# 使用Spring Boot构建RESTful Web服务以访问存储于Aerospike集群中的数据

作者 Peter Milne ，译者 **张卫滨** 发布于 2014年2月21日 |
Spring Boot是对Spring快速入门的强大工具。Spring Boot能够帮助你很容易地构建基于Spring的应用。

Aerospike是分布式和可复制的内存数据库，不管使用DRAM还是原生的flash/SSD，Aerospike都进行了优化。

Aerospike具有高可靠性并且遵循ACID。开发人员能够在不停止数据库服务的情况下，很快地将数据库集群从两个节点扩展到二十个节点。

## 你所要构建的是什么

本文将会引领你使用Spring Boot创建一个简单的RESTful Web服务。

要构建的服务接受一个HTTP GET请求。它的响应是如下的JSON：

```
{"expiration":121023390,"bins":
{"DISTANCE":2446,"DEST_CITY_NAME":"New
York","DEST":"JFK","YEAR":2012,"ORI_AIRPORT_ID":"14679","DEP
_TIME":
"802","DAY_OF_MONTH":12,"DEST_STATE_ABR":"NY","ORIGIN":"SAN"
,"FL_NUM"
:160,"CARRIER":"AA","ORI_STATE_ABR":"CA","FL_DATE":"2012/01/
12",
"AIR_TIME":291,"ORI_CITY_NAME":"San
Diego","ELAPSED_TIME":321,
```

```
"ARR_TIME":"1623","AIRLINE_ID":19805},"generation":1}
```

这里所使用的数据是商业上的飞行航班详情（包含在样例代码中，这是一个名为flights_from.csv的数据文件，它包含了大约一百万条航班信息）。



在产品化（或其他）环境中，还会有很多内置的特性添加到应用中以管理服务。这个功能来源于Spring，参见Spring指导：Building a RESTful web service。

## 你所需要的是什么

- 喜欢的文本编辑器或IDE
- JDK 7或更高版本

## 搭建工程

在构建应用的时候，你可以使用任何喜欢的构建系统，不过在这里提供了Maven的代码。如果你不熟悉Maven的话，请参考Spring指导：Building Java Projects with Maven。

你还需要构建并安装Aerospike的Java客户端到本地Maven仓库之中。下载源码发布版本，将其进行进行unzip/untar并运行如下的Maven命令：

- mvn install:install-file -Dfile=client/depends/gnu-crypto.jar -DgroupId=org.gnu -DartifactId=gnu-crypto -Dversion=2.0.1 -Dpackaging=jar
- mvn clean

- mvn package

## 创建目录结构

在你选择的工程之中，创建如下所示的子目录结构：

->src
->main
->java
->com
->aerospike
->client
->rest

## 创建Maven的pom文件

在工程的根目录下创建一个maven的pom.xml，其代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.aerospike</groupId>
      <artifactId>aerospike-restful-example</artifactId>
      <version>1.0.0</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>0.5.0.M4</version>
    </parent>
      <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
```

```
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-
actuator</artifactId>
        </dependency>
            <!-- Aerospike client. -->
        <dependency>
              <groupId>com.aerospike</groupId>
              <artifactId>aerospike-client</artifactId>
              <version>3.0.9</version>
        </dependency>
        <!-- Apache command line parser. -->
        <dependency>
              <groupId>commons-cli</groupId>
              <artifactId>commons-cli</artifactId>
              <version>1.2</version>
        </dependency>
  </dependencies>

  <properties>
      <start-
class>com.aerospike.client.rest.AerospikeRESTfulService
</start-class>
  </properties>

  <build>
      <plugins>
          <plugin>
              <artifactId>maven-compiler-plugin</artifactId>
              <version>2.3.2</version>
          </plugin>
          <plugin>
              <groupId>org.springframework.boot</groupId>
              <artifactId>spring-boot-maven-
plugin</artifactId>
```

```
            </plugin>
        </plugins>
    </build>
    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <name>Spring Snapshots</name>
            <url>http://repo.spring.io/libs-snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>spring-snapshots</id>
            <name>Spring Snapshots</name>
            <url>http://repo.spring.io/libs-snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
         </pluginRepository>
    </pluginRepositories>

</project>
```

乍看上去有些恐怖，但实际上并非如此。

## 创建一个JSON转换类

Aerospike API会返回一个Record对象，它会包含记录的generation、expiry以及bin值。但是你想让这些值以JSON格式返回。要达到这一点，最简单的方式就是使用一个转换类（translator class）。

所创建的转换类代码如下所示。这是一个工具类，能够将Aerospike Record转换为JSONObject。

```
src/main/java/com/aerospike/client/rest/JSONRecord.java
package com.aerospike.client.rest;
import java.util.Map;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import com.aerospike.client.Record;
/**
* JSONRecord is used to convert an Aerospike Record
* returned from the cluster to JSON format
*
*/
@SuppressWarnings("serial")
public class JSONRecord extends JSONObject {
      @SuppressWarnings("unchecked")
      public JSONRecord(Record record){
            put("generation", record.generation);
            put("expiration", record.expiration);
            put("bins", new JSONObject(record.bins));
            if (record.duplicates != null){
                  JSONArray duplicates = new JSONArray();
                  for (Map<String, Object> duplicate :
record.duplicates){
                        duplicates.add(new
JSONObject(duplicate));
                  }
                  put("duplicates", duplicates);
             }
       }
}
```

这个类并不复杂也很通用。你可能会希望为特定的记录指定使用你的JSON转换器。

## 创建资源控制器

在Spring中，REST端点（endpoint）是Spring MVC控制器。如下的代码能够处理对/as/{namespace}/{set}/getAll/1234的GET请求，并会返回key为1234的航班记录，

在这里{namespace}是针对Aerospike命名空间的路径变量，{set}是针对Aerospike集合的路径变量。

```
src/main/java/com/aerospike/client/rest/RESTController.java
package com.aerospike.client.rest;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;
import com.aerospike.client.AerospikeClient;
import com.aerospike.client.Bin;
import com.aerospike.client.Key;
import com.aerospike.client.Record;
import com.aerospike.client.policy.Policy;
import com.aerospike.client.policy.WritePolicy;

@Controller
public class RESTController {
      @Autowired
      AerospikeClient client;

@RequestMapping(value="/as/{namespace}/{set}/getAll/{key}",
method=RequestMethod.GET)
    public @ResponseBody JSONRecord getAll(@PathVariable
("namespace") String namespace,
            @PathVariable("set") String set,
```

```
            @PathVariable("key") String keyvalue) throws
Exception {
    Policy policy = new Policy();
    Key key = new Key(namespace, set, keyvalue);
     Record result = client.get(policy, key);
     return new JSONRecord(result);
  }
}
```

针对人类用户的控制器和针对REST端点控制器之间的区别在于响应体中要包含数据，在这个场景中也就是一个JSON对象，它代表了从Aerospike读取到的记录。

@ResponseBody注解会告知Spring MVC将返回的对象写入到响应体之中。

## 创建可执行的主类

现在要实现主方法来创建Spring MVC控制器，最简单的方式就是使用SpringApplication帮助类。

```
src/main/java/com/aerospike/client/rest/AerospikeRESTfulServ
ice.java
package com.aerospike.client.rest;
import java.util.Properties;
import javax.servlet.MultipartConfigElement;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;
import org.apache.commons.cli.PosixParser;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.EnableAutoConfigurati
on;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
```

```
import com.aerospike.client.AerospikeClient;
import com.aerospike.client.AerospikeException;
@Configuration
@EnableAutoConfiguration
@ComponentScan
public class AerospikeRESTfulService {
      @Bean
      public AerospikeClient asClient() throws
AerospikeException {
            Properties as = System.getProperties();
            return new
AerospikeClient(as.getProperty("seedHost"),
                  Integer.parseInt(as.getProperty("port")));
      }
      @Bean
      public MultipartConfigElement multipartConfigElement()
{
            return new MultipartConfigElement("");
      }
      public static void main(String[] args) throws
ParseException {
             Options options = new Options();
             options.addOption("h", "host", true,
                   "Server hostname (default: localhost)");
             options.addOption("p", "port", true, "Server
port (default: 3000)");
             // parse the command line args
             CommandLineParser parser = new PosixParser();
             CommandLine cl = parser.parse(options, args,
false);
             // set properties
             Properties as = System.getProperties();
             String host = cl.getOptionValue("h",
"localhost");
             as.put("seedHost", host);
```

```
            String portString = cl.getOptionValue("p",
"3000");
            as.put("port", portString);
            // start app

SpringApplication.run(AerospikeRESTfulService.class, args);
      }
}
```

这里添加了@EnableAutoConfiguration注解：它会对一些内容进行默认的加载（如嵌入式的servlet容器），这取决于类路径的内容以及其他的一些事情。

它还使用了@ComponentScan注解，这个注解会告诉Spring扫描rest包来查找控制器（以及其他有注解的组件类）。

最后，这个类还使用了@Configuration注解。它允许你将AerospikeClient实例配置为一个Spring的bean。

这里还定义了一个MultipartConfigElement bean。它能够让你使用这个服务处理POST操作。

主方法中大部分的主体内容都是读取命令行参数以及系统属性，以便指定Aerospike集群的seed主机和端口。

非常简单！

## 上传数据

你可能希望往这个服务中上传数据。要做到这一点的话，我们需要为RESTController类添加一个额外的方法来处理上传的文件。在这个例子中，这会是包含航行记录的CSV文件。

```
src/main/java/com/aerospike/client/rest/RESTController.java
@Controller
public class RESTController {
   . . . (code omitted) . . .
```

```
   /*
    * CSV flights file upload
    */
   @RequestMapping(value="/uploadFlights",
method=RequestMethod.GET)
   public @ResponseBody String provideUploadInfo() {
       return "You can upload a file by posting to this same
URL.";
   }
   @RequestMapping(value="/uploadFlights",
method=RequestMethod.POST)
   public @ResponseBody String
handleFileUpload(@RequestParam("name") String name,
          @RequestParam("file") MultipartFile file){
     if (!file.isEmpty()) {
           try {
                 WritePolicy wp = new WritePolicy();
                 String line = "";
                 BufferedReader br = new BufferedReader(new
InputStreamReader(file.getInputStream()));
                 while ((line = br.readLine()) != null) {
                       // use comma as separator
                       String[] flight = line.split(",");

                       /*
                        * write the record to Aerospike
                        * NOTE: Bin names must not exceed 14
characters
                        */
                           client.put(wp,
                 new Key("test", "flights",flight[0].trim()
),
                 new Bin("YEAR",
Integer.parseInt(flight[1].trim())),
                 new Bin("DAY_OF_MONTH",
Integer.parseInt(flight[2].trim())),
```

```
                new Bin("FL_DATE", flight[3].trim()),
                new Bin("AIRLINE_ID",
Integer.parseInt(flight[4].trim())),
                new Bin("CARRIER", flight[5].trim()),
                new Bin("FL_NUM",
Integer.parseInt(flight[6].trim())),
                new Bin("ORI_AIRPORT_ID",
Integer.parseInt(flight[7].trim())),
                new Bin("ORIGIN", flight[8].trim()),
                new Bin("ORI_CITY_NAME", flight[9].trim()),
                new Bin("ORI_STATE_ABR",
flight[10].trim()),
                new Bin("DEST", flight[11].trim()),
                new Bin("DEST_CITY_NAME",
flight[12].trim()),
                new Bin("DEST_STATE_ABR",
flight[13].trim()),
                new Bin("DEP_TIME",
Integer.parseInt(flight[14].trim())),
                new Bin("ARR_TIME",
Integer.parseInt(flight[15].trim())),
                new Bin("ELAPSED_TIME",
Integer.parseInt(flight[16].trim())),
                new Bin("AIR_TIME",
Integer.parseInt(flight[17].trim())),
                new Bin("DISTANCE",
Integer.parseInt(flight[18].trim()))
                            );
                            System.out.println("Flight [ID=
" + flight[0]
                + " , year=" + flight[1]
                + " , DAY_OF_MONTH=" + flight[2]
                + " , FL_DATE=" + flight[3]
                + " , AIRLINE_ID=" + flight[4]
                + " , CARRIER=" + flight[5]
                + " , FL_NUM=" + flight[6]
```

```
                + " , ORIGIN_AIRPORT_ID=" + flight[7]
                + "]");
                       }
                       br.close();
                       return "You successfully uploaded "
+ name;
                } catch (Exception e) {
          return "You failed to upload " + name + " => " +
e.getMessage();
                }
            } else {
                  return "You failed to upload " + name +
" because the file was empty.";
               }
       }
}
```

新方法handleFileUpload()响应POST请求并且会读取上传的流，每次读取一行。每一行解析后，会构建一个Key对象和多个Bin对象，据此来形成Aerospike记录。最后，调用Aerospike的put()方法，将记录存储到Aerospike集群之中。

另外一个新方法provideUploadInfo()响应GET请求，并返回一条信息来表明允许进行上传。

## 上传的客户端应用

上传可以通过任何你希望的方式来实现。不过，你可以使用下面这个单独的Java类将数据上传到服务上。

```
src/test/java/com.aerospike.client.rest/FlightsUploader.java
package com.aerospike.client.rest;
import org.junit.Before;
import org.junit.Test;
import org.springframework.core.io.FileSystemResource;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
```

```
import org.springframework.web.client.RestTemplate;

public class FilghtsUploader {
      private static final String TEST_FILE =
"flights_from.csv";
      @Before
      public void setUp() throws Exception {
      }

      @Test
      public void upload() {
        RestTemplate template = new RestTemplate();
        MultiValueMap<String, Object> parts = new
LinkedMultiValueMap
<String, Object>();
        parts.add("name", TEST_FILE);
        parts.add("file", new
FileSystemResource(TEST_FILE));
        String response = template.postForObject
("http://localhost:8080/uploadFlights",parts, String.class);
        System.out.println(response);
      }
}
```

### 航班数据

这是来自2012年的真实数据，包括了大约一百万条的记录，所以请注意它需要几分钟的时间才能完成上传。

## 构建并运行服务

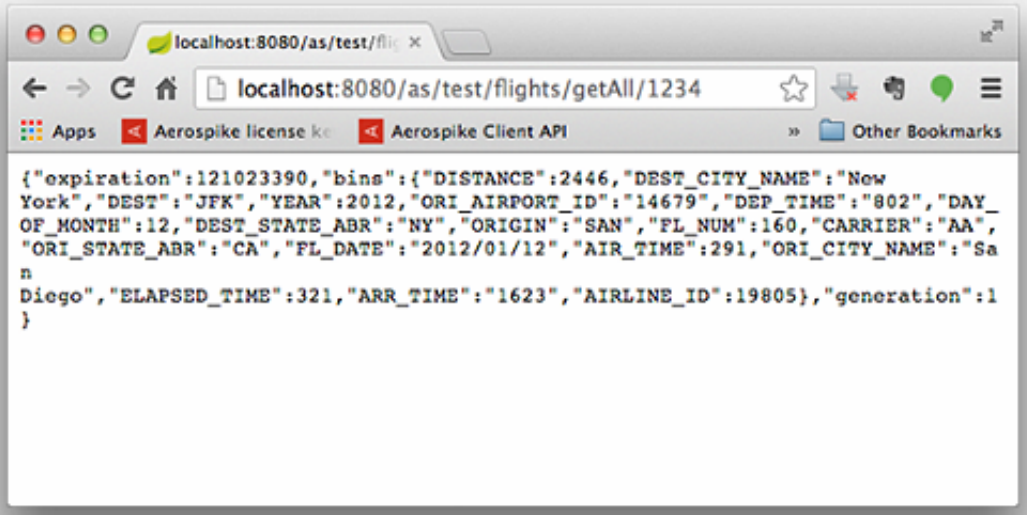
Maven的pom.xml会将服务打包为一个单独的jar文件。使用如下的命令：

**mvn clean package**

这样会生成独立的web服务应用，它会打包为一个可运行的jar文件，位于target子目录之中。这个jar文件中包含了一个Tomcat的实例，所以你可以直接运行这个jar文件，而

没有必要将其安装到应用服务器之中。

**java -jar aerospike-restful-example-1.0.0.jar**




# 总结

恭喜你！你现在已经使用Spring开发了一个简单的RESTful服务，并且连接到了Aerospike集群之中。

# 设计中的考量

目前，访问控制是通过应用来处理的，并不是通过数据库。因为认证过程会拖慢数据库的速度，实际上，所有的NoSQL数据库均不支持这种功能。我们的大多数客户更关注于提升的速度，而不是集成的认证特性。

另外一个要求的通用特性就是两个不同数据集之间的连接（join）。对于所有的分布式数据库来讲，这都是一个挑战，因为要连接的数据是分布式的。在本例中，开发人员必须在应用中实现连接。

# 关于作者

**Peter Milne**是一位很有经验的IT专业人士，对于软件开发和产品的整个生命周期都有着丰富的经验。对于小型和大型的开发团队，他都具有技术技能和管理经验。Peter最近以来在Aerospike担任高级解决方案架构师。在此之前，他在MLC担任高级分析师和编码人员，并且在iTerative Consulting担任过CTO，在此期间，他构建了一个Forte/UDS到Java的转换工具，达到了99.999%准确率。Peter在悉尼科技大学获得了分布式计算的理科硕士学位，并且具有多个直升机安全许可和证书。