

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 implements RES,
REQ {
    public GenericDTOController() {
        super();
    }
    private static final Logger logger =
LoggerFactory.getLogger(GenericDTOController.class);
    @GetMapping
    public ResponseDTO<RES> get(HttpServletRequest request) {
        logger.info("generic get call");
        try {
            long start =
System.currentTimeMillis();
            RequestDTO<REQ> requestDTO = new
RequestDTO<REQ>(request);
            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 request) {
        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; &#xA; &#xA; &#xA; import
org.slf4j.Logger; &#xA; import org.slf4j.LoggerFactory; &#xA; import
org.springframework.web.bind.annotation.PostMapping; &#xA; import
org.springframework.web.bind.annotation.RequestBody; &#xA; import
org.springframework.web.bind.annotation.RequestMapping; &#xA; import
org.springframework.web.bind.annotation.RestController; &#xA; &#xA; import
ch.std.genericdto.dto.RequestDTO; &#xA; import
ch.std.genericdto.dto.ResponseDTO; &#xA; &#xA; &#xA; @RestController &#xA; @RequestMapping(&
#34;/rest/echo&#34;)&#xA; public class EchoController extends
GenericDTOController &lt; EchoController.Echo, EchoController.Echo &gt; { &#xA; &#xA;
Logger logger = LoggerFactory.getLogger(EchoController.class); &#xA; &#xA;
@PostMapping(&#34;/real&#34;)&#xA; public ResponseDTO &lt; Echo &gt;
realPost(@RequestBody RequestDTO &lt; Echo &gt; requestDTO) { &#xA;
logger.info(&#34;real echo call&#34;); &#xA; ResponseDTO &lt; Echo &gt;
responseDTO = new ResponseDTO &lt; Echo &gt;(); &#xA;
responseDTO.setHeader(requestDTO.getHeader()); &#xA; responseDTO.setBody(new
Echo(&#34;James&#34;)); &#xA; return responseDTO; &#xA; } &#xA; &#xA; @Override &#xA;
public ResponseDTO &lt; Echo &gt; executeGet(RequestDTO &lt; Echo &gt;
requestDTO) { &#xA; logger.info(&#34;echo executeGet, requestDTO = &#34;,
requestDTO); &#xA; ResponseDTO &lt; Echo &gt; responseDTO = new
ResponseDTO &lt; Echo &gt;(); &#xA;
responseDTO.setBody((Echo) requestDTO.getBody()); &#xA; return responseDTO; &#xA;
} &#xA; &#xA; @Override &#xA; public ResponseDTO &lt; Echo &gt;
executePost(RequestDTO &lt; Echo &gt; requestDTO) { &#xA; logger.info(&#34;echo
executePost, requestDTO = &#34;, requestDTO); &#xA; ResponseDTO &lt; Echo &gt;
responseDTO = new ResponseDTO &lt; Echo &gt;(); &#xA;
responseDTO.setHeader(requestDTO.getHeader()); &#xA;
responseDTO.setBody(requestDTO.getBody()); &#xA; return responseDTO; &#xA; } &#xA; &#xA;
public static class Echo { &#xA; private String name; &#xA; public Echo() { &#xA;
} &#xA; &#xA; public Echo(String name) { &#xA; this.name = name; &#xA; } &#xA; &#xA; public
String getName() { &#xA; return name; &#xA; } &#xA; &#xA; public void setName(String name)
{ &#xA; this.name = name; &#xA; } &#xA; } &#xA; } &#xA;

```

Calc REST Endpoint

```

Das folgende Listing zeigt die Implementation eines Calc Controllers basierend auf dem Single
REST Endpoint: <code>package ch.std.genericdto.rest;</code><code>import</code>
<code>org.slf4j.Logger;</code><code>import org.slf4j.LoggerFactory;</code><code>import</code>
<code>org.springframework.web.bind.annotation.RequestMapping;</code><code>import</code>
<code>org.springframework.web.bind.annotation.RestController;</code><code>import</code>
<code>ch.std.genericdto.dto.RequestDTO;</code><code>import</code>
<code>ch.std.genericdto.dto.ResponseDTO;</code><code>@RestController</code><code>@RequestMapping(</code>"/
est/calc</code>)<code>public class CalcController extends GenericDTOController</code><code>{</code><code>Double,</code>
<code>CalcController.Calc</code><code> {</code><code>Logger logger =</code>
<code>LoggerFactory.getLogger(CalcController.class);</code><code> @Override</code><code> public</code>
<code>ResponseDTO</code><code> executeGet(RequestDTO</code><code> requestDTO</code>)<code> {</code><code> logger.info(</code>"calc executeGet, requestDTO = "</code>+ requestDTO)</code><code>,</code>
<code>Calc calc = requestDTO.getDTOFromParameterMap(Calc.class);</code><code> ResponseDTO</code><code> responseDTO = new</code>
<code>ResponseDTO</code><code>();</code><code> responseDTO.setBody(calc.calc());</code><code> return</code>
<code>responseDTO</code><code> }</code><code> @Override</code><code> public ResponseDTO</code><code> executePost(RequestDTO</code><code> requestDTO</code>)<code> {</code><code> logger.info(</code>"calc</code>
<code>executePost, requestDTO = "</code>+ requestDTO)</code><code> Calc calc = requestDTO.getBody();</code> <code>ResponseDTO</code><code> responseDTO = new</code>
<code>ResponseDTO</code><code>();</code><code> responseDTO.setBody(calc.calc());</code><code> return</code>
<code>responseDTO</code><code> }</code><code> public static class Calc</code><code> {</code><code> protected String action;</code><code> protected double a;</code><code> protected double b;</code><code> public Calc()</code><code> }</code><code> public</code>
<code>Calc(String action, double a, double b)</code><code> {</code><code> this.action = action;</code> <code>this.a = a;</code><code> this.b = b;</code><code> }</code><code> public String</code>
<code>getAction()</code><code> {</code><code> return action;</code><code> }</code><code> public void</code>
<code>setAction(String action)</code><code> {</code><code> this.action = action;</code><code> }</code><code> public double</code>

```



```

getA() {&#xA; return a;&#xA; }&#xA;&#xA; public void setA(double a) {&#xA; this.a = a;&#xA;
}&#xA;&#xA; public double getB() {&#xA; return b;&#xA; }&#xA;&#xA; public void setB(double b)
{&#xA; this.b = b;&#xA; }&#xA;&#xA; public Double calc() {&#xA; switch (this.action) {&#xA;
case &#34;add&#34;:&#xA; return this.a + this.b;&#xA; default:&#xA; return null;&#xA; }&#xA;
}&#xA; }&#xA;&#xA;}

```

Der Calc Unit Test

Das folgende Listing zeigt die Implementation des Calc Unit Integration Tests:

```

package
ch.std.genericdto.rest;&#xA;&#xA;import static
org.junit.jupiter.api.Assertions.assertEquals;&#xA;import static
org.junit.jupiter.api.Assertions.assertNotNull;&#xA;&#xA;import
javax.servlet.ServletContext;&#xA;&#xA;import org.junit.jupiter.api.BeforeEach;&#xA;import
org.junit.jupiter.api.Test;&#xA;import org.slf4j.Logger;&#xA;import
org.slf4j.LoggerFactory;&#xA;import
org.springframework.beans.factory.annotation.Autowired;&#xA;import
org.springframework.boot.test.context.SpringBootTest;&#xA;import
org.springframework.boot.test.context.SpringBootTest.WebEnvironment;&#xA;import
org.springframework.boot.test.web.client.TestRestTemplate;&#xA;import
org.springframework.boot.web.server.LocalServerPort;&#xA;import
org.springframework.core.ParameterizedTypeReference;&#xA;import
org.springframework.http.HttpEntity;&#xA;import org.springframework.http.HttpMethod;&#xA;import
org.springframework.http.ResponseEntity;&#xA;&#xA;import
ch.std.genericdto.dto.RequestDTO;&#xA;import ch.std.genericdto.dto.ResponseDTO;&#xA;import
ch.std.genericdto.rest.CalcController.Calc;&#xA;import
ch.std.genericdto.rest.EchoController.Echo;&#xA;&#xA;@SpringBootTest(webEnvironment =
WebEnvironment.RANDOM_PORT)&#xA;public class CalcControllerTests {&#xA;&#xA; Logger
logger = LoggerFactory.getLogger(CalcControllerTests.class);&#xA;&#xA; @LocalServerPort&#xA;
private int port;&#xA;&#xA; @Autowired&#xA; private ServletContext servletContext;&#xA;&#xA;
@Autowired&#xA; private TestRestTemplate restTemplate;&#xA;&#xA; @BeforeEach&#xA; public
void setup() {&#xA; logger.info(&#34;CalcControllerTests.setup&#34;);&#xA; }&#xA;&#xA;
@Test&#xA; public void testGetEcho() throws Exception {&#xA; String url =
this.getUrl(&#34;/rest/calc?action=add&#34;);&#xA;
ResponseEntity&#xA;ResponseDTO&#xA;Double&#xA;responseDTO =
this.restTemplate.exchange(url, HttpMethod.GET, null, new
ParameterizedTypeReference&#xA;ResponseDTO&#xA;Double&#xA;());&#xA;
assertNotNull(responseDTO);&#xA; assertNotNull(responseDTO.getBody());&#xA;
assertEquals(3.0, responseDTO.getBody().getBody());&#xA; }&#xA;&#xA; @Test&#xA; public void
testPostEcho() throws Exception {&#xA; String url = this.getUrl(&#34;/rest/calc&#34;);&#xA;
RequestDTO&#xA;Calc&#xA;requestDTO = new RequestDTO&#xA;Calc&#xA;();&#xA;
requestDTO.setBody(new Calc(&#34;add&#34;, 1.0,2.0));&#xA;
HttpEntity&#xA;RequestDTO&#xA;Calc&#xA;httpEntityRequest = new
HttpEntity&#xA;requestDTO);&#xA;
ResponseEntity&#xA;ResponseDTO&#xA;Double&#xA;responseDTO =
this.restTemplate.exchange(url, HttpMethod.POST, httpEntityRequest, new
ParameterizedTypeReference&#xA;ResponseDTO&#xA;Double&#xA;());&#xA;
assertNotNull(responseDTO);&#xA; assertNotNull(responseDTO.getBody());&#xA;
assertEquals(3.0, responseDTO.getBody().getBody());&#xA; }&#xA;&#xA; @Test&#xA; public void
testPostEcho2() throws Exception {&#xA; String url = this.getUrl(&#34;/rest/echo/real&#34;);&#xA;
RequestDTO&#xA;Echo&#xA;requestDTO = new RequestDTO&#xA;Echo&#xA;();&#xA;
requestDTO.setHeadValue(&#34;name&#34;, &#34;James&#34;);&#xA; requestDTO.setBody(new
Echo(&#34;James&#34;));&#xA; HttpEntity&#xA;RequestDTO&#xA;Echo&#xA;httpEntityRequest = new HttpEntity&#xA;requestDTO);
ResponseEntity&#xA;ResponseDTO&#xA;Echo&#xA;responseDTO =
this.restTemplate.exchange(url, HttpMethod.POST, httpEntityRequest, new
ParameterizedTypeReference&#xA;ResponseDTO&#xA;Echo&#xA;());&#xA;
assertNotNull(responseDTO);&#xA; assertNotNull(responseDTO.getBody());&#xA; }&#xA;&#xA;
public String getUrl(String path) {&#xA; return &#34;http://localhost:&#34; + port +
servletContext.getContextPath() + path;&#xA; }&#xA;}

```

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 Haftungsausschluss.
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/setSuccess>