



21天微服务实战营-Day7

华为云DevCloud & ServiceStage服务联合出品



Day7 CSE实战之框架扩展机制

大纲

- Handler扩展机制
- Filter扩展机制
- 异常转换扩展机制
- 请求处理流程简介

Handler扩展机制

Handler机制工作于用户业务代码接收REST请求之前和发送REST请求之后，支持默认/服务两个级别的配置。

多个handler之间是链式工作的，每个handler的handle方法处理完成后，由下一个handler继续处理该次请求。

```
public interface Handler {  
    /**  
    * 每次有请求经过handler链时，都会被这个方法处理一次  
    * @param invocation invocation中记录了本次请求相关的信息  
    * @param asyncResp asyncResp用于异步返回处理结果  
    */  
    void handle(Invocation invocation, AsyncResponse asyncResp) throws Exception;  
}
```

Handler扩展机制——开发一个Handler

```
package microservice.demo.training21days.provider.handler;

import javax.ws.rs.core.Response.Status;

import org.apache.servicecomb.core.Handler;
import org.apache.servicecomb.core.Invocation;
import org.apache.servicecomb.swagger.invocation.AsyncResponse;
import org.apache.servicecomb.swagger.invocation.exception.CommonExceptionData;
import org.apache.servicecomb.swagger.invocation.exception.InvocationException;

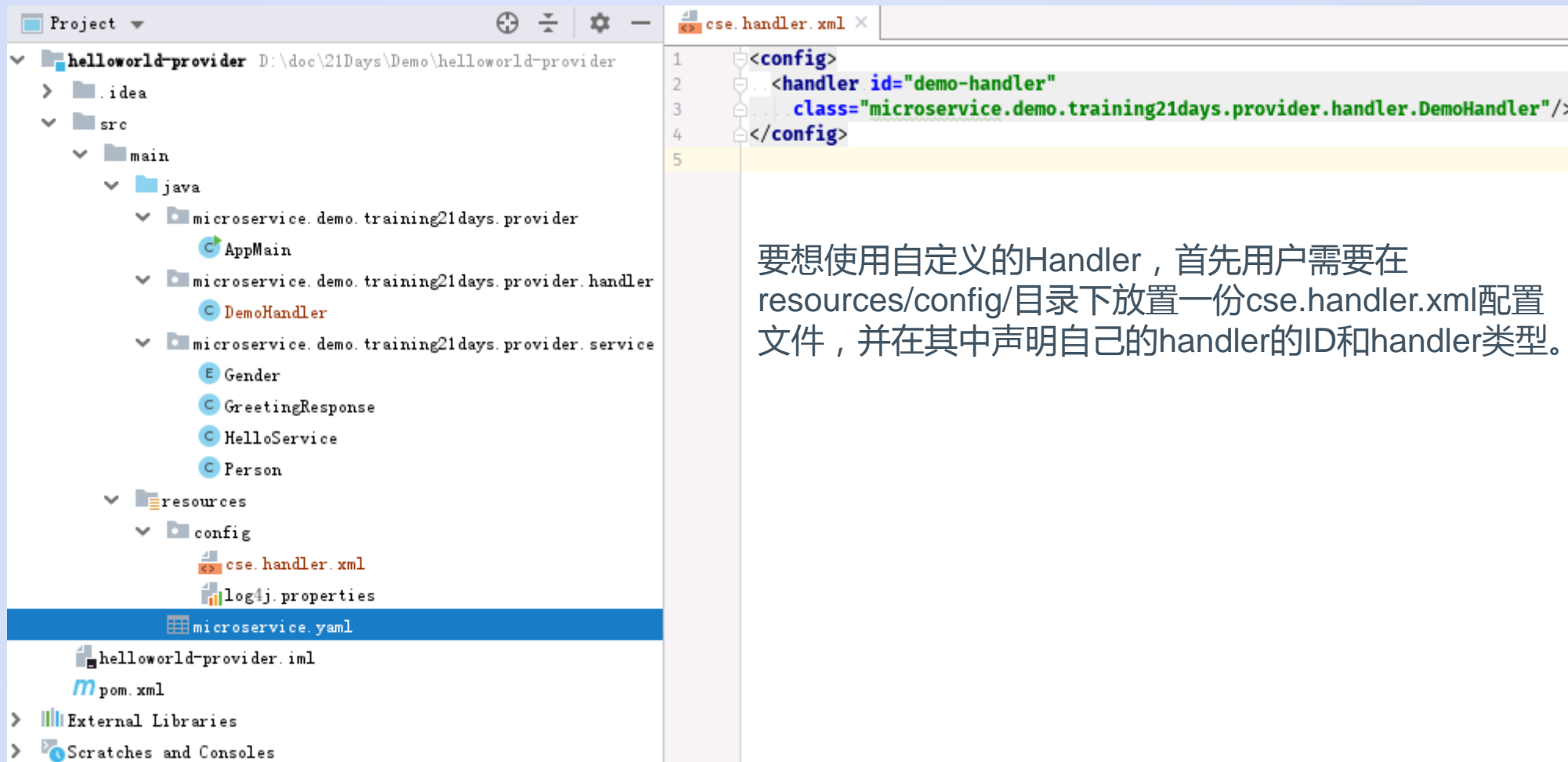
public class DemoHandler implements Handler {
    @Override
    public void handle(Invocation invocation, AsyncResponse asyncResp) throws Exception {
        // 从这里可以取出本次请求调用的方法的完整名字，格式是 serviceName.schemaId.operationId
        String operationName = invocation.getOperationMeta().getMicroserviceQualified_name();
        // 这里我们只检查 sayHello 方法的参数
        if ("provider.hello.sayHello".equals(operationName)) {
            Object name = invocation.getSwaggerArgument("idx: 0");
            // 如果 name=stranger，则拒绝请求，返回 403
            if ("stranger".equalsIgnoreCase((String) name)) {
                asyncResp.producerFail(new InvocationException(Status.FORBIDDEN, new CommonExceptionData("Don't know you :(")));
                return;
            }
        }
        // 通过检查，继续执行后面的逻辑
        invocation.next(asyncResp);
    }
}
```

这里在provider服务里开发了一个handler，该handler会检查调用sayHello方法的请求：

- 如果name是stranger，则返回403错误
- 如果是其他名字则允许调用

注意：handle方法内要么调用Invocation#next方法将请求向后传递；要么调用AsyncResponse的各种方法终止请求处理流程，返回一个应答。但是绝不能存在逻辑分支，既不向后传递请求，也不返回应答，这样会将该请求挂住，导致请求无法发往下游服务。

Handler扩展机制——开发一个Handler



The screenshot displays an IDE interface. On the left, the 'Project' view shows a directory tree for 'helloworld-provider'. The tree includes folders like '.idea', 'src', and 'main', with sub-folders for 'java' and 'resources'. The 'resources' folder contains a 'config' sub-folder with 'cse.handler.xml' and 'log4j.properties'. The 'cse.handler.xml' file is selected. On the right, the 'cse.handler.xml' file is open, showing an XML configuration snippet:

```
<config>
  <handler id="demo-handler"
    class="microservice.demo.training21days.provider.handler.DemoHandler"/>
</config>
```

Below the code editor, a text box contains the following instruction:

要想使用自定义的Handler，首先用户需要在resources/config/目录下放置一份cse.handler.xml配置文件，并在其中声明自己的handler的ID和handler类型。

Handler扩展机制——开发一个Handler

要在handler链中加载并使用handler，还需要在microservice.yaml配置文件中显式声明handler链的配置，将demo-handler加进去。

在一开始搭建微服务时引入的cse-solution-service-engine包内带有一份默认的handler链配置，大家可以打开这个包观察一下它提供的microservice.yaml配置文件。为了不损失原有的功能，我们复制一份默认的provider端handler链配置，并将自己的demo-handler加在handler链的开头。

```
cse:
  # 处理链配置
  handler:
    chain:
      Provider:
        default: demo-handler,qps-flowcontrol-provider,bizkeeper-provider
```

注意：consumer端handler链中有一个比较特殊，就是loadbalance，该handler在每次consumer服务准备发送请求时，从provider服务实例列表中选取一个实例作为本次调用的目标。如果consumer端handler链没有loadbalance handler，就会在调用时碰到找不到provider服务实例的问题。

Handler扩展机制——开发一个Handler

启动provider服务，调用它的sayHello方法，如果传入的名字不是stranger，则provider返回200的响应；如果传入的名字是stranger，则provider返回403响应。

The screenshot displays two HTTP requests in a web browser's developer tools. The first request is a GET to `http://127.0.0.1:8080/provider/v0/hello/bob`, which returns a `200 OK` status with a body of `"Hello bob"`. The second request is a GET to `http://127.0.0.1:8080/provider/v0/hello/stranger`, which returns a `403 Forbidden` status with a JSON body: `{ "message": "Don't know you :(" }`.

Request	Method	URL	Status	Time	Size	Body
1	GET	<code>http://127.0.0.1:8080/provider/v0/hello/bob</code>	200 OK	6 ms	97 B	<code>"Hello bob"</code>
2	GET	<code>http://127.0.0.1:8080/provider/v0/hello/stranger</code>	403 Forbidden	138 ms	124 B	<code>{ "message": "Don't know you :(" }</code>

Filter扩展机制

- Filter机制有两个接口，即HttpServerFilter和HttpClientFilter。
- Filter扩展机制工作于Handler扩展机制的外层，HttpServerFilter在provider端handler前工作，HttpClientFilter在consumer端handler后工作。
- Filter机制只有全局级别的生效范围。

Filter扩展机制

HttpServerFilter中的常用方法介绍如下：

```
public interface HttpServerFilter {  
    /**  
     * HttpServerFilter的优先级  
     */  
    int getOrder();  
  
    /**  
     * 是否启用该Filter，默认启用  
     */  
    default boolean enabled() {  
        return true;  
    }  
  
    /**  
     * 微服务作为provider接到请求后，会依次调用各HttpServerFilter的afterReceiveRequest方法进行处理。  
     * 注意：如果您不了解框架的底层逻辑，建议对于invocation和requestEx两个参数只读不写，否则很容易导致意料之外的问题。  
     * @param invocation 包含了本次请求的相关信息  
     * @param requestEx 包含了一些请求的原始信息。例如，不在服务契约中声明的header是不会存放在Invocation中传递给下游的provider端  
     * Handler链的，但是在requestEx里我们可以拿到这些header信息  
     * @return 如果返回null，则该请求还会继续向下执行；否则会将该方法返回的Response作为应答发给调用方，不再执行接下来的请求处理逻辑  
     */  
    Response afterReceiveRequest(Invocation invocation, HttpServletRequestEx requestEx);  
  
    /**  
     * 在应答发送给调用方之前，依次调用各HttpServerFilter的beforeSendResponse方法进行处理。默认不处理。  
     * 注意：如果您不了解框架的底层逻辑，建议对于invocation和requestEx两个参数只读不写。  
     * @param invocation 与afterReceiveRequest方法接收到的invocation相同  
     * @param responseEx 即将发送给调用方的应答。responseEx内可以自定义一些返回的header信息，但建议不要修改body。  
     */  
    default void beforeSendResponse(Invocation invocation, HttpServletResponseEx responseEx) {  
    }  
}
```

Filter扩展机制

HttpClientFilter中的常用方法介绍如下：

```
public interface HttpClientFilter {  
    /**  
     * 是否启用该Filter，默认启用  
     */  
    default boolean enabled() {  
        return true;  
    }  
  
    /**  
     * HttpClientFilter的优先级  
     */  
    int getOrder();  
  
    /**  
     * 微服务作为consumer发送请求时，会依次调用各HttpClientFilter的beforeSendRequest方法进行处理。  
     * @param invocation 包含了本次请求的相关信息，包括服务契约相关的信息  
     * @param requestEx 本次请求相关的参数信息，可以在这里自定义一些header信息，但建议不要修改body  
     */  
    void beforeSendRequest(Invocation invocation, HttpServletResponse requestEx);  
  
    /**  
     * 接收到服务端的应答后，会依次调用各HttpClientFilter的afterReceiveResponse方法进行处理。  
     * @param invocation 与afterReceiveRequest方法接收到的invocation相同  
     * @param responseEx 这里包含了一些应答的原始数据信息，如header等  
     * @return 如果返回null则继续调用下一个HttpClientFilter处理，否则将该方法返回的Response作为应答返回给业务逻辑  
     */  
    Response afterReceiveResponse(Invocation invocation, HttpServletResponse responseEx);  
}
```

Filter扩展机制——开发一个HttpServerFilter

现在我们在edge服务中定义一个HttpServerFilter：

```
public class DemoFilter implements HttpServerFilter {  
  
    private static final String LET_STRANGER_PASS = "LetStrangerPass";  
  
    @Override  
    public int getOrder() {  
        return 0;  
    }  
  
    @Override  
    public Response afterReceiveRequest(Invocation invocation, HttpServletRequestEx httpServletRequestEx) {  
        // 从请求中取出一个header  
        String letStrangerPass = httpServletRequestEx.getHeader(LET_STRANGER_PASS);  
        if (!StringUtils.isEmpty(letStrangerPass)) {  
            // 如果此header存在则将它存入到InvocationContext中，InvocationContext可以从上游服务自动传递给下游服务  
            invocation.addContext(LET_STRANGER_PASS, letStrangerPass);  
        }  
        return null;  
    }  
}
```

这里的Filter的目的是从请求中获取名为LetStrangerPass的header，将其存入InvocationContext中向下游的provider服务传递。

当provider服务的DemoHandler从InvocationContext中取出LetStrangerPass并且其值为true时，就允许stranger请求访问sayHello方法。

Filter扩展机制——开发一个HttpServerFilter

为了edge服务能够加载该filter，我们还需要定义一份SPI配置文件：



SPI机制 (Service Provider Interface) 是一种JDK内置的服务提供发现机制。
开发者扩展了哪个接口，那么对应的SPI配置文件的名字必须是与该接口相同的，文件的内容是该接口的实现类的名字。

Filter扩展机制——开发一个HttpServerFilter

我们修改一下provider服务中的DemoHandler的逻辑，如果能从InvocationContext中取出LetStrangerPass，并且其值为true，则允许stranger访问sayHello方法：

```
public class DemoHandler implements Handler {
    ..@Override
    public void handle(Invocation invocation, AsyncResponse asyncResp) throws Exception {
        ..// 从这里可以取出本次请求调用的方法的完整名字，格式是 .serviceName.schemaId.operationId
        String operationName = invocation.getOperationMeta().getMicroserviceQualified_name();
        ..// 这里我们只检查 sayHello 方法的参数
        if ("provider.hello.sayHello".equals(operationName)) {
            Object name = invocation.getSwaggerArgument( idx: 0);
            ..// 如果 name=stranger，则拒绝请求，返回403
            if (!"true".equalsIgnoreCase(invocation.getContext( key: "LetStrangerPass")))
                && "stranger".equalsIgnoreCase((String) name)) {
                asyncResp.producerFail(new InvocationException(Status.FORBIDDEN, new CommonExceptionData("Don't know you :(")));
                return;
            }
        }
        ..// 通过检查，继续执行后面的逻辑
        invocation.next(asyncResp);
    }
}
```

Filter扩展机制——开发一个HttpServerFilter

The image displays two sequential screenshots of the Edge browser's developer tools, specifically the Network tab, illustrating the effect of the `LetStrangerPass` header.

Top Screenshot (Failed Request):

- Method: GET
- URL: `http://localhost:8000/rest/provider/v0/hello/stranger`
- Status: 403 Forbidden
- Time: 6 ms
- Size: 124 B
- Headers (2):
- Body (JSON):

```
{  "message": "Don't know you :("}
```

Bottom Screenshot (Successful Request):

- Method: GET
- URL: `http://localhost:8000/rest/provider/v0/hello/stranger`
- Status: 200 OK
- Time: 6 ms
- Size: 102 B
- Headers (1):
- Body (Text): `"Hello stranger"`

In the bottom screenshot, the `LetStrangerPass` header is checked and set to `true`, which results in a successful response with the message `"Hello stranger"`.

通过edge调用provider时，如果设置header `LetStrangerPass=true`，则使用`name=stranger`的参数也可以调用`sayHello`方法。

异常转换扩展机制

如果用户的业务代码抛出了InvocationException异常，则框架会将InvocationException中的data直接序列化为响应消息的body。如果是其他的异常，则返回的响应消息状态码为490/590，响应body为CommonExceptionData。我们也提供了异常转换扩展机制，允许用户捕获不同类型的异常后，将其转换为响应消息：

- 该机制允许按优先级排列和选取异常转换器
- 可以定义特定类型的异常转换器，转换该类型及其子类的异常

异常转换扩展机制——开发一个异常转换器

我们来定义一个IllegalArgumentException的转换器作为例子。

首先我们在provider服务的greeting方法中增加一个检查，如果person参数的属性缺失，则抛出一个IllegalArgumentException。

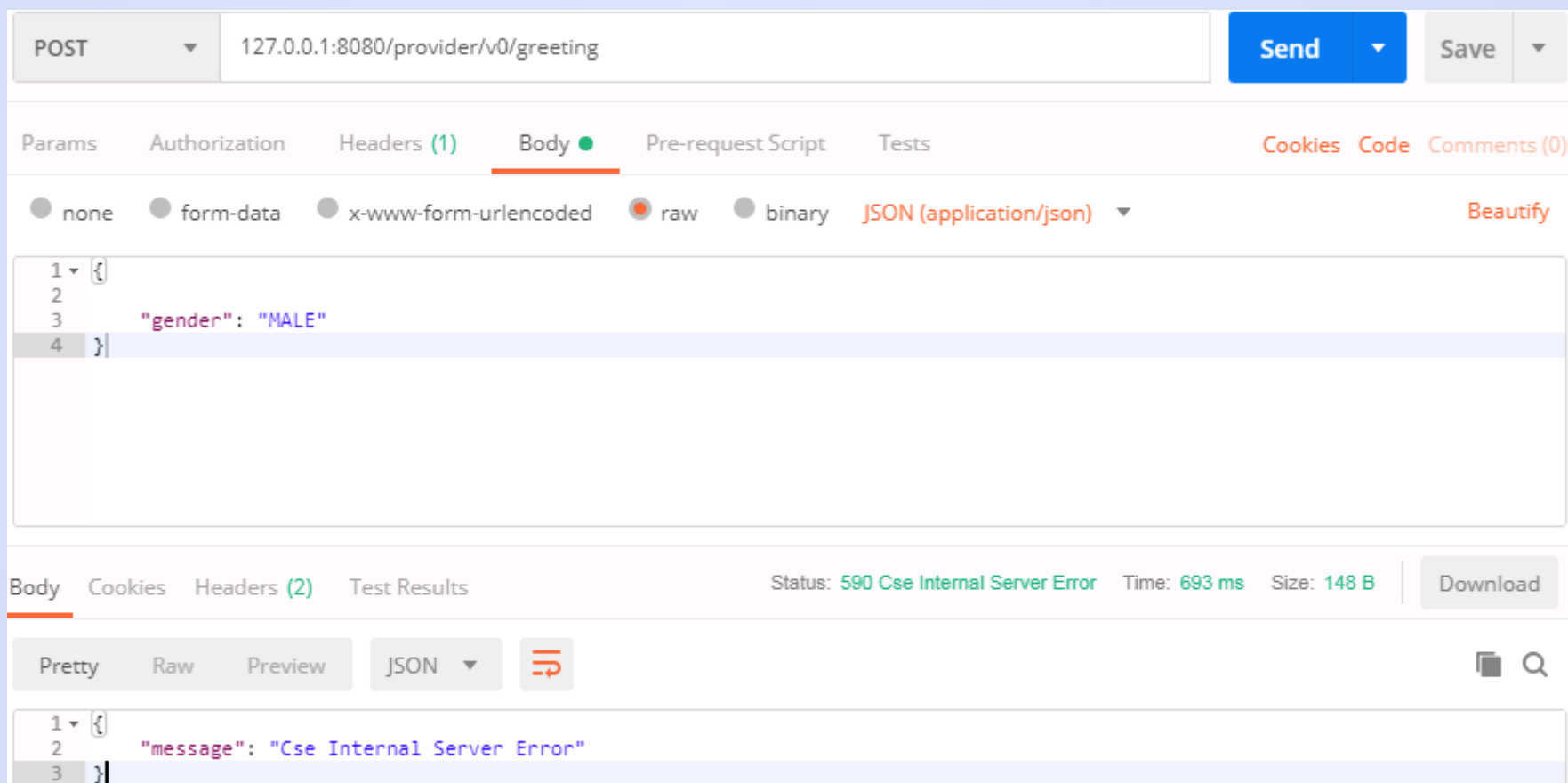
```
@PostMapping(path = "/greeting")
public GreetingResponse greeting(@RequestBody Person person) {
    if (StringUtils.isEmpty(person.getName()) || null == person.getGender()) {
        throw new IllegalArgumentException("Lack of property");
    }
    GreetingResponse greetingResponse = new GreetingResponse();

    if (Gender.MALE.equals(person.getGender())) {
        greetingResponse.setMsg("Hello, Mr." + person.getName());
    } else {
        greetingResponse.setMsg("Hello, Ms." + person.getName());
    }
    greetingResponse.setTimestamp(new Date());

    return greetingResponse;
}
```


异常转换扩展机制——开发一个异常转换器

调用provider服务的greeting方法，不传name字段，可以看到provider服务返回的状态码是590，错误信息是Cse Internal Server Error，此时请求错误的信息没有展示出来。



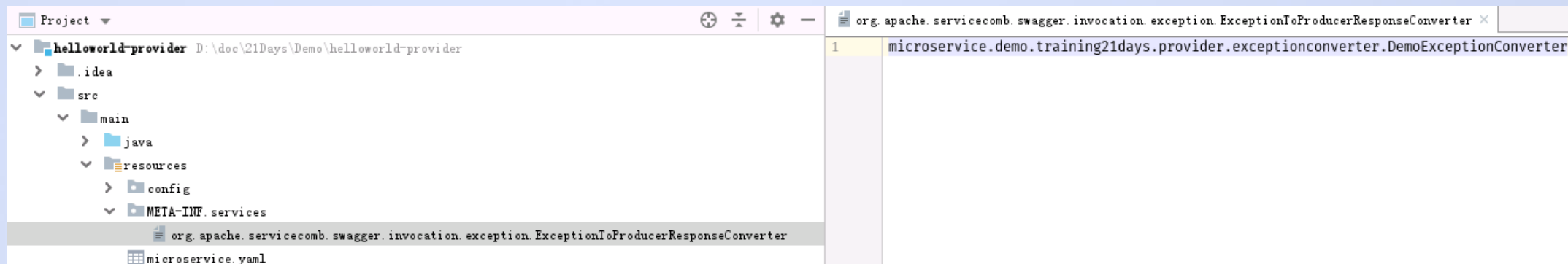
异常转换扩展机制——开发一个异常转换器

现在我们来定义一个针对IllegalArgumentExceotion的转换器

```
public class DemoExceptionHandler implements ExceptionToProducerResponseConverter<IllegalArgumentException> {  
    /**  
     * 该方法的返回值表明DemoExceptionHandler处理IllegalArgumentException及其子类的异常。  
     */  
    @Override  
    public Class<IllegalArgumentException> getExceptionClass() {  
        return IllegalArgumentException.class;  
    }  
  
    /**  
     * 当业务代码抛出的IllegalArgumentException被捕获后，会传入该方法进行处理。  
     * @param swaggerInvocation 本次业务调用相关的信息  
     * @param e 被捕获的异常  
     * @return 转换后的响应消息，将会发送给调用方  
     */  
    @Override  
    public Response convert(SwaggerInvocation swaggerInvocation, IllegalArgumentException e) {  
        return Response.consumerFailResp(  
            new InvocationException(Status.BAD_REQUEST,  
                new CommonExceptionData(  
                    swaggerInvocation.getInvocationQualifiedName() + " gets illegal param: " + e.getMessage()))  
        );  
    }  
}
```

异常转换扩展机制——开发一个异常转换器

ExceptionToProducerResponseConverter的扩展类也是通过SPI机制加载的，因此需要定义一份SPI配置文件



异常转换扩展机制——开发一个异常转换器

再次调用provider服务的greeting方法，此时响应消息是转换后的错误信息

POST 127.0.0.1:8080/provider/v0/greeting

Send Save

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code Comments (0)

none form-data x-www-form-urlencoded raw binary JSON (application/json) Beautify

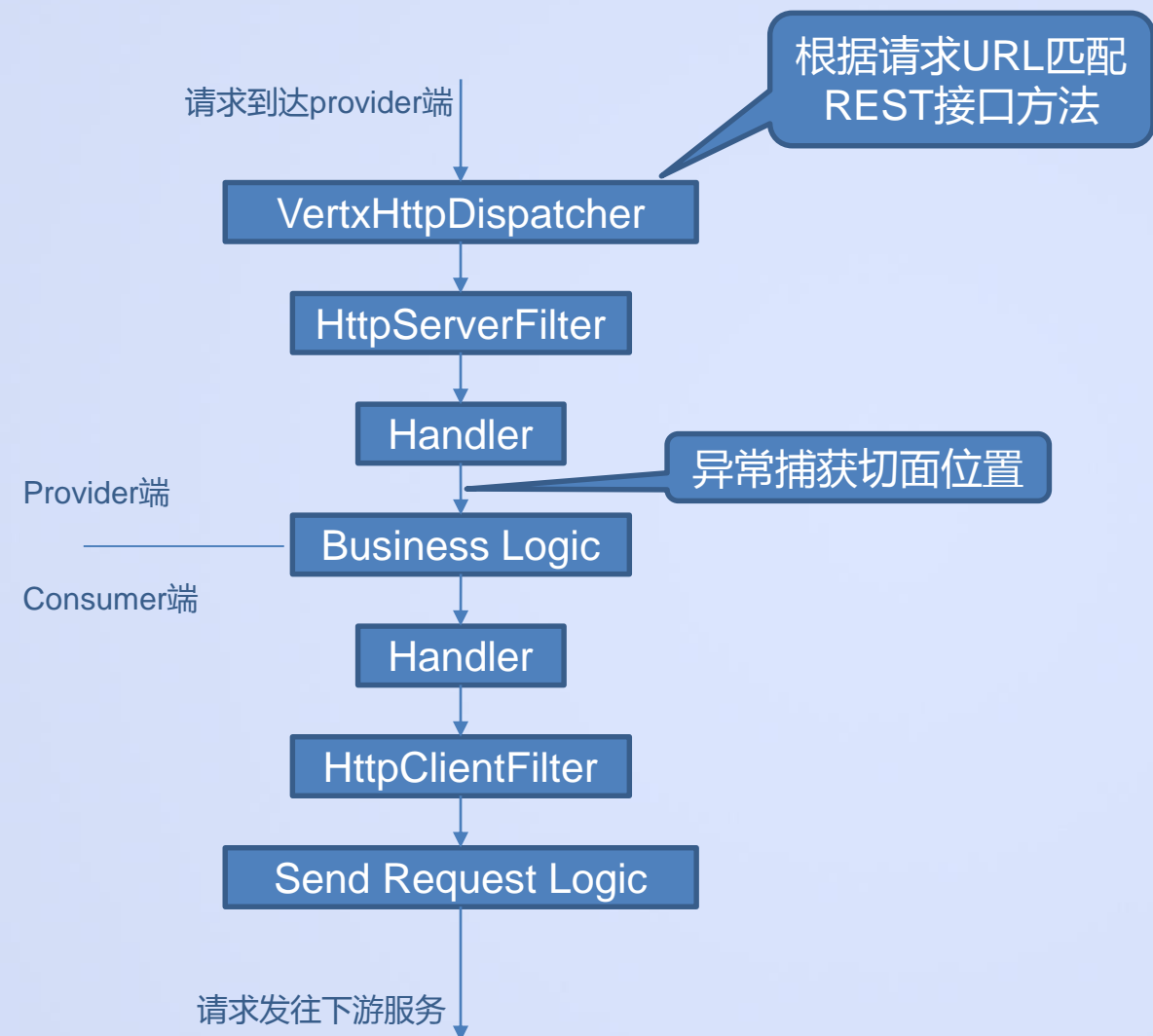
```
1 {  
2  
3   "gender": "MALE"  
4 }
```

Body Cookies Headers (2) Test Results Status: 400 Bad Request Time: 4 ms Size: 183 B Download

Pretty Raw Preview JSON

```
1 {  
2   "message": "PRODUCER rest provider.hello.greeting gets illegal param: Lack of property"  
3 }
```

请求处理流程简介



一个请求发送到某微服务，触发服务执行业务逻辑，调用下游服务方法的总体流程如图所示。

- Filter机制只有全局作用范围，Handler机制有全局和服务级作用范围
- 注意异常转换扩展捕获异常的位置，如果异常不是由业务逻辑抛出，而是由Handler等抛出的，则不在它的处理范围内
- Handler由handler.xml定义加载，Filter和异常转换机制由SPI机制加载，这些机制的实现类都不是由Spring Bean机制加载和管理的，因此@Autowired等Bean自动注入功能无法在这些扩展类里使用
- 如果要在这些扩展类里获取Spring Bean，可以考虑使用BeanUtils#getBean方法，但要注意获取时机不能太早，否则可能对应的Spring Bean还没有被Spring框架实例化
- EdgeService网关服务只有consumer端handler，没有provider端handler

Thank You

