

What's New in Spring Boot 2.5

JSUG 勉強会 2021/7/2
池谷 智行

自己紹介

- ・日本Springユーザ会(JSUG)スタッフ
- ・某SIerでソフトウェアアーキテクト
(実態は管理業務多め・・・)
- ・某書の筆者の一人

<https://www.shoeisha.co.jp/book/detail/9784798142470>



参考：過去のWhat's Newシリーズ

VMWare 槙さんの講演資料を参照



<https://docs.google.com/presentation/d/1rPM00F3ptJ3m9UHNrdTx2ZPrBoecb4Wca0GujfViBP0/edit#slide=id.p1>

https://docs.google.com/presentation/d/111ePUTXHwFqODW_8np1RRqsPtrF7rVnZhocfqOLgFk/edit#slide=id.g8294c00475_0_0

アジェンダ

- ライフサイクル等
- 新機能／改善
- 既存機能の変更や影響
- ドキュメント改善



<https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.5-Release-Notes>

ライフサイクル

- Spring Boot 2.2からは半年1回頻度でリリース（JDKと対応）
- Spring Boot 2.5のEOLは2023/2月

Version	Released	Expected End of Life	依存Spring Framework	依存Spring Cloud	対応Javaバージョン
2.5.x	May 2021	February 2023	5.3	2020.0.3	16~8
2.4.x	November 2020	August 2022	5.3	2020.0.3 2020.0.0	15~8
2.3.x	May 2020	February 2022	5.2	Hoxton.SR11	14~8
2.2.x	October 2019	July 2021	5.2	Hoxton.SR11	13~8

<https://github.com/spring-projects/spring-boot/wiki/Supported-Versions#released-versions>

<https://start.spring.io/actuator/info>

参考：Spring BootのOSSサポートポリシ

OSSとしての致命的バグや脆弱性へのサポートのポリシ
(重要な話なので、正式には公式URLで確認を)

- ・ メジャーバージョンはリリースから**最低3年**はサポートされる
(ただしサポート対象のマイナーバージョンである必要)
- ・ マイナーバージョンは**最低12ヶ月**はサポートされる
- ・ EOLはリリースより**21ヶ月**

<https://github.com/spring-projects/spring-boot/wiki/Supported-Versions#spring-boot-support-policy>

アジェンダ

- ・ ライフサイクル等
- ・ 新機能／改善
- ・ 既存機能の変更や影響
- ・ ドキュメント改善



<https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.5-Release-Notes>

Spring Boot 2.5新機能／改善

- ・ 環境変数のプレフィックス
- ・ Spring Data Repositoryのメトリクス公開対応
- ・ DataSource Beanの生成順序の改善
- ・ R2DBCでのデータソース初期化
- ・ その他

環境変数のプレフィックス

これまで、同一環境（同一コンテナ等）で複数Spring Bootを起動する場合、Spring Boot用環境変数が競合してしまっていた。

環境変数への外出しは
最近ではMust!?

```
$export SERVER_PORT=8090  
$export FUGA_SERVER_PORT=8091
```

2.4.x

```
$java -jar hoge.jar
```

```
...
```

```
$java -jar fuga.jar --server.port=$FUGA_SERVER_PORT
```

```
...
```

設定の増減の時、いちいち
マッピングが面倒

環境変数のプレフィックス

アプリケーション毎にプレフィックスを設定可能になり、競合を避けることが可能。

2.5.x

```
public static void main(String[] args) {  
    SpringApplication application = new SpringApplication(Application.class);  
    application.setEnvironmentPrefix("fuga");  
    application.run(args);  
}
```

環境変数のプレフィックス
を指定

プレフィックス付きの
環境変数を定義

```
$export SERVER_PORT=8090  
$export FUGA_SERVER_PORT=8091
```

```
$java -jar hoge.jar
```

...

```
$java -jar fuga.jar
```

...

将来、カスタムしたい変数
が増えても安心

Spring Boot 2.5新機能／改善

- ・ 環境変数のプレフィックス
- ・ Spring Data Repositoryのメトリクス公開対応
- ・ DataSource Beanの生成順序の改善
- ・ R2DBCでのデータソース初期化
- ・ その他

Spring Data Repositoryのメトリクス公開対応

Spring Data Repositoryの利用実績が、ActuatorのmetricsエンドポイントやMicrometerに公開できるようになった。

Metricsエンドポイントの例

<http://localhost:8080/actuator/metrics/spring.data.repository.invocations>

```
{  
  "name": "spring.data.repository.invocations",  
  "description": null,  
  "baseUnit": "seconds",  
  "measurements": [  
    {  
      "statistic": "COUNT",  
      "value": 4  
    },  
    {  
      "statistic": "TOTAL_TIME",  
      "value": 0.216646629  
    },  
    {  
      "statistic": "MAX",  
      "value": 0.211294863  
    }  
  ],  
  "availableTags": [  
    {  
      "tag": "exception",  
      "values": ["None"]  
    },  
    {  
      "tag": "method",  
      "values": ["findAll"]  
    },  
    {  
      "tag": "state",  
      "values": ["SUCCESS"]  
    },  
    {  
      "tag": "repository",  
      "values": ["TodoRepository"]  
    }]  
}
```

Spring Data Repositoryのメトリクス公開対応

Spring Data Repositoryの利用実績が、ActuatorのmetricsエンドポイントやMicrometerに公開できるようになった。

Metricsエンドポイントの例

<http://localhost:8080/actuator/metrics/spring.data.repository.invocations>

```
{  
  "name": "spring.data.repository.invocations",  
  "description": null,  
  "baseUnit": "seconds",  
  "measurements": [  
    {  
      "statistic": "COUNT",  
      "value": 4  
    },  
    {  
      "statistic": "TOTAL_TIME",  
      "value": 0.216646629  
    },  
    {  
      "statistic": "MAX",  
      "value": 0.211294863  
    }  
  ],  
  "availableTags": [  
    {  
      "tag": "exception",  
      "values": ["None"]  
    },  
    {  
      "tag": "method",  
      "values": ["findAll"]  
    },  
    {  
      "tag": "state",  
      "values": ["SUCCESS"]  
    },  
    {  
      "tag": "repository",  
      "values": ["TodoRepository"]  
    }]  
}
```

呼び出し回数

総所用時間

最大所用時間

発生例外

メソッド名

実行結果状態

対象リポジトリ

Spring Data Repositoryのメトリクス公開対応

Spring Data Repositoryの利用実績が、ActuatorのmetricsエンドポイントやMicrometerに公開できるようになった。

Prometheus出力の例

<http://localhost:8080/actuator/prometheus>

```
# HELP spring_data_repository_invocations_seconds_max
# TYPE spring_data_repository_invocations_seconds_max gauge
spring_data_repository_invocations_seconds_max{exception="None",method="findAll",repository="TodoRepository",state="SUCCESS",} 0.111220909
# HELP spring_data_repository_invocations_seconds
# TYPE spring_data_repository_invocations_seconds summary
spring_data_repository_invocations_seconds_count{exception="None",method="findAll",repository="TodoRepository",state="SUCCESS",} 1.0
spring_data_repository_invocations_seconds_sum{exception="None",method="findAll",repository="TodoRepository",state="SUCCESS",} 0.111220909
```

Spring Data Repositoryのメトリクス公開対応

Spring Data Repositoryの利用実績が、ActuatorのmetricsエンドポイントやMicrometerに公開できるようになった。

Prometheus出力の例

<http://localhost:8080/actuator/prometheus>

```
# HELP spring_data_repository_invocations_seconds_max
# TYPE sp
spring_da どのテーブルにどのくらいアクセスしているか、所要時間はどのく method="fi
ndAll",re
# HELP sp
# TYPE sp
spring_da うまく加工・可視化すれば性能チューニングで活用できそう ,method="
findAll",
spring_da
findAll",re
↓
(今回は間に合わず可視化は省略)
```

Spring Data Repositoryのメトリクス公開対応

手順1：Actuatorの設定を変更し、metricsエンドポイントやPrometheusの公開設定を行う。

application.properties

```
management.endpoints.web.exposure.include=health,info,metrics,prometheus
```

手順2：micrometer-registry-prometheusの追加
(Prometheus出力の場合)

pom.xml

```
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
  <scope>runtime</scope>
</dependency>
```

Spring Data Repositoryのメトリクス公開対応

手順3：Spring Data Repositoryを利用する。
初回アクセス以降に記録が開始される。

```
public interface TodoRepository extends CrudRepository<Todo, String> {
```

```
@GetMapping("todo/selectall")
@Transactional(readOnly = true)
public List<Todo> selectAll() {
    List<Todo> todoList = new ArrayList<>();
    todoRepository.findAll().iterator().forEachRemaining(todoList::add);
    return todoList;
}
```

Spring Boot 2.5新機能／改善

- ・ 環境変数のプレフィックス
- ・ Spring Data Repositoryのメトリクス公開対応
- ・ DataSource Beanの生成順序の改善
- ・ R2DBCでのデータソース初期化
- ・ その他

DataSource Beanの生成順序の改善

これまで、DataSourceに関するBeanの生成順序が制御されておらず、循環依存エラーが発生するケースがあった。

```
@Bean  
public DataSource dataSource1(DataSourceProperties properties) {  
    ...  
}  
  
@Bean  
public DataSource dataSource2(DataSourceProperties properties) {  
    ...  
}  
  
@Bean  
@Primary  
public AbstractRoutingDataSource routingDataSource(  
    @Qualifier("dataSource1") DataSource dataSource1,  
    @Qualifier("dataSource2") DataSource dataSource2) {  
    ...  
}
```

DataSource Beanの生成順序の改善

```
*****  
APPLICATION FAILED TO START  
*****
```

Description:

The dependencies of some of the beans in the application context form a cycle:

```
helloController defined in file [/Users/ikeyat/Documents/dev/git/spring-boot-2.5-showcase/sprin  
↓  
todoRepository  
↓  
jdbcDialect defined in class path resource [org/springframework/boot/autoconfigure/data/jdbc/Jd  
↓  
namedParameterJdbcTemplate defined in class path resource [org/springframework/boot/autoconfigu  
↓  
jdbcTemplate defined in class path resource [org/springframework/boot/autoconfigure/jdbc/JdbcTe  
↓  
| routingDataSource defined in com.example.springboot2.to25.Application  
↑ ↓  
| dataSource1 defined in com.example.springboot2.to25.Application  
↑ ↓  
| org.springframework.boot.autoconfigure.jdbc.DataSourceInitializerInvoker
```

循環依存エラーが発生

2.4.x

DataSource Beanの生成順序の改善

全く同じコードでも、Spring Boot 2.5.xでは循環エラーとならず正常に起動する。

```

.
-----
\ \ / _ _ ' _ _ _ ( ) _ _ _ _ _ \ \ \ \
( ( ) \ _ _ | ' _ | ' _ | ' _ \ \ _ | \ \ \ \
\ \ \ _ _ ) | | _ ) | | | | | | ( | | ) ) ) )
' | _ _ | . _ | | | | | | \ _ , | / / /
=====|_|=====|_|=====|_|=/_/_/_/
:: Spring Boot ::      (v2.5.2-SNAPSHOT)

2021-07-01 02:00:13.078 INFO 17672 --- [           main] c.example.springboot2.to25.Application : Starting Application using Java 11 on Tomcat/9.0.52
2021-07-01 02:00:13.082 INFO 17672 --- [           main] c.example.springboot2.to25.Application : No active profile set, falling back to default(s): []
2021-07-01 02:00:14.547 INFO 17672 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repository...
2021-07-01 02:00:14.618 INFO 17672 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning
2021-07-01 02:00:15.427 INFO 17672 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-07-01 02:00:15.446 INFO 17672 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-07-01 02:00:15.447 INFO 17672 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.52]
2021-07-01 02:00:15.591 INFO 17672 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplication...
2021-07-01 02:00:15.591 INFO 17672 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization...
2021-07-01 02:00:16.028 INFO 17672 --- [           main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 [Tomcat-localhost]
2021-07-01 02:00:16.302 INFO 17672 --- [           main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 [Tomcat-localhost]
2021-07-01 02:00:17.284 INFO 17672 --- [           main] o.s.b.a.e.web.EndpointLinksResolver : 
2021-07-01 02:00:17.353 INFO 17672 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-07-01 02:00:17.374 INFO 17672 --- [           main] c.example.springboot2.to25.Application : Started Application in 1.398 seconds (process completed in 1398 ms)
2021-07-01 02:00:26.512 INFO 17672 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-07-01 02:00:26.513 INFO 17672 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-07-01 02:00:26.514 INFO 17672 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms

```

エラーなく正常起動

Spring Boot 2.5新機能／改善

- ・ 環境変数のプレフィックス
- ・ Spring Data Repositoryのメトリクス公開対応
- ・ DataSource Beanの生成順序の改善
- ・ R2DBCでのデータソース初期化
- ・ その他

データベース初期化機能のR2DBC対応

これまで、R2DBCデータソース (ConnectionFactory) ではDB初期化機能 (schema.sql/data.sql) が利用できず、以下実装が必要だったが、**2.5.xから実装が不要**となった。

```
// 2.5.x からは実装不要になった  
@Configuration(proxyBeanMethods = false)  
static class DatabaseInitializationConfiguration {  
  
    @Autowired  
    void initializeDatabase(ConnectionFactory connectionFactory) {  
        ResourceLoader resourceLoader = new DefaultResourceLoader();  
        Resource[] scripts = new Resource[] { resourceLoader.getResource("classpath:schema.sql"),  
            resourceLoader.getResource("classpath:data.sql") };  
        new ResourceDatabasePopulator(scripts).populate(connectionFactory).block();  
    }  
}
```

2.4.x

<https://docs.spring.io/spring-boot/docs/2.4.7/reference/htmlsingle/#howto-initialize-a-database-using-r2dbc>

その他

- Docker Image Building Support (槇さんハンズオンにて紹介)
 - Custom Buildpacks <https://spring-boot-cnb-hol.apps.pcfone.io/#9>
 - ボリュームのbinding <https://spring-boot-cnb-hol.apps.pcfone.io/#7>
 - 実行可能WarのDockerImageサポート <https://spring-boot-cnb-hol.apps.pcfone.io/#2>
- Layered Wars
- HTTP/2 over TCP (h2c)
- OpenMetrics for Prometheus
- WebFluxでの@Timed
- MongoDB Metrics
- Liquibase DataSource
- GET requests to actuator/startup
- Actuator Endpoint for Quartz

<https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.5-Release-Notes>

アジェンダ

- ・ ライフサイクル等
- ・ 新機能／改善
- ・ 既存機能の変更や影響
- ・ ドキュメント改善



<https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.5-Release-Notes>

既存機能の変更や影響

- データベース初期化機能のリファクタ
- デフォルトでの/infoエンドポイントの保護
- Default Error Viewのメッセージ非表示
- AbstractRoutingDataSourceのヘルスチェック仕様変更
- ErrorControllerのgetErrorPath()の廃止
- その他

データベース初期化機能のリファクタ

ScriptでのDB初期化機能 (schema.sql/data.sql) の内部構造の変更があり、プロパティが変更になっている。

2.4.x以前 (2.5.xから非推奨)	2.5.x以降	説明
spring.datasource.initialization-mode	spring.sql.init.enabled (2.5.0) spring.sql.init.mode (2.5.1+)	DB初期化機能を作動させる条件
spring.datasource.platform	spring.sql.init.platform	data-*.sql、 schema-*.sqlの*部分の指定
spring.datasource.schema	spring.sql.init.schema-locations	schema.sqlのリソースパス
spring.datasource.schema-username	spring.sql.init.username	schema.sqlのDBユーザ名
spring.datasource.schema-password	spring.sql.init.password	schema.sqlのDBユーザパスワード
spring.datasource.data	spring.sql.init.data-locations	data.sqlのリソースパス
spring.datasource.data-username	spring.sql.init.username	data.sqlのDBユーザ名
spring.datasource.data-password	spring.sql.init.password	data.sqlのDBユーザパスワード
spring.datasource.continue-on-error	spring.sql.init.continue-on-error	Script途中でエラーが起きた時に継続するか
spring.datasource.separator	spring.sql.init.separator	Scriptの区切り文字
spring.datasource.sql-script-encoding	spring.sql.init.encoding	Scriptの文字エンコーディング

データベース初期化機能のリファクタ

ScriptでのDB初期化機能 (schema.sql/data.sql) の内部構造の変更があり、プロパティが変更になっている。

2.4.x以前 (2.5.xから非推奨)	2.5.x以降	説明	
spring.datasource.initialization-mode	spring.sql.init.enabled (2.5.0) spring.sql.init.mode (2.5.1+)	DB初期化機能を作動させる条件	
spring.datasource.url		data-* .sql、 schema-* .sql の * 部分の URL	
spring.datasource.username		データベース名	
spring.datasource.password		データベースパス	
spring.datasource.driver-class-name		データベース名	
spring.datasource.url		データベースパス	
spring.datasource.username		データベース名	
spring.datasource.password		データベース名	
spring.datasource.driver-class-name		データベース名	
作動条件	2.4.x	2.5.0	2.5.1+
組込DBの場合のみ	embedded*	-	embedded*
常に作動	always	true*	always
無効	never	false	never
spring.datasource.url			* デフォルト値
spring.datasource.sql-script-encoding	spring.sql.init.encoding	Scriptの文字エンコーディング	

データベース初期化機能のリファクタ

ScriptでのDB初期化機能 (schema.sql/data.sql) の内部構造の変更があり、プロパティが変更になっている。

2.4.x以前

注意点②

spring.datasource.schema

- 2.4.x以前はschema.sqlとdata.sqlを別のDBユーザで実行できた

spring.datasource.data

- 2.5.xでは同一DBユーザでのみ実行できる（設定も一本化）

spring.datasource.schema	spring.sql.init.schema-locations	schema.sqlのリソースパス
spring.datasource.schema-username	spring.sql.init.username	schema.sqlのDBユーザ名
spring.datasource.schema-password	spring.sql.init.password	schema.sqlのDBユーザパスワード
spring.datasource.data	spring.sql.init.data-locations	data.sqlのリソースパス
spring.datasource.data-username	spring.sql.init.username	data.sqlのDBユーザ名
spring.datasource.data-password	spring.sql.init.password	data.sqlのDBユーザパスワード
spring.datasource.continue-on-error	spring.sql.init.continue-on-error	Script途中でエラーが起きた時に継続するか
spring.datasource.separator	spring.sql.init.separator	Scriptの区切り文字
spring.datasource.sql-script-encoding	spring.sql.init.encoding	Scriptの文字エンコーディング

既存機能の変更や影響

- ・ データベース初期化機能のリファクタ
- ・ デフォルトでの/infoエンドポイントの保護
- ・ Default Error Viewのメッセージ非表示
- ・ AbstractRoutingDataSourceのヘルスチェック仕様変更
- ・ ErrorControllerのgetErrorPath()の廃止
- ・ その他

デフォルトでの/infoエンドポイントの保護

アプリケーション情報を開示するActuatorの「/info」がデフォルトで公開されなくなった。

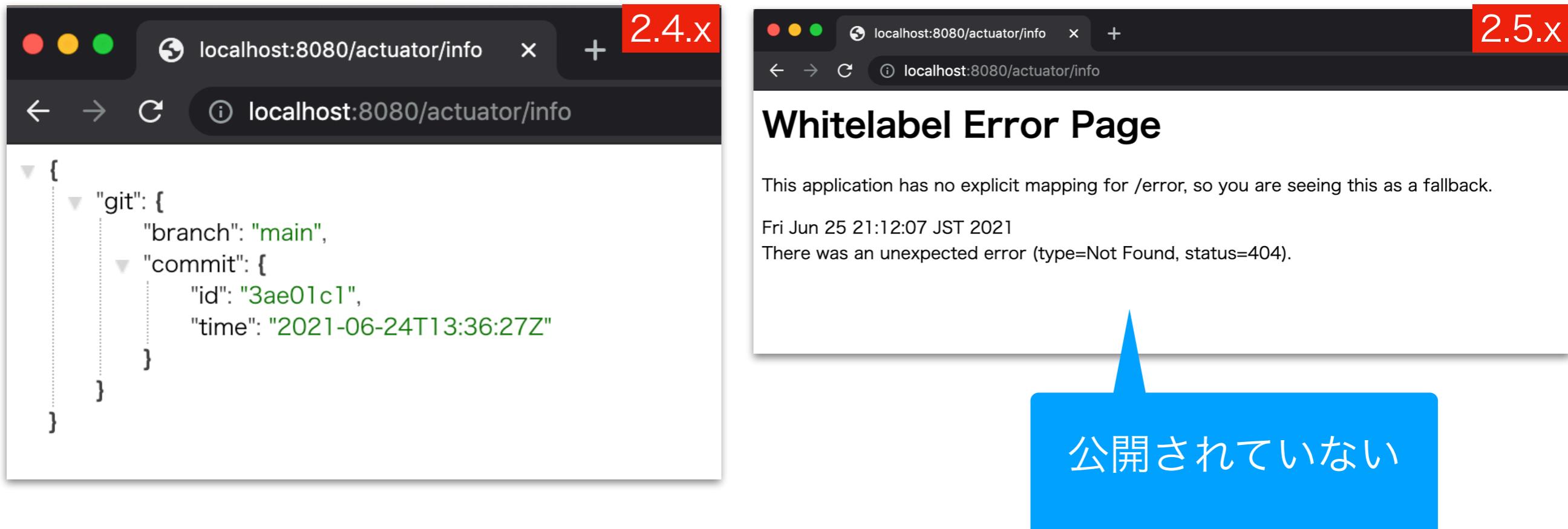
ID	JMX	Web
auditevents	Yes	No
beans	Yes	No
caches	Yes	No
conditions	Yes	No
configprops	Yes	No
env	Yes	No
flyway	Yes	No
health	Yes	Yes
heapdump	N/A	No
httptrace	Yes	No
info	Yes	No

2.4.x Yes
↓
2.5.x No

<https://docs.spring.io/spring-boot/docs/2.5.0/reference/html/actuator.html#actuator>

デフォルトでの/infoエンドポイントの保護

Spring Security依存無しのアプリケーションの場合



「git-commit-id-plugin」を用いてビルド時にgit.propertiesを生成した場合の例

デフォルトでの/infoエンドポイントの保護

Spring Security依存ありのアプリケーションの場合

The image displays three browser windows illustrating the change in endpoint protection between Spring Boot 2.4.x and 2.5.x.

- 2.4.x:** Shows the JSON response for the `/actuator/info` endpoint, which includes detailed information about the git repository (branch: "main", commit ID: "3ae01c1", time: "2021-06-24T13:36:27Z").
- 2.5.x:** Shows a "Please sign in" login screen with fields for "Username" and "Password" and a "Sign in" button. A blue arrow points from this screen down to the 2.5.x browser window below.
- 2.5.x:** Shows a "Whitelabel Error Page" with the message: "This application has no explicit mapping for /error, so you are seeing this as a fallback." It also shows the timestamp "Fri Jun 25 21:25:03 JST 2021" and the error message "There was an unexpected error (type=Not Found, status=404)."

A blue callout box contains the following Japanese text:

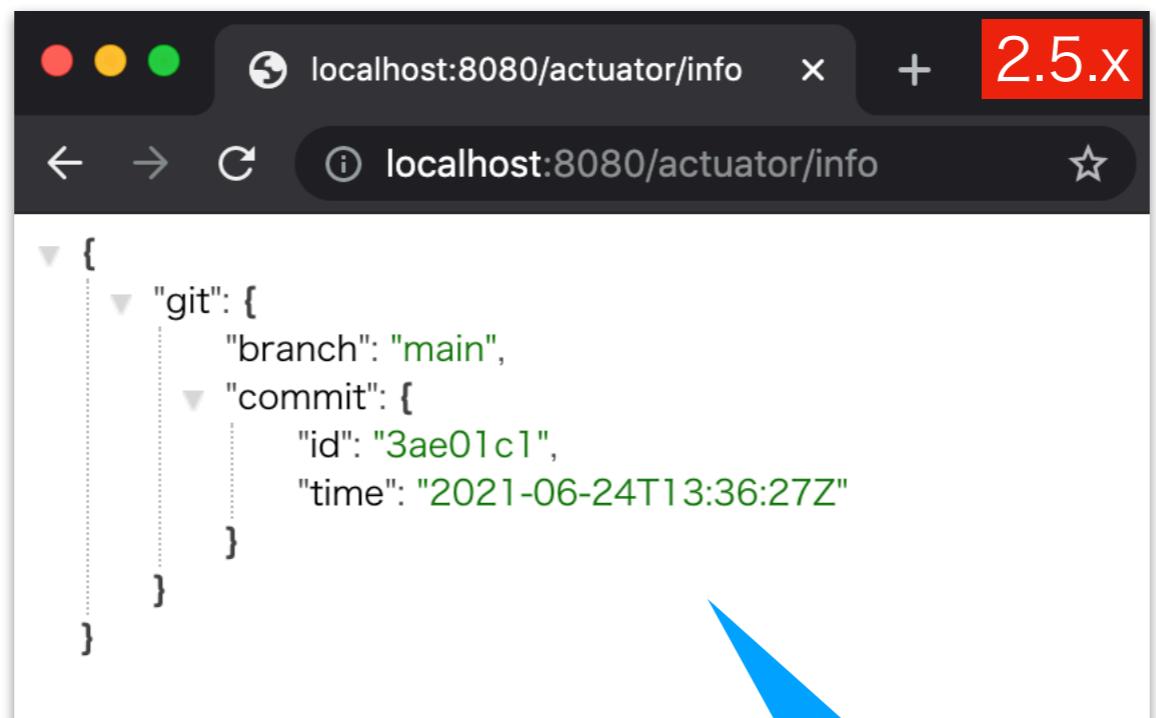
- 認証対象となった
- 公開されていない

デフォルトでの/infoエンドポイントの保護

保護を解除する方法

application.properties

```
management.endpoints.web.exposure.include=health,info
```



公開された

既存機能の変更や影響

- ・ データベース初期化機能のリファクタ
- ・ デフォルトでの/infoエンドポイントの保護
- ・ Default Error Viewのメッセージ非表示
- ・ AbstractRoutingDataSourceのヘルスチェック仕様変更
- ・ ErrorControllerのgetErrorPath()の廃止
- ・ その他

Default Error Viewのメッセージ非表示

Spring Boot 2.3にてデフォルトで空文字置き換えされるようになったエラーメッセージが、2.5からは項目ごと削除された。

2.2.x

```
{  
  "timestamp": "2021-06-25T13:40:28.785+0000",  
  "status": 400,  
  "error": "Bad Request",  
  "message": "TEST ERROR MESSAGE",  
  "path": "/error_message"  
}
```

```
@GetMapping("error_message")  
public String errorMessage() {  
    throw new ResponseStatusException(  
        HttpStatus.BAD_REQUEST,  
        "TEST ERROR MESSAGE");  
}
```

2.3.x～2.4.x

```
{  
  "timestamp": "2021-06-25T13:17:34.277+00:00",  
  "status": 400,  
  "error": "Bad Request",  
  "message": "",  
  "path": "/error_message"  
}
```

messageがデフォで
空文字になった

2.5.x

```
{  
  "timestamp": "2021-06-25T13:09:26.444+00:00",  
  "status": 400,  
  "error": "Bad Request",  
  "path": "/error_message"  
}
```

message項目
自体がなくなった

Default Error Viewのメッセージ非表示

メッセージを表示したい場合は、Spring Boot2.3～2.4での対応と同様の設定を行う。

application.properties

```
server.error.include-message=always
```

```
{  
    "timestamp": "2021-06-25T13:55:18.712+00:00",  
    "status": 400,  
    "error": "Bad Request",  
    "messageTEST ERROR MESSAGE",  
    "path": "/error_message"  
}
```

2.5.x

message項目
とメッセージが復活した

既存機能の変更や影響

- ・ データベース初期化機能のリファクタ
- ・ デフォルトでの/infoエンドポイントの保護
- ・ Default Error Viewのメッセージ非表示
- ・ AbstractRoutingDataSourceのヘルスチェック仕様変更
- ・ ErrorControllerのgetErrorPath()の廃止
- ・ その他

AbstractRoutingDataSourceのヘルスチェック仕様変更

AbstractRoutingDataSourceに対するヘルスチェックが正確に機能するようになった。

2.4.x

```
localhost:8080/actuator/health
```

```
{  
    "status": "UP",  
    "components": {  
        "db": {  
            "status": "UNKNOWN",  
            "details": {  
                "routing": true  
            }  
        },  
        "diskSpace": {  
            "status": "UP",  
            "details": {  
                "total": 250685575168,  
                "free": 95114199040,  
                "threshold": 10485760,  
                "exists": true  
            }  
        },  
        "ping": {  
            "status": "UP"  
        }  
    }  
}
```

2.5.x

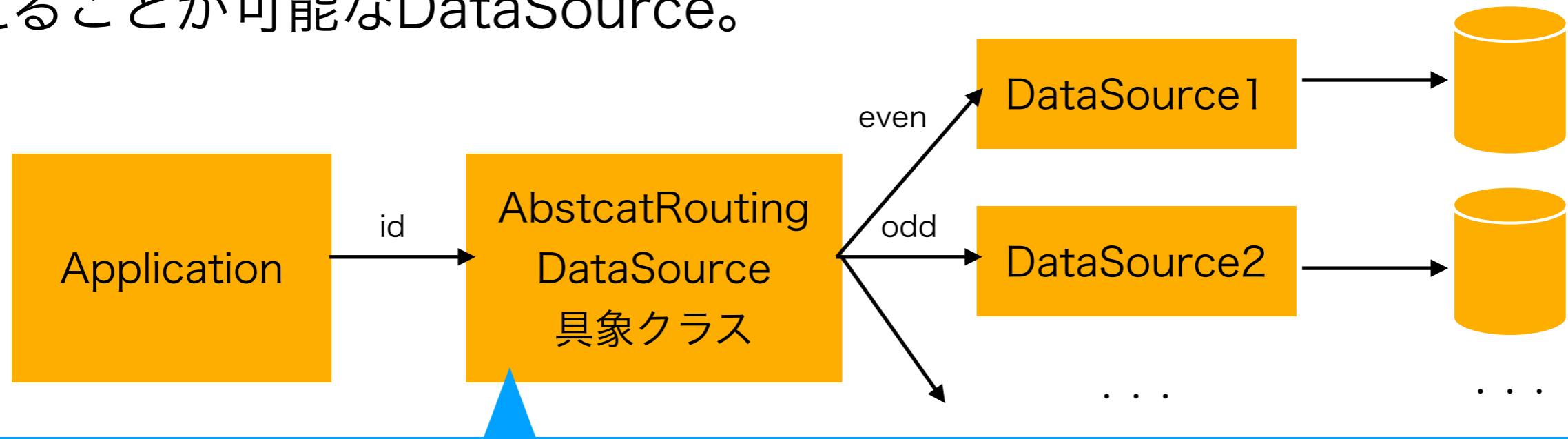
```
localhost:8080/actuator/health
```

```
{  
    "status": "UP",  
    "components": {  
        "even": {  
            "status": "UP",  
            "details": {  
                "database": "H2",  
                "validationQuery": "isValid()"  
            }  
        },  
        "odd": {  
            "status": "UP",  
            "details": {  
                "database": "H2",  
                "validationQuery": "isValid()"  
            }  
        }  
    }  
}
```

management.endpoint.health.show-details=always

参考 AbstractRoutingDataSourceについて

何かしらのルールに基づいて、複数のDataSourceを動的に切り替えることが可能なDataSource。

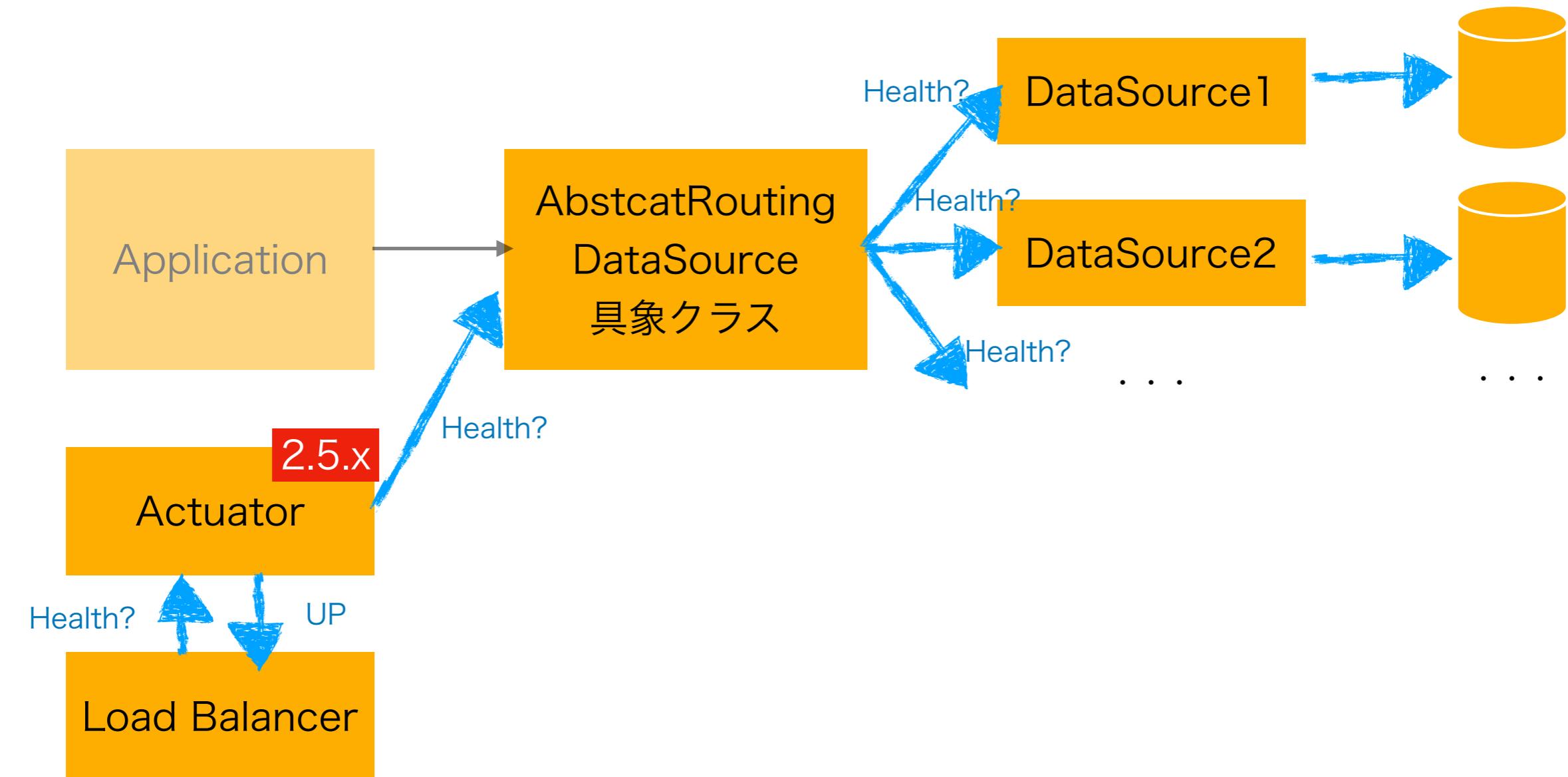


DataSourceの振り分けルールを実装

```
AbstractRoutingDataSource routingDataSource = new AbstractRoutingDataSource() {  
    @Override  
    protected Object determineCurrentLookupKey() {  
        return idHolder.getId() % 2 == 0 ? "even" : "odd";  
    }  
};  
  
routingDataSource.setTargetDataSources(  
    Map.of("even", dataSource1, "odd", dataSource2));
```

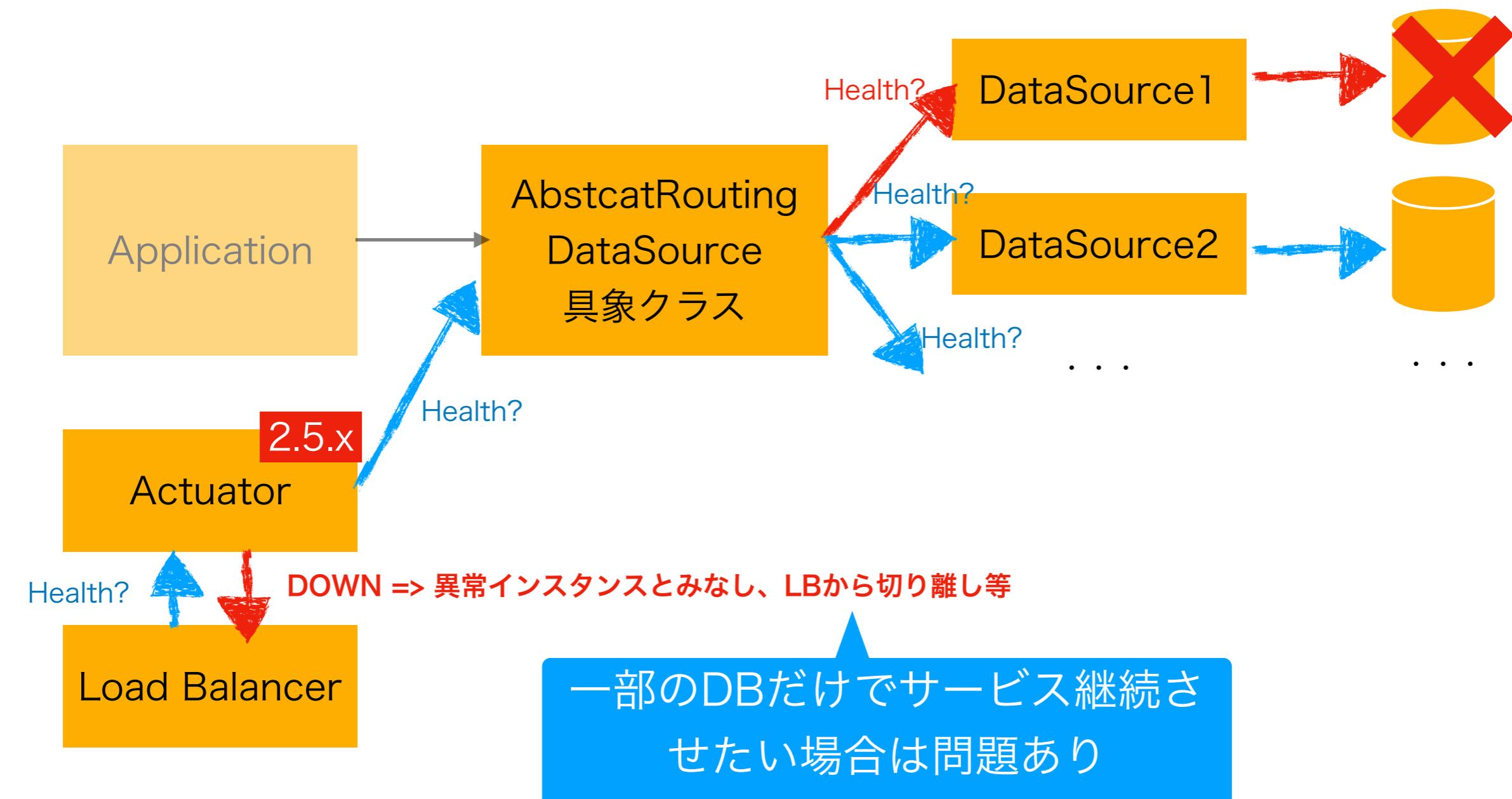
AbstractRoutingDataSourceのヘルスチェック仕様変更

Spring Boot 2.5.xのヘルスチェックでは切り替え対象の
DataSource全てのヘルスチェックを行う。



AbstractRoutingDataSourceのヘルスチェック仕様変更

DBが1つでも障害が起きるとヘルスチェックがNGとなるようになった。



AbstractRoutingDataSourceのヘルスチェック仕様変更

無効にしたい場合は、以下で設定をOFFにする。

application.properties

2.5.x

```
# AbstractRoutingDataSource以外のDataSourceが存在する場合  
management.health.db.ignore-routing-data-sources=true  
  
# 存在しない場合  
management.health.db.enabled=false
```

既存機能の変更や影響

- ・ データベース初期化機能のリファクタ
- ・ デフォルトでの/infoエンドポイントの保護
- ・ Default Error Viewのメッセージ非表示
- ・ AbstractRoutingDataSourceのヘルスチェック仕様変更
- ・ ErrorControllerのgetErrorPath()の廃止
- ・ その他

ErrorControllerのgetErrorPath()の廃止

Spring Boot 2.3で非推奨（かつ無効化）されていたが、非推奨ポリシに従い2.5で削除された。コンパイルエラーとなるため、メソッド削除が必要。

```
public class MyErrorController implements ErrorController {  
    @Override  
    public String getErrorPath() {  
        return null;  
    }  
}
```



コンパイルエラー

<https://github.com/spring-projects/spring-boot/issues/19844>

ErrorControllerのgetErrorPath()の廃止

- そもそも、Spring Boot 2.3以降では「server.error.path」プロパティでエラー時の遷移先パスを指定する必要がある。
- ErrorControllerはもはやマーカーインターフェイスに過ぎず、利用しなくても良い。

getErrorPath

2.4.x

```
@Deprecated  
String getErrorPath()
```

Deprecated. since 2.3.0 in favor of setting the property `server.error.path`

The return value from this method is not used; the property `server.error.path` must be set to override the default error page path.

Returns:

the error path

メソッドは残っていたが
使われていない代物！

application.properties

```
server.error.path=/errorpage
```

その他

- Gradle Default jar and war Tasks
 - 従来設定のままだと2重にjar/warが生成されるらしい
- Gradleはver6.8+が最低要件になった
- デフォルトEL実装の変更
 - glassfish:jakarta.el -> tomcat-embed-el
- Spring Data Solrのauto-config廃止
- Logging Shutdown Hooksがデフォルトで有効になった
- Flyway and Liquibase JDBC URLs
- Cassandra Throttling Properties
- Customizing jOOQ's DefaultConfiguration
- Spring Boot 2.3の非推奨対象の削除

アジェンダ

- ・ ライフサイクル等
- ・ 新機能／改善
- ・ 既存機能の変更や影響
- ・ ドキュメント改善



<https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.5-Release-Notes>

ドキュメントの改善

The `Application.java` file would declare the `main` method, along with the basic `@SpringBootApplication`, as follows:

2.4.x

```
package com.example.myapplication;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

JAVA

<https://docs.spring.io/spring-boot/docs/2.4.7/reference/html/using-spring-boot.html#using-boot-locating-the-main-class>

The `MyApplication.java` file would declare the `main` method, along with the basic `@SpringBootApplication`, as follows:

2.5.x

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MyApplication {

    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }

}
```

JAVA

<https://docs.spring.io/spring-boot/docs/2.5.1/reference/html/using.html#using-structuring-your-code.locating-the-main-class>

ドキュメントの改善

The `Application.java` file would declare the `main` method, along with the basic `@SpringBootApplication`, as follows:

2.4.x

```
package com.example.myapplication;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

JAVA

<https://docs.spring.io/spring-boot/docs/2.4.7/reference/html/using-spring-boot.html#using-boot-locating-the-main-class>

The `MyApplication.java` file would declare the `main` method, along with the basic `@SpringBootApplication`, as follows:

2.5.x

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MyApplication {

    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }

}
```

JAVA

• テキストカラー

- import表示切替
- コピペ



<https://docs.spring.io/spring-boot/docs/2.5.1/reference/html/using.html#using-structuring-your-code.locating-the-main-class>

まとめ

- Spring Boot 2.5は目立った新機能がなく地味な印象
- 逆に捉えると、VerUpしやすい？
- 2.5.0の人はすぐにVerUpを（データソース初期化のバグ）
- Gradleユーザは要注意（二重jar生成、最低バージョン要件）