



21天微服务实战营-Day2

华为云DevCloud & ServiceStage服务联合出品



Day2 微服务入门之编写HelloWorld

大纲

- 开发第一个微服务
- 服务契约
- 开发服务调用者

开发第一个微服务

创建一个空的maven工程，然后在pom文件中加入如下依赖：

```
<properties>
... <cse.version>2.3.62</cse.version>
... </properties>

<dependencyManagement>
... <dependencies>
... <dependency>
... <groupId>com.huawei.paas.cse</groupId>
... <artifactId>cse-dependency</artifactId>
... <version>${cse.version}</version>
... <type>pom</type>
... <scope>import</scope>
... </dependency>
... </dependencies>
... </dependencyManagement>

<dependencies>
... <dependency>
... <groupId>com.huawei.paas.cse</groupId>
... <artifactId>cse-solution-service-engine</artifactId>
... </dependency>
... </dependencies>
```

创建一个简单的CSEJavaSDK微服务只需要引入cse-solution-service-engine包，但我们仍然推荐大家使用<dependencyManagement>管理依赖依赖，这在项目依赖关系复杂时可以有效降低依赖管理复杂度。

开发第一个微服务

创建一个main类：

```
public class AppMain {  
    public static void main(String[] args) throws Exception {  
        Log4jUtils.init(); // 初始化默认的日志组件  
        BeanUtils.init(); // 加载Spring bean定义文件，正式开始启动流程  
    }  
}
```

CSEJavaSDK使用的默认日志组件是Log4J，并在此基础上进行了一些默认的配置，可以开箱即用。

创建服务的REST接口类：

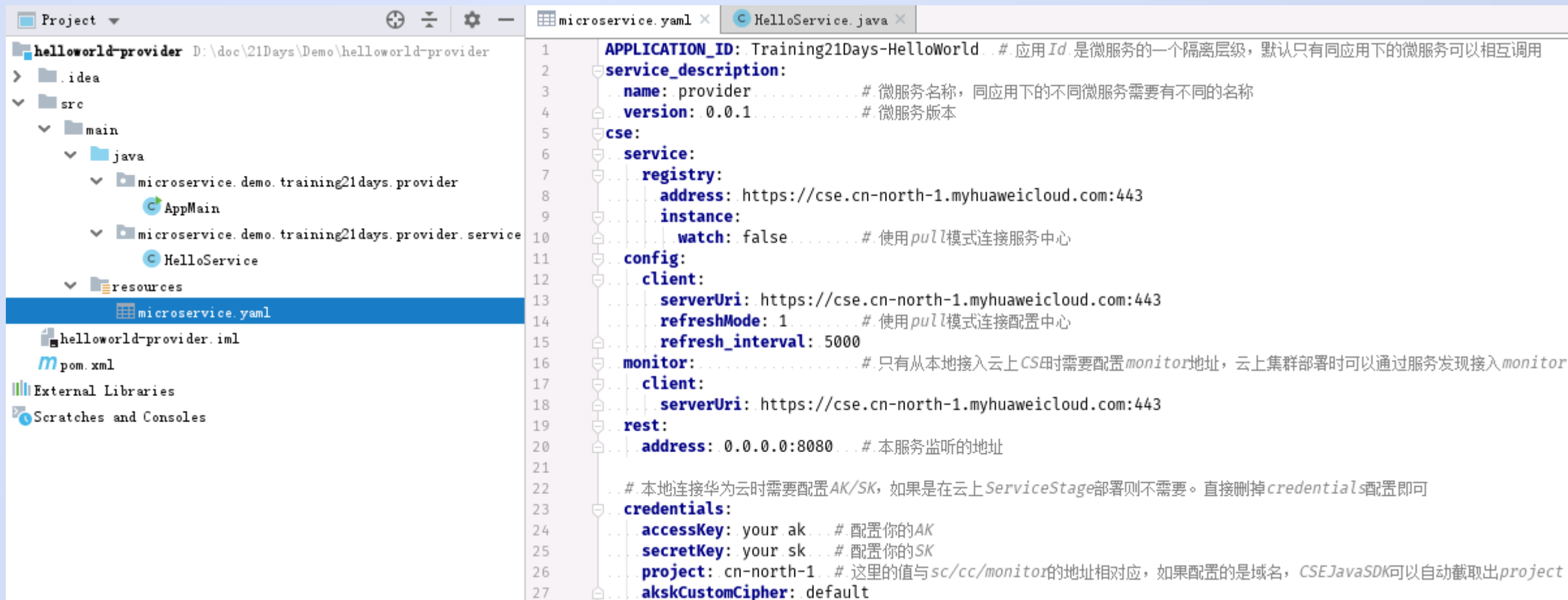
```
@RestSchema(schemaId = "hello") // 该注解声明这是一个REST接口类，CSEJavaSDK会扫描到这个类，根据它的代码生成接口契约  
@RequestMapping(path = "/provider/v0") // @RequestMapping是Spring的注解，这里在使用Spring MVC风格开发REST接口  
public class HelloService {  
    @RequestMapping(path = "/hello/{name}", method = RequestMethod.GET)  
    public String sayHello(@PathVariable(value = "name") String name) {  
        return "Hello," + name;  
    }  
}
```

一个微服务可以有多个接口契约。这里使用@RestSchema注解声明HelloService是一个契约id为hello的REST接口，同时会在启动时生成相应的契约。

这里的REST接口是以Spring MVC风格开发的。CSEJavaSDK支持的开发风格有REST([JAX-RS](#)、[Spring MVC](#))和[RPC](#)，开发者可以自由选用。

开发第一个微服务

在src\main\resources\目录下创建一份microservice.yaml文件



The screenshot displays an IDE interface with two main panels. The left panel shows the project structure for 'helloworld-provider' located at 'D:\doc\21Days\Demo\helloworld-provider'. The structure includes a 'src' directory with 'main' and 'resources' subdirectories. The 'resources' directory contains the 'microservice.yaml' file, which is currently selected. The right panel shows the content of 'microservice.yaml' with line numbers 1 through 27. The file contains configuration for a microservice, including application ID, service description, CSE configuration, registry address, instance watch, config, client, monitor, rest, and credentials.

```
1 APPLICATION_ID: Training21Days-HelloWorld... # 应用Id 是微服务的一个隔离层级，默认只有同应用下的微服务可以相互调用
2 service_description:
3   name: provider... # 微服务名称，同应用下的不同微服务需要有不同的名称
4   version: 0.0.1... # 微服务版本
5 cse:
6   service:
7     registry:
8       address: https://cse.cn-north-1.myhuaweicloud.com:443
9       instance:
10        watch: false... # 使用pull模式连接服务中心
11    config:
12      client:
13        serverUri: https://cse.cn-north-1.myhuaweicloud.com:443
14        refreshMode: 1... # 使用pull模式连接配置中心
15        refresh_interval: 5000
16    monitor: ... # 只有从本地接入云上CSE时需要配置monitor地址，云上集群部署时可以通过服务发现接入monitor
17    client:
18      serverUri: https://cse.cn-north-1.myhuaweicloud.com:443
19    rest:
20      address: 0.0.0.0:8080... # 本服务监听的地址
21
22    # 本地连接华为云时需要配置AK/SK，如果是在云上ServiceStage部署则不需要。直接删掉credentials配置即可
23    credentials:
24      accessKey: your ak... # 配置你的AK
25      secretKey: your sk... # 配置你的SK
26      project: cn-north-1... # 这里的值与sc/cc/monitor的地址相对应，如果配置的是域名，CSEJavaSDK可以自动截取出project
27      akskCustomCipher: default
```

开发第一个微服务

运行AppMain类，可以在ServiceStage的微服务控制台看到provider服务

 华为云

北京一

控制台 总览 资源管理 应用开发 应用上线 应用运维 软件中心

Q 费用 资源 工单 备案 yhs0092

76



微服务控制台
Cloud Service Engine

仪表盘

服务目录

服务治理

全局配置

事务看板

服务目录

您可以从应用、微服务和实例的维度查看微服务详细信息。如果微服务较多，您可以通过搜索功能查找目标微服务。[如何维护微服务？](#)

应用列表 微服务列表 实例列表

创建微服务

删除

Training21Days-Hello... 微服务名称

<input type="checkbox"/>	微服务名称	所属应用	版本数	实例数	创建时间	操作
<input type="checkbox"/>	provider	Training21Days-Hell...	1	1	2019/02/12 20:26:10 GMT+0...	删除

体验新版

你已经开发出第一个微服务了！

调用 <http://127.0.0.1:8080/provider/v0/hello/Bob> ，可以得到provider服务的应答

The screenshot displays a REST client interface with the following components:

- URL Bar:** Shows the URL `http://127.0.0.1:8080/provider/v0/hello/Bob`.
- Method and Path:** A dropdown menu is set to `GET`, and the path `http://127.0.0.1:8080/provider/v0/hello/Bob` is entered.
- Buttons:** `Send` (blue) and `Save` (grey) buttons are present.
- Tabs:** `Params`, `Authorization`, `Headers`, `Body`, `Pre-request Script`, and `Tests` are visible. `Params` is the active tab.
- Params Table:**

KEY	VALUE	DESCRIPTION
Key	Value	Description
- Response Section:**
 - Status:** `200 OK`
 - Time:** `4 ms`
 - Size:** `97 B`
 - Download** button.
- Response Body:** The response is displayed in `JSON` format, showing `{\"Hello,Bob\"}`.

服务契约

微服务 provider

所属应用 Training21Days-HelloWo... 状态 ● 在线 正常 1 异常 0 实例

基本信息 动态配置 灰度发布

版本 0.0.1

创建时间 2019/02/12 20:26:10 GMT+08:00

描述

标签 标签管理

实例列表 被调用服务 调用服务 服务契约

hello

Swagger Yaml

```
1 ---
2 swagger: "2.0"
3 info:
4   version: "1.0.0"
5   title: "swagger definition for microservice.demo.training21days.provider.service.HelloService"
6   x-java-interface: "cse.gen.Training21Days_HelloWorld.provider.hello.HelloServiceIntf"
7 basePath: "/provider/v0"
8 consumes:
9   - "application/json"
10 produces:
11   - "application/json"
12 paths:
13   /hello/{name}:
14     get:
15       operationId: "sayHello"
16       parameters:
17         - name: "name"
18           in: "path"
19           required: true
20           type: "string"
21       responses:
22         200:
23           description: "response of 200"
24           schema:
25             type: "string"
```

点击provider服务的记录查看详情，我们能看到一份服务契约。

服务契约描述了微服务的接口，是在启动过程中由CSEJavaSDK根据微服务REST接口类（这里是HelloService.java）自动生成的。如果你观察一下provider服务的启动日志，会发现在日志里也将生成的契约打印出来了。

服务契约不仅仅是一份接口文档，它也约束了CSEJavaSDK运行时接收请求和返回应答的行为。

CSEJavaSDK使用的服务契约是Swagger契约，用户可以从网上搜索到相关资料。关于REST接口定义的约束，可以参考[接口定义和数据类型](#)。

TIPS：

服务契约描述了服务的接口，因此契约内容的变化可以认为是服务接口变化了。在正式的生产环境中这应该是不允许随意发生的。修改provider服务的接口，重启服务，可以发现服务启动失败，因为它的契约与服务中心中保存的契约不一致。如果碰到这种问题，在正式的生产环境中推荐的处理方式是在microservice.yaml文件中升级微服务版本；开发环境也可以考虑删除服务中心里的provider服务记录，或者配置service_description.environment=development。

服务契约



参见ServiceComb开源资料[ServiceComb-Java-Chassis微服务系统架构](#)，服务契约的作用贯穿ServiceComb的三个模型，而不是简单地作为接口文档。服务契约将三个模型解耦，这使得运行模型中的同一套微服务治理逻辑既可以用于不同的开发风格代码，也可以用于不同的通信方式，让框架的功能扩展能力更好。同时接口契约规范了provider和consumer双方的交互行为，让开发与测试之间、不同微服务的开发之间的沟通协作效率更高。

CSEJavaSDK作为一个带服务契约的REST开发框架，在使用上不会像传统的Servlet开发方式那么随心所欲，但随着系统规模扩大、复杂度提升，服务契约带来的好处将会明显大于其在开发方式上的限制。

开发微服务调用者

开发一个consumer服务来调用provider服务，pom.xml文件和main类完全相同，定义一个REST接口类接收外部请求并调用provider服务：

```
@RestSchema(schemaId = "helloConsumer")
@Path("/consumer/v0") // 这里使用 JAX-RS 风格开发的 consumer 服务
public class HelloConsumerService {
    // RPC调用方式需要声明一个 provider 服务的 REST 接口代理
    @RpcReference(microserviceName = "provider", schemaId = "hello")
    private HelloService helloService;

    // RestTemplate调用方式需要创建一个 ServiceComb 的 RestTemplate
    private RestTemplate restTemplate = RestTemplateBuilder.create();

    @Path("/hello")
    @GET
    public String sayHello(@QueryParam("name") String name) {
        // RPC 调用方式体验与本地调用相同
        return helloService.sayHello(name);
    }

    @Path("/helloRT")
    @GET
    public String sayHelloRestTemplate(@QueryParam("name") String name) {
        // RestTemplate 使用方式与原生的 Spring RestTemplate 相同，可以直接参考原生 Spring 的资料
        // 注意 URL 不是 http://{IP}:{port}，而是 cse://{provider端服务名}，其他部分如 path/query 等与原生调用方式一致
        ResponseEntity<String> responseEntity =
            restTemplate.getForEntity("cse://provider/provider/v0/hello/" + name, String.class);
        return responseEntity.getBody();
    }
}
```

```
// 定义 RPC 调用方式所使用的代理接口
public interface HelloService {
    // 方法名称与 provider 服务契约中的 operationId 保持一致
    // 参数顺序与 provider 服务契约中定义的顺序保持一致
    String sayHello(String name);
}
```

CSEJavaSDK支持[JAX-RS](#)、[Spring MVC](#)和[RPC](#)三种开发风格，一般我们推荐用户使用前两种，配合CSEJavaSDK自动生成服务契约的能力开发更方便。

CSEJavaSDK提供了两种微服务调用方式，[RPC方式](#)和[RestTemplate方式](#)。

开发微服务调用者

consumer服务的microservice.yaml文件与provider服务基本一致，但是注意它的服务名称需要改为“consumer”，服务监听地址改为“0.0.0.0:9090”。

启动consumer服务，可以在ServiceStage的微服务控制台看到consumer服务。调用consumer服务的两个接口，可以看到通过RPC方式和RestTemplate方式都能够成功调用provider。

The screenshot displays two REST client interactions. Each interaction shows a GET request to a specific URL, followed by a response with a 200 OK status, a response time of 8 ms, and a response size of 99 B. The response body for both is a JSON string: "Hello,Alice".

Request 1:

- Method: GET
- URL: `http://127.0.0.1:9090/consumer/v0/hello?name=Alice`
- Status: 200 OK
- Time: 8 ms
- Size: 99 B
- Body: `"Hello,Alice"`

Request 2:

- Method: GET
- URL: `http://127.0.0.1:9090/consumer/v0/helloRT?name=Alice`
- Status: 200 OK
- Time: 8 ms
- Size: 99 B
- Body: `"Hello,Alice"`

Thank You

