linguagem de programação ii

prof. lucas nadalete

```
public void getAlunos(Javalin javalin) {
    javalin.get( path: "/alunos", context -> {
        List<Aluno> alunos = alunoService.getAlunos();
        context.json(alunos);
    });
}
```

GET http://localhost:7001/alunos

Pretty Raw Preview Visualize BETA JSON


```
[
    {
        "nome": "Maria",
        "sobrenome": "Clara"
    },
    {
        "nome": "Robson",
        "sobrenome": "Sousa"
    },
    {
        "nome": "Matheus",
        "sobrenome": "Luiz"
    },
    {
        "nome": "Otavio",
        "sobrenome": "Raposo"
    },
    {
        "nome": "Kleber",
        "sobrenome": "Nascimento"
    }
]
```

o que é?

- *web framework* leve para Java e Kotlin
- suporta *Websockets* HTTP2 e solicitações assíncronas.

o que é?

Principais objetivos:

1. simplicidade, propiciando uma ótima experiência aos desenvolvedores
2. interoperabilidade, entre Java e Kotlin.

o que é?

- está mais para uma biblioteca do que para um *framework*: não há necessidade de estender classes ou utilizar *@Annotations*.
- é construído em cima do *Jetty* e tem performance parecida com a do *Jetty* sem nenhuma aplicação.

o início

O Javalin começou como um *fork* do *framework SparkJava*, mas rapidamente foi reescrito.

Também foi influenciado pelo *framework JavaScript* koa.js.

adicionando Javalin ao Projeto Java

Maven | Gradle | SBT | Grape | Leiningen | Buildr | Ivy

```
<dependency>
    <groupId>io.javalin</groupId>
    <artifactId>javalin</artifactId>
    <version>3.6.0</version>
</dependency>
```

Maven | Gradle | SBT | Grape | Leiningen | Buildr | Ivy

```
compile 'io.javalin:javalin:3.6.0'
```

características

GET http://localhost:7001/caracteristicas

Pretty Raw Preview Visualize BETA JSON


```
[
    {
        "nome": "Simples"
    },
    {
        "nome": "Ativo"
    },
    {
        "nome": "Leve"
    },
    {
        "nome": "Flexível"
    },
    {
        "nome": "Interoperável"
    },
    {
        "nome": "Educacional"
    }
]
```

handlers

O Javalin possui três handlers principais:

1. *before-handlers*
2. *Endpoint-handlers*
3. *after-handlers*

Há também *handlers* de erros e exceções.

before handlers

Java | Kotlin

```
app.before(ctx -> {
    // runs before all requests
});
app.before("/path/*", ctx -> {
    // runs before request to /path/*
});
```

- *Before Handlers* são utilizados antes de cada requisição e podem incluir arquivos estáticos, caso seja ativado

endpoint handlers

Java | Kotlin

```
app.get("/", ctx -> {
    // some code
    ctx.json(object);
});

app.post("/", ctx -> {
    // some code
    ctx.status(201);
});
```

- *Endpoint Handlers* são utilizados na ordem em que são definidos. Seus principais métodos são: *POST, GET, UPDATE e DELETE*

after handlers

Java | Kotlin

```java
app.after(ctx -> {
    // run after all requests
});
app.after("/path/*", ctx -> {
    // runs after request to /path/*
});
```

- *After Handlers* são executados após cada solicitação (mesmo se ocorrer uma exceção).

nossos handlers

Em nosso projeto, acabamos por utilizar apenas os endpoint handlers.

endpoint handlers

```
public void postProblema(Javalin javalin) {

    javalin.post( path: "/maratona", context -> {
        Gson gson = new Gson();
        Problema problema = gson.fromJson(context.body(), Problema.class);
        Resultado resultado = problemaService.rodaMaratona(problema);
        context.json(resultado);
    });
}
```

endpoint handlers

```
public void getResultados(Javalin javalin) {
    javalin.get( path: "/resultados", context -> {
        List<Resultado> resultados = problemaService.getResultados();
        context.json(resultados);
    });
}

public void getResultadosId(Javalin javalin) {
    javalin.get( path: "/maratona/id/:id", context -> {
        Resultado resultadoEncontrado =
                problemaService.buscarId(context.pathParam( key: ":id"));
        context.json(resultadoEncontrado);
    });
}
```

endpoint handlers

```
public void getResultadosStatus(Javalin javalin) {
    javalin.get( path: "/maratona/status/:status", context -> {
        List<Resultado> resultadoEncontrado =
                problemaService.buscarStatus(context.pathParam( key: ":status"));
        context.json(resultadoEncontrado);
    });
}

public void getResultadosData(Javalin javalin) {
    javalin.get( path: "/maratona/data/:data", context -> {

        List<Resultado> resultadoEncontrado =
                problemaService.buscarData(context.pathParam( key: ":data"));
        context.json(resultadoEncontrado);
    });
}
```

context

- objeto *Context* fornece ferramentas para tratar requisições http
- engloba *servlet-requests* e *servlet-responses*
- diversos *getters* - principalmente para *requests*, e *setters* - exclusivamente para responses.

request methods

```
ctx.body()                          // get body as string (consumes underlying request body if not cached)
ctx.bodyAsBytes()                   // get body as bytes (consumes underlying request body if not cached)
ctx.bodyAsClass(class)              // get body as class (consumes underlying request body if not cached)
ctx.bodyValidator(class)            // get typed validator for body (consumes underlying body request if not cached)
ctx.uploadedFile(name)              // get uploaded file by name
ctx.uploadedFiles(name)             // get uploaded file(s) by name
ctx.formParam(key)                  // get form parameter
ctx.formParam(key, default)         // get form parameter (or default value)
ctx.formParam(key, class)           // get form parameter as class
ctx.formParam(key, class, default)  // get form parameter (or default value) as class
ctx.formParams(key)                 // get form parameters (multiple)
ctx.formParamMap()                  // get form parameter map
ctx.pathParam(key)                  // get path parameter
ctx.pathParam(key, class)           // get path as class
ctx.pathParamMap()                  // get path parameter map
ctx.basicAuthCredentials()          // get basic auth credentials (username/pwd)
ctx.attribute(key)                  // get request attribute
ctx.attributeMap()                  // get request attribute map
ctx.contentLength()                 // get request content length
ctx.contentType()                   // get request content type
ctx.cookie(name)                    // get request cookie
ctx.cookieMap()                     // get request cookie map
ctx.header(header)                  // get request header
ctx.headerMap()                     // get request header map
```

request methods

```
ctx.host()                          // get request host
ctx.ip()                            // get request host
ctx.isMultipart()                   // check if request is multipart
ctx.isMultipartFormData()           // check if request is multipart/form data
ctx.method()                        // get request method
ctx.path()                          // get request path
ctx.port()                          // get request port
ctx.protocol()                      // get request protocol
ctx.queryParam(key)                 // get query parameter
ctx.queryParam(key, default)        // get query parameter (or default value)
ctx.queryParam(key, class)          // get query parameter as class
ctx.queryParam(key, class, default) // get query parameter (or default value) as class
ctx.queryParams(key)                // get query parameters (multiple)
ctx.queryParamMap()                 // get query parameter map
ctx.queryString()                   // get query string
ctx.scheme()                        // get request scheme
ctx.sessionAttribute(key, value)    // set session attribute (server side attribute)
ctx.sessionAttribute(key)           // get session attribute
ctx.sessionAttributeMap()           // get attribute map
ctx.url()                           // get request url
ctx.fullUrl()                       // get request url + query param
ctx.contextPath()                   // get request context path
ctx.userAgent()                     // get request user agent
```

response methods

```
ctx.result(resultString)                // set a string result that will be sent to the client
ctx.resultString()                      // get the string result that will be sent to the client
ctx.result(resultStream)                // set a stream result that will be sent to the client
ctx.resultStream()                      // get the stream that will be sent to the client
ctx.result(future)                      // set a future result that will be sent to the client (async)
ctx.resultFuture()                      // get the future result that will be sent to the client
ctx.contentType(contentType)            // set the response content type
ctx.header(name, value)                 // set a response header
ctx.redirect(location)                  // send a redirect response to location
ctx.redirect(location, httpStatusCode)  // send a redirect response to location with status code
ctx.status(statusCode)                  // set response status
ctx.status()                            // get response status
ctx.cookie(name, value)                 // set cookie by name and value
ctx.cookie(cookie)                      // set cookie
ctx.removeCookie(name, path)            // remove a cookie
ctx.html(html)                          // call result(string).contentType("text/html")
ctx.json(obj)                           // call result(JavalinJson.toJson(obj)).contentType("application/json")
ctx.json(future)                        // call result(future(JavalinJson.toJson(obj))).contentType("application/json")
ctx.render(filePath, model)             // call html(JavalinRenderer.render(filePath, model)
```

comparação

FRAMEWORK	LINGUAGENS DE PROGRAMAÇÃO			
	JAVA	KOTLIN	GROOVY	SCALA
JAVALIN	V	V	F	F
MICRONAUT	V	V	V	F
QUARKUS	V	F	F	F
PLAY	V	F	F	V

documentação

FRAMEWORK	DOCUMENTAÇÃO			
	RUIM	BOM	EXCELENTE	COMPLEXO
JAVALIN	X			X
MICRONAUT		X		X
QUARKUS			X	X
PLAY		X		X

annotations

FRAMEWORK	Utiliza Annotations	
	SIM	NÃO
JAVALIN		X
MICRONAUT	X	
QUARKUS	X	
PLAY	X	