玩转Spring全家桶

极客时间视频课程《玩转Spring全家桶》课程课件及代码示例。

第一章: 初识 Spring

macos动态切换java版本

在macOS中安装jdk，并查看是否安装成功

```
jameszou@JamesZOUdeMacBook-Pro target % /usr/libexec/java_home -V
Matching Java Virtual Machines (2):
    11.0.7 (x86_64) "Oracle Corporation" - "Java SE 11.0.7"
/Library/Java/JavaVirtualMachines/jdk-11.0.7.jdk/Contents/Home
    1.8.0_251 (x86_64) "Oracle Corporation" - "Java SE 8"
/Library/Java/JavaVirtualMachines/jdk1.8.0_251.jdk/Contents/Home
/Library/Java/JavaVirtualMachines/jdk-11.0.7.jdk/Contents/Home
```

打开根目录下的隐藏文件.bash_profile进行环境配置

打开 .bash_profile，没有的话创建

```
vim ~/.bash_profile
```

写入以下内容

```
export
JAVA_8_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_251.jdk/Contents/
Home
export JAVA_11_HOME=/Library/Java/JavaVirtualMachines/jdk-
11.0.7.jdk/Contents/Home
alias jdk8="export JAVA_HOME=$JAVA_8_HOME" #编辑一个命令jdk8，输入则转至
jdk1.8
alias jdk11="export JAVA_HOME=$JAVA_11_HOME" #编辑一个命令jdk11，输入则转至
jdk1.11
export JAVA_HOME=`/usr/libexec/java_home`  #最后安装的版本，这样当自动更新时，
始终指向最新版本
```

执行命令生效

```
source ~/.bash_profil
```

使用jdk？实现终端命令的自由切换

```
jameszou@JamesZOUdeMacBook-Pro helloworld % jdk11
jameszou@JamesZOUdeMacBook-Pro helloworld % java -version
java version "11.0.7" 2020-04-14 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.7+8-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.7+8-LTS, mixed mode)

jameszou@JamesZOUdeMacBook-Pro helloworld % jdk8
jameszou@JamesZOUdeMacBook-Pro helloworld % java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)
```

生成骨架

Spring Initializr

访问

localhost:8080/hello

localhost:8080/actuator/health

localhost:8080/actuator/beans

编译

```
mvn clean package -Dmaven.test.skip
```

编译之后在target目录生成文件

helloworld-0.0.1-SNAPSHOT.jar

```
jameszou@JamesZOUdeMacBook-Pro target % ls -al
total 37096
drwxr-xr-x   8 jameszou  staff       256  5 17 09:54 .
drwxr-xr-x@ 14 jameszou  staff       448  5 17 09:54 ..
drwxr-xr-x   4 jameszou  staff       128  5 17 09:54 classes
drwxr-xr-x   3 jameszou  staff        96  5 17 09:54 generated-sources
-rw-r--r--   1 jameszou  staff  18985235  5 17 09:54 helloworld-0.0.1-
SNAPSHOT.jar
-rw-r--r--   1 jameszou  staff      3053  5 17 09:54 helloworld-0.0.1-
SNAPSHOT.jar.original
drwxr-xr-x   3 jameszou  staff        96  5 17 09:54 maven-archiver
drwxr-xr-x   3 jameszou  staff        96  5 17 09:54 maven-status
```

执行

停止ide中运行的进程

```
java -jar helloworld-0.0.1-SNAPSHOT.jar
```

第2章: Spring中的数据操作

Demo1 – datasource

org.springframework.boot.CommandLineRunner

在应用服务启动时，需要在所有Bean生成之后，加载一些数据和执行一些应用的初始化。例如：删除临时文件，清楚缓存信息，读取配置文件，数据库连接，这些工作类似开机自启动的概念，CommandLineRunner、ApplicationRunner 接口是在容器启动成功后的最后一步回调（类似开机自启动）。

org.springframework.boot.ApplicationRunner

ApplicationRunner接口源码定义如下：

```
package org.springframework.boot;

import org.springframework.core.Ordered;
import org.springframework.core.annotation.Order;

/**
 * Interface used to indicate that a bean should <em>run</em> when it is
contained within
 * a {@link SpringApplication}. Multiple {@link ApplicationRunner} beans
can be defined
 * within the same application context and can be ordered using the
{@link Ordered}
 * interface or {@link Order @Order} annotation.
 *
 * @author Phillip Webb
 * @since 1.3.0
 * @see CommandLineRunner
 */
```

```
@FunctionalInterface
public interface ApplicationRunner {

    /**
     * Callback used to run the bean.
     * @param args incoming application arguments
     * @throws Exception on error
     */
    void run(ApplicationArguments args) throws Exception;

}
```

在对该接口的注释中，可以看到两个接口的应用场景，甚至注释都是完全一样的。唯一的区别是接口中的函数run的参数，一个是与main函数同样的(String[] args)，而另外一个是ApplicationArgumens类型。在一般情况下，开发时是不需要添加命令行参数的，因此两个接口的区别对于这样的场景也就完全一样了。但如果真的需要类型–foo=bar的option arguments，为了方便起见，可以使用ApplicationRunner来读取类似的参数。

CommandLineRunner和ApplicationRunner区别

从上面的分析可以看出，CommandLineRunner和ApplicationRunner接口的作用是完全一致的，唯一不同的则是接口中待实现的run方法，其中CommandLineRunner的run方法参数类型与main一样是原生的String[] 类型，而ApplicationRunner的run方法参数类型为ApplicationArguments类型。

demo2 - pure-spring-datasource

不使用spring boot的配置

demo3 - multi-datasource

配置多个数据源

```
@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class,
        DataSourceTransactionManagerAutoConfiguration.class,
        JdbcTemplateAutoConfiguration.class})
```

通过

localhost:8080/actuator/beans

进行查看

demo4 – druid

阿里巴巴的druid数据库连接池

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
  <exclusions>
    <exclusion>
      <artifactId>HikariCP</artifactId>
      <groupId>com.zaxxer</groupId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid-spring-boot-starter</artifactId>
  <version>1.1.10</version>
</dependency>
```

druid-filter.properties

ConnectionLogFilter.java

demo 5 - simple jdbc

spring jdbc简单操作

demo 6 - programmatic-transaction

编程式事务

JTA(Java Transaction Manager)

JTA是如何实现多数据源的事务管理呢?

主要的原理是**两阶段提交**

demo 7 - declarative-transaction

声明式事务

```
        <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-aop</artifactId>
        </dependency>
        <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-aspects</artifactId>
        </dependency>
```

demo 8 - errorcode

spring的jdbc异常抽象

第3章: ORM

demo 1 - jpa demo

demo 2 - jpa complex demo

repository中的这些方法在哪定义呢?

demo 3 - mybatis-demo

MoneyTypeHandler什么时候会用到

demo 4 - mybatis-generator-demo

配置

```
src/main/resource/generatorConfig.xml
```

自动生成的有

```
src/main/java/geektime/spring/data/mybatis/mapper
src/main/java/geektime/spring/data/mybatis/model
src/main/resource/mapper
```

demo 5 - mybatis-pagehelper-demo

demo 6 - springbucks

第4章: NoSQL实践

demo 1 - mongo-demo

demo 2 - mongo-repository-demo

pom.xml

```
        <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
```

demo 3 - jedis-demo

demo 4 - cache-demo

demo 5 - cache-with-redis-demo

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

demo 6 - redis-demo

demo 7 - redis-repository-demo

第5章: 数据访问进阶

demo 1 - simple-reactor-demo

demo 2 - redis-demo

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis-reactive</artifactId>
</dependency>
```

demo 3 - mongodb-demo

pom.xml

```
        <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
        </dependency>
```

demo 4 - simple-r2dbc-demo

spring Milestone的依赖不在主仓库里

```
      <repositories>
        <repository>
          <id>spring-milestone</id>
          <name>Spring Maven MILESTONE Repository</name>
          <url>http://repo.spring.io/libs-milestone</url>
        </repository>
      </repositories>
```

demo 5 - r2dbc-repository-demo

demo 6 - performance-aspect-demo

p6spy

demo 7 - reactive-springbucks

第6章: Web哪些事

demo 1 - simple-controller-demo

idea中有rest插件

demo 2 - context-hierarchy-demo

spring的应用程序上下文

demo 3 - complex-controller-demo

demo 4 - more-complex-controller-demo

```
    @PostMapping(path = "/", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
    @ResponseBody
    @ResponseStatus(HttpStatus.CREATED)
    public List<Coffee> batchAddCoffee(@RequestParam("file")
MultipartFile file) {
       ...
    }
```

上传文件

使用postman，body -> form-data -> 填入key -> 选择文件

demo 5 - json-view-demo

```
        <!-- 增加Jackson的Hibernate类型支持 -->
        <dependency>
          <groupId>com.fasterxml.jackson.datatype</groupId>
          <artifactId>jackson-datatype-hibernate5</artifactId>
          <version>2.9.8</version>
        </dependency>
        <!-- 增加Jackson XML支持 -->
        <dependency>
          <groupId>com.fasterxml.jackson.dataformat</groupId>
          <artifactId>jackson-dataformat-xml</artifactId>
          <version>2.9.0</version>
        </dependency>
```

```
    @PostMapping(path = "/", consumes =
    MediaType.APPLICATION_JSON_UTF8_VALUE)
```

序列化

```
    @JsonComponent
```

demo 6 - thymeleaf-view-demo

模版引擎

CoffeeOrderController.java

```
        @GetMapping(path = "/")
        public ModelAndView showCreateForm() {
            return new ModelAndView("create-order-form");
        }
```

demo 7 - cache-demo

静态资源与缓存

如何使用，如何看出来缓存了

demo 8 - exception-demo

异常处理

demo 9 - springbucks

第7章: 访问web资源

demo 1 - simple-resttemplate-demo

RestTemplate

demo 2 - complex-resttemplate-demo

RestTemplate高阶用法

demo 3 - advanced-resttemplate-demo

简单定制RestTemplate

demo 4 - webclient-demo

通过WebClient访问web资源

demo 5 - customer-service

第8章: Web开发进阶

demo 1 - hateoas-waiter-service

第10章: 运行中的spring boot

docker-demo

依次执行如下命令

```
mvn clean package -Dmaven.test.skip

# 执行命令会找到springbucks/waiter-server
docker images

# 执行镜像
docker run --name waiter-service -d -p 8080:8080 springbucks/waiter-
service:0.0.1-SNAPSHOT

# 观察日志
docker logs
docker ps

# 测试
curl http://localhost:8080/coffee/1

# 推到仓库中去
docker deploy xxx
```

第11章: Spring Cloud云原生应用

第12章: 服务发现与注册

Docker

mongodb

```
docker pull mongo
# -v 宿主目录:容器目录，挂载磁盘卷
# -e，设置环境变量
# -d，后台运行容器
# docker run --name mongo -p 27017:27017 -v ~/dockerdata/mongo:/data/db
-e MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INITDB_ROOT_PASSWORD=admin
-d mongo
docker run --name mongo -p 27017:27017 -v ~/dockerdata/mongo:/data/db -e
MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INITDB_ROOT_PASSWORD=admin -d
mongo
# 停止
docker stop mongo

# 登录到 MongoDB 容器器中
docker exec -it mongo bash
# 通过 Shell 连接 MongoDB
mongo -u admin -p admin
```

```
# 查看库
show dbs;
# 查看用户
show users;
```

redis

```
docker pull redis
docker run --name redis -d -p 6379:6379 redis
docker exec -it redis bash
```