

GET Requests: Validating the Body



Andrejs Doronins

Overview



Extract the body from the response

Extract specific values from simple and complex JSON

Use Fluent Interface verifications

```
// headers
```

```
status: 200 OK
```

```
content-type: application/json; charset=utf-8
```

```
// body
```

```
{
```

```
id: 123
```

```
name: xyz
```

```
nested: {
```

```
    field1: value1
```

```
    }
```

```
}
```

```
// headers
```

```
String actual = response.getHeader( "headerX" );
```

```
// body
```

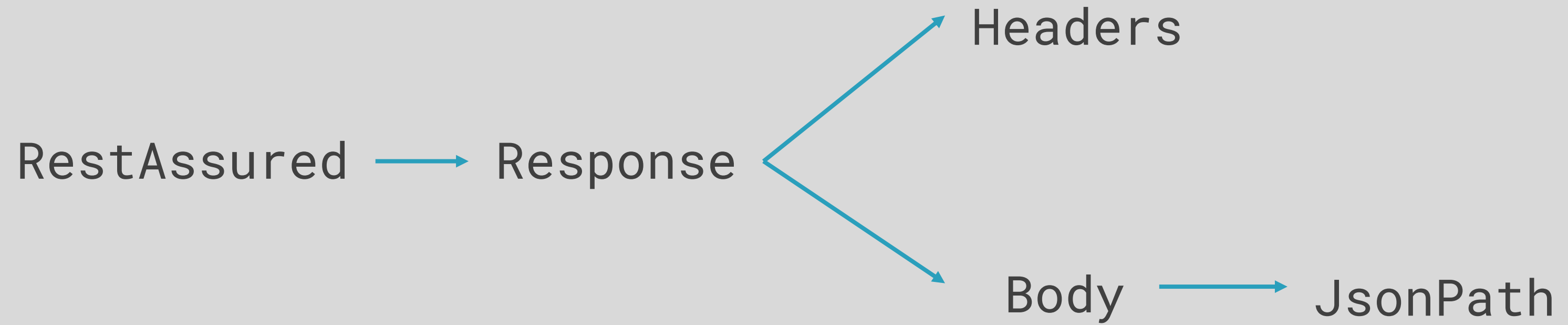
```
ResponseBody<?> body = response.getBody();
```

```
ResponseBody<?> body1 = response.body();
```

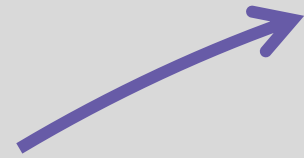


ResponseBody methods:

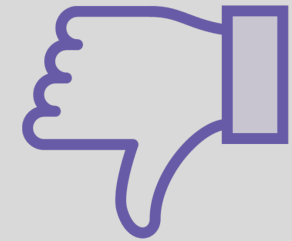
- `peek()`, `print()`, etc.
- `asByteArray()`, `asInputStream()`
- `asString()`
- `as()`
- `jsonPath()`, `xmlPath()`



```
{
  resources: {
    core: {
      limit: 60,
      remaining: 60
    },
    search: {
      limit: 10,
      remaining: 10
    }
  }
}
```



`.asString().substring(...).contains("60");`



`.jsonPath().get()`

Map > Map > Map

`.jsonPath().get("path.to.node")`

Map > Map

```
{
  resources: {
    core: {
      limit: 60,
      remaining: 60
    },
    search: {
      limit: 10,
      remaining: 10
    }
  }
}
```

 **.get()**


```
{
```

```
  resources: {
```

```
    core: {
```

```
      limit: 60,
```

```
      remaining: 60
```

```
    },
```

```
    search: {
```

```
      limit: 10,
```

```
      remaining: 10
```

```
    }
```

```
  }
```

→ `.get("resources.core")`

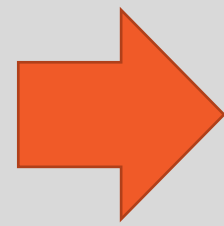
```
{  
  resources: {  
    core: {  
      limit: 60,  
      remaining: 60  
    },  
    search: {  
      limit: 10,  
      remaining: 10  
    }  
  }  
}
```



A blue arrow points from the value '10' in the 'search.limit' field of the JSON object to the string 'resources.search.limit' in the .get() method call, illustrating the path to the value.

`.get("resources.search.limit")`

```
int v =  
response.statusCode();
```



```
then().statusCode(200)
```

```
int v = jPath.get( "..."); ➡ then().body( "...", Matcher m)
```

```
RestAssured.get(url)
    .then()
    .body(Matcher m, Matcher ms)
    .body(String path, Matcher m, Object... ms)
    .body(String path, ResponseAwareMatcher m)
    [...]
```

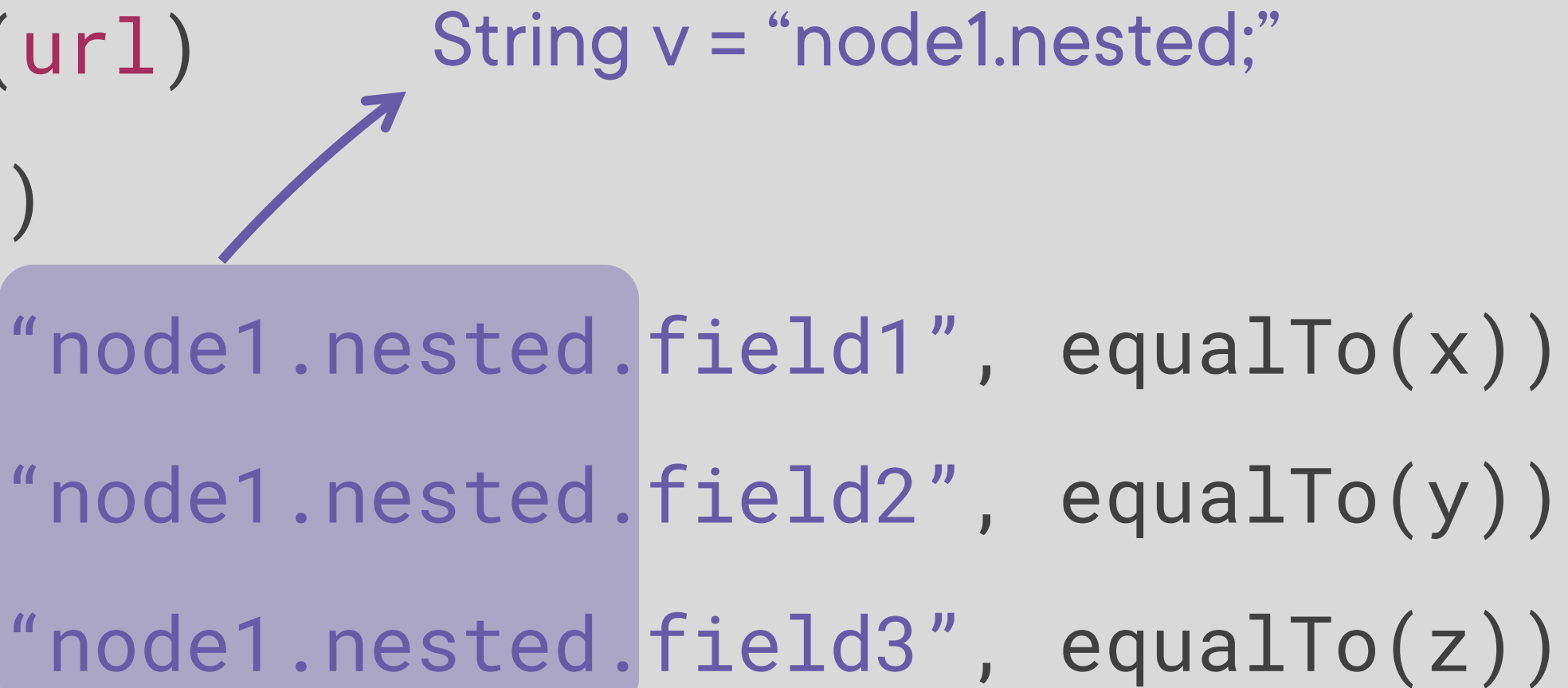
```
{  
  field1: "value1"  
  field2: "value2"  
}  
  
{  
  node1: {  
    field3: "value3"  
  }  
}
```

→ .get("field1")

→ .get("node1.field3")

```
RestAssured.get(url)
    .then()
    .body("node1.nested.field1", equalTo(x))
    .body("node1.nested.field2", equalTo(y))
    .body("node1.nested.field3", equalTo(z))
```

String v = "node1.nested;"



The diagram illustrates the relationship between a variable declaration and its use in a REST Assured assertion. A purple arrow points from the text "String v = 'node1.nested;'" to the highlighted box containing the assertion paths "node1.nested.field1", "node1.nested.field2", and "node1.nested.field3". The highlighted box is a light purple rectangle that encloses the three assertion lines in the code snippet.

```
RestAssured.get(url)
    .then()
    .rootPath("node1.nested")
        .body("field1", equalTo(x))
        .body("field2", equalTo(y))
        .body("field3", equalTo(z))
```



```
RestAssured.get(url)
```

```
.then()
```

```
.rootPath("node1.nested")
```

```
.body("field1", equalTo(x))
```

```
.body("field2", equalTo(y))
```

we keep going!



```
.rootPath("node2.nested2")
```

```
.body("fieldX", equalTo(z))
```

```
RestAssured.get(url)
```

```
.then()
```

```
.rootPath("node1.nested")
```

```
.body("field1", equalTo(x))
```

```
.rootPath("node2.nested2")
```

```
.body("fieldX", equalTo(z))
```

reset



```
.noRootPath()
```

```
.body("full.path.again", equalTo(a))
```



Tests should be small,
focused and self-
contained



```
java.lang.AssertionError: 1 expectation failed.
```

```
JSON path data.id doesn't match.
```

```
Expected: <1>
```

```
    Actual: <[1, 2, 3, 4, 5, 6]>
```



```
.body("data.first_name", contains("Eve", "Emma"))  
.body("data.first_name", containsInAnyOrder("Eve", "Emma"))  
  
.body("data.first_name", hasItem("Emma"))  
.body("data.first_name", hasItem(endsWith("ma"))))
```



Summary



ResponseBody useful methods

body() and jsonPath()

rootPath()

Dealing with repeating items

defaultParser()

Up Next:

Mapping Payload Data to Objects
