springboot(十三)：springboot小技巧

 2017/06/22

一些springboot小技巧、小知识点

**初始化数据**

我们在做测试的时候经常需要初始化导入一些数据，如何来处理呢？会有两种选择，一种是使用Jpa，另外一种是Spring JDBC。两种方式各有区别下面来详细介绍。

**使用Jpa**

在使用spring boot jpa的情况下设置spring.jpa.hibernate.ddl-auto的属性设置为 create or create-drop的时候，spring boot 启动时默认会扫描classpath下面（项目中一般是resources目录）是否有import.sql，如果有机会执行import.sql脚本。

**使用Spring JDBC**

使用Spring JDBC 需要在配置文件中添加以下配置

spring:

datasource:

schema: database/data.sql

sql-script-encoding: utf-8

jpa:

hibernate:

ddl-auto: none

* schema ：设置脚本的路径
* sql-script-encoding：设置脚本的编码

spring boot项目启动的时候会自动执行脚本。

**ddl-auto 四个值的解释**

1. create： 每次加载hibernate时都会删除上一次的生成的表，然后根据你的model类再重新来生成新表，哪怕两次没有任何改变也要这样执行，这就是导致数据库表数据丢失的一个重要原因。
2. create-drop ：每次加载hibernate时根据model类生成表，但是sessionFactory一关闭,表就自动删除。
3. update：最常用的属性，第一次加载hibernate时根据model类会自动建立起表的结构（前提是先建立好数据库），以后加载hibernate时根据 model类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行。要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等 应用第一次运行起来后才会。
4. validate ：每次加载hibernate时，验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值。 5、 none : 什么都不做。

**不同点**

第一种方式启动的时候Jpa会自动创建表，import.sql只负责创建表单后的初始化数据。第二种方式启动的时候不会创建表，需要在初始化脚本中判断表是否存在，再初始化脚本的步骤。

在生产中，这两种模式都建议慎用！

参考：[howto-database-initialization](https://docs.spring.io/spring-boot/docs/current/reference/html/howto-database-initialization.html)

**thymeleaf 设置不校验html标签**

默认配置下，thymeleaf对.html的内容要求很严格，比如，如果少封闭符号/，就会报错而转到错误页。也比如你在使用Vue.js这样的库，然后有<div v-cloak></div>这样的html代码，也会被thymeleaf认为不符合要求而抛出错误。

通过设置thymeleaf模板可以解决这个问题，下面是具体的配置:

spring.thymeleaf.cache=false

spring.thymeleaf.mode=LEGACYHTML5

LEGACYHTML5需要搭配一个额外的库NekoHTML才可用 项目中使用的构建工具是Maven添加如下的依赖即可完成:

<dependency>

<groupId>net.sourceforge.nekohtml</groupId>

<artifactId>nekohtml</artifactId>

<version>1.9.22</version>

</dependency>

参考：[thymeleaf模板对没有结束符的HTML5标签解析出错的解决办法](http://blog.csdn.net/yalishadaa/article/details/60768811)

**随机端口**

为Spring Cloud的应用实用随机端口非常简单，主要有两种方法：

设置server.port=0，当应用启动的时候会自动的分配一个随机端口，但是该方式在注册到Eureka的时候会一个问题：所有实例都使用了同样的实例名（如：Lenovo-test:hello-service:0），这导致只出现了一个实例。所以，我们还需要修改实例ID的定义，让每个实例的ID不同，比如使用随机数来配置实例ID：

server.port=0

eureka.instance.instance-id=${spring.application.name}:${random.int}

除了上面的方法，实际上我们还可以直接使用random函数来配置server.port。这样就可以指定端口的取值范围，比如：

server.port=${random.int[10000,19999]}

由于默认的实例ID会由server.port拼接，而此时server.port设置的随机值会重新取一次随机数，所以使用这种方法的时候不需要重新定义实例ID的规则就能产生不同的实例ID了。

参考：[Spring Cloud实战小贴士：随机端口](http://blog.didispace.com/spring-cloud-tips-2/)

[**示例代码-github**](https://github.com/ityouknow/spring-boot-examples)

[**示例代码-码云**](https://gitee.com/ityouknow/spring-boot-examples)