REST server

Альтман Евгений Анатольевич

Омск, 2019

ОмГУПС, каф. АиСУ

REST server

REST Theory

REST

Representational State Transfer (REST) is architectural style.

- Client–server architecture
- Statelessness (HTTP)
- Layered system

REST Layer

- 0 HTTP
- 1 Resources
- 2 Verbs
- 3 Hypermedia Controls

HTTP

▾ **General**

Request URL: `http://127.0.0.1:8080/api/games`

Request Method: `GET`

Status Code: 🟢 `200 OK`

Remote Address: `127.0.0.1:8080`

Referrer Policy: `no-referrer-when-downgrade`

▾ **Response Headers** view parsed

```
HTTP/1.1 200 OK
Content-Length: 3
Content-Type: application/json; charset=UTF-8
Connection: keep-alive
```

▾ **Request Headers** view source

Accept: `text/html,application/xhtml+xml,application/xml;q=0.9,image`

Accept-Encoding: `gzip, deflate, br`

Accept-Language: `ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7`

Resources

Uniform Resource Identifier (URI):

https://server/resources/

https://server/resources/item

https://server/resources/item/items_resources

REST verbs

4 basic functions of persistent storage (CRUD):

- Create (HTTP method Post)
- Read (Get)
- Update (Put)
- Delete (Delete)

Representation of object

- XML (eXtensible Markup Language)
- JSON (JavaScript Object Notation)
- YAML (Yet Another Markup Language)
- ...

Representation of object

JSON:

```
{
 "firstname": "Sheldon",
 "lastname": "Cooper",
 "data": [
  {"value": 1}
  {"value": 2}
 ]

}
```

REST Hypermedia Controls

HATEOAS (Hypermedia as the Engine of Application State):

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">100.00</balance>
    <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
    <link rel="withdraw" href="https://bank.example.com/accounts/12345/withdraw" />
    <link rel="transfer" href="https://bank.example.com/accounts/12345/transfer" />
    <link rel="close" href="https://bank.example.com/accounts/12345/close" />
</account>
```

(https://ru.wikipedia.org/wiki/HATEOAS)

REST server

Tools

Jackson

```
dependencies {
    compile "org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlin_versi
    compile "io.ktor:ktor-server-netty:$ktor_version"
    compile "ch.qos.logback:logback-classic:$logback_version"
    compile "io.ktor:ktor-server-core:$ktor_version"
    compile "io.ktor:ktor-html-builder:$ktor_version"
    compile "io.ktor:ktor-jackson:$ktor_version"
    testCompile "io.ktor:ktor-server-tests:$ktor_version"
}
```

```
install(ContentNegotiation) { this: ContentNegotiation.Configuration
    jackson { this: ObjectMapper
        enable(SerializationFeature.INDENT_OUTPUT)
    }
}
```

REST server

Data

Player.kt

```
package data

class Player(val name: String)
```

Data.kt

```
package data

class Game(val player1: Player,
           val player2: Player
) {
    val steps = ArrayList<Step>()
    constructor(player1: Player,
                player2: Player,
                word:String):this(player1,player2){
        steps.add(Step( player: 0, cell: 0, char: ' ',word))
    }

    class Step (val player:Int,
                val cell:Int,
                val char:Char,
                val word:String)
}
```

REST server

DTO

Link.kt

```
package rest

class Link(
    val rel: String,
    val href: String
)

class Links(
    val links: Array<Link>
)
```

PlayerDTO.kt

```
class PlayerDTO(
    val name: String,
    val links: Links
){
    constructor(player: Player):
            this(
                player.name,
                Links(Array<Link>( size: 1) { player.getLink() })
            )
}
```

PlayerDTO.kt

```kotlin
import data.Player
import host
import players
import rootURI

fun Player.getLink(): Link {
    val id : Int  = players.indexOf(this)
    return Link( rel: "self",  href: "$host$rootURI/players/$id")
}
```

GameDTO.kt

```
class GameDTO(
    val player1: PlayerDTO,
    val player2: PlayerDTO,
    val steps: ArrayList<Game.Step>,
    val links: Links
){
    constructor(game: Game):
            this(
                PlayerDTO(game.player1),
                PlayerDTO(game.player2),
                game.steps,
                Links(
                    arrayOf(
                        game.getLink(),
                        game.getStepLink()
                    )
                )
            )
}
```

GameDTO.kt

```
class StepDTO(
    val player: PlayerDTO,
    val cell:Int,
    val char:Char,
    val word:String
){
    fun getStep(game: Game): Game.Step {
        val playerNum :Int =
            if(player.name==game.player1.name) 1 else 2
        return Game.Step(playerNum, cell, char, word)
    }
}
```

GameDTO.kt

```
import data.Game
import games
import host
import rootURI

fun Game.getLink(): Link {
    val id :Int = games.indexOf(this)
    return Link( rel: "self", href: "$host$rootURI/games/$id")
}

fun Game.getStepLink(): Link {
    val id :Int = games.indexOf(this)
    return Link( rel: "steps", href: "$host$rootURI/games/$id/steps")
}
```

REST server

REST

GameREST.kt

```
fun Application.gamesRest(){
    routing { this: Routing

        get( path: "$rootURI/games") { this: PipelineContext<Unit, ApplicationCall>
            call.respond(games.map { GameDTO(it) })
        }
        post( path: "$rootURI/games") { this: PipelineContext<Unit, ApplicationCal
            val game : Game = call.receive<Game>()
            games.add(game)
            call.respond(GameDTO(game))
        }
```

GameREST.kt

```
get( path: "$rootURI/games/{id}") { this: PipelineContext<Unit, ApplicationCall>
    val id :Int  = call.parameters["id"]?.toInt()?:0
    call.respond(GameDTO(games[id]))
}

post( path: "$rootURI/games/{id}/steps"){ this: PipelineContext<Unit, Applicatio
    val id :Int  = call.parameters["id"]?.toInt()?:0
    val step :StepDTO  = call.receive<StepDTO>()
    val game :Game  = games[id]
    game.steps.add(step.getStep(game))
    call.respond(GameDTO(games[id]))
}
```

PlayerREST.kt

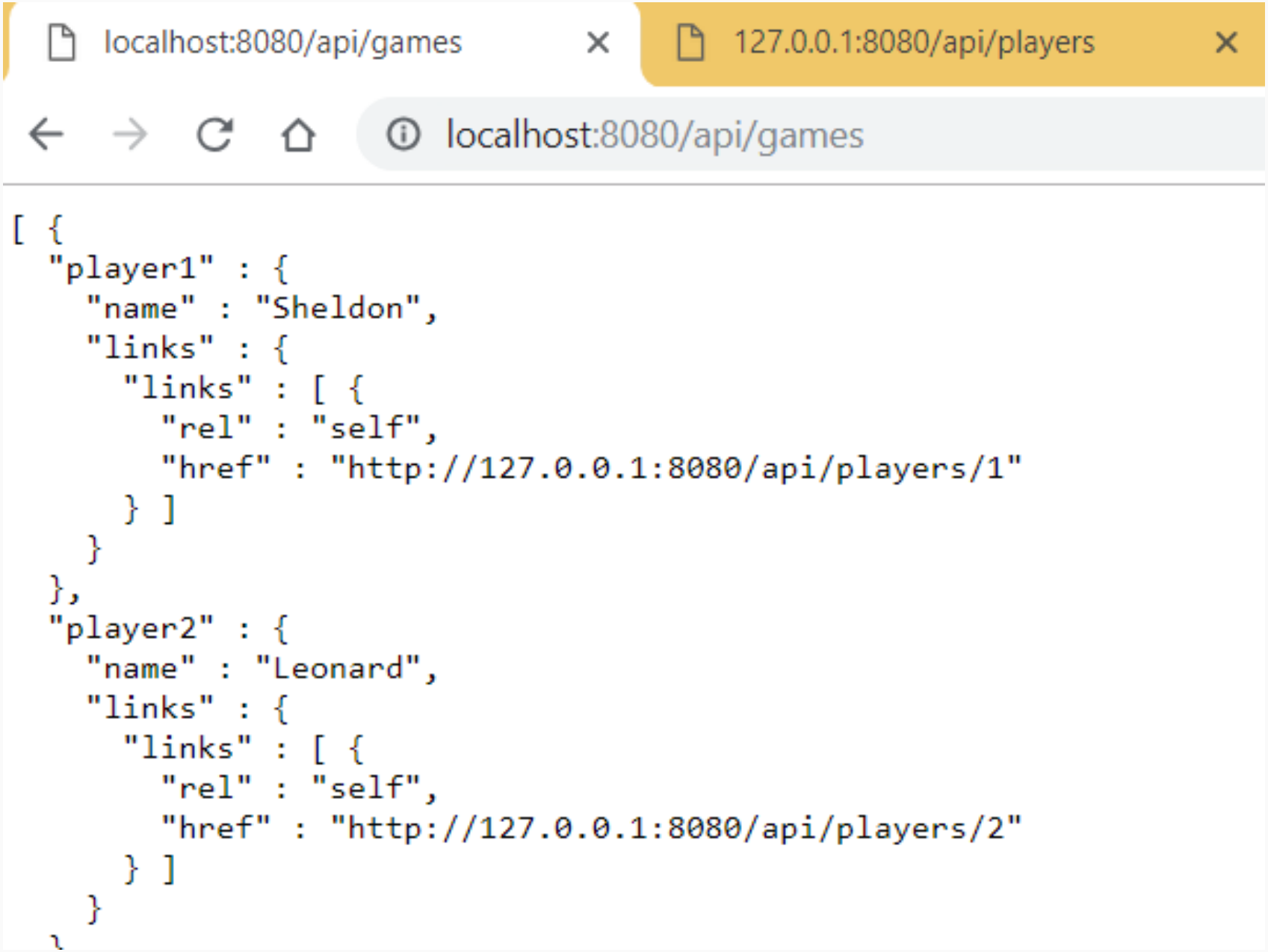
```
get( path: "$rootURI/players") { this: PipelineContext<Unit, ApplicationCall>
    call.respond(players.map { PlayerDTO(it) })
}
post( path: "$rootURI/players") { this: PipelineContext<Unit, ApplicationCall>
    val player : Player = call.receive<Player>()
    players.add(player)
    call.respond(PlayerDTO(player))
}

get( path: "$rootURI/players/{id}") { this: PipelineContext<Unit, ApplicationCall
    val id : Int = call.parameters["id"]?.toInt()?:0
    call.respond(PlayerDTO(players[id]))
}
```

REST server

Test

Check with Chrome



Test with kotlin.test

```
@Test
fun testPlayers(){
    withTestApplication({ this: Application
        module()
        playersRest()
        gamesRest()
    }) { this: TestApplicationEngine
        handleRequest(HttpMethod.Get, uri: "/api/players").apply { this: TestApplicationCall
            assertEquals(HttpStatusCode.OK, response.status())
            val players : ArrayList<PlayerDTO> = jacksonObjectMapper()
                .readValue<ArrayList<PlayerDTO>>( content: response.content ?: "")
            assertEquals( expected: 4, players.size)
        }
```

Test with kotlin.test

```
    handleRequest(HttpMethod.Post, uri: "/api/players"){ this: TestApplicationRequest
        setBody(jacksonObjectMapper().writeValueAsString(Player( name: "Howard")))
        addHeader( name: "Content-Type", value: "application/json")
        addHeader( name: "Accept", value: "application/json")
    }.apply { this: TestApplicationCall
        assertEquals(HttpStatusCode.OK, response.status())
    }

    ^withTestApplication handleRequest(HttpMethod.Get, uri: "/api/players").apply { this: TestApp
        assertEquals(HttpStatusCode.OK, response.status())
        val players : ArrayList<PlayerDTO> = jacksonObjectMapper()
            .readValue<ArrayList<PlayerDTO>>( content: response.content ?: "")
        assertEquals( expected: 5, players.size)
        assertEquals( expected: "Howard", players.last().name)
    }
}
```

REST server

Summary

REST Theory

- REST style
- REST Layer
- HTTP
- Resource identification
- REST verbs
- Object representation
- HATEOUS

Application structure

- Dependency and module
- Data layer
- DTO layer
- REST layer
- Test REST