arnaud.nauwynck@gmail.com

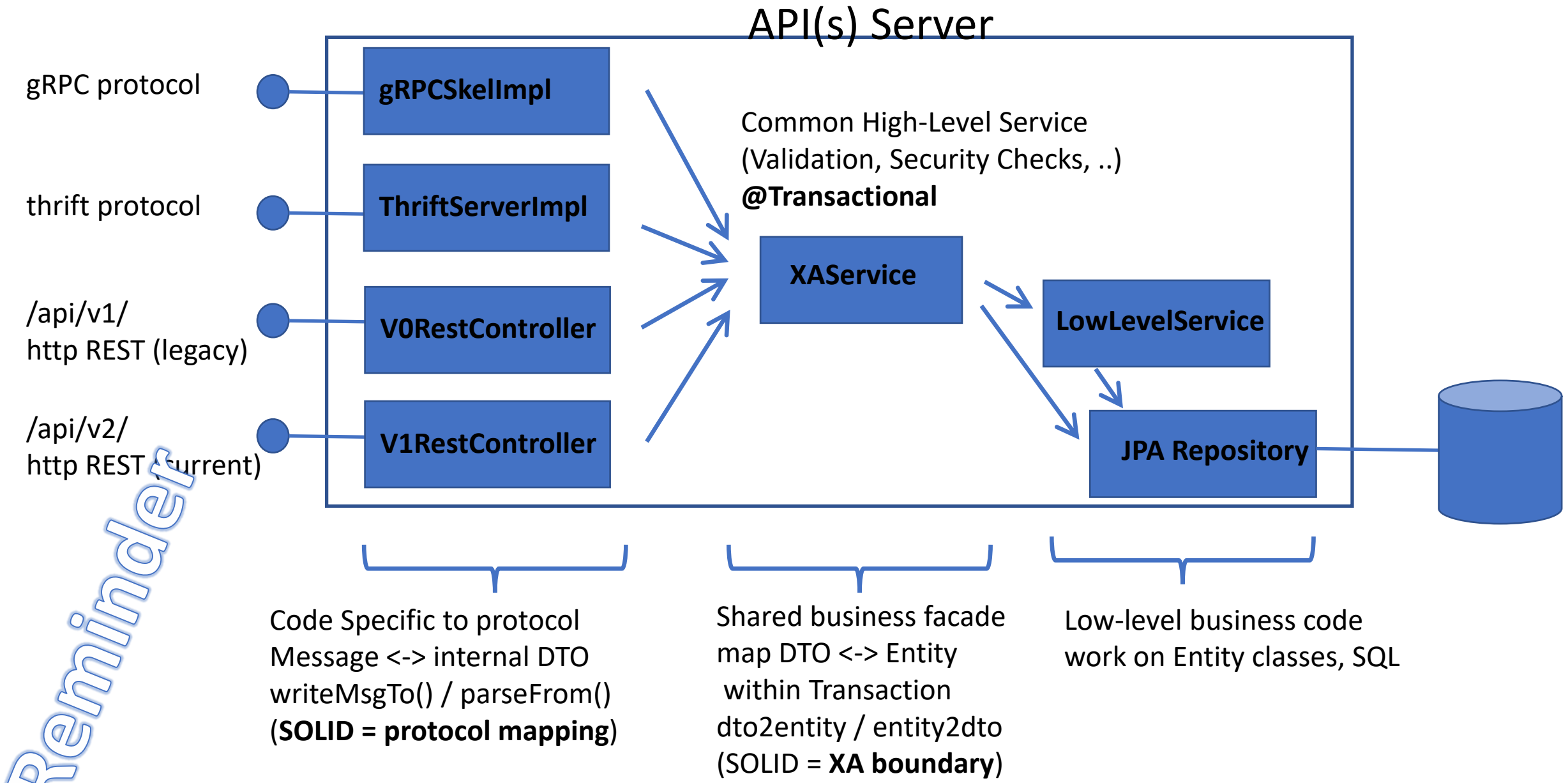
Part 4 : Rest Json to DTO Class

Json, Mapping Json to Java class, Jackson

Mapping Rest http to springboot Controller methods

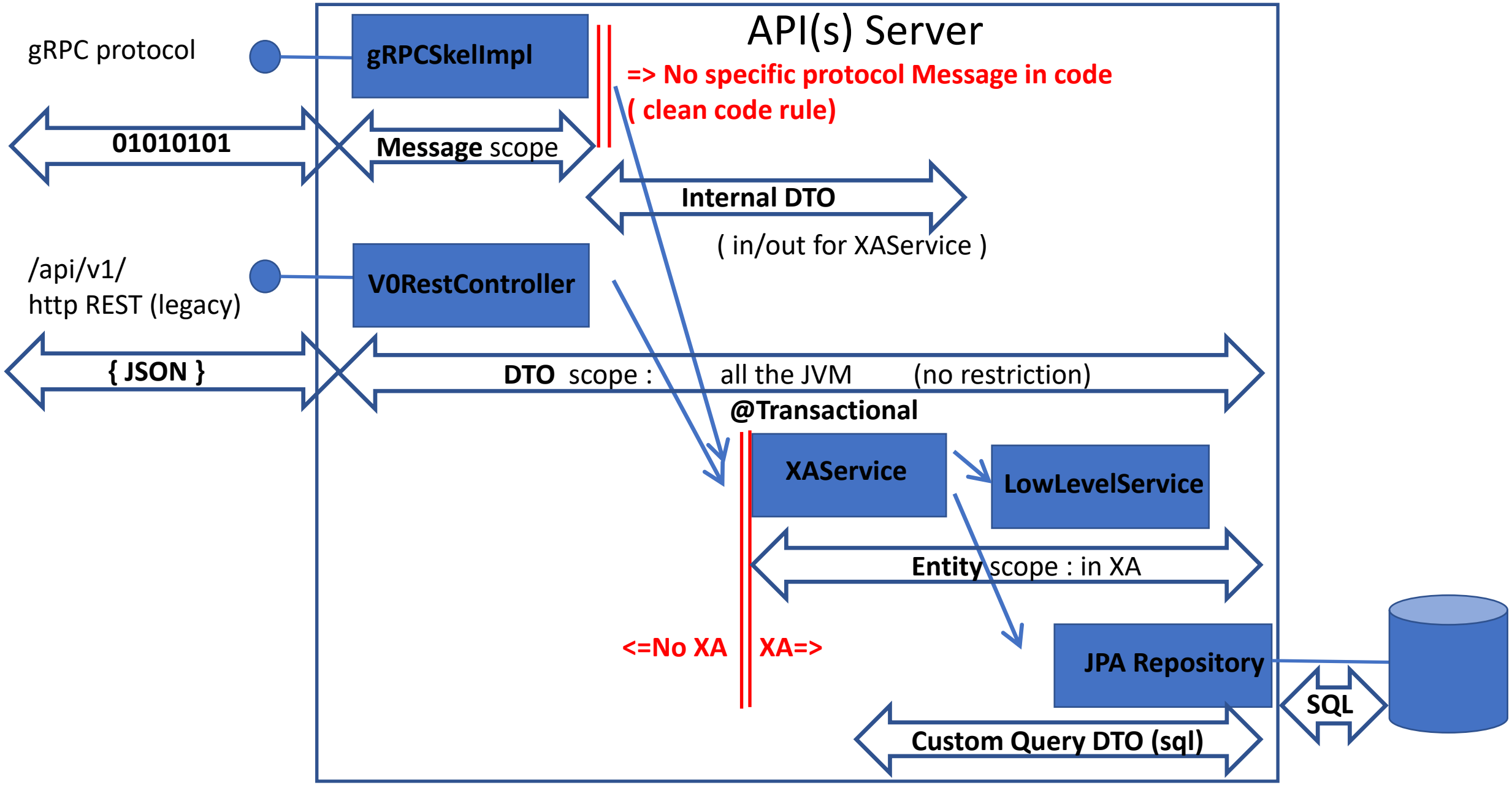
This document:
http://github.com/arnaud-nauwynck/Presentations/java/Architecture-Design-part4-RestJsonToDTO.pdf

Design code for several protocols / versionned APIs



Reminder

Message / Api DTO != Internal DTO



gRPC .. Sample protocol efficient Marshalling supporting evolutivity / backward compatibility

Reminder

Binary (protobuf,..) with Schema = OK

What about JSON ?
Schema-Less / No code generator

JavaScript <-> Json (JavaScript Object Notation) <-> Java

Script (untyped interpreter)

```javascript
let object = {
    name: 'arnaud',
    skills: [ 'IT', 'math']
};
console.log(`Hello ${object.name}`);

// JS untyped: Any: map<String,Any>
let anotherObj: any = new Object();
// anotherObj.prototype ... JS dark-magic
anotherObj.name = 'fabien';
anotherObj['skills'] = [ 'IT', 'other' ];
```

DATA format (no schema)

```json
{
    "name": "arnaud",
    "skills": [ "IT", "math"]
}
```

Langage (typed)

```java
public class User {
    public String name;
    public List<String> skills;
}
```

TypeScript

JavaScript

{JSON}

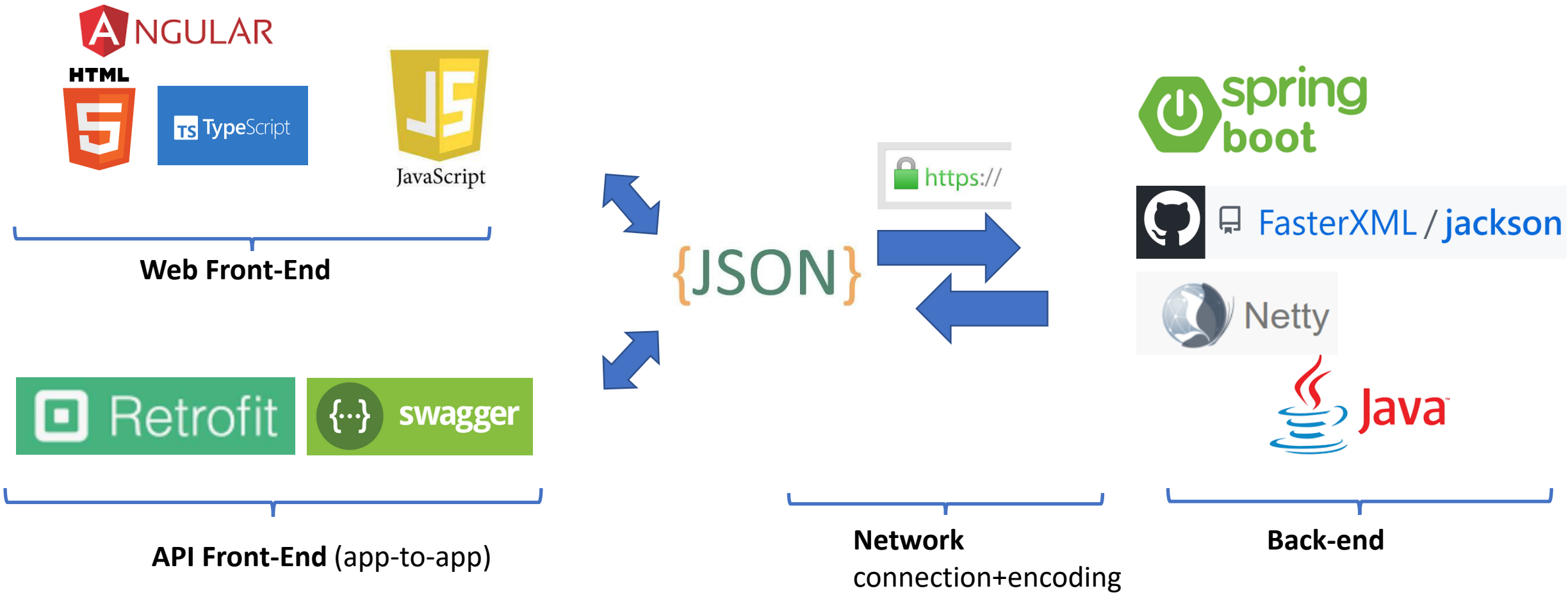
Java

Web Front-End

Network encoding

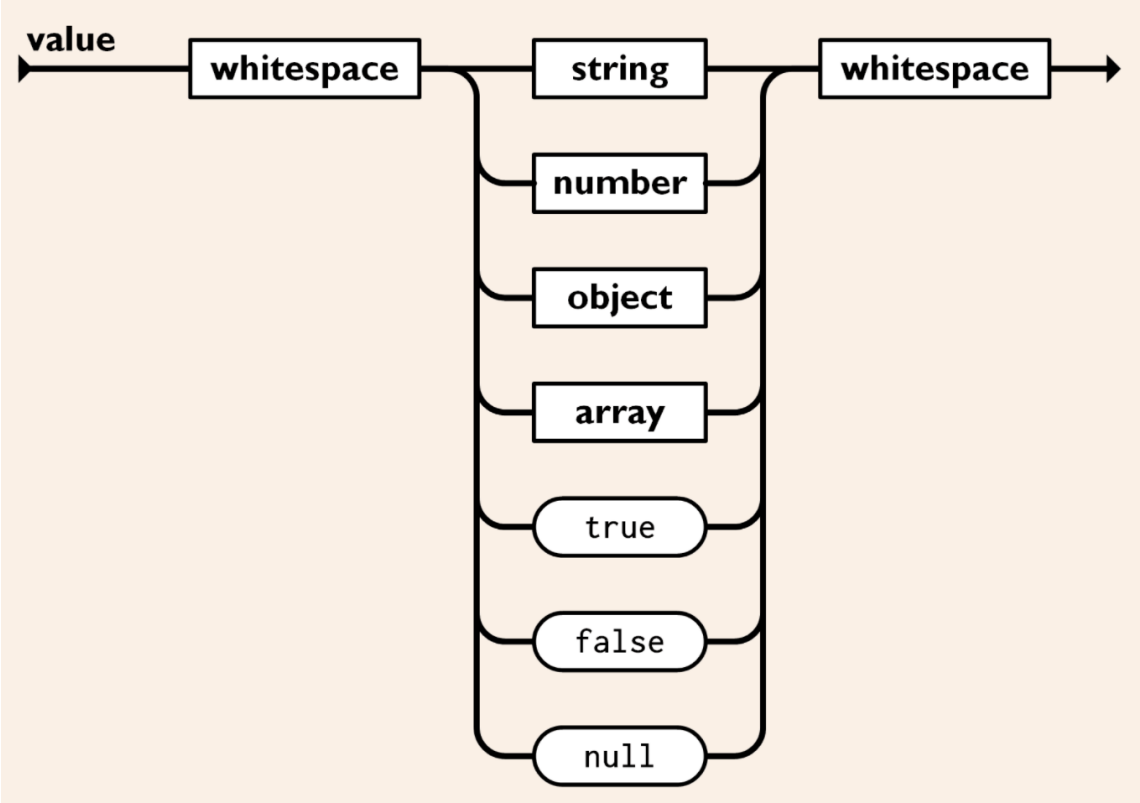
Back-end

http JSON : Open & DeFacto Standard for Portability, Simplicity, Frameworks

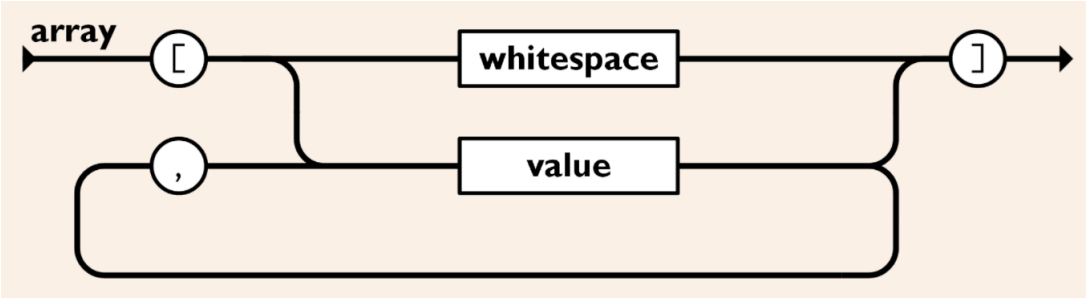


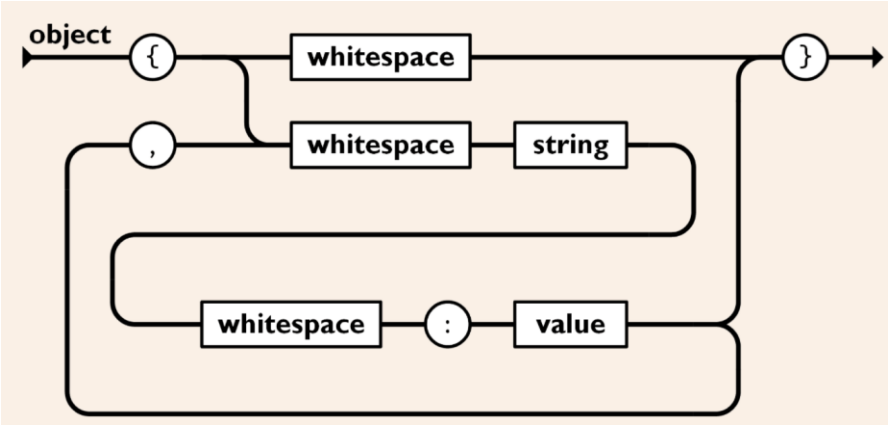
Grammar of JSON
https://www.json.org/

Literal value (terminal)



Array / Object composition





Jackson
(default dependencies from spring)

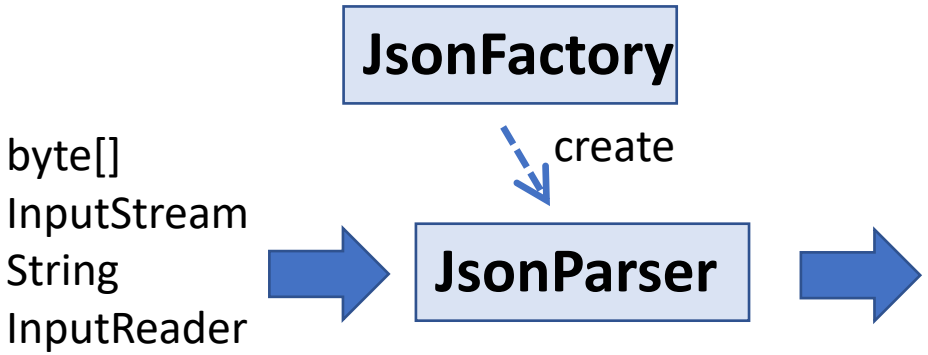
Dependency Hierarchy

- spring-boot-starter-web : 2.6.0 [compile]
  - spring-boot-starter-json : 2.6.0 [compile]
    - jackson-databind : 2.13.0 [compile]
      - jackson-annotations : 2.13.0 [compile]
      - jackson-core : 2.13.0 [compile]
    - jackson-datatype-jdk8 : 2.13.0 [compile]
      - jackson-core : 2.13.0 [compile]
      - jackson-databind : 2.13.0 [compile]
    - jackson-datatype-jsr310 : 2.13.0 [compile]
      - jackson-annotations : 2.13.0 [compile]
      - jackson-core : 2.13.0 [compile]
      - jackson-databind : 2.13.0 [compile]
    - jackson-module-parameter-names : 2.13.0 [compile]
      - jackson-core : 2.13.0 [compile]
      - jackson-databind : 2.13.0 [compile]

Jackson-Core: Json Streaming api for Parsing

jackson-core-2.13.0.jar

com.fasterxml.jackson.core



JsonToken
- NOT_AVAILABLE
- START_OBJECT
- END_OBJECT
- START_ARRAY
- END_ARRAY
- FIELD_NAME
- VALUE_EMBEDDED_OBJECT
- VALUE_STRING
- VALUE_NUMBER_INT
- VALUE_NUMBER_FLOAT
- VALUE_TRUE
- VALUE_FALSE
- VALUE_NULL

- nextToken() : JsonToken
- currentToken() : JsonToken
- currentTokenId() : int

- getValueAsInt() : int
- getValueAsLong() : long
- getValueAsString() : String

..

Sample Json Streaming Parsing

```
String json = "[ 123, true, { \"f\": \"abcd\" } ]";

JsonFactory jsonFactory = new JsonFactory();
JsonParser parser = jsonFactory.createParser(json);

JsonToken tk;
tk = parser.nextToken(); assertEquals(JsonToken.START_ARRAY, tk);
tk = parser.nextToken(); assertEquals(JsonToken.VALUE_NUMBER_INT, tk);
int i = parser.getIntValue(); assertEquals(123, i);

tk = parser.nextToken(); assertEquals(JsonToken.VALUE_TRUE, tk);

tk = parser.nextToken(); assertEquals(JsonToken.START_OBJECT, tk);
tk = parser.nextToken(); assertEquals(JsonToken.FIELD_NAME, tk);
String name = parser.getCurrentName(); assertEquals("f", name);
tk = parser.nextToken(); assertEquals(JsonToken.VALUE_STRING, tk);
String text = parser.getText(); assertEquals("abcd", text);
tk = parser.nextToken(); assertEquals(JsonToken.END_OBJECT, tk);

tk = parser.nextToken(); assertEquals(JsonToken.END_ARRAY, tk);
tk = parser.nextToken(); assertNull(tk);
```

Jackson-databind : JsonNode Class Hierarchy

jackson-databind-2.13.0.jar

com.fasterxml.jackson.databind

AST Class Hierarchy <-> Json Grammar

- JsonNode
  - BaseJsonNode
    - ContainerNode<T>
      - ArrayNode ⟷ [1, 2 ..]
      - ObjectNode ⟷ {"field1" : 1, .. }
    - ValueNode ⟷ "value"
true|false
1.234
null
  - ValueNode
    - BinaryNode
    - BooleanNode
    - MissingNode
    - NullNode
    - NumericNode
      - BigIntegerNode
      - DecimalNode
      - DoubleNode
      - FloatNode
      - IntNode
      - LongNode
      - ShortNode
    - POJONode
    - TextNode

Parsing Json to in-memory Tree (no user-defined class)

{ JSON} → JsonNode Tree

JsonNode Tree → { JSON}

Could not use typed List<T>
.. only List<JsonNode>

```
String json = "[123, { \"name\": \"abcd\" }]";

ObjectMapper om = new ObjectMapper();
JsonNode tree = om.readTree(json);

assertEquals(JsonToken.START_ARRAY, tree.asToken());
JsonNode elt0 = tree.get(0), elt1 = tree.get(1);
assertEquals(123, elt0.asInt());
assertEquals(JsonToken.START_OBJECT, elt1.asToken());
JsonNode nameNode = elt1.findValue("name");
assertEquals(JsonToken.VALUE_STRING, nameNode.asToken());
assertEquals("abcd", nameNode.asText());
```

```
JsonNodeFactory f = new JsonNodeFactory(true);
ArrayNode arrayNode = f.arrayNode();
arrayNode.add(f.numberNode(123));
ObjectNode objectNode = f.objectNode();
objectNode.set("name", f.textNode("abcd"));
arrayNode.add(objectNode);

String json = arrayNode.toString();
// arrayNode.toPrettyString();

assertEquals("[123,{\"name\":\"abcd\"}]", json);
```

Jackson-databind: ObjectMapper

{ JSON} → Object

Object → { JSON}

```
public class User {
    public String name;
}
```

```
String json = "{ \"name\": \"abcd\" }";

ObjectMapper om = new ObjectMapper();
User bean = om.readValue(json, User.class);

assertEquals("abcd", bean.name);
```

```
User bean = new User();
bean.name = "abcd";

ObjectMapper om = new ObjectMapper();
String json = om.writeValueAsString(bean);

assertEquals("{\"name\":\"abcd\"}", json);
```

Type discriminant for « class / sub-classes »

```
@JsonTypeInfo(use=Id.NAME, property="type")
@JsonSubTypes({
    @Type(name="dog", value=Dog.class),
    @Type(name="duck", value=Duck.class),
})
public static abstract class Animal {

}

public static class Dog extends Animal {
    public String barking;
}

public static class Duck extends Animal {
    public String quacking;
```

JSON

```
{"type":"dog", "barking":"houah"}
```

⟷

```
{"type":"duck", "quacking":"coin"}
```

⟷

```
Dog dog = new Dog();
dog.barking = "houah";
Animal animal = dog;

ObjectMapper om = new ObjectMapper();
String json = om.writeValueAsString(animal);

assertEquals("{\"type\":\"dog\",\"barking\":\"houah\"}", json);

Animal a = om.readValue(json, Animal.class); // parse any from abstract class
assertTrue(a instanceof Dog); // instanciated sub-class
assertEquals("houah", ((Dog) a).barking);
```

Field not set (not in JSON) => 0, false, null (in Java)

```
{
  "name":   "abcd"
}
```

null
0
false

```
public static class DetailedUser {
    public String name;
    public String address; // => null  if unset
    public int age;        // => 0     if unset
    public boolean adult;  // => false if unset
}
```

disable FAIL Unknown properties

```
String json = "{ \"name\": \"abcd\", \"xxx\": 123 }";

ObjectMapper om = new ObjectMapper();
om.disable(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES); // override
User bean = om.readValue(json, User.class);

assertEquals("abcd", bean.name);
// field "xxx" not mapped to bean
```

```
String json = "{ \"name\": \"abcd\", \"xxx\": 123 }";

ObjectMapper om = new ObjectMapper();
// om.enable(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES); // default
try {
    om.readValue(json, User.class);
    Assert.fail(); //
} catch(UnrecognizedPropertyException ex) {
    // ok !
}
```

Ignored / Unset / Unknown fields

```
{
  "field1":  "value1",
  "field2":  "value2",

  "unkownField3": "value3",
  "unkownField4": "value4"
}
```

```
class UserDefinedDTO {

  public String field1;
  public String field2;

  public String field5;
  public String field6;

  // private, no getter => ignored
  private String ignoreField7;
  @JsonIgnore // explicitly ignored
  public String ignoreField8;
}
```

(not in Json) UNSET… 0

UNKOWN (not in Java)

Ignored in java

Mixing Generic JsonNode in Type-safe DTO

{ JSON} → DTO with JsonNode

DTO with JsonNode → { JSON}

```
public class UntypedDataDTO {
    public String field1;
    public JsonNode untypedData;
}
```

```
String json = "{\"field1\":\"abcd\",\"untypedData\":[123,\"abc\"]}";

ObjectMapper om = new ObjectMapper();
UntypedDataDTO bean = om.readValue(json, UntypedDataDTO.class);

assertEquals("abcd", bean.field1);
ArrayNode arrayNode = (ArrayNode) bean.untypedData;
assertEquals(2, arrayNode.size());
JsonNode elt0 = arrayNode.get(0), elt1 = arrayNode.get(1);
assertEquals(123, elt0.asInt());
assertEquals("abc", elt1.asText());
```

```
UntypedDataDTO bean = new UntypedDataDTO();
bean.field1 = "abcd";
JsonNodeFactory f = new JsonNodeFactory(true);
ArrayNode arrayNode = f.arrayNode();
arrayNode.add(f.numberNode(123));
arrayNode.add(f.textNode("abc"));
bean.untypedData = arrayNode;

ObjectMapper om = new ObjectMapper();
String json = om.writeValueAsString(bean);

assertEquals("{\"field1\":\"abcd\",\"untypedData\":[123,\"abc\"]}", json
```

Support for Any Getter/Setter Properties

Known properties GO here

All **others** unknown properties GO here

```
public class ExtraDataDTO {
    public String field1;

    private Map<String,Object> extraData = new LinkedHashMap<>();
    public Object getExtraData(String key) {
        return extraData.get(key);
    }

    @JsonAnyGetter
    public Map<String,Object> getExtraData() {
        return extraData;
    }
    @JsonAnySetter
    public void setExtraData(String key, Object value) {
        this.extraData.put(key, value);
    }
}
```

Any Getter/Setter Properties … preserve future Unknown

{ JSON} → DTO with Any Property

```
String json = "{" +
        "\"field1\":\"abcd\"," +
        "\"field2\":\"bcde\"," +
        "\"field3\":123" +
        "}";

ObjectMapper om = new ObjectMapper();
ExtraDataDTO bean = om.readValue(json, ExtraDataDTO.class);

assertEquals("abcd", bean.field1);
Object value2 = bean.getExtraData("field2");
Object value3 = bean.getExtraData("field3");
assertEquals(2, bean.getExtraData().size());
assertEquals(123, ((Integer) value2).intValue());
assertEquals("abc", (String) value3);
```

DTO with Any Property → { JSON}

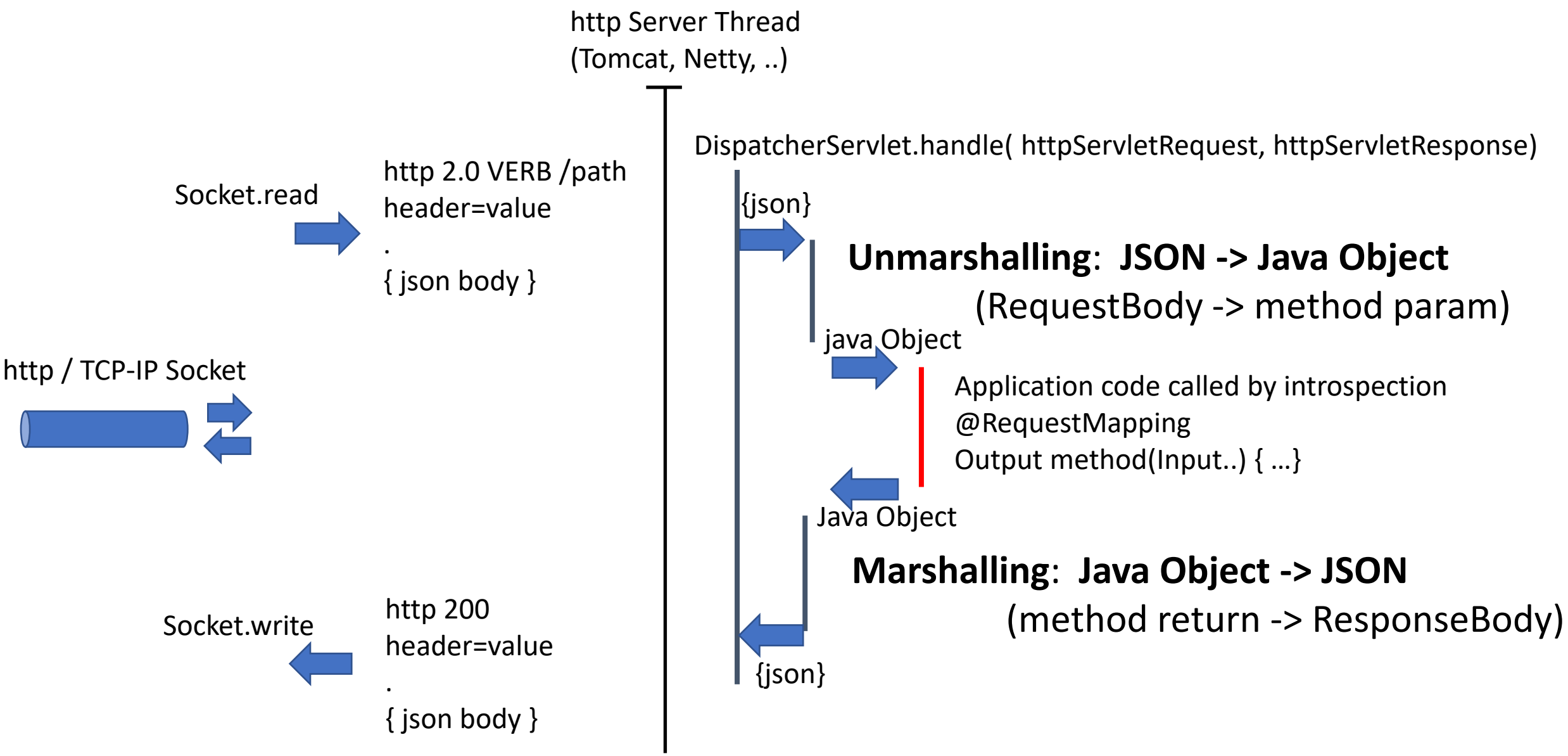
```
ExtraDataDTO bean = new ExtraDataDTO();
bean.field1 = "abcd";
bean.setExtraData("field2", "bcde");
bean.setExtraData("field3", 123);

ObjectMapper om = new ObjectMapper();
String json = om.writeValueAsString(bean);

String expectedJson = "{" +
        "\"field1\":\"abcd\"," +
        "\"field2\":\"bcde\"," +
        "\"field3\":123" +
        "}";
assertEquals(expectedJson, json);
```

JSON : schema-less
well integrated in Java (Jackson)

even more portable than protobuf
but much more verbose…

Mapping Rest Request <-> server Method Call

Http Json Request <-> Java method + Object param

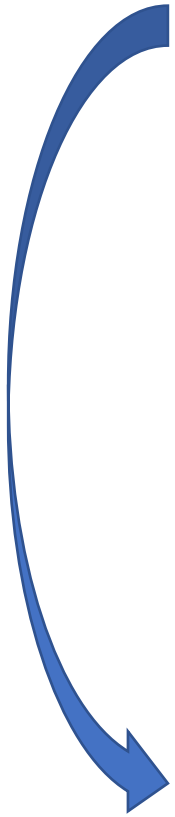


Http Request <-> Spring Java Mappings
@RequestMapping, @{Get|Post|..}Mapping, @RequestBody ..

```java
@PostMapping
public TodoDTO postTodo(
        @RequestBody TodoDTO req // => from outside, spring dispatcher...
                                 // request body as json text, is converted to java Object using Jackson
        ) {
    Log.info("http POST /api/todo");
    TodoDTO res = service.createTodo(req);
    return res;
}
```

```java
@GetMapping("/{id}")
public TodoDTO get(@PathVariable("id") int id) {
    TodoDTO res = service.get(id);
    return res;
} // => outside, spring dispatcher... return java Object is converted to json using Jackson
```

Equivalent Explicit Json Unmarshalling



```
@PostMapping
public TodoDTO postTodo(
        @RequestBody TodoDTO req) {
    Log.info("http POST /api/todo");
    TodoDTO res = service.createTodo(req);
    return res;
}

@Autowired
ObjectMapper jsonMapper;

// equivalent
@PostMapping(consumes = "application/json")
public TodoDTO postTodo2(
        @RequestBody byte[] reqBodyContent) throws Exception {
    Log.info("http POST /api/todo");
    TodoDTO req = jsonMapper.readValue(reqBodyContent, TodoDTO.class);
    TodoDTO res = service.createTodo(req);
    return res;
}
```

Equivalent Explicit Json Marshalling

```
@GetMapping("/{id}")
public TodoDTO get(@PathVariable("id") int id) {
    return service.get(id);
}
```

```
@Autowired
ObjectMapper jsonMapper;

// implicit equivalent..
@GetMapping(path = "/equivalent1/{id}",
        produces = "application/json")
public byte[] get1(@PathVariable("id") int id) throws JsonProcessingException {
    TodoDTO res = service.get(id);
    return jsonMapper.writeValueAsBytes(res);
}
```

Explicit Equivalent, with http Status + Headers

```
// implicit equivalent.. with extra header
@GetMapping(path = "/equivalent2/{id}",
        produces = "application/json")
public ResponseEntity<byte[]> get2(@PathVariable("id") int id) throws JsonProcessingException {
    TodoDTO res = service.get(id);
    byte[] content = jsonMapper.writeValueAsBytes(res);
    return ResponseEntity.status(HttpStatus.OK)
            .header("some-response-header", "value")
            .body(content);
}

// implicit equivalent.. with extra header
@GetMapping(path = "/equivalent2/{id}",
        produces = "application/json")
public void get2(@PathVariable("id") int id,
        HttpServletResponse serlvetResponse
        ) throws IOException {
    TodoDTO res = service.get(id);
    byte[] content = jsonMapper.writeValueAsBytes(res);
    serlvetResponse.setStatus(200);
    serlvetResponse.addHeader("some-response-header", "value");
    serlvetResponse.getOutputStream().write(content);
}
```