Day 16

What is JSON?

- Platform independent data interchange format
  - Others data interchange format includes XML, Protobuf, etc.
- Properties
  - Text based
  - Key value pair
  - Supports the following data types - string, boolean, numbers, array and objects
- Supported natively in JavaScript/TypeScript
  - Require libraries when used in other languages
  - Eg. Java - JSON-P

JSON Example

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY", "postalCode": 10021 },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234" },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

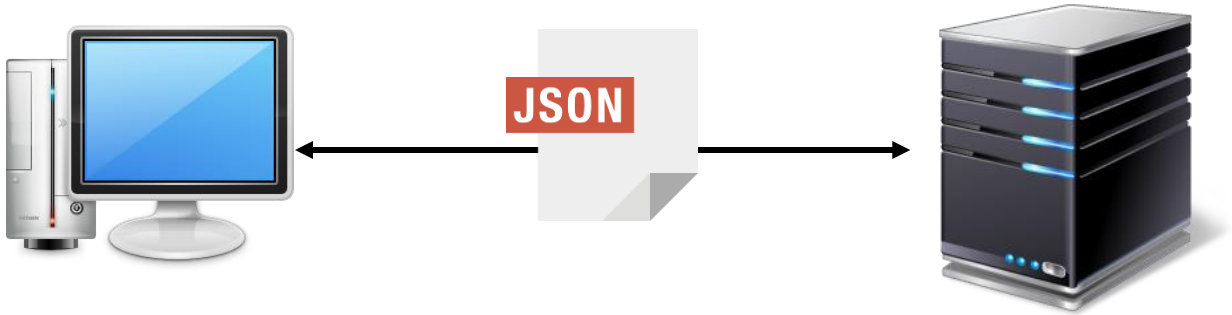
Value

Object

Array

Example from http://en.wikipedia.org/wiki/JSON

JSON as Data Interchange



- Language neutral format for exchanging structured data between applications on different systems

JSON-P

- One of the many Java API for processing JSON
- Create a builder `Json.createXXXXBuilder()`
  - `Json.createObjectBuider()` returns `JsonObjectBuilder`
  - `Json.createArrayBuilder()` returns `JsonArrayBuilder`
- Add attribute to builder with add()
  - `add(propName, String/Number/Boolean)`
  - `add(propName, JsonObjectBuilder/JsonObject)`
  - `add(propName, JsonArrayBuilder/JsonArray)`
- Call `build()` to build either `JsonObject` or `JsonArray`
  - `JsonObject` and `JsonArray` are immutable

JSON-P Dependencies

```
<dependency>
   <groupId>org.glassfish</groupId>
   <artifactId>jakarta.json</artifactId>
   <version>2.0.1</version>
</dependency>
```

Example - Using JSON-P

```
JsonObjectBuilder empBuilder =
    Json.createObjectBuilder();
empBuilder.add("firstName", "John")
    .add("lastName", "Smith")
    .add("age", 25);

JsonObjectBuilder addrBuilder =
    Json.createObjectBuilder();
addrBuilder.add("streetAddress", ...)
    .add("city", "New York")
    ...;
empBuilder.add("address", addrBuilder);

JsonObjectBuilder phBuilder = ...
JsonObjectBuilder faxBuilder = ...
JsonArrayBuilder phsBuilder =
    Json.createArrayBuilder();
phsBuilder.add(pbBuilder).add(faxBuilder);
empBuilder.add("phoneNumbers", phsBuilder);

JsonObject employee = empBuilder.build();
```

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,

  "address": {
    "streetAddress": "21 2nd
Street",
    "city": "New York",
    "state": "NY", "postalCode":
10021 },

  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234" },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

Build the actual JSON object

Example - JSON-P Fluent Style

```
JsonObject employee = Json.createObjectBuilder()
   .add("firstName", "John")
   .add("lastName", "Smith")
   .add("age", 25)
   .add("address"
      , Json.createObjectBuilder()
         .add("streetAddress", ...)
         .add("city", "..."))
   .add("phoneNumbers"
      , Json.createArrayBuilder()
         .add(Json.createObjectBuilder()
            .add("type", "home")
            .add(...))
         .add(Json.createObjectBuilder()
            .add("type", "fax")
            .add(...)))
   .build();
```

Marshalling and Unmarshall from String

- Parsing a Json string to JsonObject
  - Use `readArray()` or `readObject()` depending on the expected type

```
String jsonString = ... //

try (InputStream is = new ByteArrayInputStream(jsonString.getBytes()) {
    JsonReader reader = Json.createReader(is);
    JsonObject data = reader.readObject();
    // do something with data
}
```

Create an `InputStream` from String

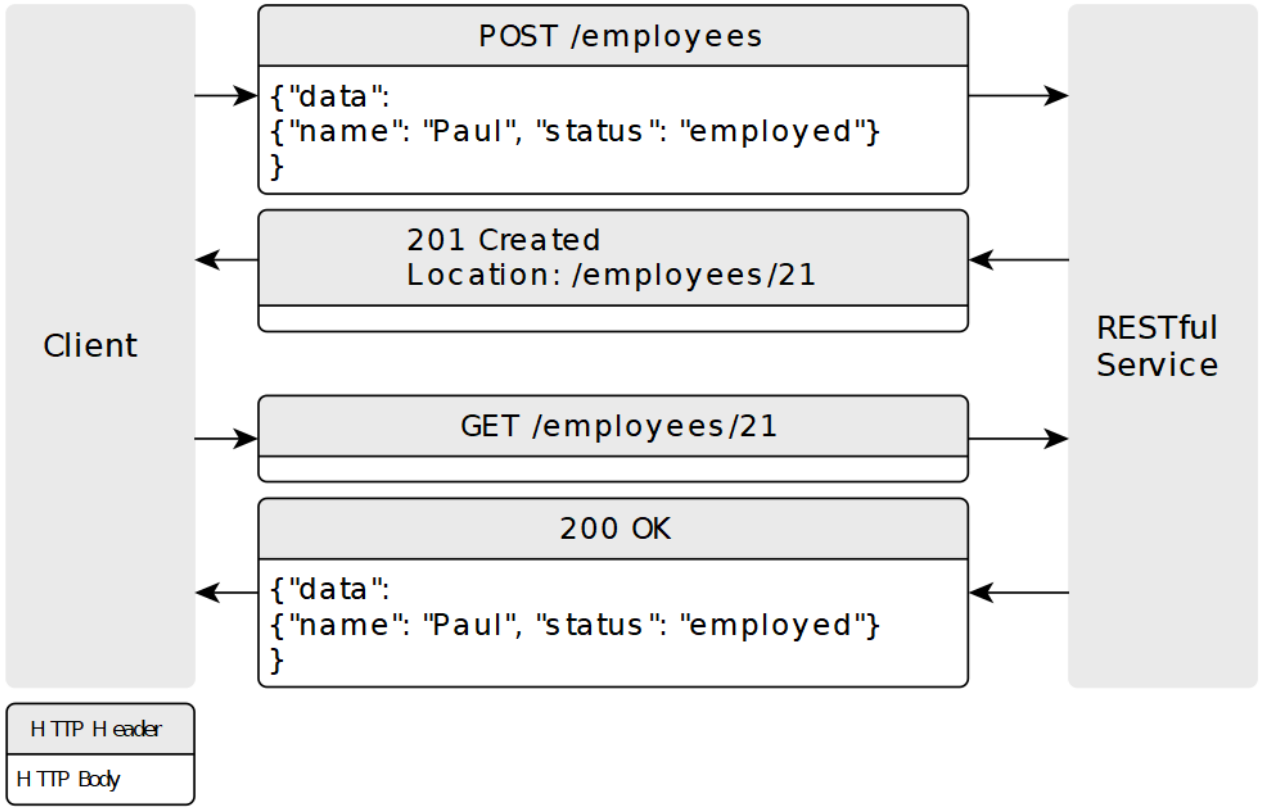
Create a reader from the `InputStream`

Parse the string to `JsonObject`

- 'Stringify' a JSONObject

```
data.toString()
```

HTTP REST Endpoint



- REST - REpresentational State Transfer
- HTTP resources returns structured data instead of HTML
  - Meant for machine consumption
  - JSON and XML are the most common

`@RestController`

- The annotation is used to create HTTP REST endpoint
- Very similar in use to `@Controller`
  - Returns JSON object
- Returns `ResponseEntity` object instead of a Thymeleaf template
  - Use `ResponseEntity` to construct the response

Example - GET Endpoint

Annotates this class as a RESTful resource

Content type that the handler will be producing
Sets the `Content-Type` header

Annotate the method use to handle the HTTP GET

Response type will be a string

Sets the HTTP status code to 200 and stringify the JSON object

```
@RestController
@RequestMapping(path="/user", produces="application/json")
public class UserController {
    @Autowired
    private UserService userSvc;

    @GetMapping(path="{userId}")
    public ResponseEntity<String> getUser(
            @PathVariable(name="userId") String userId) {

        final User user = userSvc.get(userId);

        final JsonObject resp = Json.createObject()
                .add("name", user.getName())
                ...
                .build();

        return ResponseEntity.ok(resp.toString());
    }
}
```

Example - POST Endpoint

If consumes matches the request's `Content-Type` HTTP header

Inject the payload as string into the request handler

```
@PostMapping(consumes="application/json")
public ResponseEntity<String> postUser(@RequestBody String payload) {
    JsonObject body;
    try (InputStream is = new ByteArrayInputStream(payload.getBytes()) {
        JsonReader reader = Json.createReader(is);
        body = read.readObject();
    } catch(Exception ex) {
        body = Json.createObject()
            .add("error", ex.getMessage()).build();
        return ResponseEntity.internalServerError().body(body.toString());
    }
    // continue with processing
}
```

Common ResponseEntity

- OK result

```
ResponseEntity.ok(entity).build()
```

- OK with custom header

```
HttpHeaders headers = new HttpHeaders();
headers.add("X-AppName", "myapp");
new ResponseEntity(entity, headers, HttpStatus.OK)
```

- Resource not found

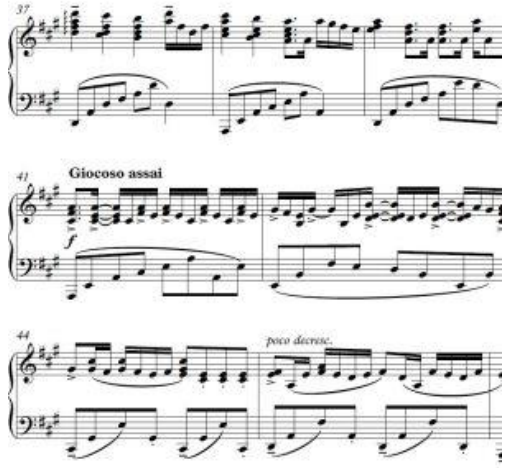
```
ResponseEntity.status(HttpStatus.NOT_FOUND)
    .body(entity).build()
```

Representation

- A data can be presented in more than one way

Canon in D

As music sheet



As MP3 file

Representation

http://www.kerfers.com/menu

Accept:

```
<html>
 <body>
  <ul>
   <li>Espresso</li>
   <li>Espresso Macchiato</li>
   <li>Espresso con Panna</li>
   <li>Caffe Latte</li>
   <li>Flat White</li>
   <li>Cafe Breve</li>
  <ul>
 </body>
</html>
```

text/html

```
Espresso
Espresso Macchiato
Espresso con Panna
Cafe Latte
Flat White
Cafe Breve
```

text/plain

```
{ "coffees": [
        "Espresso",
        "Espresso Macchiato",
        "Espresso con Panna",
        "Cafe Latte",
        "Flat White",
        "Cafe Breve"
]}
```

application/json

Media type

Content-Type:

Representation

```
<html>
 <body>
  <ul>
   <li>Espresso</li>
   <li>Espresso Macchiato</li>
   <li>Espresso con Panna</li>
   <li>Caffe Latte</li>
   <li>Flat White</li>
   <li>Cafe Breve</li>
  <ul>
 </body>
</html>
```

text/html

Browser

```
Espresso
Espresso Macchiato
Espresso con Panna
Cafe Latte
Flat White
Cafe Breve
```

text/plain

Human

```
{ "coffees": [
        "Espresso",
        "Espresso Macchiato",
        "Espresso con Panna",
        "Cafe Latte",
        "Flat White",
        "Cafe Breve"
]}
```

application/json

JavaScript

Content Negotiation

- `Accept` HTTP headers tells the server what data format the client expects
- `Content-Type` HTTP header tells the client what data format the server is returning
  - `Content-Type` may also be present in a request; it is to form the server the format of the data present in the request eg. user registration

Content Negotiation

```
GET /time HTTP/1.1
Host: localhost
Connection: Keep-Alive
Accept: text/html
Accept-Language: us-en, fr, cn
```

Can I have /time resource in text/html format?

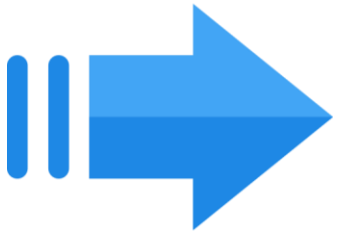
```
HTTP/1.1 200 OK

Content-Type: text/html
Content-Length: 1970
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100

<!doctype html>
...
```

Here is the content in text/html format

Content Negotiation



```
GET /user/fred
Accept: application/json
```

```
@Controller
@RequestMapping(path="/user")
public class UserController {
    @GetMapping(path="{user}",produces="text/html")
    public String getUser(@PathVariable String userId) {
        ...
}
```

Controller to process the request is selected based on the `Accept` header

Returns 406 Not Acceptable status code if match the entities

```
@RestController
@RequestMapping(path="/user")
public class UserRestController {
    @GetMapping(path="{user}", produces="application/json")
    public String getUser(@PathVariable String userId) {
        ...
}
```