SpringBoot使用ResponseBodyAdvice和RequestBodyAdvice实现请求体解密、响应体加密

@[TOC]

一、写在前面

项目中经常需要对接第三方平台，每次对接都需要对接收的参数进行加密、响应参数进行解密，所以通过SpringMVC的扩展点，实现一个统一的方法，对请求体进行加解密。

二、实现细节

1、定义加解密注解

```
import java.lang.annotation.*;

/**
 * @description: : 请求参数解密
 */
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface RequestDecryption {

    DecryptionType type() default DecryptionType.Type0;
}
```

```
import java.lang.annotation.*;
```

```
/**
 * @description: 响应参数加密
 */
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface ResponseEncryption {

    DecryptionType type() default DecryptionType.Type0;
}

```

```
/**
 * 加密类型
 * 根据业务类型进行扩展
 */
public enum DecryptionType {
    Type0(”不加密”),
    Type1(”业务1”),
    Type2(”业务2”),
    Type3(”业务3”),
    Type4(”业务4”),
    ;

    private String name;

    DecryptionType(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}

```

2、请求体解密逻辑

```
```

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import org.springframework.core.MethodParameter;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpInputMessage;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.util.StreamUtils;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.servlet.mvc.method.annotation.RequestBodyAd
vice;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.lang.reflect.Type;
import java.util.HashMap;
import java.util.Map;

/**
 * @description: 请求参数解密，针对post请求
 */

@ControllerAdvice
public class DecryptRequestBodyAdvice implements RequestBodyAdvice
{

    /**
     * 方法上有DecryptionAnnotation注解的，进入此拦截器
     * 只处理post请求
     *
     * @param methodParameter 方法参数对象
     * @param targetType      参数的类型
     * @param converterType   消息转换器
     * @return true，进入，false，跳过
     */
    @Override
    public boolean supports(MethodParameter methodParameter, Type
targetType, Class<? extends HttpMessageConverter<?>> converterType)
{
        return
methodParameter.hasMethodAnnotation(RequestDecryption.class) &&
methodParameter.hasMethodAnnotation(PostMapping.class);
    }

    @Override
    public HttpInputMessage beforeBodyRead(HttpInputMessage
```

```
inputMessage, MethodParameter parameter, Type targetType, Class<? extends HttpMessageConverter<?>> converterType) throws IOException {
        RequestDecryption requestDecryption = parameter.getMethodAnnotation(RequestDecryption.class);
        if (requestDecryption == null) {
            return inputMessage;
        }

        if (requestDecryption.type() == DecryptionType.Type1) {
            // 解密逻辑  这里可以做成可扩展的
            byte[] body = StreamUtils.copyToByteArray(inputMessage.getBody());

            // TODO 解密逻辑
            Map<String, String> map = new HashMap<>();
            JSONObject jsonObject = JSON.parseObject(new String(body));
            // 进行解密
            map.put("id", jsonObject.get("entryId") + "99999");
            map.put("name", jsonObject.get("entryName") + "99999");

            final ByteArrayInputStream bais = new ByteArrayInputStream(JSON.toJSONBytes(map)); // 再将字节转为输入流
            return new HttpInputMessage() {
                @Override
                public InputStream getBody() throws IOException {
                    return bais;  // 再将输入流返回
                }
                @Override
                public HttpHeaders getHeaders() {
                    return inputMessage.getHeaders();
                }
            };
        }


        return inputMessage;
    }

    /**
     * 转换之后，执行此方法，解密，赋值
     *
     * @param body          spring解析完的参数
     * @param inputMessage  输入参数
     * @param parameter     参数对象
     * @param targetType    参数类型
     * @param converterType 消息转换类型
     * @return 真实的参数
```

```
     */
    @Override
    public Object afterBodyRead(Object body, HttpInputMessage
inputMessage, MethodParameter parameter, Type targetType, Class<?
extends HttpMessageConverter<?>> converterType) {
        System.out.println("解密后的请求报文:" + body);
        return body;
    }


    /**
     * 如果body为空，转为空对象
     *
     * @param body          spring解析完的参数
     * @param inputMessage  输入参数
     * @param parameter     参数对象
     * @param targetType    参数类型
     * @param converterType 消息转换类型
     * @return 真实的参数
     */
    @Override
    public Object handleEmptyBody(Object body, HttpInputMessage
inputMessage, MethodParameter parameter, Type targetType, Class<?
extends HttpMessageConverter<?>> converterType) {
        return body;
    }

}
```

3、响应体加密逻辑

```
import org.springframework.core.MethodParameter;
import org.springframework.http.MediaType;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.http.server.ServerHttpRequest;
import org.springframework.http.server.ServerHttpResponse;
import org.springframework.web.bind.annotation.ControllerAdvice;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseBodyA
dvice;
```

```
@ControllerAdvice
public class EncryptResponseAdvice implements
ResponseBodyAdvice<Object> {


    /**
     * 响应匹配
     * @param returnType
     * @param converterType
     */
    @Override
    public boolean supports(MethodParameter returnType, Class<?
extends HttpMessageConverter<?>> converterType) {
        return
returnType.hasMethodAnnotation(ResponseEncryption.class);
    }

    // 这里可以重写响应体的内容
    @Override
    public Object beforeBodyWrite(Object body, MethodParameter
returnType, MediaType selectedContentType,
                                  Class<? extends HttpMessageConverter<?>>
selectedConverterType, ServerHttpRequest request, ServerHttpResponse
response) {

        RequestDecryption requestDecryption =
returnType.getMethodAnnotation(RequestDecryption.class);
        if (requestDecryption == null) {
            return body;
        }

        if (requestDecryption.type() == DecryptionType.Type1) {
            // 加密逻辑
            User user = (User) body;

            UserEntry userEntry = new UserEntry();

            userEntry.setEntryName("加密后的名字");
            userEntry.setEntryId("加密后的id");
            return userEntry;
        }

        return body;
    }
}

```

4、测试类

```
/**
 * 加密后传输的内容
 */
public class UserEntry {

    private String entryId;

    private String entryName;

    public String getEntryId() {
        return entryId;
    }

    public void setEntryId(String entryId) {
        this.entryId = entryId;
    }

    public String getEntryName() {
        return entryName;
    }

    public void setEntryName(String entryName) {
        this.entryName = entryName;
    }

    @Override
    public String toString() {
        return "UserEntry{" +
                "entryId='" + entryId + '\'' +
                ", entryName='" + entryName + '\'' +
                '}';
    }
}
```

```
import java.io.Serial;

/**
```

```
 * 解密后传输的内容
 */
public class User implements java.io.Serializable{

    @Serial
    private static final long serialVersionUID = -7369845805375954031L;

    private String id;

    private String name;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "User{" +
                "id='" + id + '\'' +
                ", name='" + name + '\'' +
                '}';
    }
}


```

```
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/test")
```

```
public class TestController {


    @RequestDecryption(type = DecryptionType.Type1)
    @ResponseEncryption(type = DecryptionType.Type1)
    @PostMapping("/test1")
    public User test1(@RequestBody User user) {
        System.out.println(user);
        return user;
    }
}


```

5、测试结果

我们可以实现对请求的内容进行转换、对响应的内容也进行了转换。

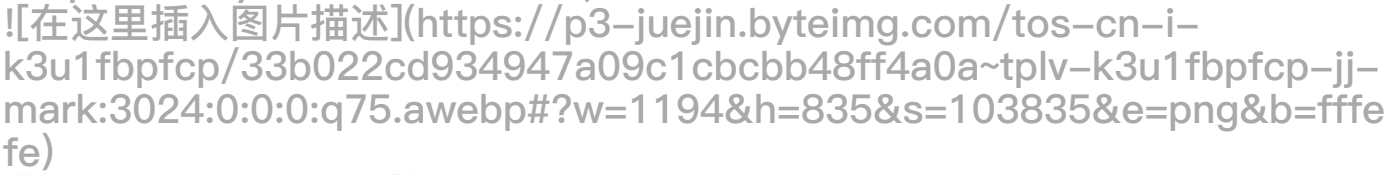
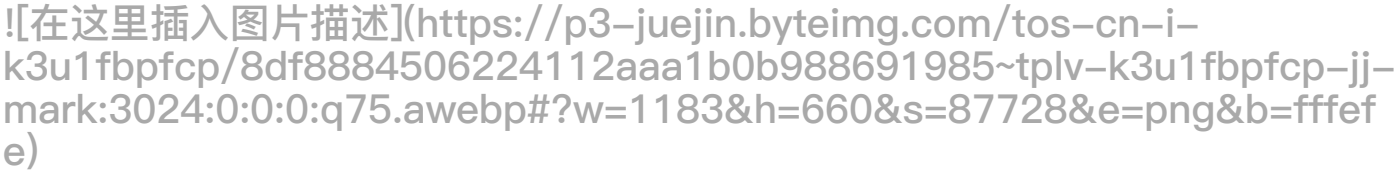
三、源码分析
======

1、RequestResponseBodyMethodProcessor

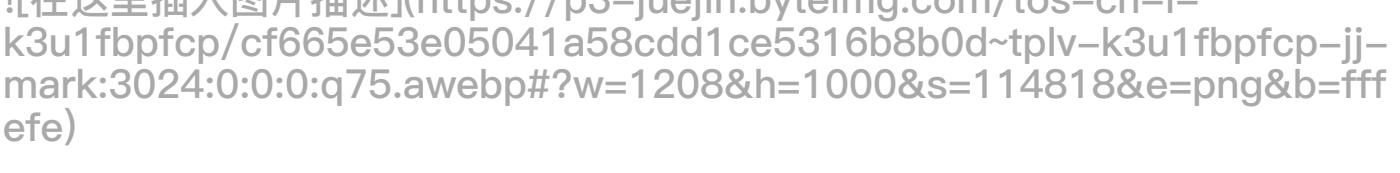
RequestResponseBodyMethodProcessor是一个请求参数的处理器，只处理@RequestBody标注的参数，所以只能用于Post请求。

在处理的过程中，会获取所有的RequestBodyAdvice，调用其中的方法进行额外的处理：

2、RequestBodyAdvice

RequestBodyAdvice是一个接口，提供了三个方法：



3、ResponseBodyAdvice



原文链接: https://juejin.cn/post/7362849699244998691