-10T21:27:38Z",
      "id": "df027cf"
    },
    "branch": "master"
  },
  "build": {
    "version": "1.0.3",
    "artifact": "application",
    "group": "com.example"
  }
}
```

12.1.1. Response Structure

The response contains general information about the application. Each section of the response is contributed by an InfoContributor. Spring Boot provides build and git contributions.

**build Response Structure**

The following table describe the structure of the build section of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifact</td>
<td>String</td>
<td>Artifact ID of the application, if any.</td>
</tr>
<tr>
<td>group</td>
<td>String</td>
<td>Group ID of the application, if any.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the application, if any.</td>
</tr>
</tbody>
</table>
**Path** | **Type** | **Description**
---|---|---
version | String | Version of the application, if any.
time | Varies | Timestamp of when the application was built, if any.

### git Response Structure

The following table describes the structure of the `git` section of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch</td>
<td>String</td>
<td>Name of the Git branch, if any.</td>
</tr>
<tr>
<td>commit</td>
<td>Object</td>
<td>Details of the Git commit, if any.</td>
</tr>
<tr>
<td>commit.time</td>
<td>Varies</td>
<td>Timestamp of the commit, if any.</td>
</tr>
<tr>
<td>commit.id</td>
<td>String</td>
<td>ID of the commit, if any.</td>
</tr>
</tbody>
</table>
Chapter 13. Spring Integration graph
(integrationgraph)

The integrationgraph endpoint exposes a graph containing all Spring Integration components.

13.1. Retrieving the Spring Integration graph

To retrieve the information about the application, make a GET request to /actuator/integrationgraph, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/integrationgraph' -i -X GET
```

The resulting response is similar to the following:
13.1.1. Response Structure

The response contains all Spring Integration components used within the application, as well as the links between them. More information about the structure can be found in the reference documentation.

13.2. Rebuilding the Spring Integration graph

To rebuild the exposed graph, make a POST request to /actuator/integrationgraph, as shown in the following curl-based example:

$ curl 'http://localhost:8080/actuator/integrationgraph' -i -X POST
This will result in a **204 - No Content** response:

```
HTTP/1.1 204 No Content
```
Chapter 14. Liquibase (liquibase)

The liquibase endpoint provides information about database change sets applied by Liquibase.

14.1. Retrieving the Changes

To retrieve the changes, make a GET request to /actuator/liquibase, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/liquibase' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 688

{
   "contexts": {
      "application": {
         "liquibaseBeans": {
            "liquibase": {
               "changeSets": [{
                  "author": "marceloverdijk",
                  "changeLog": "classpath:/db/changelog/db.changelog-master.yaml",
                  "comments": "",
                  "contexts": [],
                  "dateExecuted": "2020-10-29T10:24:14.939Z",
                  "deploymentId": "3967054928",
                  "description": "createTable tableName=customer",
                  "execType": "EXECUTED",
                  "id": "1",
                  "labels": [],
                  "checksum": "8:46deb252c5e6d7b25e28ddeb9fc4bf6",
                  "orderExecuted": 1
               }
            }
         }
      }
   }
}
```

14.1.1. Response Structure

The response contains details of the application’s Liquibase change sets. The following table describes the structure of the response:
<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contexts</td>
<td>Object</td>
<td>Application contexts keyed by id</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets</td>
<td>Array</td>
<td>Change sets made by the Liquibase beans, keyed by bean name.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].author</td>
<td>String</td>
<td>Author of the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].changeLog</td>
<td>String</td>
<td>Change log that contains the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].comments</td>
<td>String</td>
<td>Comments on the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].contexts</td>
<td>Array</td>
<td>Contexts of the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].dateExecuted</td>
<td>String</td>
<td>Timestamp of when the change set was executed.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].deploymentId</td>
<td>String</td>
<td>ID of the deployment that ran the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].description</td>
<td>String</td>
<td>Description of the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].execType</td>
<td>String</td>
<td>Execution type of the change set (EXECUTED, FAILED, SKIPPED, RERAN, MARK_RAN).</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].id</td>
<td>String</td>
<td>ID of the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].labels</td>
<td>Array</td>
<td>Labels associated with the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].checksum</td>
<td>String</td>
<td>Checksum of the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].orderExecuted</td>
<td>Number</td>
<td>Order of the execution of the change set.</td>
</tr>
<tr>
<td>contexts.<em>.liquibaseBeans.</em>.changeSets[].tag</td>
<td>String</td>
<td>Tag associated with the change set, if any.</td>
</tr>
<tr>
<td>contexts.*.parentId</td>
<td>String</td>
<td>Id of the parent application context, if any.</td>
</tr>
</tbody>
</table>
Chapter 15. Log File (logfile)

The `logfile` endpoint provides access to the contents of the application’s log file.

15.1. Retrieving the Log File

To retrieve the log file, make a GET request to `/actuator/logfile`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/logfile' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Type: text/plain;charset=UTF-8
Content-Length: 4723

```

```
2017-08-08 17:12:30.910  INFO 19866 --- [           main]  s.f.SampleWebFreeMarkerApplication       : Starting SampleWebFreeMarkerApplication on
host.local with PID 19866
2017-08-08 17:12:30.913  INFO 19866 --- [           main]  s.f.SampleWebFreeMarkerApplication       : No active profile set, falling back to
default profiles: default
2017-08-08 17:12:30.952  INFO 19866 --- [           main]  ConfigServletWebServerApplicationContext : Refreshing
org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplication:
2017-08-08 17:12:31.878  INFO 19866 --- [           main]  o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080
(http)
Tomcat/8.5.16
WebApplicationContext
```

```
15.2. Retrieving Part of the Log File

Retrieving part of the log file is not supported when using Jersey.

To retrieve part of the log file, make a GET request to /actuator/logfile by using the Range header, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/logfile' -i -X GET \
   -H 'Range: bytes=0-1023'
```

The preceding example retrieves the first 1024 bytes of the log file. The resulting response is similar to the following:

```
HTTP/1.1 206 Partial Content
Accept-Ranges: bytes
Content-Type: text/plain;charset=UTF-8
Content-Range: bytes 0-1023/4723
Content-Length: 1024
```
Chapter 16. Loggers (loggers)

The loggers endpoint provides access to the application’s loggers and the configuration of their levels.

16.1. Retrieving All Loggers

To retrieve the application’s loggers, make a GET request to /actuator/loggers, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/loggers' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 283

{
    "levels" : [ "OFF", "FATAL", "ERROR", "WARN", "INFO", "DEBUG", "TRACE" ],
    "loggers" : {
        "ROOT" : {
            "configuredLevel" : "INFO",
            "effectiveLevel" : "INFO"
        },
        "com.example" : {
            "configuredLevel" : "DEBUG",
            "effectiveLevel" : "DEBUG"
        }
    }
}
```

16.1.1. Response Structure

The response contains details of the application’s loggers. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>levels</td>
<td>Array</td>
<td>Levels support by the logging system.</td>
</tr>
<tr>
<td>loggers</td>
<td>Object</td>
<td>Loggers keyed by name.</td>
</tr>
<tr>
<td>loggers.*.configuredLevel</td>
<td>String</td>
<td>Configured level of the logger, if any.</td>
</tr>
<tr>
<td>loggers.*.effectiveLevel</td>
<td>String</td>
<td>Effective level of the logger.</td>
</tr>
</tbody>
</table>
16.2. Retrieving a Single Logger

To retrieve a single logger, make a GET request to `/actuator/loggers/{logger.name}`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/loggers/com.example' -i -X GET
```

The preceding example retrieves information about the logger named `com.example`. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Disposition: inline;filename=f.txt
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 61

{
  "configuredLevel" : "INFO",
  "effectiveLevel" : "INFO"
}
```

16.2.1. Response Structure

The response contains details of the requested logger. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuredLevel</td>
<td>String</td>
<td>Configured level of the logger, if any.</td>
</tr>
<tr>
<td>effectiveLevel</td>
<td>String</td>
<td>Effective level of the logger.</td>
</tr>
</tbody>
</table>

16.3. Setting a Log Level

To set the level of a logger, make a POST request to `/actuator/loggers/{logger.name}` with a JSON body that specifies the configured level for the logger, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/loggers/com.example' -i -X POST \
-H 'Content-Type: application/json' \
-d '{"configuredLevel":"debug"}''
```

The preceding example sets the configuredLevel of the `com.example` logger to `DEBUG`.

16.3.1. Request Structure

The request specifies the desired level of the logger. The following table describes the structure of
the request:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuredLevel</td>
<td>String</td>
<td>Level for the logger. May be omitted to clear the level.</td>
</tr>
</tbody>
</table>

## 16.4. Clearing a Log Level

To clear the level of a logger, make a **POST** request to `/actuator/loggers/{logger.name}` with a JSON body containing an empty object, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/loggers/com.example' -i -X POST \
  -H 'Content-Type: application/json' \
  -d '{}'
```

The preceding example clears the configured level of the `com.example` logger.
Chapter 17. Mappings (mappings)

The mappings endpoint provides information about the application's request mappings.

17.1. Retrieving the Mappings

To retrieve the mappings, make a GET request to /actuator/mappings, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:40297/actuator/mappings' -i -X GET
```

The resulting response is similar to the following:

```json
{
    "contexts": {
        "application": {
            "mappings": {
                "dispatcherServlets": {
                    "dispatcherServlet": [
                        {
                            "handler": "ResourceHttpRequestHandler [class path resource [META-INF/resources/], class path resource [resources/], class path resource [static/], class path resource [public/], ServletContext resource [/], class path resource [ ]]",
                            "predicate": "/**/favicon.ico"
                        },
                        {
                            "handler": "Actuator web endpoint 'mappings'",
                            "predicate": "@GET /actuator/mappings, produces [application/vnd.spring-boot.actuator.v2+json || application/json]",
                            "details": {
                                "handlerMethod": {
                                    "name": "handle",
                                    "descriptor": "(Ljavax/servlet/http/HttpServletRequest;Ljava/util/Map;)Ljava/lang/Object;"
                                },
                                "requestMappingConditions": {
                                    "consumes": [ ],
                                    "headers": [ ],
                                    "methods": [ "GET" ],
                                    "params": [ ],
                                    "patterns": [ "/actuator/mappings" ]
                                }
                            }
                        }
                    ]
                }
            }
        }
    }
}
```
"handler": "Actuator root web endpoint",
"predicate": "{GET /actuator, produces [application/vnd.spring-boot.actuator.v2+json || application/json]}",
"details": {
  "handlerMethod": {
    "className": "org.springframework.boot.actuate.endpoint.web.servlet.WebMvcEndpointHandlerMapping.WebMvcLinksHandler",
    "name": "links",
    "descriptor": "(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;){Ljavax/lang/Object;}",
    "requestMappingConditions": {
      "consumes": [ ],
      "headers": [ ],
      "methods": [ "GET" ],
      "params": [ ],
      "patterns": [ "/actuator" ],
      "produces": [ {
        "mediaType": "application/vnd.spring-boot.actuator.v2+json",
        "negated": false
      }, {
        "mediaType": "application/json",
        "negated": false
      } ]
    }
  }
},
"handler": "public java.lang.String
org.springframework.boot.actuate.autoconfigure.endpoint.web.documentation.MappingsEndPointServletDocumentationTests$ExampleController.example()",
"predicate": "{POST /, params [a!=alpha], headers [X-Custom=Foo], consumes [application/json || !application/xml], produces [text/plain]}",
"details": {
  "handlerMethod": {
    "className": "org.springframework.boot.actuate.autoconfigure.endpoint.web.documentation.MappingsEndPointServletDocumentationTests.ExampleController",
    "name": "example",
    "descriptor": "()Ljava/lang/String;"}
"requestMappingConditions": {
  "consumes": [
    {
      "mediaType": "application/json",
      "negated": false
    },
    {
      "mediaType": "application/xml",
      "negated": true
    }
  ],
  "headers": [
    {
      "name": "X-Custom",
      "value": "Foo",
      "negated": false
    }
  ],
  "methods": [ "POST" ],
  "params": [
    {
      "name": "a",
      "value": "alpha",
      "negated": true
    }
  ],
  "patterns": [ "/" ],
  "produces": [
    {
      "mediaType": "text/plain",
      "negated": false
    }
  ]
},

"handler": "ResourceHttpRequestHandler ["classpath:/META-INF/resources/webjars/"]",
"predicate": "/webjars/**",

"servletFilters": [
  {
    "servletNameMappings": [],
    "urlPatternMappings": [ "/*" ],
    "name": "requestContextFilter",
    "className": "org.springframework.boot.web.servlet.filter.OrderedRequestContextFilter"
  },
  {
    "servletNameMappings": [],
    "urlPatternMappings": [ "/*" ],
    "name": "hiddenHttpMethodFilter",
    "className": "org.springframework.boot.web.servlet.filter.OrderedHiddenHttpMethodFilter"
  },
  {
    "servletNameMappings": [],
    "urlPatternMappings": [ "/*" ],
    "name": "handler".
"predicate": "/webjars/**"
}]
}
17.1.1. Response Structure

The response contains details of the application’s mappings. The items found in the response depend on the type of web application (reactive or Servlet-based). The following table describes the structure of the common elements of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contexts</td>
<td>Object</td>
<td>Application contexts keyed by id.</td>
</tr>
<tr>
<td>contexts.*.mappings</td>
<td>Object</td>
<td>Mappings in the context, keyed by mapping type.</td>
</tr>
<tr>
<td>contexts.*.mappings.dispatcherServlets</td>
<td>Object</td>
<td>Dispatcher servlet mappings, if any.</td>
</tr>
<tr>
<td>contexts.*.mappings.servletFilters</td>
<td>Array</td>
<td>Servlet filter mappings, if any.</td>
</tr>
<tr>
<td>contexts.*.mappings.servlets</td>
<td>Array</td>
<td>Servlet mappings, if any.</td>
</tr>
<tr>
<td>contexts.*.mappings.dispatcherHandlers</td>
<td>Object</td>
<td>Dispatcher handler mappings, if any.</td>
</tr>
<tr>
<td>contexts.*.parentId</td>
<td>String</td>
<td>Id of the parent application context, if any.</td>
</tr>
</tbody>
</table>

The entries that may be found in contexts.*.mappings are described in the following sections.

17.1.2. Dispatcher Servlets Response Structure

When using Spring MVC, the response contains details of any DispatcherServlet request mappings beneath contexts.*.mappings.dispatcherServlets. The following table describes the structure of this section of the response:
<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Array</td>
<td>Dispatcher servlet mappings, if any, keyed by dispatcher servlet bean name.</td>
</tr>
<tr>
<td>*.[].details</td>
<td>Object</td>
<td>Additional implementation-specific details about the mapping. Optional.</td>
</tr>
<tr>
<td>*.[].handler</td>
<td>String</td>
<td>Handler for the mapping.</td>
</tr>
<tr>
<td>*.[].predicate</td>
<td>String</td>
<td>Predicate for the mapping.</td>
</tr>
<tr>
<td>*.[].details.handlerMethod</td>
<td>Object</td>
<td>Details of the method, if any, that will handle requests to this mapping.</td>
</tr>
<tr>
<td>*.[].details.handlerMethod.className</td>
<td>Varies</td>
<td>Fully qualified name of the class of the method.</td>
</tr>
<tr>
<td>*.[].details.handlerMethod.name</td>
<td>Varies</td>
<td>Name of the method.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions</td>
<td>Object</td>
<td>Details of the request mapping conditions.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.consumes</td>
<td>Varies</td>
<td>Details of the consumes condition</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.consumes.[].mediaType</td>
<td>Varies</td>
<td>Consumed media type.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.consumes.[].negated</td>
<td>Varies</td>
<td>Whether the media type is negated.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.headers</td>
<td>Varies</td>
<td>Details of the headers condition.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.headers.[].name</td>
<td>Varies</td>
<td>Name of the header.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.headers.[].value</td>
<td>Varies</td>
<td>Required value of the header, if any.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.headers.[].negated</td>
<td>Varies</td>
<td>Whether the value is negated.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.methods</td>
<td>Varies</td>
<td>HTTP methods that are handled.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.params</td>
<td>Varies</td>
<td>Details of the params condition.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.params.[].name</td>
<td>Varies</td>
<td>Name of the parameter.</td>
</tr>
</tbody>
</table>
### 17.1.3. Servlets Response Structure

When using the Servlet stack, the response contains details of any Servlet mappings beneath contexts.*.mappings.servlets. The following table describes the structure of this section of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[].mappings</td>
<td>Array</td>
<td>Mappings of the servlet.</td>
</tr>
<tr>
<td>[].name</td>
<td>String</td>
<td>Name of the servlet.</td>
</tr>
<tr>
<td>[].className</td>
<td>String</td>
<td>Class name of the servlet</td>
</tr>
</tbody>
</table>

### 17.1.4. Servlet Filters Response Structure

When using the Servlet stack, the response contains details of any Filter mappings beneath contexts.*.mappings.servletFilters. The following table describes the structure of this section of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[].servletNameMappings</td>
<td>Array</td>
<td>Names of the servlets to which the filter is mapped.</td>
</tr>
<tr>
<td>[].urlPatternMappings</td>
<td>Array</td>
<td>URL pattern to which the filter is mapped.</td>
</tr>
<tr>
<td>[].name</td>
<td>String</td>
<td>Name of the filter.</td>
</tr>
<tr>
<td>[].className</td>
<td>String</td>
<td>Class name of the filter</td>
</tr>
</tbody>
</table>

### 17.1.5. Dispatcher Handlers Response Structure

When using Spring WebFlux, the response contains details of any DispatcherHandler request
mappings beneath `contexts.*.mappings.dispatcherHandlers`. The following table describes the structure of this section of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Array</td>
<td>Dispatcher handler mappings, if any, keyed by dispatcher handler bean name.</td>
</tr>
<tr>
<td>*[].details</td>
<td>Object</td>
<td>Additional implementation-specific details about the mapping. Optional.</td>
</tr>
<tr>
<td>*[].handler</td>
<td>String</td>
<td>Handler for the mapping.</td>
</tr>
<tr>
<td>*[].predicate</td>
<td>String</td>
<td>Predicate for the mapping.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions</td>
<td>Object</td>
<td>Details of the request mapping conditions.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.consumes</td>
<td>Array</td>
<td>Details of the consumes condition</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.consumes[].mediaType</td>
<td>String</td>
<td>Consumed media type.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.consumes[].negated</td>
<td>Boolean</td>
<td>Whether the media type is negated.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.headers</td>
<td>Array</td>
<td>Details of the headers condition.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.headers[].name</td>
<td>String</td>
<td>Name of the header.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.headers[].value</td>
<td>String</td>
<td>Required value of the header, if any.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.headers[].negated</td>
<td>Boolean</td>
<td>Whether the value is negated.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.methods</td>
<td>Array</td>
<td>HTTP methods that are handled.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.params</td>
<td>Array</td>
<td>Details of the params condition.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.params[].name</td>
<td>String</td>
<td>Name of the parameter.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.params[].value</td>
<td>String</td>
<td>Required value of the parameter, if any.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.params[].negated</td>
<td>Boolean</td>
<td>Whether the value is negated.</td>
</tr>
<tr>
<td>*[].details.requestMappingConditions.patterns</td>
<td>Array</td>
<td>Patterns identifying the paths handled by the mapping.</td>
</tr>
<tr>
<td>Path</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.produces</td>
<td>Array</td>
<td>Details of the produces condition.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.produces[].mediaType</td>
<td>String</td>
<td>Produced media type.</td>
</tr>
<tr>
<td>*.[].details.requestMappingConditions.produces[].negated</td>
<td>Boolean</td>
<td>Whether the media type is negated.</td>
</tr>
<tr>
<td>*.[].details.handlerMethod</td>
<td>Object</td>
<td>Details of the method, if any, that will handle requests to this mapping.</td>
</tr>
<tr>
<td>*.[].details.handlerMethod.className</td>
<td>String</td>
<td>Fully qualified name of the class of the method.</td>
</tr>
<tr>
<td>*.[].details.handlerMethod.name</td>
<td>String</td>
<td>Name of the method.</td>
</tr>
<tr>
<td>*.[].details.handlerFunction</td>
<td>Object</td>
<td>Details of the function, if any, that will handle requests to this mapping.</td>
</tr>
<tr>
<td>*.[].details.handlerFunction.className</td>
<td>String</td>
<td>Fully qualified name of the class of the function.</td>
</tr>
</tbody>
</table>
Chapter 18. Metrics (metrics)

The metrics endpoint provides access to application metrics.

18.1. Retrieving Metric Names

To retrieve the names of the available metrics, make a GET request to /actuator/metrics, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/metrics' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 154

{
  "names" : [ "jvm.memory.max", "jvm.memory.used", "jvm.memory.committed", "jvm.buffer.memory.used", "jvm.buffer.count", "jvm.buffer.total.capacity" ]
}
```

18.1.1. Response Structure

The response contains details of the metric names. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>names</td>
<td>Array</td>
<td>Names of the known metrics.</td>
</tr>
</tbody>
</table>

18.2. Retrieving a Metric

To retrieve a metric, make a GET request to /actuator/metrics/{metric.name}, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/metrics/jvm.memory.max' -i -X GET
```

The preceding example retrieves information about the metric named jvm.memory.max. The resulting response is similar to the following:
18.2.1. Query Parameters

The endpoint uses query parameters to drill down into a metric by using its tags. The following table shows the single supported query parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>A tag to use for drill-down in the form name:value.</td>
</tr>
</tbody>
</table>

18.2.2. Response structure

The response contains details of the metric. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the metric</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>Description of the metric</td>
</tr>
<tr>
<td>baseUnit</td>
<td>String</td>
<td>Base unit of the metric</td>
</tr>
<tr>
<td>measurements</td>
<td>Array</td>
<td>Measurements of the metric</td>
</tr>
<tr>
<td>Path</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>measurements[].statistic</td>
<td>String</td>
<td>Statistic of the measurement. (TOTAL, TOTAL_TIME, COUNT, MAX, VALUE, UNKNOWN, ACTIVE_TASKS, DURATION).</td>
</tr>
<tr>
<td>measurements[].value</td>
<td>Number</td>
<td>Value of the measurement.</td>
</tr>
<tr>
<td>availableTags</td>
<td>Array</td>
<td>Tags that are available for drill-down.</td>
</tr>
<tr>
<td>availableTags[].tag</td>
<td>String</td>
<td>Name of the tag.</td>
</tr>
<tr>
<td>availableTags[].values</td>
<td>Array</td>
<td>Possible values of the tag.</td>
</tr>
</tbody>
</table>

### 18.3. Drilling Down

To drill down into a metric, make a GET request to `/actuator/metrics/{metric.name}` using the tag query parameter, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/metrics/jvm.memory.max?tag=area%3Anonheap&tag=id%3AClass+Space' -i -X GET
```

The preceding example retrieves the `jvm.memory.max` metric, where the `area` tag has a value of `nonheap` and the `id` attribute has a value of `Compressed Class Space`. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Disposition: inline;filename=f.txt
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 263

{
   "name" : "jvm.memory.max",
   "description" : "The maximum amount of memory in bytes that can be used for memory management",
   "baseUnit" : "bytes",
   "measurements" : [ {
       "statistic" : "VALUE",
       "value" : 1.073741824E9
   } ],
   "availableTags" : [ ]
}
```
Chapter 19. Prometheus (prometheus)

The prometheus endpoint provides Spring Boot application’s metrics in the format required for scraping by a Prometheus server.

19.1. Retrieving the Metrics

To retrieve the metrics, make a GET request to /actuator/prometheus, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/prometheus' -i -X GET
```

The resulting response is similar to the following:
HTTP/1.1 200 OK
Content-Type: text/plain;version=0.0.4;charset=utf-8
Content-Length: 2371

# HELP jvm_buffer_memory_used_bytes An estimate of the memory that the Java virtual machine is using for this buffer pool
# TYPE jvm_buffer_memory_used_bytes gauge
jvm_buffer_memory_used_bytes{id="direct"}, 442501.0
jvm_buffer_memory_used_bytes{id="mapped"}, 0.0

# HELP jvm_buffer_total_capacity_bytes An estimate of the total capacity of the buffers in this pool
# TYPE jvm_buffer_total_capacity_bytes gauge
jvm_buffer_total_capacity_bytes{id="direct"}, 442500.0
jvm_buffer_total_capacity_bytes{id="mapped"}, 0.0

# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="heap",id="PS Survivor Space"}, 2.8311552E7
jvm_memory_committed_bytes{area="heap",id="PS Old Gen"}, 3.145728E8
jvm_memory_committed_bytes{area="heap",id="PS Eden Space"}, 3.00941312E8
jvm_memory_committed_bytes{area="nonheap",id="Metaspace"}, 1.62095104E8
jvm_memory_committed_bytes{area="nonheap",id="Code Cache"}, 4.5285376E7
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space"}, 2.3199744E7

# HELP jvm_buffer_count_buffers An estimate of the number of buffers in the pool
# TYPE jvm_buffer_count_buffers gauge
jvm_buffer_count_buffers{id="direct"}, 15.0
jvm_buffer_count_buffers{id="mapped"}, 0.0

# HELP jvm_memory_used_bytes The amount of used memory
# TYPE jvm_memory_used_bytes gauge
jvm_memory_used_bytes{area="heap",id="PS Survivor Space"}, 1.92714E7
jvm_memory_used_bytes{area="heap",id="PS Old Gen"}, 1.64152912E8
jvm_memory_used_bytes{area="heap",id="PS Eden Space"}, 1.90245968E8
jvm_memory_used_bytes{area="nonheap",id="Metaspace"}, 1.5255588E8
jvm_memory_used_bytes{area="nonheap",id="Code Cache"}, 4.4475456E7
jvm_memory_used_bytes{area="nonheap",id="Compressed Class Space"}, 2.1309824E7

# HELP jvm_memory_max_bytes The maximum amount of memory in bytes that can be used for memory management
# TYPE jvm_memory_max_bytes gauge
jvm_memory_max_bytes{area="heap",id="PS Survivor Space"}, 2.8311552E7
jvm_memory_max_bytes{area="heap",id="PS Old Gen"}, 7.16177408E8
jvm_memory_max_bytes{area="heap",id="PS Eden Space"}, 3.00941312E8
jvm_memory_max_bytes{area="nonheap",id="Metaspace"}, -1.0
jvm_memory_max_bytes{area="nonheap",id="Code Cache"}, 2.5165824E8
jvm_memory_max_bytes{area="nonheap",id="Compressed Class Space"}, 1.073741824E9
Chapter 20. Scheduled Tasks (scheduledtasks)

The scheduledtasks endpoint provides information about the application’s scheduled tasks.

20.1. Retrieving the Scheduled Tasks

To retrieve the scheduled tasks, make a GET request to /actuator/scheduledtasks, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/scheduledtasks' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 629

{
  "cron" : [ {
    "runnable" : {
      "target" : "com.example.Processor.processOrders"
    },
    "expression" : "0 0 0/3 1/1 * ?"
  } ],
  "fixedDelay" : [ {
    "runnable" : {
      "target" : "com.example.Processor.purge"
    },
    "initialDelay" : 5000,
    "interval" : 5000
  } ],
  "fixedRate" : [ {
    "runnable" : {
      "target" : "com.example.Processor.retrieveIssues"
    },
    "initialDelay" : 10000,
    "interval" : 3000
  } ],
  "custom" : [ {
    "runnable" : {
      "target" : "com.example.Processor$CustomTriggeredRunnable"
    },
    "trigger" : "com.example.Processor$CustomTrigger@3b6ce423"
  } ]
}
```
## 20.1.1. Response Structure

The response contains details of the application’s scheduled tasks. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cron</td>
<td>Array</td>
<td>Cron tasks, if any.</td>
</tr>
<tr>
<td>cron[].runnable.target</td>
<td>String</td>
<td>Target that will be executed.</td>
</tr>
<tr>
<td>cron[].expression</td>
<td>String</td>
<td>Cron expression.</td>
</tr>
<tr>
<td>fixedDelay</td>
<td>Array</td>
<td>Fixed delay tasks, if any.</td>
</tr>
<tr>
<td>fixedDelay[].runnable.target</td>
<td>String</td>
<td>Target that will be executed.</td>
</tr>
<tr>
<td>fixedDelay[].initialDelay</td>
<td>Number</td>
<td>Delay, in milliseconds, before first execution.</td>
</tr>
<tr>
<td>fixedDelay[].interval</td>
<td>Number</td>
<td>Interval, in milliseconds, between the end of the last execution and the start of the next.</td>
</tr>
<tr>
<td>fixedRate</td>
<td>Array</td>
<td>Fixed rate tasks, if any.</td>
</tr>
<tr>
<td>fixedRate[].runnable.target</td>
<td>String</td>
<td>Target that will be executed.</td>
</tr>
<tr>
<td>fixedRate[].interval</td>
<td>Number</td>
<td>Interval, in milliseconds, between the start of each execution.</td>
</tr>
<tr>
<td>fixedRate[].initialDelay</td>
<td>Number</td>
<td>Delay, in milliseconds, before first execution.</td>
</tr>
<tr>
<td>custom</td>
<td>Array</td>
<td>Tasks with custom triggers, if any.</td>
</tr>
<tr>
<td>custom[].runnable.target</td>
<td>String</td>
<td>Target that will be executed.</td>
</tr>
<tr>
<td>custom[].trigger</td>
<td>String</td>
<td>Trigger for the task.</td>
</tr>
</tbody>
</table>
Chapter 21. Sessions (sessions)

The sessions endpoint provides information about the application’s HTTP sessions that are managed by Spring Session.

21.1. Retrieving Sessions

To retrieve the sessions, make a GET request to /actuator/sessions, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/sessions?username=alice' -i -X GET
```

The preceding examples retrieves all of the sessions for the user whose username is alice. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 753

{
  "sessions": [
    {
      "id": "31f62eb1-72c0-4c48-9655-3d66433d2fe0",
      "attributeNames": [],
      "creationTime": "2020-10-28T22:24:54.333Z",
      "lastAccessedTime": "2020-10-29T10:24:09.333Z",
      "maxInactiveInterval": 1800,
      "expired": false
    },
    {
      "id": "fbf12ae1-8b9d-43f8-b90b-6c44d0e77779",
      "attributeNames": [],
      "creationTime": "2020-10-29T08:24:54.334Z",
      "lastAccessedTime": "2020-10-29T10:24:42.334Z",
      "maxInactiveInterval": 1800,
      "expired": false
    },
    {
      "id": "4db5efcc-99cb-4d05-a52c-b49acfb7ea9",
      "attributeNames": [],
      "creationTime": "2020-10-29T05:24:54.334Z",
      "lastAccessedTime": "2020-10-29T10:24:17.334Z",
      "maxInactiveInterval": 1800,
      "expired": false
    }
  ]
}
```
21.1.1. Query Parameters

The endpoint uses query parameters to limit the sessions that it returns. The following table shows the single required query parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the user.</td>
</tr>
</tbody>
</table>

21.1.2. Response Structure

The response contains details of the matching sessions. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sessions</td>
<td>Array</td>
<td>Sessions for the given username.</td>
</tr>
<tr>
<td>sessions[].id</td>
<td>String</td>
<td>ID of the session.</td>
</tr>
<tr>
<td>sessions[].attributeNames</td>
<td>Array</td>
<td>Names of the attributes stored in the session.</td>
</tr>
<tr>
<td>sessions[].creationTime</td>
<td>String</td>
<td>Timestamp of when the session was created.</td>
</tr>
<tr>
<td>sessions[].lastAccessedTime</td>
<td>String</td>
<td>Timestamp of when the session was last accessed.</td>
</tr>
<tr>
<td>sessions[].maxInactiveInterval</td>
<td>Number</td>
<td>Maximum permitted period of inactivity, in seconds, before the session will expire.</td>
</tr>
<tr>
<td>sessions[].expired</td>
<td>Boolean</td>
<td>Whether the session has expired.</td>
</tr>
</tbody>
</table>

21.2. Retrieving a Single Session

To retrieve a single session, make a GET request to /actuator/sessions/{id}, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/sessions/4db5efcc-99cb-4d05-a52c-b49acfbb7ea9'
```

The preceding example retrieves the session with the id of 4db5efcc-99cb-4d05-a52c-b49acfbb7ea9. The resulting response is similar to the following:
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 228

{
    "id": "4db5efcc-99cb-4d05-a52c-b49acfb7ea9",
    "attributeNames": [],
    "creationTime": "2020-10-29T05:24:54.334Z",
    "lastAccessedTime": "2020-10-29T10:24:17.334Z",
    "maxInactiveInterval": 1800,
    "expired": false
}

21.2.1. Response Structure

The response contains details of the requested session. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>ID of the session.</td>
</tr>
<tr>
<td>attributeNames</td>
<td>Array</td>
<td>Names of the attributes stored in the session.</td>
</tr>
<tr>
<td>creationTime</td>
<td>String</td>
<td>Timestamp of when the session was created.</td>
</tr>
<tr>
<td>lastAccessedTime</td>
<td>String</td>
<td>Timestamp of when the session was last accessed.</td>
</tr>
<tr>
<td>maxInactiveInterval</td>
<td>Number</td>
<td>Maximum permitted period of inactivity, in seconds, before the session will expire.</td>
</tr>
<tr>
<td>expired</td>
<td>Boolean</td>
<td>Whether the session has expired.</td>
</tr>
</tbody>
</table>

21.3. Deleting a Session

To delete a session, make a DELETE request to /actuator/sessions/{id}, as shown in the following curl-based example:

    $ curl 'http://localhost:8080/actuator/sessions/4db5efcc-99cb-4d05-a52c-b49acfb7ea9'
          -i -X DELETE

The preceding example deletes the session with the id of 4db5efcc-99cb-4d05-a52c-b49acfb7ea9.
Chapter 22. Shutdown (shutdown)

The shutdown endpoint is used to shut down the application.

22.1. Shutting Down the Application

To shut down the application, make a POST request to /actuator/shutdown, as shown in the following curl-based example:

```bash
$ curl 'http://localhost:8080/actuator/shutdown' -i -X POST
```

A response similar to the following is produced:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 41

{
  "message" : "Shutting down, bye..."
}
```

22.1.1. Response Structure

The response contains details of the result of the shutdown request. The following table describes the structure of the response:

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>String</td>
<td>Message describing the result of the request.</td>
</tr>
</tbody>
</table>
Chapter 23. Thread Dump (threaddump)

The threaddump endpoint provides a thread dump from the application’s JVM.

23.1. Retrieving the Thread Dump

To retrieve the thread dump, make a GET request to /actuator/threaddump, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/threaddump' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 5529

{
  "threads": [
    {
      "threadName": "Thread-103",
      "threadId": 555,
      "blockedTime": -1,
      "blockedCount": 0,
      "waitedTime": -1,
      "waitedCount": 1,
      "lockName": "java.util.concurrent.CountDownLatch$Sync@6e51f4dd",
      "lockOwnerId": -1,
      "inNative": false,
      "suspended": false,
      "threadState": "WAITING",
      "stackTrace": [
        {
          "methodName": "park",
          "fileName": "Unsafe.java",
          "lineNumber": -2,
          "className": "sun.misc.Unsafe",
          "nativeMethod": true
        },
        {
          "methodName": "park",
          "fileName": "LockSupport.java",
          "lineNumber": 175,
          "className": "java.util.concurrent.locks.LockSupport",
          "nativeMethod": false
        },
        {
          "methodName": "parkAndCheckInterrupt",
          "fileName": "AbstractQueuedSynchronizer.java",
          "lineNumber": 836,
          "className": "java.util.concurrent.locks.AbstractQueuedSynchronizer",
          "nativeMethod": false
        }
      ]
    }
  ]
}
```

},
{
  "methodName" : "doAcquireSharedInterruptibly",
  "fileName" : "AbstractQueuedSynchronizer.java",
  "lineNumber" : 997,
  "className" : "java.util.concurrent.locks.AbstractQueuedSynchronizer",
  "nativeMethod" : false
},
{
  "methodName" : "acquireSharedInterruptibly",
  "fileName" : "AbstractQueuedSynchronizer.java",
  "lineNumber" : 1304,
  "className" : "java.util.concurrent.locks.AbstractQueuedSynchronizer",
  "nativeMethod" : false
},
{
  "methodName" : "await",
  "fileName" : "CountDownLatch.java",
  "lineNumber" : 231,
  "className" : "java.util.concurrent.CountDownLatch",
  "nativeMethod" : false
},
{
  "methodName" : "lambda\$threadDump\$0",
  "fileName" : "ThreadDumpEndpointDocumentationTests.java",
  "lineNumber" : 53,
  "className" :
  "org.springframework.boot.actuate.autoconfigure.endpoint.web.documentation.ThreadDumpEndpointDocumentationTests",
  "nativeMethod" : false
},
{
  "methodName" : "run",
  "lineNumber" : -1,
  "className" :
  "org.springframework.boot.actuate.autoconfigure.endpoint.web.documentation.ThreadDumpEndpointDocumentationTests\$\$Lambda\$2692/1993335720",
  "nativeMethod" : false
},
{
  "methodName" : "run",
  "fileName" : "Thread.java",
  "lineNumber" : 748,
  "className" :
  "java.lang.Thread",
  "nativeMethod" : false
}
],
"lockedMonitors" : [],
"lockedSynchronizers" : [
{
  "className" : "java.util.concurrent.locks.ReentrantLock\$NonfairSync",
  "identityHashCode" : 60753930
}
],
"lockInfo" : {
  "className" : "java.util.concurrent.CountDownLatch\$Sync",
  "identityHashCode" : 1850864861
}
},
{
  "threadName" : "Thread-101",
  "lockedMonitors" : [],
  "lockedSynchronizers" : [
    {  
      "className" : "java.util.concurrent.locks.ReentrantLock\$NonfairSync",
      "identityHashCode" : 60753930
    }
  ],
  "lockInfo" : {  
    "className" : "java.util.concurrent.CountDownLatch\$Sync",
    "identityHashCode" : 1850864861
  }
}
"threadId" : 553,
"blockedTime" : -1,
"blockedCount" : 0,
"waitedTime" : -1,
"waitedCount" : 1,
"lockOwnerId" : -1,
"inNative" : false,
"suspended" : false,
"threadState" : "TIMED_WAITING",
"stackTrace" : [ {
  "methodName" : "sleep",
  "fileName" : "Thread.java",
  "lineNumber" : -2,
  "className" : "java.lang.Thread",
  "nativeMethod" : true
}, {
  "methodName" : "performShutdown",
  "fileName" : "ShutdownEndpoint.java",
  "lineNumber" : 65,
  "className" : "org.springframework.boot.actuate.context.ShutdownEndpoint",
  "nativeMethod" : false
}, {
  "methodName" : "run",
  "lineNumber" : -1,
  "className" : "org.springframework.boot.actuate.context.ShutdownEndpoint$$Lambda$2691/1591301832",
  "nativeMethod" : false
}
],
"lockedMonitors" : [ ],
"lockedSynchronizers" : [ ]
},
{
  "threadName" : "pool-10-thread-1",
  "threadId" : 545,
  "blockedTime" : -1,
  "blockedCount" : 0,
  "waitedTime" : -1,
  "waitedCount" : 0,
  "lockOwnerId" : -1,
  "inNative" : false,
  "suspended" : false,
  "threadState" : "RUNNABLE",
  "stackTrace" : [ {
    "methodName" : "take",
    "fileName" : "ScheduledThreadPoolExecutor.java",
    "lineNumber" : 1085,
  }
  ]
}
23.1.1. Response Structure

The response contains details of the JVM's threads. The following table describes the structure of the response:
<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threads</td>
<td>Array</td>
<td>JVM’s threads.</td>
</tr>
<tr>
<td>threads[].blockedCount</td>
<td>Number</td>
<td>Total number of times that the thread has been blocked.</td>
</tr>
<tr>
<td>threads[].blockedTime</td>
<td>Number</td>
<td>Time in milliseconds that the thread has spent blocked. -1 if thread contention monitoring is disabled.</td>
</tr>
<tr>
<td>threads[].daemon</td>
<td>Boolean</td>
<td>Whether the thread is a daemon thread. Only available on Java 9 or later.</td>
</tr>
<tr>
<td>threads[].inNative</td>
<td>Boolean</td>
<td>Whether the thread is executing native code.</td>
</tr>
<tr>
<td>threads[].lockName</td>
<td>String</td>
<td>Description of the object on which the thread is blocked, if any.</td>
</tr>
<tr>
<td>threads[].lockInfo</td>
<td>Object</td>
<td>Object for which the thread is blocked waiting.</td>
</tr>
<tr>
<td>threads[].lockInfo.className</td>
<td>String</td>
<td>Fully qualified class name of the lock object.</td>
</tr>
<tr>
<td>threads[].lockInfo.identityHashCode</td>
<td>Number</td>
<td>Identity hash code of the lock object.</td>
</tr>
<tr>
<td>threads[].lockedMonitors</td>
<td>Array</td>
<td>Monitors locked by this thread, if any</td>
</tr>
<tr>
<td>threads[].lockedMonitors[].className</td>
<td>String</td>
<td>Class name of the lock object.</td>
</tr>
<tr>
<td>threads[].lockedMonitors[].identityHashCode</td>
<td>Number</td>
<td>Identity hash code of the lock object.</td>
</tr>
<tr>
<td>threads[].lockedMonitors[].lockedStackDepth</td>
<td>Number</td>
<td>Stack depth where the monitor was locked.</td>
</tr>
<tr>
<td>threads[].lockedMonitors[].lockedStackFrame</td>
<td>Object</td>
<td>Stack frame that locked the monitor.</td>
</tr>
<tr>
<td>threads[].lockedSynchronizers</td>
<td>Array</td>
<td>Synchronizers locked by this thread.</td>
</tr>
<tr>
<td>threads[].lockedSynchronizers[].className</td>
<td>String</td>
<td>Class name of the locked synchronizer.</td>
</tr>
<tr>
<td>threads[].lockedSynchronizers[].identityHashCode</td>
<td>Number</td>
<td>Identity hash code of the locked synchronizer.</td>
</tr>
<tr>
<td>threads[].lockOwnerId</td>
<td>Number</td>
<td>ID of the thread that owns the object on which the thread is blocked. -1 if the thread is not blocked.</td>
</tr>
<tr>
<td>Path</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>threads[].lockOwnerName</td>
<td>String</td>
<td>Name of the thread that owns the object on which the thread is blocked, if any.</td>
</tr>
<tr>
<td>threads[].priority</td>
<td>Number</td>
<td>Priority of the thread. Only available on Java 9 or later.</td>
</tr>
<tr>
<td>threads[].stackTrace</td>
<td>Array</td>
<td>Stack trace of the thread.</td>
</tr>
<tr>
<td>threads[].stackTrace[].classLoaderName</td>
<td>String</td>
<td>Name of the class loader of the class that contains the execution point identified by this entry, if any. Only available on Java 9 or later.</td>
</tr>
<tr>
<td>threads[].stackTrace[].className</td>
<td>String</td>
<td>Name of the class that contains the execution point identified by this entry.</td>
</tr>
<tr>
<td>threads[].stackTrace[].fileName</td>
<td>String</td>
<td>Name of the source file that contains the execution point identified by this entry, if any.</td>
</tr>
<tr>
<td>threads[].stackTrace[].lineNumber</td>
<td>Number</td>
<td>Line number of the execution point identified by this entry. Negative if unknown.</td>
</tr>
<tr>
<td>threads[].stackTrace[].methodName</td>
<td>String</td>
<td>Name of the method.</td>
</tr>
<tr>
<td>threads[].stackTrace[].moduleName</td>
<td>String</td>
<td>Name of the module that contains the execution point identified by this entry, if any. Only available on Java 9 or later.</td>
</tr>
<tr>
<td>threads[].stackTrace[].moduleVersion</td>
<td>String</td>
<td>Version of the module that contains the execution point identified by this entry, if any. Only available on Java 9 or later.</td>
</tr>
<tr>
<td>threads[].stackTrace[].nativeMethod</td>
<td>Boolean</td>
<td>Whether the execution point is a native method.</td>
</tr>
<tr>
<td>threads[].suspended</td>
<td>Boolean</td>
<td>Whether the thread is suspended.</td>
</tr>
<tr>
<td>threads[].threadId</td>
<td>Number</td>
<td>ID of the thread.</td>
</tr>
<tr>
<td>threads[].threadName</td>
<td>String</td>
<td>Name of the thread.</td>
</tr>
<tr>
<td>threads[].threadState</td>
<td>String</td>
<td>State of the thread (NEW, Runnable, BLOCKED, WAITING, TIMED_WAITING, TERMINATED).</td>
</tr>
<tr>
<td>Path</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>threads[].waitedCount</td>
<td>Number</td>
<td>Total number of times that the thread has waited for notification.</td>
</tr>
<tr>
<td>threads[].waitedTime</td>
<td>Number</td>
<td>Time in milliseconds that the thread has spent waiting. -1 if thread contention monitoring is disabled</td>
</tr>
</tbody>
</table>