链滴

spring-boot 自定义 endpoint，让运行时的 java 应用不再是黑盒

作者：crick77

原文链接：https://ld246.com/article/1511661974948

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

本文项目已发布到github，后续学习项目也会添加到此工程下，欢迎fork点赞。
https://github.com/wangyuheng/spring-boot-sample

你的java应用在运行时对你来说是黑盒吗？你可以查看到springboot运行时的各种信息吗？

Spring Boot Actuator

springboot提供了用于健康检测的endpoint，提供了查看系统信息、内存、环境等。

1. 添加依赖

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

2. 访问链接

应用启动后访问 http://localhost:8080/health 查看应用启动状态。
端口号可以通过配置文件management.port=9527变更

常用endpoint

1. http://localhost:8080/env 环境变量
2. http://localhost:8080/info 应用信息
3. http://localhost:8080/metrics 内存等应用基本指标
4. http://localhost:8080/dump 线程栈
5. http://localhost:8080/configprops 配置项

3. 开启全部endpoint & 关闭验权

部分请求会返回**401**， 这是因为endpoint未开启，或者开启了登录验权，可以通过配置文件进行配置

```
management.security.enabled=false
endpoints.enabled=true
```

自定义endpoint

一、实现Endpoint接口

```
public interface Endpoint<T> {
    String getId();

    boolean isEnabled();

    boolean isSensitive();
```

```
    T invoke();
}
```

1. getId()， 指定了endpoint访问url
2. isEnabled()， 表示是否启用
3. isSensitive()， 表示是否验权
4. invoke()， 页面返回值

实现类通过@Bean的形式注入后，再次启动应用，即可通过url访问，并返回invoke返回值。

```
public class CustomEndpoint implements Endpoint {

    @Override
    public String getId() {
        return "custom";
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public boolean isSensitive() {
        return false;
    }

    @Override
    public Object invoke() {
        return "hello endpoint";
    }
}

@SpringBootApplication
public class EndpointApplication {

    public static void main(String[] args) {
        SpringApplication.run(EndpointApplication.class, args);
    }


    @Bean
    public CustomEndpoint customEndpoint() {
        return new CustomEndpoint();
    }
}
```

访问http://localhost:8080/custom 可以看到invoke返回的内容。

但是这样，每个endpoint都需要单独注入，且没有层级、通配符，不方便管理，为满足需求，尝试做如下改造

二、继承EndpointMvcAdapter

2.1 自定义 EndpointAction 接口

用于定制endpoint处理行为，可以理解为invoke方法的具体实施者，名称用来管理访问路径

```
public interface EndpointAction extends Serializable {
    Object execute();

    String getName();
}
```

2.2 EndpointAction的多种实现

需要实现的功能，如：读取配置文件、查看内存信息

```
@Component
public class PropertiesAction implements EndpointAction {

    @Override
    public Object execute() {
        try {
            return PropertiesLoaderUtils.loadAllProperties("application.properties");
        } catch (IOException e) {
            return "read application fail! error: " + e.getMessage();
        }
    }

    @Override
    public String getName() {
        return "properties";
    }
}

@Component
public class VMAction implements EndpointAction {

    private static final VMAction INSTANCE = new VMAction();

    private String version;
    private String startTime;
    private String initHeap;
    private String maxHeap;
    private Set<String> arguments;

    @Override
    public Object execute() {
        INSTANCE.version = System.getProperty("java.runtime.version");
        INSTANCE.startTime = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(Man
gementFactory.getRuntimeMXBean().getStartTime());
        INSTANCE.initHeap = String.valueOf(ManagementFactory.getMemoryMXBean().getHe
pMemoryUsage().getInit() / 1024 / 1024).concat("MB");
        INSTANCE.maxHeap = String.valueOf(ManagementFactory.getMemoryMXBean().getH
apMemoryUsage().getMax() / 1024 / 1024).concat("MB");
        INSTANCE.arguments = new HashSet<>(ManagementFactory.getRuntimeMXBean().g
```

```
tInputArguments());
            return INSTANCE;
        }

        @Override
        public String getName() {
            return "vm";
        }

        public String getVersion() {
            return version;
        }

        public String getStartTime() {
            return startTime;
        }

        public String getInitHeap() {
            return initHeap;
        }

        public String getMaxHeap() {
            return maxHeap;
        }

        public Set<String> getArguments() {
            return arguments;
        }
    }

    @Component
    public class DefaultAction implements EndpointAction {


        private static final DefaultAction INSTANCE = new DefaultAction();

        public static DefaultAction getInstance() {
            return INSTANCE;
        }

        @Override
        public Object execute() {
            return "try /help for action list";
        }

        @Override
        public String getName() {
            return "default";
        }

    }
```

2.3 继承EndpointMvcAdapter

1. 注入action map，根据name获取bean实现
2. 通过url mapping匹配action

```
public class CustomEndpointAdapter extends EndpointMvcAdapter {


    private Map<String, EndpointAction> endpointActionMap = new HashMap<>();

    @Autowired
    public void setEndpointActionMap(List<EndpointAction> endpointActionList) {
        endpointActionList.forEach(endpointAction -> endpointActionMap.put(endpointActio
.getName(), endpointAction));
    }

    public CustomEndpointAdapter() {
        super(new CustomEndpoint());
    }

    @RequestMapping(value = "/{name:.*}",
            method = RequestMethod.GET, produces = {
            ActuatorMediaTypes.APPLICATION_ACTUATOR_V1_JSON_VALUE,
            MediaType.APPLICATION_JSON_VALUE}
    )
    @ResponseBody
    @HypermediaDisabled
    public Object dispatcher(@PathVariable String name) {
        if ("help".equalsIgnoreCase(name)) {
            return endpointActionMap.keySet().stream().map(key -> getName() + "/" + key).col
ect(toSet());
        } else {
            return endpointActionMap.getOrDefault(name, DefaultAction.getInstance()).execute
);
        }
    }

}
```

2.4 定制endpoint作为入口

```
public class CustomEndpoint implements Endpoint {

    @Override
    public String getId() {
        return "custom";
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public boolean isSensitive() {
```

```
        return false;
    }

    @Override
    public Object invoke() {
        return DefaultAction.getInstance().execute();
    }
}
```

2.5 启动时注入

```
@SpringBootApplication
public class EndpointApplication {

    public static void main(String[] args) {
        SpringApplication.run(EndpointApplication.class, args);
    }


    @Bean
    @ConditionalOnMissingBean
    @ConditionalOnClass(CustomEndpoint.class)
    public CustomEndpointAdapter customEndpointAdapter() {
        return new CustomEndpointAdapter();
    }
}
```

测试

揪心的测试环节，启动webEnvironment环境，通过**TestRestTemplate**访问endpoint的mapping比对返回值即可

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class PropertiesActionTest {

    private TestRestTemplate restTemplate;

    @Value("${management.port}")
    private String managementPort;

    @Before
    public void setupMockMvc() {
        restTemplate = new TestRestTemplate();
    }

    @Test
    public void test_properties_action() throws Exception {
        String path = "http://localhost:" + managementPort + "/custom/properties";
        Map<String, String> result = restTemplate.getForObject(path, HashMap.class);
        assertEquals(result.get("management.port"), managementPort);
    }
```

```
    @Test
    public void test_help() throws Exception {
        String path = "http://localhost:" + managementPort + "/custom/help";
        Set<String> result = restTemplate.getForObject(path, Set.class);
        assertTrue(result.contains("custom/properties"));
    }

    @Test
    public void test_rand() throws Exception {
        String path = "http://localhost:" + managementPort + "/custom/" + new Random().ne
tInt();
        String result = restTemplate.getForObject(path, String.class);
        assertEquals("try /help for action list", result);

    }

}
```

原文链接：spring-boot 自定义 endpoint，让运行时的 java 应用不再是黑盒