使用Spring Boot Actuator将指标导出到InfluxDB和Prometheus

原创 complone SpringForAll社区 2019-01-24

Spring Boot Actuator是Spring Boot 2发布后修改最多的项目之一。它经过了主要的改进，旨在简化定制，并包括一些新功能，如支持其他Web技术，例如新的反应模块 - `Spring WebFlux`。它还为 `InfluxDB`添加了开箱即用的支持，这是一个开源时间序列数据库，旨在处理大量带时间戳的数据。与 `Spring Boot 1.5`使用的版本相比，它实际上是一个很大的简化。您可以通过阅读我之前的一篇文章使用Grafana和InfluxDB自定义指标可视化来了解自己有多少。我在那里描述了如何使用 `@ExportMetricsWriter` bean将[Spring Boot Actuator生成的指标导出到InfluxDB。示例Spring Boot应用程序已在分支主文件中的GitHub存储库sample-spring-graphite上提供该文章。对于本文，我创建了分支spring2，它展示了如何实现与使用Spring Boot 2.0版本之前相同的功能。弹簧启动执行器。

另外，我将向您展示如何将相同的指标导出到另一个流行的监控系统，以便有效地存储时间序列数据 - `Prometheus`。在 `InfluxDB`和 `Prometheus`之间导出指标的模型之间存在一个主要区别。第一个是基于推送的系统，而第二个是基于拉的系统。因此，我们的示例应用程序需要主动将数据发送到 `InfluxDB`监控系统，而使用 `Prometheus`时，它只需要公开将定期获取数据的端点。让我们从 `InfluxDB`开始吧。

> 运行InfluxDB

在上一篇文章中，我没有写太多关于这个数据库及其配置的内容。所以，现在我说一些关于它的话。第一步是我的示例的典型步骤 - 我们将使用 InfluxDB运行 Docker容器。这是在本地计算机上运行 InfluxDB并在 8086端口上公开 HTTP API的最简单命令。 $ docker run -d --name influx -p 8086:8086 influxdb

一旦我们启动了该容器，您可能希望在那里登录并执行一些命令。没有比这更简单的了，只需运行以下命令即可。登录后，您应该看到目标Docker容器上运行的InfluxDB版本。

```
$ docker exec -it influx influx
Connected to http://localhost:8086 version 1.5.2
InfluxDB shell version: 1.5.2
```

第一步是创建数据库。正如您可能猜到的，可以使用命令 create database来实现。然后切换到新创建的数据库。

```
$ create database springboot
$ use springboot
```

这种语义对你来说是否熟悉？是的， InfluxDB为 SQL提供了非常相似的查询语言。它被称为 InluxQL，允许您定义 SELECT语句， GROUP BY或 INTO子句等等。但是，在执行此类查询之前，我们应该将数据存储在数据库中，对吗？现在，让我们继续下一步，以生成一些测试指标。

> 将Spring Boot应用程序与InfluxDB集成

如果您将工件 micrometer-registry-Influx包含在项目的依赖项中，则会自动启用对InfluxDB的导出。当然，我们还需要包括 spring-boot-starter-actuator

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-influx</artifactId>
</dependency>
```

您唯一要做的就是覆盖 InfluxDB 的默认地址，因为我们在 VM 上运行 InfluxDB Docker 容器。默认情况下， Spring Boot Data 尝试连接名为 mydb 的数据库。但是，我已经创建了数据库 springboot，所以我也应该覆盖这个默认值。在 Spring Boot 的第2版中，与 Spring Boot Actuator 端点相关的所有配置属性都已移至 management .* 部分。

```
management:
  metrics:
    export:
      influx:
        db: springboot
        uri: http://192.168.99.100:8086
```

在使用类路径中包含的执行器启动 Spring Boot 应用程序后，您可能会感到惊讶，它默认只显示两个HTTP端点/执行器/信息和/执行器/运行状况。这就是为什么在最新版本的 Spring Boot 中，出于安全目的，默认情况下禁用除 / health 和 / info 之外的所有执行器。要启用所有执行器连接点，必须将属性 management . endpoints . web . exposure . include 设置为'*'。

在最新版本的 Spring Boot 中， HTTP 指标的监控得到了显着改善。我们可以通过将属性 management . metrics . web . server . auto – time – requests 设置为 true 来启用收集所有 Spring MVC 指标。或者，当它设置为 false 时，您可以通过使用 @Timed 对其进行注释来启用特定REST控制器的度量标准。您还可以在控制器内注释单个方法，以仅为特定端点生成度量。

应用程序启动后，您可以通过调用端点 GET / actuator / metrics 来查看生成的指标的完整列表。默认情况下， Spring MVC 控制器的度量标准以名称 http . server . requests 生成。可以通过设置 management . metrics . web . server . requests – metric – name 属性来自定义此名称。如果您运行我的 GitHub 存储库中可用的示例应用程序，则默认情况下可以使用uder端口 2222.现在，您可以通过调用端点 GET / actuator / metrics / { requiredMetricName } 来查看为单个度量标准生成的统计信息列表，如下图所示

构建 Spring Boot 应用程序

用于生成度量的示例 Spring Boot 应用程序由单个控制器组成，该控制器实现用于操作 Person 实体，存储库 bean 和实体类的基本 CRUD 操作。应用程序使用提供 CRUD 实现的 Spring Data JPA 存储库连接到 MySQL 数据库。这是控制器类。

```
@RestController
@Timed
public class PersonController {

    protected Logger logger = Logger.getLogger(PersonController.class.getName());

    @Autowired
    PersonRepository repository;

    @GetMapping("/persons/pesel/{pesel}")
    public List findByPesel(@PathVariable("pesel") String pesel) {
        logger.info(String.format("Person.findByPesel(%s)", pesel));
        return repository.findByPesel(pesel);
    }
```

```
    @GetMapping("/persons/{id}")
    public Person findById(@PathVariable("id") Integer id) {
        logger.info(String.format("Person.findById(%d)", id));
        return repository.findById(id).get();
    }

    @GetMapping("/persons")
    public List findAll() {
        logger.info(String.format("Person.findAll()"));
        return (List) repository.findAll();
    }

    @PostMapping("/persons")
    public Person add(@RequestBody Person person) {
        logger.info(String.format("Person.add(%s)", person));
        return repository.save(person);
    }

    @PutMapping("/persons")
    public Person update(@RequestBody Person person) {
        logger.info(String.format("Person.update(%s)", person));
        return repository.save(person);
    }

    @DeleteMapping("/persons/{id}")
    public void remove(@PathVariable("id") Integer id) {
        logger.info(String.format("Person.remove(%d)", id));
        repository.deleteById(id);
    }

}
```

在运行应用程序之前，我们设置了 MySQL 数据库。实现它的最方便的方法是通过 MySQL Docker 镜像。这是使用数据库 grafana 运行容器的命令，定义用户和密码，并在端口 33306 上公开 MySQL 5。

```
docker run -d --name mysql -e MYSQL_DATABASE=grafana -e MYSQL_USER=grafana -e MYSQ
```

然后我们需要在应用程序端设置一些数据库配置属性。所有必需的表都将在应用程序启动时创建，这要归功于设置属性 spring.jpa.properties.hibernate.hbm2ddl.auto 进行更新。

```
spring:
  datasource:
    url: jdbc:mysql://192.168.99.100:33306/grafana?useSSL=false
    username: grafana
    password: grafana
    driverClassName: com.mysql.jdbc.Driver
  jpa:
    properties:
```

```
    hibernate:
      dialect: org.hibernate.dialect.MySQL5Dialect
      hbm2ddl.auto: update
```

> 生成指标

在启动应用程序和所需的 Docker 容器之后，唯一需要做的就是生成一些测试统计信息。我创建了 JUnit 测试类，它生成一些测试数据并在循环中调用应用程序公开的端点。这是该测试方法的片段。

```
int ix = new Random().nextInt(100000);
Person p = new Person();
p.setFirstName("Jan" + ix);
p.setLastName("Testowy" + ix);
p.setPesel(new DecimalFormat("0000000").format(ix) + new DecimalFormat("000").form
p.setAge(ix%100);
p = template.postForObject("http://localhost:2222/persons", p, Person.class);
LOGGER.info("New person: {}", p);

p = template.getForObject("http://localhost:2222/persons/{id}", Person.class, p.ge
p.setAge(ix%100);
template.put("http://localhost:2222/persons", p);
LOGGER.info("Person updated: {} with age={}", p, ix%100);

template.delete("http://localhost:2222/persons/{id}", p.getId());
```

现在，让我们回到第1步。您可能还记得，我已经向您展示了如何在 InfluxDB Docker 容器中运行涌入客户端。经过几分钟的工作后，测试单元应多次调用暴露的端点。我们可以查看 Influx 上存储的度量标准 http_server_requests 的值。以下查询返回最近3分钟内收集的测量值列表。

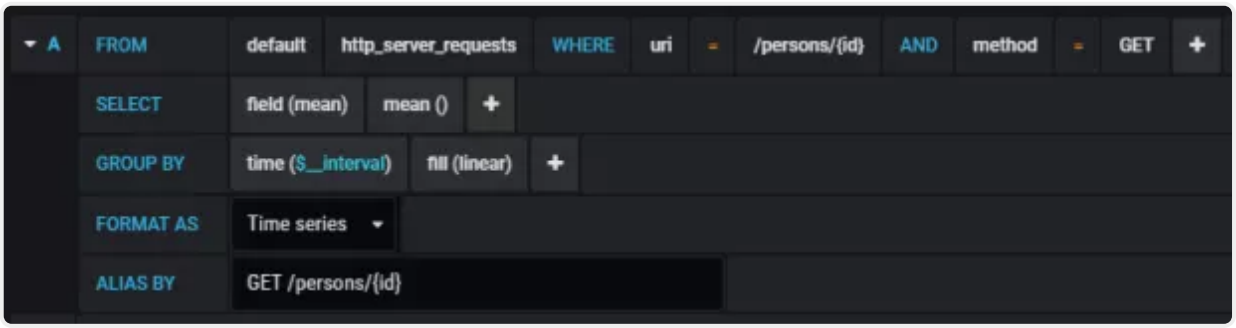
```
> select * from http_server_requests where time > now()-3m
name: http_server_requests
time                count exception mean      method metric_type status sum         upper      uri
----                ----- --------- ----      ------ ----------- ------ ---         -----      ---
1525943080593000000 372   None      6.372438  DELETE histogram   200    2370.547017 63.658014  /persons/{id}
1525943080594000000 372   None      8.338496  GET    histogram   200    3101.920406 60.845445  /persons/{id}
1525943080607000000 373   None      11.766607 POST   histogram   200    4388.944557 155.269215 /persons
1525943080623000000 372   None      5.699041  PUT    histogram   200    2120.043211 37.652505  /persons
1525943140596000000 367   None      6.020041  DELETE histogram   200    2209.355028 40.377836  /persons/{id}
1525943140597000000 367   None      7.669823  GET    histogram   200    2814.825178 58.474202  /persons/{id}
1525943140610000000 366   None      11.521371 POST   histogram   200    4216.82166  297.517716 /persons
1525943140617000000 367   None      5.364256  PUT    histogram   200    1968.682011 37.652505  /persons
1525943200602000000 362   None      6.481632  DELETE histogram   200    2346.350624 157.083865 /persons/{id}
1525943200603000000 362   None      9.467898  GET    histogram   200    3427.37907  376.283863 /persons/{id}
1525943200607000000 362   None      13.024394 POST   histogram   200    4732.930655 297.517716 /persons
1525943200610000000 362   None      5.847097  PUT    histogram   200    2116.649177 52.520634  /persons
```

如您所见， Spring Boot Actuator 生成的所有指标都标有以下信息： method, uri, status 和 exception。由于这些标签，我们可以轻松地为每个信号端点分组指标，包括失败和成功百分比。我们来看看如何在 Grafana 中配置和查看它。

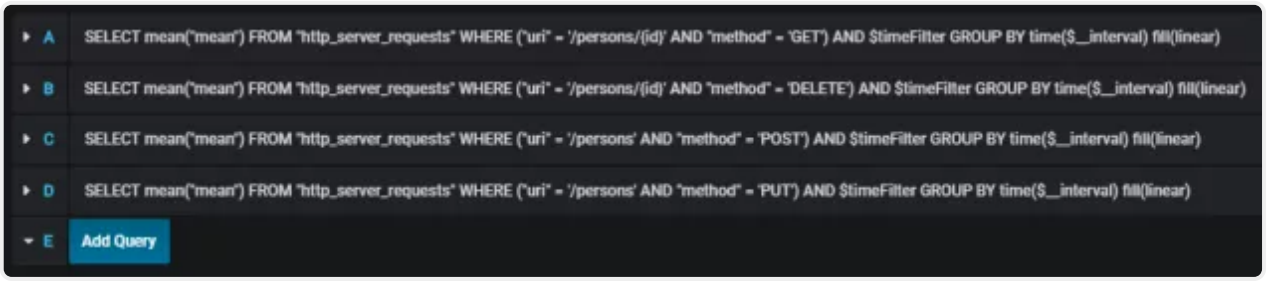
> 使用 Grafana 进行度量标准可视化

一旦我们将成功的指标导出到 InfluxDB，就可以使用 Grafana将它们可视化了。首先，让我们用 Grafana运行 Docker容器。 $ docker run – d –– name grafana – p 3000 : 3000 grafana / grafana

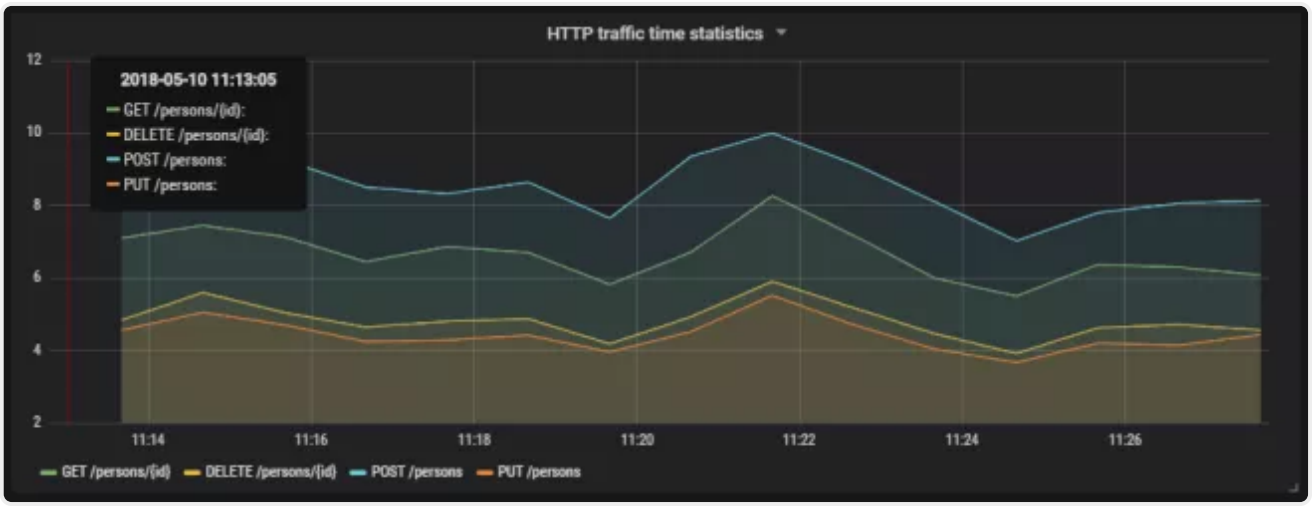
Grafana为用户提供了用于创建大量涌入查询的界面。我们定义了一个图形，可视化每个呼叫端点的请求处理时间和应用程序接收的请求总数。如果我们按方法类型和 uri 过滤存储在表 http_server_requests 中的统计信息，我们将收集每个端点生成的所有度量标准。



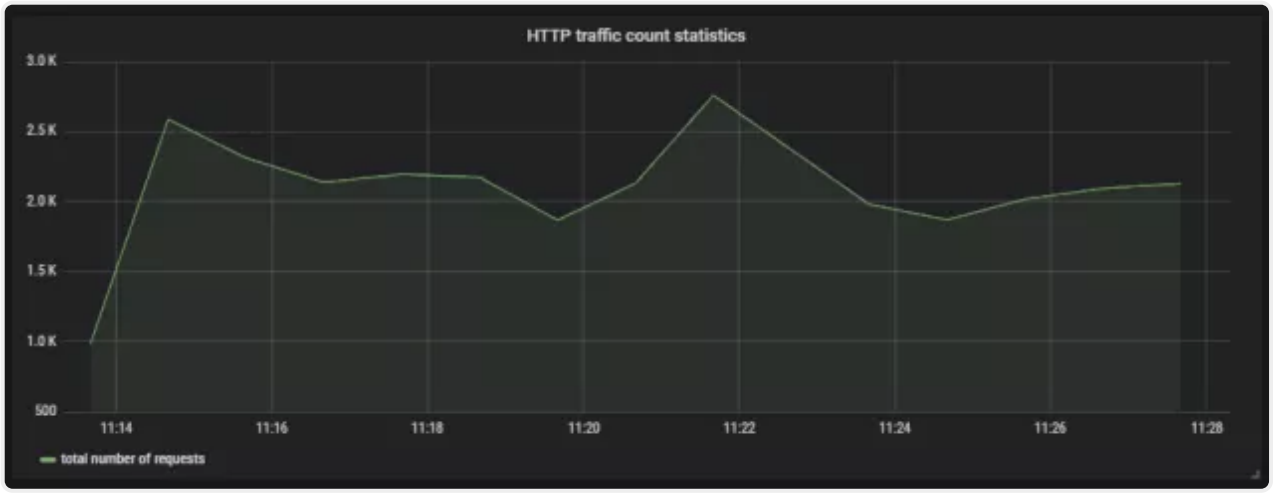
应为其他端点创建类似的定义。我们将在一张图上说明它们。



这是最终的结果。



这是可视化发送到应用程序的请求总数的图表。



> 运行 Prometheus

在本地运行 Prometheus 最合适的方法显然是通过 Docker 容器。 API在端口 9090 下公开。我们还应该传递初始配置文件和 Docker 网络的名称。为什么？ 您将在本步骤说明的下一部分找到所有的答案。

```
docker run -d --name prometheus -p 9090:9090 -v /tmp/prometheus.yml:/etc/prometheu
```

与 InfluxDB 相比， Prometheus 从应用程序中提取指标。因此，我们需要启用公开 Prometheus 指标的执行器端点，默认情况下禁用该指标。要启用它，请将property management . endpoint . prometheus . enabled 设置为 true ，如下面的配置片段所示。

```
management:
 endpoint:
   prometheus:
     enabled: true
```

然后我们应该在 Prometheus 配置文件中设置应用程序公开的执行器端点的地址。 scrape_config 部分负责指定一组目标和参数，描述如何与它们连接。默认情况下， Prometheus 会尝试每分钟从定义的目标端点收集数据。

```
scrape_configs:
  - job_name: 'springboot'
    metrics_path: '/actuator/prometheus'
    static_configs:
    - targets: ['person-service:2222']
```

与 InfluxDB 的集成类似，我们需要将以下工件包含在项目的依赖项中。

```
<dependency>
```

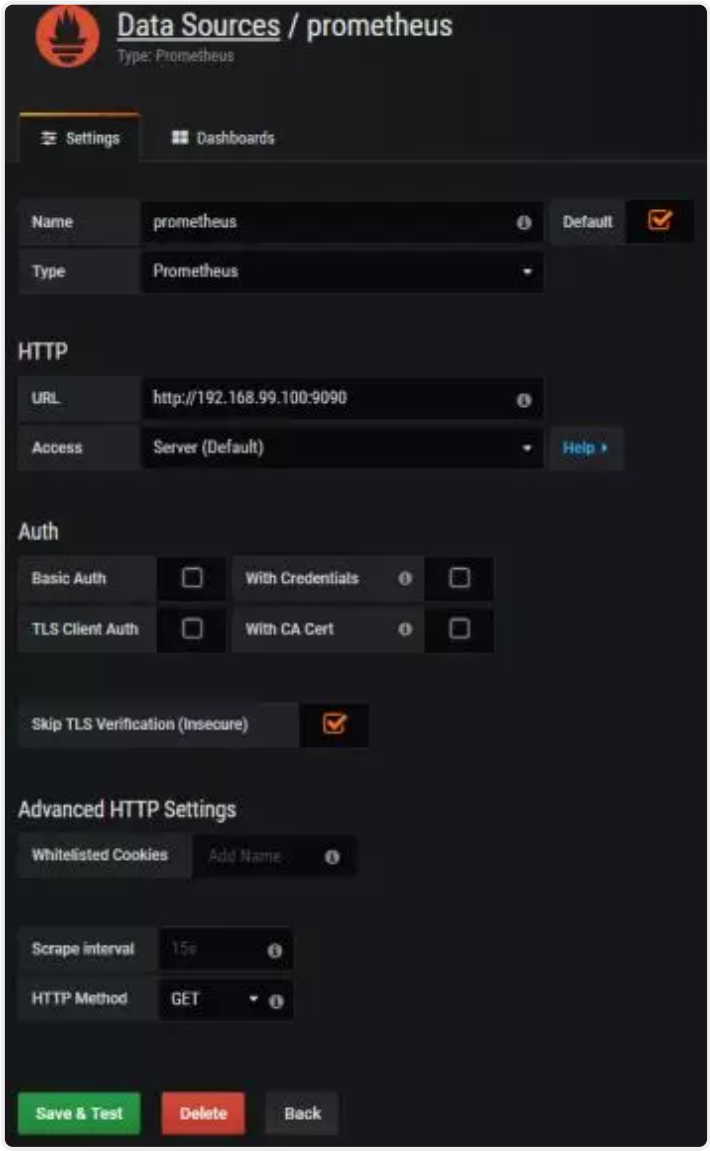
```
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

在我的例子中， Docker在 VM上运行，并且在IP 192.168 . 99.100下可用。如果我想要作为 Docker容器启动的 Prometheus能够连接我的应用程序，我也应该将它作为 Docker容器启动。链接两个独立容器的最方便方法是通过 Docker网络。如果两个容器都分配到同一网络，则它们可以使用容器的名称作为目标地址相互连接。 Dockerfile位于示例应用程序源代码的根目录中。下面显示的第二个命令（ docker build）不是必需的，因为我的 Docker Hub存储库中提供了所需的图像 piomin / person - service

```
$ docker network create springboot
$ docker build -t piomin/person-service .
$ docker run -d --name person-service -p 2222:2222 --network springboot piomin/per
```

> 将 Prometheus整合进Grafana

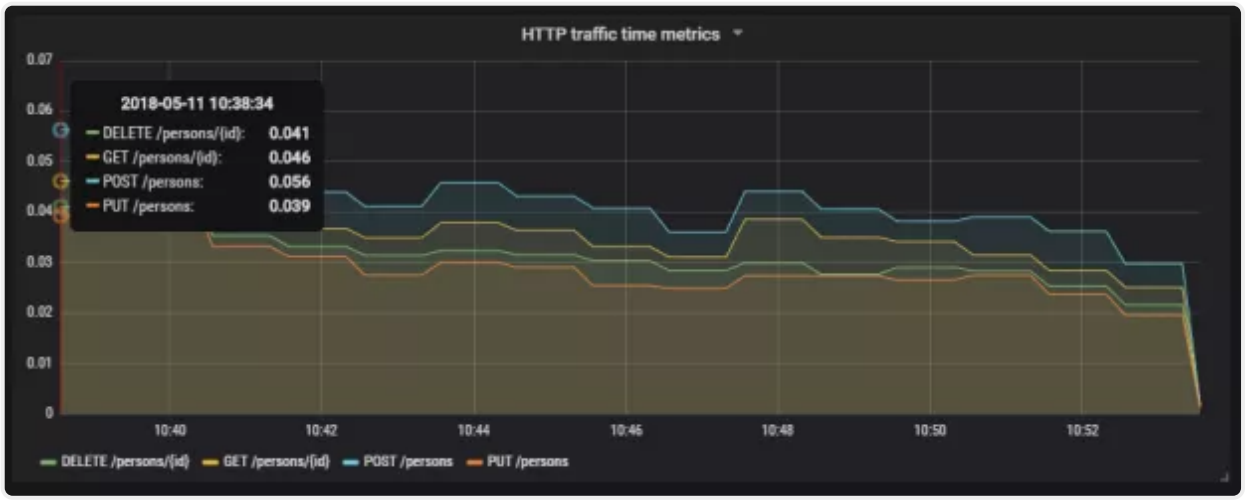
Prometheus在地址 192.168 . 99.100 : 9090下公开 Web控制台，您可以在其中指定带有指标的查询和显示图形。但是，我们可以将它与 Grafana集成，以利用此工具提供的更好的可视



化。首先，您应该创建 Prometheus 数据源。

然后我们应该定义从 Prometheus API 收集指标的查询。 Spring Boot Actuator 公开了与 HTTP 流量相关的三种不同指标： http_server_requests_seconds_count，http_server_requests_seconds_sum 和 http_server_requests_seconds_max。例如，我们可以计算 http_server_requests_seconds_sum 的时间序列的每秒平均增长率，它返回使用 rate() 函数处理请求所花费的总秒数。可以使用方法和 uri 使用 {} 内的表达式过滤这些值。下图说明了每个端点的rate（）函数配置。

这是图表。



> 总结

Spring Boot 版本 1.5 和 2.0 之间的度量标准生成的改进非常重要。将数据导出到诸如 InfluxDB 或 Prometheus 之类的流行监控系统现在比以前容易得多，并且不需要任何额外的开发。由于标签指示了 HTTP 请求的uri，类型和状态，因此与HTTP流量相关的指标更加详细，并且可以轻松地与特定端点关联。我认为 Spring Boot Actuator 中与 Spring Boot 的早期版本相关的修改可能是将应用程序迁移到最新版本的主要动机之一。

> 原文链接：https://piotrminkowski.wordpress.com/2018/05/11/exporting-metrics-to-influxdb-and-prometheus-using-spring-boot-actuator/
>
> 作者：piotr
>
> 译者：complone

推荐： SpringBoot WebFlux 入门案例

上一篇：Spring Boot微服务中Chaos Monkey的应用

关注公众号