链滴

SpringCloud 系列 --2. 服务实例健康自检 SpringBoot Actuator

作者：289306290

原文链接：https://ld246.com/article/1583837309532

来源网站：链滴

说明:

在默认情况下,Eureka的客户端每隔30秒会发送一次心跳给服务端,告知仍然存活,但是一些情况下(比数据库挂了),客户端表面上可以正常发送心跳，但实际上无法提供服务。

这时可以利用Eureka的健康检查控制器(哪个模块对外提供服务需要自检,在哪个模块实现)。

需要在pom.xml中引入

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-actuator</artifactId>
        <version>1.5.3.RELEASE</version>
    </dependency>
```

SpringBoot Actuator

第一步: 可以实现一个自定义的HealthIndicator 来根据是能访问数据库决定自身的健康状态,

```
package org.crazyit.cloud;

import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.Status;
import org.springframework.boot.actuate.health.Health.Builder;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

/**
 * 健康指示器
 * @author 杨恩雄
 *
 */
@Component
public class MyHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
        if(HealthController.canVisitDb) {
            // 成功连接数据库，返回UP
            return new Health.Builder(Status.UP).build();
        } else {
            // 连接数据库失败，返回 out of service
            return new Health.Builder(Status.DOWN).build();
        }
    }
}
```

模拟数据库状态用一个控制器来控制HealthController中的静态变量canVisitDb

```
package org.crazyit.cloud;

import javax.servlet.http.HttpServletRequest;
```

```
import org.springframework.http.MediaType;
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.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HealthController {

    // 标识当前数据库是否可以访问
    static Boolean canVisitDb = false;

    @RequestMapping(value = "/db/{canVisitDb}", method = RequestMethod.GET)
    @ResponseBody
    public String setConnectState(@PathVariable("canVisitDb") Boolean canVisitDb) {
        this.canVisitDb = canVisitDb;
        return "当前数据库是否正常: " + this.canVisitDb;
    }
}
```

第二步: 服务提供者把自身健康状态告知服务器.

实现"健康检查处理器", 将应用的健康状态保存在内存中,状态一旦发生改变,就会重新向服务器进行注，其他的客户端将拿不到这些不可用的实例。

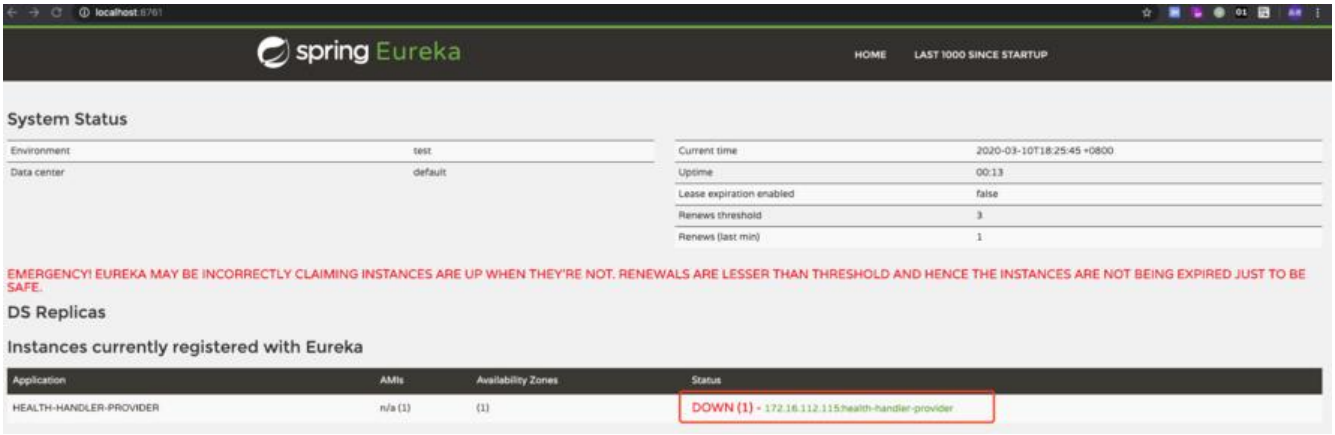
```
package org.crazyit.cloud;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.actuate.health.Status;
import org.springframework.stereotype.Component;

import com.netflix.appinfo.HealthCheckHandler;
import com.netflix.appinfo.InstanceInfo.InstanceStatus;

/**
 * 健康检查处理器
 * @author 杨恩雄
 *
 */
@Component
public class MyHealthCheckHandler implements HealthCheckHandler {

    @Autowired
    private MyHealthIndicator indicator;

    public InstanceStatus getStatus(InstanceStatus currentStatus) {
            System.out.println("Eureka定时器调用了哈，每隔eureka.client.instanceInfoReplication
ntervalSeconds时间调用一次 。。。。。。。");
        Status s = indicator.health().getStatus();
        if(s.equals(Status.UP)) {
            System.out.println("数据库正常连接");
```

```
            return InstanceStatus.UP;
        } else {
            System.out.println("数据库无法连接");
            return InstanceStatus.DOWN;
        }
    }
}
```

上述代码根据健康指示器结果来返回不通的状态。 Eureka会启动一个定时器,定时刷新本地实例的信息并且执行"处理器"中的getStatus方法,再将服务实例状态"更新"到服务器中, 默认定时器每隔30秒执行次,可以通过修改配置 eureka.client.instanceInfoReplicationIntervalSeconds 来改变，如下:

```
spring:
  application:
    name: health-handler-provider
eureka:
  instance:
    hostname: localhost
  client:
    healthcheck:
      enabled: true
    instanceInfoReplicationIntervalSeconds: 10
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

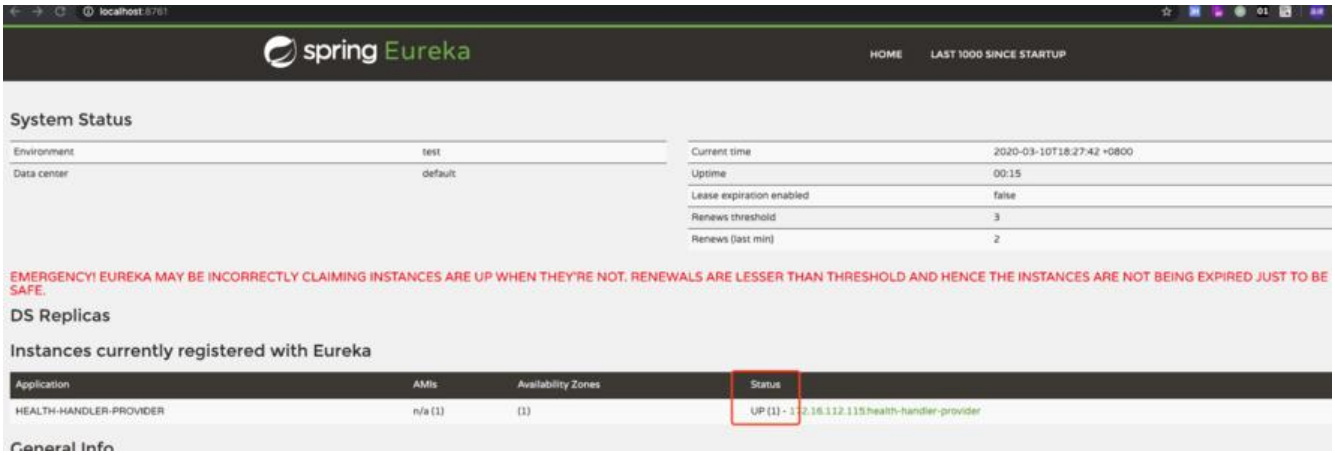
因为默认数据库canVisitDb = false;所有服务默认是 DOWN的状态



访问 http://localhost:8080/db/true 将数据库状态设置为true

当前数据库是否正常: true

再看状态就是UP



第三步 服务查询

查看集群中的服务,可以使用SpringCloud的 discoveryClient类, 或者 eurekaClient类,SpringCloud Eureka进行了封装。本例中使用 discoveryClient 的方法来查询服务实例，输出服务实例的状态等信
。

```
package org.crazyit.cloud;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.cloud.netflix.eureka.EurekaDiscoveryClient.EurekaServiceInstance;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

import com.netflix.appinfo.InstanceInfo;

@RestController
@Configuration
public class InvokerController {

    @Autowired
    private DiscoveryClient discoveryClient;

    @RequestMapping(value = "/router", method = RequestMethod.GET)
    @ResponseBody
    public String router() {
        // 查找服务列表
        List<ServiceInstance> ins = getServiceInstances();
        // 输出服务信息及状态
```

```
        for (ServiceInstance service : ins) {
            EurekaServiceInstance esi = (EurekaServiceInstance) service;
            InstanceInfo info = esi.getInstanceInfo();
            System.out.println(info.getAppName() + "---" + info.getInstanceId()
                    + "---" + info.getStatus());
        }
        return "";
    }

    /**
     * 查询可用服务
     */
    private List<ServiceInstance> getServiceInstances() {
        List<String> ids = discoveryClient.getServices();
        List<ServiceInstance> result = new ArrayList<ServiceInstance>();
        for (String id : ids) {
            List<ServiceInstance> ins = discoveryClient.getInstances(id);
            result.addAll(ins);
        }
        return result;
    }
}
```

1.访问服务 http://localhost:9000/router, 输出如下(说明此时db正常):

HEALTH-HANDLER-PROVIDER---172.16.112.115:health-handler-provider---UP

HEALTH-HANDLER-INVOKER---172.16.112.115:health-handler-invoker:9000---UP

2. 访问http://localhost:8080/db/false 将db设置为false

3.再访问http://localhost:9000/router 服务 输出如下:

HEALTH-HANDLER-INVOKER---172.16.112.115:health-handler-invoker:9000---UP

说明,可用的服务只剩下调用者自己了，服务的提供者已经不存在于服务列表中.

综上,

想要服务健康自检，需要两步（**实测,仅需要第二步实现 HealthCheckHandler**）：

1.实现 HealthIndicator 来返回服务的状态。

```
package org.crazyit.cloud;

import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.Status;
import org.springframework.boot.actuate.health.Health.Builder;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

/**
 * 健康指示器
 * @author 杨恩雄
 *
 */
```

```
@Component
public class MyHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
        if(HealthController.canVisitDb) {
            // 成功连接数据库，返回UP
            return new Health.Builder(Status.UP).build();
        } else {
            // 连接数据库失败，返回 out of service
            return new Health.Builder(Status.DOWN).build();
        }
    }
}
```

2. 实现 HealthCheckHandler 来让Eureka调用获取getStatus方法来获取状态并通知服务端

```
package org.crazyit.cloud;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.actuate.health.Status;
import org.springframework.stereotype.Component;

import com.netflix.appinfo.HealthCheckHandler;
import com.netflix.appinfo.InstanceInfo.InstanceStatus;

/**
 * 健康检查处理器
 * @author 杨恩雄
 *
 */
@Component
public class MyHealthCheckHandler implements HealthCheckHandler {

    @Autowired
    private MyHealthIndicator indicator;

    public InstanceStatus getStatus(InstanceStatus currentStatus) {
        System.out.println("Eureka定时器调用了哈，每隔eureka.client.instanceInfoReplicationInter
alSeconds时间调用一次 。。。。。。。");
        Status s = indicator.health().getStatus();
        if(s.equals(Status.UP)) {
            System.out.println("数据库正常连接");
            return InstanceStatus.UP;
        } else {
            System.out.println("数据库无法连接");
            return InstanceStatus.DOWN;
        }
    }
}
```