REST APIs

Szolgáltatásorientált rendszerintegráció

Service-Oriented System Integration

Dr. Balázs Simon

BME, IIT

Outline

- JSON binding APIs
  - Java: JAXB
  - .NET: DataContract
- REST APIs
  - Java: JAX-RS
  - .NET: ASP.NET Core Web API

JAXB

JAXB

- Java Architecture for XML Binding
- Java annotations
- Strongly typed mapping between:
  - Java classes and XSD
  - Java objects+values and XML
- Can be used for JSON binding

JAXB Annotations

- @XmlSchema
  - used on packages
  - use the empty XML namespace with JSON!
- @XmlType
  - used on classes
  - maps to JSON object
- @XmlRootElement
  - used together with @XmlType on a class
  - indicates that the type can be the root element of a JSON
  - Required for the root element on serialization!

JAXB Annotations

- @XmlEnum
  - used on enum types
  - maps to a JSON string
- @XmlEnumValue
  - used on enum values
  - maps to a JSON string
- @XmlElement/@XmlAttribute
  - used on properties (getter-setter) or fields
  - maps to a JSON attribute

@XmlSchema

package-info.java:

```
@javax.xml.bind.annotation.XmlSchema(
    namespace = "",
    elementFormDefault =
        javax.xml.bind.annotation.XmlNsForm.QUALIFIED)
package soirestapi;
```

@XmlType

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType
public class Complex {
    @XmlElement
    private double re;
    @XmlElement
    private double im;

    // getters & setters...
}
```

JSON:

```
{"complex":{"re":3.1,"im":4.7}}
```

@XmlType, @XmlSeeAlso for lists

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ComplexList", propOrder = {
    "values"
})
@XmlSeeAlso(Complex.class)
public class ComplexValues {
  @XmlElement(name = "value",
    type = Complex.class,
    nillable = true)
  protected List<Complex> values;

  public List<Complex> getValues() {
    if (values == null) {
      values = new ArrayList<Complex>();
    }
    return this.values;
  }
}
```

```
JSON:
{ "complexValues": {
    "value": [
    { "@type": "complex",
      "re": 3.1,
      "im": 4.7 },
    { "@type": "complex",
      "re": 3.1,
      "im": 4.7 },
    { "@type": "complex",
      "re": 3.1,
      "im": 4.7 }
  ]
 }
}
```

@XmlEnum

JSON:
```
{"lamp":{"color":"Green"}}
```

```
@XmlEnum
public enum Colors {
    @XmlEnumValue(value = "Red")
    RED,
    @XmlEnumValue(value = "Green")
    GREEN,
    @XmlEnumValue(value = "Blueish")
    BLUE
}
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType
public class Lamp {
    @XmlElement
    private Colors color;
    // getters & setters...
}
```

@XmlRootElement

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType
public class Complex {
    @XmlElement
    private double re;
    @XmlElement
    private double im;

    // getters & setters...
}
```

Serialization and deserialization

- JAX-RS automatically handles JSON serialization on the server side
- No standard API for standalone applications
  - use a third party library
  - e.g. Jettison

Marshalling

```
// Creating the object to serialize:
Complex complex = new Complex(); complex.setRe(3.1); complex.setIm(4.7);
// Creating the file stream:
FileOutputStream os = new FileOutputStream("complex.json");
// Jettison namespace mapping:
Configuration config = new Configuration();
MappedNamespaceConvention con = new MappedNamespaceConvention(config);
config.getXmlToJsonNamespaces().
   put("http://www.w3.org/2001/XMLSchema-instance", "");
Writer writer = new OutputStreamWriter(os);
// Using Jettison for JSON serialization:
XMLStreamWriter xmlStreamWriter = new MappedXMLStreamWriter(con, writer);
// Creating the marshaller:
JAXBContext jc = JAXBContext.newInstance(Complex.class);
Marshaller marshaller = jc.createMarshaller();
// Writing the Complex object:
marshaller.marshal(complex, xmlStreamWriter);
// Closing the stream:
xmlStreamWriter.close();
```

Unmarshalling

```
// Loading the contents of the file into a string:
byte[] encoded = Files.readAllBytes(Paths.get("complex.json"));
String json = new String(encoded, "UTF8");
// Creating a Jettison JSONObject:
JSONObject obj = new JSONObject(json);
// Jettison namespace mapping:
Configuration config = new Configuration();
MappedNamespaceConvention con = new MappedNamespaceConvention(config);
// Using Jettison for JSON deserialization:
XMLStreamReader xmlStreamReader = new MappedXMLStreamReader(obj, con);
// Creating the unmarshaller:
JAXBContext jc = JAXBContext.newInstance(Complex.class);
Unmarshaller unmarshaller = jc.createUnmarshaller();
// Reading the Complex object:
Complex complex = (Complex) unmarshaller.unmarshal(xmlStreamReader);
```

DataContract

WCF DataContract

- Strongly typed mapping between:
  - .NET classes and XSD
  - .NET objects+values and XML
- Fast, easy to use
- Can be used for JSON

WCF DataContract Attributes

- [DataContract]
  - used on a class or enum
  - maps to a JSON object
- [DataMember]
  - used on a field or property
  - maps to a JSON attribute
- [EnumMember]
  - used on an enum value
  - maps to an integer value
- [CollectionDataContract]
  - used on a class inherited from a collection
  - maps to a JSON array

[DataContract], [DataMember]

```
[DataContract]
public class Complex
{
  [DataMember]
  public double Re { get; set; }
  [DataMember]
  public double Im { get; set; }
}
```

JSON:

```
{"Im":4.7,"Re":3.1}
```

[CollectionDataContract]

```
[CollectionDataContract]
public class ComplexValues : List<Complex>
{
}
```

JSON:

```
[{"Im":4.7,"Re":3.1},
 {"Im":5.6,"Re":3.9},
 {"Im":1.33,"Re":5.34}]
```

[EnumMember]

```
[DataContract]
public enum Colors
{
    [EnumMember]
    Red,
    [EnumMember]
    Green,
    [EnumMember(Value = "Blueish")]
    Blue
}

[DataContract]
public class Lamp
{
    [DataMember]
    public Colors Color { get; set; }
}
```

JSON (for Blue):
{"Color":2}

Serialization

```
// Creating the object to serialize:
var complex = new Complex();
complex.Re = 3.1;
complex.Im = 4.7;
// Creating the serializer:
var dcjs = new DataContractJsonSerializer(typeof(Complex));
// Creating the file:
using (FileStream file =
          new FileStream("complex.json", FileMode.Create))
{
    // Writing the Complex object:
    dcjs.WriteObject(file, complex);
}
```

Deserialization

```
// Creating the serializer:
var dcjs = new DataContractJsonSerializer(typeof(Complex));
// Opening the file:
using (FileStream file =
          new FileStream("complex.json", FileMode.Open))
{
    // Reading the Complex object:
    Complex complex = (Complex)dcjs.ReadObject(file);
}
```

JAX-RS

JAX-RS

- Java API for RESTful Web Services
- Goal:
  - Mapping between RESTful web services and Java
  - Uses Java annotations
- Implementations:
  - Apache CXF
  - Oracle WebLogic, Sun Jersey
  - JBoss: RESTeasy
  - IBM: WebSphere

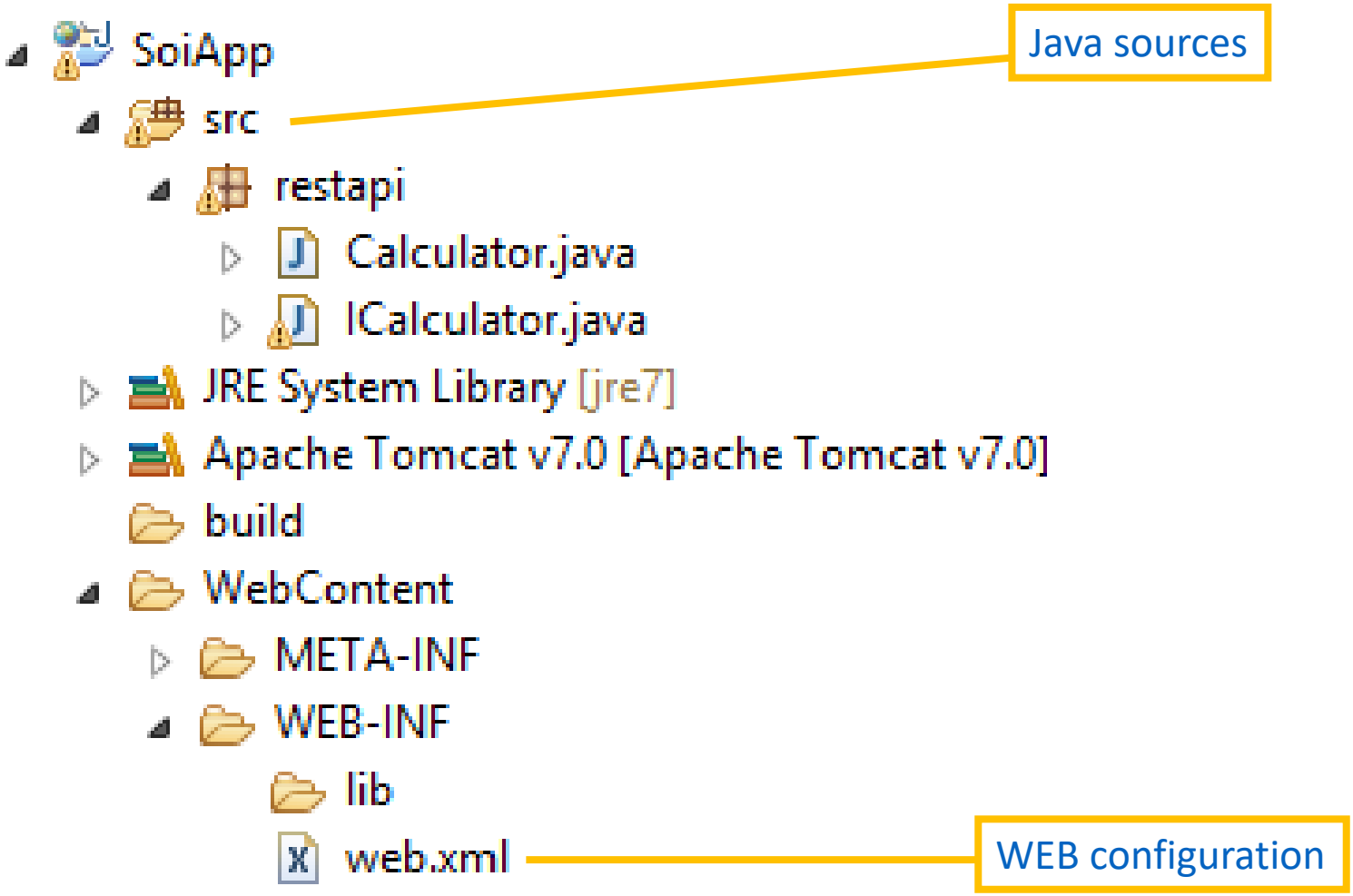
JAX-RS Annotations

- @Path
  - used on a class or method
  - specifies a relative path
    - from the web application root for the REST application resource
    - from the REST application resource to the class or method
- @GET, @PUT, @POST, @DELETE, @HEAD
  - used on a class or method
  - specifies the HTTP request method to be used to invoke the methods
  - may specify a route template for parameters in the URL

JAX-RS Annotations

- @Consumes, @Produces
  - used on a class or method
  - specifies the format of the input and output data
- @PathParam, @QueryParam, @MatrixParam, @HeaderParam, @CookieParam, @FormParam
  - used on a parameter
  - defines how the input parameters of the method are passed in the request

Publish on a server



web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  ...
 <servlet-mapping>
  <servlet-name>javax.ws.rs.core.Application</servlet-name>
  <url-pattern>/api/*</url-pattern>
 </servlet-mapping>
</web-app>
```

Query parameters

```
http://localhost:8080/SoiApp/api/calculator/add?left=5&right=8
```

```
@Path("calculator")
public interface ICalculator {
  @GET
  @Path("add")
  double add(@QueryParam("left") double left,
             @QueryParam("right") double right);
  @GET
  @Path("multiply")
  double multiply(@QueryParam("left") double left,
               @QueryParam("right") double right);
}
```

Form parameters

http://localhost:8080/SoiApp/api/calculator/add

```
@Path("calculator")
public interface ICalculator {
  @POST
  @Path("add")
  double add(@FormParam("left") double left,
             @FormParam("right") double right);
  @POST
  @Path("multiply")
  double multiply(@FormParam("left") double left,
                  @FormParam("right") double right);
}
```

Path parameters

http://localhost:8080/SoiApp/api/calculator/add/5/8

```
@Path("calculator")
public interface ICalculator {
  @GET
  @Path("add/{left}/{right}")
  double add(@PathParam("left") double left,
             @PathParam("right") double right);
  @GET
  @Path("multiply/{left}/{right}")
  double multiply(@PathParam("left") double left,
                  @PathParam("right") double right);
}
```

Implementation

```
public class Calculator implements ICalculator {

  public double add(double left, double right) {
    return left+right;
  }

  public double multiply(double left, double right) {
    return left*right;
  }
}
```

Exceptions: the easy way

```
public class Calculator implements ICalculator {
  @Override
  public double divide(double left, double right) {
    if (right == 0) {
      throw new WebApplicationException(
                    Response.Status.BAD_REQUEST);
    }
    return left / right;
  }
}
```

Exceptions: custom exception

```
public class MathException extends WebApplicationException {
    public MathException(String message) {
        super(Response.status(Response.Status.BAD_REQUEST)
            .entity(message).type(MediaType.TEXT_PLAIN).build());
    }
}

public class Calculator implements ICalculator {
    @Override
    public double divide(double left, double right) {
        if (right == 0) {
            throw new MathException("Division by zero.");
        }
        return left / right;
    }
}
```

Body parameters: XML format

```
@Path("calculator")
public interface ICalculator {
  @POST
  @Path("add")
  @Consumes(MediaType.APPLICATION_XML)
  @Produces(MediaType.APPLICATION_XML)
  Complex add(Operands operands);
}
```

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType
public class Operands {
    @XmlElement
    private Complex left;
    @XmlElement
    private Complex right;

    // getters & setters...
}
```

Body parameters: JSON format

```
@Path("calculator")
public interface ICalculator {
  @POST
  @Path("add")
  @Consumes(MediaType.APPLICATION_JSON)
  @Produces(MediaType.APPLICATION_JSON)
  Complex add(Operands operands);
}
```

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType
public class Operands {
    @XmlElement
    private Complex left;
    @XmlElement
    private Complex right;

    // getters & setters...
}
```

XML and JSON format at the same time

```
@Path("calculator")
@Consumes({MediaType.APPLICATION_XML,
           MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_XML,
           MediaType.APPLICATION_JSON})
public interface ICalculator {
  @POST
  @Path("add")
  Complex add(Operands operands);
}
```

Can be specified on the class, too

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType
public class Operands {
    @XmlElement
    private Complex left;
    @XmlElement
    private Complex right;

    // getters & setters...
}
```

Other format

```
@Path("calculator")
public interface ICalculator {
  @POST
  @Path("add")
  @Produces("text/plain")
  double add(@QueryParam("left") double left,
             @QueryParam("right") double right);
}
```

REST client

- No standard API
- Use a third party library
  - e.g. RESTeasy from JBoss
- RESTeasy:
  - provides untyped access using the builder design pattern
  - provides typed access using a Java interface annotated with JAX-RS annotations

RESTeasy client

```
ResteasyClient client = new ResteasyClientBuilder().build();
ResteasyWebTarget target = client.target(
    "http://localhost:8080/SoiApp/api/");

ICalculator calculator = target.proxy(ICalculator.class);
calculator.add(5,8);
```

ASP.NET Core Web API

ASP.NET Core Web API

- Goal:
  - Mapping between RESTful web services and .NET classes
  - Uses .NET attributes

General Web API Attributes

- [ApiController]
  - used on a class
  - defines a Controller
- [Route]
  - used on a class or a method
  - specifies a relative path from the parent (application, controller)
- [HttpGet], [HttpPost], [HttpPut], [HttpDelete], [HttpHead], [HttpPatch]
  - used on a method
  - specifies the HTTP verb and a relative path within the controller
  - may specify a route template for parameters in the URL

Attributes for Parameters

- [FromQuery]
  - gets value from the query string
- [FromRoute]
  - gets value from route data
- [FromForm]
  - gets value from posted form fields
- [FromBody]
  - gets value from the request body
- [FromHeader]
  - gets value from HTTP headers

Parameter binding

- By default, model binding gets data in the form of key-value pairs from the following sources in an HTTP request:
  - Form fields
  - Request body
  - Route data
  - Query string parameters
  - Uploaded files
- Route data and query string values are used only for simple types
- Uploaded files are bound only to target types that implement `IFormFile` or `IEnumerable<IFormFile>`

Registering the controller in Program.cs

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();

var app = builder.Build();
app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.Run();
```

Register API controllers

Query parameters

```
http://localhost/api/calculator/add?left=5&right=8
```

```
[ApiController]
[Route("api/[controller]")]
public class CalculatorController : Controller
{
    [HttpGet("add")]
    public double Add([FromQuery] double left,
                      [FromQuery] double right) { ... }

    [HttpGet("multiply")]
    public double Multiply([FromQuery] double left,
                           [FromQuery] double right) { ... }
}
```

Form parameters

http://localhost/api/calculator/add

```
[ApiController]
[Route("api/[controller]")]
public class CalculatorController : Controller
{
    [HttpPost("add")]
    public double Add([FromForm] double left,
                      [FromForm] double right) { ... }

    [HttpPost("multiply")]
    public double Multiply([FromForm] double left,
                      [FromForm] double right) { ... }
}
```

Path parameters

```
http://localhost/api/calculator/add/5/8
```

```
[ApiController]
[Route("api/[controller]")]
public class CalculatorController : Controller
{
    [HttpGet("add/{left}/{right}")]
    public double Add([FromRoute] double left,
                      [FromRoute] double right) { ... }

    [HttpGet("multiply/{left}/{right}")]
    public double Multiply([FromRoute] double left,
                      [FromRoute] double right) { ... }
}
```

Returning an error

```
http://localhost/api/calculator/divide?left=5&right=0
```

```
[ApiController]
[Route("api/[controller]")]
public class CalculatorController : Controller
{
  [HttpGet("divide")]
  public double Divide(double left, double right)
  {
    if (right == 0)
    {
      throw new HttpResponseException(HttpStatusCode.BadRequest);
    }
    return left/right;
  }
}
```

JSON and XML formatter configuration

Both are supported by default

Uses Json.NET by default

```
var json = GlobalConfiguration.Configuration.Formatters
                .JsonFormatter;
json.UseDataContractJsonSerializer = true;
```

Switch to DataContract serialization

```
var xml = GlobalConfiguration.Configuration.Formatters
                .XmlFormatter;
```

Uses DataContract serialization by default

Body parameters: XML and JSON format at the same time

```
[ApiController]
[Route("api/[controller]")]
public class CalculatorController : Controller
{
  [HttpGet("divide")]
  public Complex Add(Operands operands)
  { ... }
}
```

DataContract attribute is optional. All public properties are serialized:

```
public class Operands
{
  public Complex Left
  { get; set; }

  public Complex Right
  { get; set; }
}
```

Only DataMembers are serialized:

```
[DataContract]
public class Operands
{
  [DataMember]
  public Complex Left
  { get; set; }

  [DataMember]
  public Complex Right
  { get; set; }
}
```

Client: only untyped

```
// Creating a client:
var client = new HttpClient();
client.BaseAddress =
    new Uri("http://localhost/SoiApp/api/calculator");

// Calling the service:
var operands = new Operands() { ... }
var response = await client.PostAsJsonAsync("add", operands);
var result = await response.Content.ReadFromJsonAsync<Complex>();
```

HTML Clients for REST

Plain HTML: query parameters

```
http://localhost/SoiApp/Calculator.svc/add?left=5&right=8
```

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <form method="get"
     action="http://localhost/SoiApp/Calculator.svc/add">
        Left: <input type="text" name="left"/><br/>
        Right: <input type="text" name="right"/><br/>
        <input type="submit" value="Add"/>
    </form>
</body>
</html>
```

Plain HTML: form parameters

```
http://localhost/SoiApp/Calculator.svc/add
```

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <form method="post"
     action="http://localhost/SoiApp/Calculator.svc/add">
        Left: <input type="text" name="left"/><br/>
        Right: <input type="text" name="right"/><br/>
        <input type="submit" value="Add"/>
    </form>
</body>
</html>
```

jQuery AJAX

```
$.ajax({
    url: 'http://localhost/SoiApp/Calculator.svc/add',
    type: 'POST',
    data: $('#form').serializeArray(),
    success: function () { alert('POST completed'); }
});
```

Advanced REST Client

Advanced REST client

from restforchrome.blogspot.com

★★★★★ (6981) | Developer Tools | 655,275 users

AVAILABLE ON CHROME

OVERVIEW | REVIEWS | SUPPORT | RELATED

g+1 2.5k

Advanced Rest Client

http://restforchrome.blogspot.com/feeds/posts/default?alt=json&max-results=1

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Headers

Request
Socket
Projects
Saved
History
Settings
About

Add new header

User-Agent: Chrome-extension2

Cookie: none

A | value

Accept
Accept-Charset
Accept-Encoding
Accept-Language
Authorization

Files (0) Payload

application/x-www-form-urlencoded — Set "Content-Type" header to overwrite this value.

Scroll to top

Status: 200 OK Loading time: 208 ms

Request headers: User-Agent: Chrome-extension2
Cookie: none

Response headers: Date: Mon, 15 Oct 2012 23:07:59 GMT
Server: GSE
Content-Type: application/json
Expires: Mon, 15 Oct 2012 22:10:34 GMT
Cache-Control: private, max-age=0
Last-Modified: Mon, 08 Oct 2012 07:41:17 GMT
ETag: W/"CEcDRnsyfl7ImA9WhJaFkU."
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block

The web developers helper program to create and test custom HTTP requests.

Advanced REST client for Google Chrome.

Rate it, comment it, share it!

Users reviews:
- "This should be a benchmark for what chrome apps should be like.", Karan Kapoor
- "The best and the simplest client I've ever used. The new GWT look is much better. Thumbs up!!! Keep up the awesome work!", Denis Jajčević
- "very powerful yet simple to use", Diego Guidi

Website

Report Abuse

Version: **3.1.9**

Updated: **November 21, 2014**

Size: **411KB**

Language: **English (United States)**

Postman

Postman API / **Postman Echo GET**

Save

GET https://postman-echo.com/get?id=123&type=VIP Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	ooo	Bulk Edit
☑	id	123			
☑	type	VIP			

Body Cookies (1) Headers (7) Test Results Security 200 OK 465 ms 853 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "args": {
        "id": "123",
        "type": "VIP"
    },
    "headers": {
        "x-forwarded-proto": "https",
        "x-forwarded-port": "443",
        "host": "postman-echo.com",
        "x-amzn-trace-id": "Root=1-632a3c57-3c0f306e028d5632546fa347",
        "user-agent": "PostmanRuntime/7.29.2",
        "accept": "*/*",
        "cache-control": "no-cache",
        "postman-token": "315c86c5-b435-48f1-978c-a77a17a21766",
        "accept-encoding": "gzip, deflate, br"
```

Summary

Summary

- Java
  - JAXB
  - JAX-RS
- .NET
  - DataContract
  - ASP.NET Core Web API