Single REST DTO Service Endpoint

Einführung

Die Programmierung von REST Services mit Spring Boot ist gemäss Lehrbuch eine relativ einfache Sache.Jede Klasse kann als Rest Service funktionieren und Daten im Format JSON verarbeiten. In der Regel erfolgt dies über Data Transfer Objekte (DTO's).Mit dem Lauf der Entwicklung nimmt die Anzahl REST Service Endpoints zu und damit auch die Komplexität und Redundanz. Projekte mit über 100 REST Endpoints sind schnell möglich und damit befinden wir uns in einem stetigen Update Prozess, da die REST Endpoints mit der zunehmenden Anzahl vermehrt angepasst werden. Es fehlt ein zentrales Error Handling oder ein Überwachungspunkt (Single REST Endpoint). Jede Anpassung löst sofort an vielen anderen Stellen Korrekturen aus.Ein Single REST Endpoint arbeitet wie ein Portier, alle Zu- und Abgänge werden über einen Punkt abgewickelt. Er zwingt uns ein Protokoll für die Kommunikation zu definieren und damit die REST Endpoints zu standardisieren. Genau hier hilft das Konzept des Single REST DTO Service Endpoints.

Das Protokoll

Zuerst definieren wir das REST Protokoll mit generischen DTO Klassen und setzen auf das HEAD-BODY-Pattern. Das UML Modell:Die generische Klasse Base definiert die HEAD-BODY Struktur:package ch.std.genericdto.dto;

import java.util.HashMap;
import java.util.Map;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.fasterxml.jackson.annotation.JsonIgnore;

import ch.std.genericdto.tools.StringUtil;

public class BaseDTO<T> {

 private static final Logger logger = LoggerFactory.getLogger(BaseDTO.class);

 private Map<String, Object> header;
 private T body;

 public BaseDTO() {
 this.header = new HashMap<String, Object>();
 }

 public Map<String, Object> getHeader() {
 return header;
 }

 public void setHeader(Map<String, Object> header) {
 this.header = header;
 }

 public T getBody() {
 return body;
 }

 public void setBody(T body) {
 this.body = body;
 }

 @JsonIgnore
 public Object getHeadValue(String key) {
 return this.header.get(key);
 }

 @JsonIgnore
 public void setHeadValue(String key, Object value) {
 this.header.put(key, value);
 }

 @JsonIgnore
 public String getStatus() {
 try {
 return this.header.get("status").toString();
 } catch (Exception e) {
 return null;
 }
 }

 @JsonIgnore
 public String getStatusMessage() {
 try {
 return this.header.get("statusMessage").toString();
 } catch (Exception e) {
 return null;
 }
 }

 @JsonIgnore
 public boolean isSuccess() {
 return "success".equals(this.getStatus());
 }

 @JsonIgnore
 public void setSuccess(String message) {
 try {
 this.header.put("status", "success");
 this.header.put("statusMessage", message);
 } catch (Exception e) {
 logger.error(e.getMessage(), e);
 }
 }

 @JsonIgnore
 public void setStatusSuccessIfNotSet(String message) {
 if (this.header.get("status") != null) {
 return;
 }
 this.setSuccess(message);
 }

 @JsonIgnore
 public boolean isFailure() {
 return "failure".equals(this.getStatus());
 }

 @JsonIgnore
 public void setFailure(String message) {
 try {
 this.header.put("status", "failure");
 this.header.put("statusMessage", message);
 } catch (Exception e) {
 logger.error(e.getMessage(), e);
 }
 }

 @JsonIgnore
 public void setFailure(Throwable t) {
 try {
 this.header.put("status", "failure");
 String message = t.getMessage();
 if (StringUtil.isNullOrEmpty(message)) {
 message = t.toString();
 }
 this.header.put("statusMessage", message);
 } catch (Exception e) {
 logger.error(e.getMessage(), e);
 }
 }
}Die generische Klasse RequestDTO repräsentiert den REST Request:
package ch.std.genericdto.dto;

import java.util.HashMap;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import com.fasterxml.jackson.databind.ObjectMapper;

public class RequestDTO<T> extends BaseDTO<T> {

 private

Map<String, Object> parameterMap;

 public RequestDTO() {
 }

 public RequestDTO(HttpServletRequest httpServletRequest) {
 this.parameterMap = new HashMap<String, Object>();
 httpServletRequest.getParameterMap().forEach((key, value) -> {
 if (value.length > 0) {
 this.parameterMap.put(key, value[0]);
 }
 });
 }

 public T getDTOFromParameterMap(Class<T> c) {
 final ObjectMapper mapper = new ObjectMapper(); // jackson's objectmapper
 return mapper.convertValue(parameterMap, c);
 }
}Die generische Klasse ResponseDTO repräsentiert die REST Response:package ch.std.genericdto.dto;
public class ResponseDTO<T> extends BaseDTO<T> {
}Damit ist das HEAD-BODY Protokoll abgedeckt. Im Header sind beliebige Key/Value Parameter definierbar. Der Body ist frei für die durch den generischen Type definierten DTO Instanzen.

Single REST Endpoint

Der Single Rest Endpoint Controller bildet den zentralen Butler, über den alle REST Calls laufen. Das Beispiel zeigt das zentrale Exception und Status Handling:
package ch.std.genericdto.rest;

import javax.servlet.http.HttpServletRequest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.GetMapping;
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;

import ch.std.genericdto.dto.RequestDTO;
import ch.std.genericdto.dto.ResponseDTO;

@RestController // default scope is singleton
@RequestMapping(value = "/rest/generic", produces = MediaType.APPLICATION_JSON_VALUE)
public class GenericDTOController<RES, REQ> {

 public GenericDTOController() {
 super();
 }

 private static final Logger logger = LoggerFactory.getLogger(GenericDTOController.class);

 @GetMapping
 public ResponseDTO<RES> get(HttpServletRequest httpServletRequest) {
 logger.info("generic get call");
 try {
 long start = System.currentTimeMillis();
 RequestDTO<REQ> requestDTO = new RequestDTO<REQ>(httpServletRequest);
 ResponseDTO<RES> responseDTO = executeGet(requestDTO);
 long end = System.currentTimeMillis();
 responseDTO.setStatusSuccessIfNotSet("successful");
 responseDTO.setHeadValue("durationmillis", (end - start));
 return responseDTO;
 } catch (Exception e) {
 logger.error(e.getMessage(), e);
 ResponseDTO<RES> responseDTO = new ResponseDTO<RES>();
 responseDTO.setFailure(e);
 return responseDTO;
 }
 }

 @PostMapping
 public ResponseDTO<RES> post(@RequestBody RequestDTO<REQ> requestDTO, HttpServletRequest httpServletRequest) {
 logger.info("generic post call");
 try {
 long start = System.currentTimeMillis();
 ResponseDTO<RES> responseDTO = executePost(requestDTO);
 long end = System.currentTimeMillis();
 responseDTO.setStatusSuccessIfNotSet("successful");
 responseDTO.setHeadValue("durationmillis", (end - start));
 return responseDTO;
 } catch (Exception e) {
 logger.error(e.getMessage(), e);
 ResponseDTO<RES> responseDTO = new ResponseDTO<RES>();
 responseDTO.setFailure(e);
 return responseDTO;
 }
 }

 public ResponseDTO<RES> executeGet(RequestDTO<REQ> requestDTO) throws Exception {
 throw new UnsupportedOperationException();
 }

 public ResponseDTO<RES> executePost(RequestDTO<REQ> requestDTO) throws Exception {
 throw new UnsupportedOperationException();
 }

}

Echo REST Endpoint

Das folgende Listing zeigt die Implementation eines Echo Controllers basierend auf dem Single REST Endpoint:package ch.std.genericdto.rest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
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;

import ch.std.genericdto.dto.RequestDTO;
import ch.std.genericdto.dto.ResponseDTO;

@RestController
@RequestMapping("/rest/echo")
public class EchoController extends GenericDTOController<EchoController.Echo, EchoController.Echo> {

 Logger logger = LoggerFactory.getLogger(EchoController.class);

 @PostMapping("/real")
 public ResponseDTO<Echo> realPost(@RequestBody RequestDTO<Echo> requestDTO) {
 logger.info("real echo call");
 ResponseDTO<Echo> responseDTO = new ResponseDTO<Echo>();
 responseDTO.setHeader(requestDTO.getHeader());
 responseDTO.setBody(new Echo("James"));
 return responseDTO;
 }

 @Override
 public ResponseDTO<Echo> executeGet(RequestDTO<Echo> requestDTO) {
 logger.info("echo executeGet, requestDTO = ", requestDTO);
 ResponseDTO<Echo> responseDTO = new ResponseDTO<Echo>();
 responseDTO.setBody((Echo)requestDTO.getBody());
 return responseDTO;
 }

 @Override
 public ResponseDTO<Echo> executePost(RequestDTO<Echo> requestDTO) {
 logger.info("echo executePost, requestDTO = ", requestDTO);
 ResponseDTO<Echo> responseDTO = new ResponseDTO<Echo>();
 responseDTO.setHeader(requestDTO.getHeader());
 responseDTO.setBody(requestDTO.getBody());
 return responseDTO;
 }

 public static class Echo {
 private String name;

 public Echo() {
 }

 public Echo(String name) {
 this.name = name;
 }

 public String getName() {
 return name;
 }

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

Calc REST Endpoint

Das folgende Listing zeigt die Implementation eines Calc Controllers basierend auf dem Single REST Endpoint:
package ch.std.genericdto.rest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import ch.std.genericdto.dto.RequestDTO;
import ch.std.genericdto.dto.ResponseDTO;

@RestController
@RequestMapping("/rest/calc")
public class CalcController extends GenericDTOController<Double, CalcController.Calc> {

 Logger logger = LoggerFactory.getLogger(CalcController.class);

 @Override
 public ResponseDTO<Double> executeGet(RequestDTO<Calc> requestDTO) {
 logger.info("calc executeGet, requestDTO = ", requestDTO);
 Calc calc = requestDTO.getDTOFromParameterMap(Calc.class);
 ResponseDTO<Double> responseDTO = new ResponseDTO<Double>();
 responseDTO.setBody(calc.calc());
 return responseDTO;
 }

 @Override
 public ResponseDTO<Double> executePost(RequestDTO<Calc> requestDTO) {
 logger.info("calc executePost, requestDTO = ", requestDTO);
 Calc calc = requestDTO.getBody();
 ResponseDTO<Double> responseDTO = new ResponseDTO<Double>();
 responseDTO.setBody(calc.calc());
 return responseDTO;
 }

 public static class Calc {
 protected String action;
 protected double a;
 protected double b;

 public Calc() {
 }

 public Calc(String action, double a, double b) {
 this.action = action;
 this.a = a;
 this.b = b;
 }

 public String getAction() {
 return action;
 }

 public void setAction(String action) {
 this.action = action;
 }

 public double

getA() {
  return a;
  }

  public void setA(double a) {
  this.a = a;
  }

  public double getB() {
  return b;
  }

  public void setB(double b) {
  this.b = b;
  }

  public Double calc() {
  switch (this.action) {
  case "add":
  return this.a + this.b;
  default:
  return null;
  }
  }
 }

}

Der Calc Unit Test

Das folgende Listing zeigt die Implementation des Calc Unit Integration Tests:package ch.std.genericdto.rest;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;

import javax.servlet.ServletContext;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.boot.web.server.LocalServerPort;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;

import ch.std.genericdto.dto.RequestDTO;
import ch.std.genericdto.dto.ResponseDTO;
import ch.std.genericdto.rest.CalcController.Calc;
import ch.std.genericdto.rest.EchoController.Echo;

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
public class CalcControllerTests {

 Logger logger = LoggerFactory.getLogger(CalcControllerTests.class);

 @LocalServerPort
 private int port;

 @Autowired
 private ServletContext servletContext;

 @Autowired
 private TestRestTemplate restTemplate;

 @BeforeEach
 public void setup() {
  logger.info("CalcControllerTests.setup");
 }

 @Test
 public void testGetEcho() throws Exception {
  String url = this.getUrl("/rest/calc?action=add&a=1.0&b=2.0");
 ResponseEntity<ResponseDTO<Double>> responseDTO = this.restTemplate.exchange(url, HttpMethod.GET, null, new ParameterizedTypeReference<ResponseDTO<Double>>() {});
 assertNotNull(responseDTO);
  assertNotNull(responseDTO.getBody());
 assertEquals(3.0, responseDTO.getBody().getBody());
 }

 @Test
 public void testPostEcho() throws Exception {
  String url = this.getUrl("/rest/calc");
 RequestDTO<Calc> requestDTO = new RequestDTO<Calc>();
 requestDTO.setBody(new Calc("add", 1.0,2.0));
 HttpEntity<RequestDTO<Calc>> httpEntityRequest = new HttpEntity<>(requestDTO);
 ResponseEntity<ResponseDTO<Double>> responseDTO = this.restTemplate.exchange(url, HttpMethod.POST, httpEntityRequest, new ParameterizedTypeReference<ResponseDTO<Double>>() {});
 assertNotNull(responseDTO);
  assertNotNull(responseDTO.getBody());
 assertEquals(3.0, responseDTO.getBody().getBody());
 }

 @Test
 public void testPostEcho2() throws Exception {
  String url = this.getUrl("/rest/echo/real");
 RequestDTO<Echo> requestDTO = new RequestDTO<Echo>();
  requestDTO.setHeadValue("name", "James");
  requestDTO.setBody(new Echo("James"));
  HttpEntity<RequestDTO<Echo>> httpEntityRequest = new HttpEntity<>(requestDTO);
 ResponseEntity<ResponseDTO<Echo>> responseDTO = this.restTemplate.exchange(url, HttpMethod.POST, httpEntityRequest, new ParameterizedTypeReference<ResponseDTO<Echo>>() {});
 assertNotNull(responseDTO);
  assertNotNull(responseDTO.getBody());
 }

 public String getUrl(String path) {
  return "http://localhost:" + port + servletContext.getContextPath() + path;
 }
}

Full Sample Download

Das gesamte Beispiel finden Sie unter dem Link genericdtocontroller.zip.

Feedback

War dieser Blog für Sie wertvoll. Wir danken für jede Anregung und Feedback

Kontakt

Simtech AG
Finkenweg 23
3110 Münsingen
Schweiz

Impressum

Das Copyright für sämtliche Inhalte dieser Website liegt bei Simtech AG, Schweiz.
Beachten Sie auch unsere Hinweise zum Urheberrecht, Datenschutz und Haftungsauschluss.
Jeder Hinweis auf Fehler nehmen wir gerne entgegen.

Copyright

2024 Simtech AG, All rights reserved, Powered by stack.ch written in Golang by Daniel Schmutz

https://www.simtech-ag.ch/requestDTO