

8.1.1 三层微服务源码分析

为了方面我们能深入理解 Istio 的功能，我们对这套应用的源码进行分析。由于篇幅有限，在源码分析时，仅列出关键的方法，完整的代码书中会提供链接。

1. Customer 微服务源码分析

Custmer 微服务是基于 Springboot 实现的。源码地址为：

<https://github.com/ocp-msa-devops/istio-tutorial/blob/master/customer/java/springboot/src/main/java/com/redhat/developer/demos/customer>。在该目录下，有两个源码文件，如下图 8-48 所示：






This branch is even with redhat-developer-demos:master.		 Pull request	 Compare
	lordofthejars Adds Zero Downtime with database example (redhat-developer-demos#177)		Latest commit 1626c96 on 8 Jan
..			
	CustomerApplication.java	Changed folder structure	a year ago
	CustomerController.java	Adds Zero Downtime with database example (redhat-developer-demos#177)	4 months ago

图 8-48 应用源码文件

第一个文件 CustomerApplication.java 定义了一个 main 函数，用于 Springboot 加载时使用，里面没有业务逻辑，我们不进行讨论。我们主要分析 CustomerController.java，代码如下：

```
@RestController

public class CustomerController {

    private static final String RESPONSE_STRING_FORMAT = "customer
=> %s\n";

    private final Logger logger =
LoggerFactory.getLogger(getClass());

    private final RestTemplate restTemplate;
```

```

    @Value("${preferences.api.url:http://preference:8080}")

    private String remoteURL;


    @RequestMapping(value = "/", method = RequestMethod.POST,
consumes = "text/plain")

    public ResponseEntity<String> addRecommendation(@RequestBody
String body) {

        try {

            return restTemplate.postForEntity(remoteURL, body,
String.class);

        } catch (HttpStatusCodeException ex) {

            logger.warn("Exception trying to post to preference
service.", ex);

            return
ResponseEntity.status(HttpStatus.SERVICE_UNAVAILABLE)

                .body(String.format("%d %s", ex.getRawStatusCode(),

    @RequestMapping(value = "/", method = RequestMethod.GET)


    ResponseEntity<String> responseEntity =
restTemplate.getForEntity(remoteURL, String.class);

    String response = responseEntity.getBody();

    return
ResponseEntity.ok(String.format(RESPONSE_STRING_FORMAT,
response.trim()));

```

在 Springboot 中:

- 使用 `@RestController` 注解来处理 http 请求。`RestController` 使用的效果是将方法返

回的对象直接在浏览器上展示成 json 格式。

- 如果只是使用 `@RestController` 注解 Controller 类，则方法无法返回 jsp 页面。则需要对应的方法上加上 `@ResponseBody` 注解。
- `@RequestMapping` 是 Spring Web 应用程序中最常被用到的注解之一。这个注解会将 HTTP 请求映射到 MVC 和 REST 控制器的处理方法上。

从上面的 `CustomerController.java` 源码片段中，我们可以得出如下结论：

- 本源码文件声明了对 http 请求的相应，并且返回值是 `customer => %s`
- `customer` 微服务响应请求的 URL 是 /，可以响应 POST 和 GET。
- 当请求时 POST 的时候，`customer` 微服务将传递新来的内容追加到 `remoteURL` 定义的微服务。
- 定请求是 GET 时，本微服务会调用 `remoteURL` 定义的微服务，然后将获取到的信息的 body 返回到前台。也就是 %s 的内容
- 源码中定义了本微服务的 `remoteURL` 是 `http://preference:8080`。

2. preference 微服务源码分析

`preference` 微服务也是基于 Spingboot 运行的。在 `ocp-msa-devops/istio-tutorial/tree/master/preference/java/springboot/src/main/java/com/redhat/developer/demos/preference` 目录下，有两个源码文件，如下图 8-49 所示：

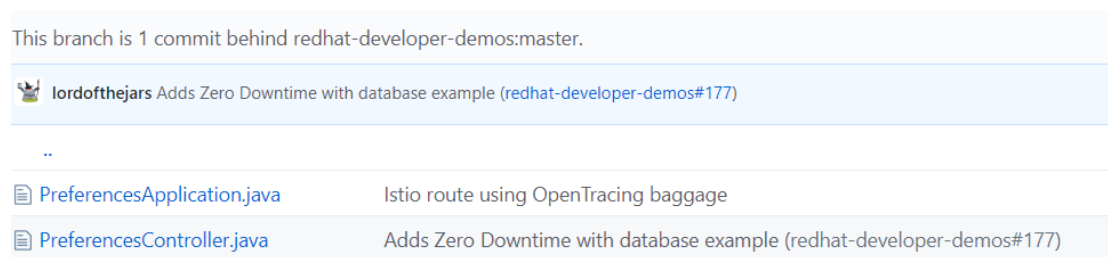


图 8-49 应用源码文件

与 `customer` 源码类似，`PreferencesApplication.java` 主要是 Spingboot 启动的配置，不包含业务逻辑。我们主要看 `PreferencesController.java`，内容如下：

```
@RestController

public class PreferencesController {
```

```

        private static final String RESPONSE_STRING_FORMAT = "preference
=> %s\n";

        @Value("${recommendations.api.url:http://recommendations:8080}")
        private String remoteURL;

        public PreferencesController(RestTemplate restTemplate) {

            this.restTemplate = restTemplate;

        }

        @RequestMapping(value = "/", method = RequestMethod.POST,
consumes = "text/plain")

        public ResponseEntity<String> addRecommendation(@RequestBody
String body) {

            try {

                return restTemplate.postForEntity(remoteURL, body,
String.class);

            }

        }

        @RequestMapping("/")

        public ResponseEntity<?> getPreferences() {

            try {

                ResponseEntity<String> responseEntity =
restTemplate.getForEntity(remoteURL, String.class);

                String response = responseEntity.getBody();

                return
ResponseEntity.ok(String.format(RESPONSE_STRING_FORMAT,
response.trim()));

```

从上面的源码片段中，我们可以得出如下结论：

- 本源码文件声明了对 HTTP 请求的相应，并且返回值是 `preference => %s`。
- 响应请求的 URL 是 /，可以响应 POST 和 GET。
- 当请求时 POST 的时候，本微服务将传递新来的内容追加到 `remoteURL` 定义的微服务。
- 定请求是 GET 时，本微服务会调用 `remoteURL` 定义的微服务，然后将获取到的信息的 body 返回到前台。也就是 %s 的内容。
- 源码中定义了本微服务的 `remoteURL` 是 `http://recommendations:8080`。

3. recommendation 微服务源码分析

`recommendation` 是 `vertx` 应用程序。在目录 `ocp-msa-devops/istio-tutorial/tree/master/recommendation/java/vertx/src/main/java/com/redhat/developer/demos/recommendation` 下，有两个源码文件，如下图 8-50 所示：

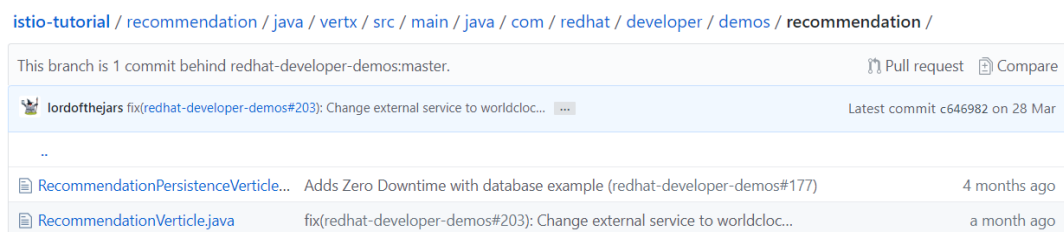


图 8-50 应用源码文件

我们在编译代码的使用，使用的是 `RecommendationVerticle.java` 源码。因此我们对此进行分析：

```
public class RecommendationVerticle extends AbstractVerticle {

    private static final String RESPONSE_STRING_FORMAT =
"recommendation v1 from '%s': %d\n";

    private static final String HOSTNAME =
parseContainerIdFromHostname(
```

```

        System.getenv().getOrDefault("HOSTNAME", "unknown")

    );

    static String parseContainerIdFromHostname(String hostname) {

        return hostname.replaceAll("recommendation-v\\d+-", "");

    }

    private int count = 0;

    /**

     * Flag for throwing a 503 when enabled

     */

    private boolean misbehave = false;

```

本源码定义的内容是：

- 当 `recommendation` 被请求时，返回内容为：`recommendation v1 from '%s': %d\n`。
- `%s` 是获取的 `docker` 运行时容器的主机名。
- `%d` 是被请求访问的计数。

除此之外，该源码中还定义了端点：`misbehave`。也就是说，当 `RoutingContext ctx` 为 `misbehave` 时，`recommendations` 微服务的返回都是 503。当 `RoutingContext` 不是 `misbehave` 时，正常返回。

```

    private void getRecommendations(RoutingContext ctx) {

        if (misbehave) {

            count = 0;

            logger.info(String.format("Misbehaving %d", count));

            ctx.response().setStatusCode(503).end(String.format("recommendation
misbehavior from '%s'\n", HOSTNAME));

        } else {

            count++;

```

```
ctx.response().end(String.format(RESPONSE_STRING_FORMAT, HOSTNAME,
count));

    }

}
```

而 RoutingContext ctx 的可以通过 webclient 进行设置。当前台设置 RoutingContext ctx 为 misbehave, 返回结果为: Following requests to '/' will return a 503。

```
private void misbehave(RoutingContext ctx) {

    this.misbehave = true;

    logger.info("'misbehave' has been set to 'true'");

    ctx.response().end("Following requests to '/' will return a
503\n");

}
```

当前台设置 RoutingContext ctx 为 misbehave, 返回结果为: Following requests to '/' will return a 503; 当前台设置当前台设置 RoutingContext ctx 不为 misbehave, 返回结果为: Following requests to '/' will return a 200。

```
private void behave(RoutingContext ctx) {

    this.misbehave = false;

    logger.info("'misbehave' has been set to 'false'");

    ctx.response().end("Following requests to '/' will return a
200\n");

}
```

Misbehave 端点在本章后面将会用到。

因此, 三个微服务的调用逻辑是: 当我们通过路由发起对 customer 微服务的请求时, customer 会调用微服务 preference; 微服务 preference 会调用 recommendation 微服务。最终显示的结果是三个微服务代码中定义的 RESPONSE 拼接起来, 即:

customer => preference => recommendation v1 from 'recommendation 容器的 id': 被调用次数。