

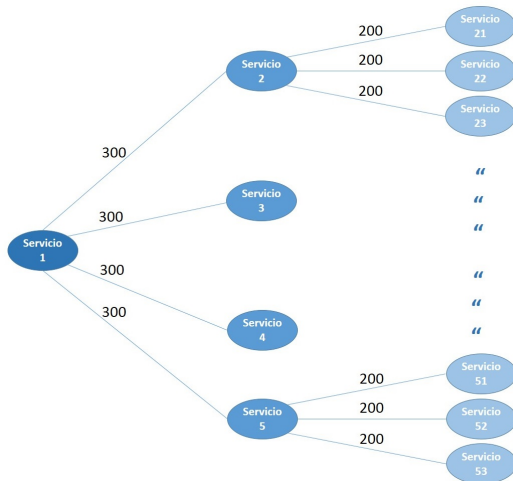
Programación asincrónica

Carlos Lombardi

Universidad Nacional de Quilmes – Argentina

Sesiones sobre microservicios

Asincronismo – por qué



Ejecución sincrónica:
 $(300 + 200 * 3) * 4 = 3600$ ms.

Ejecución asincrónica:
 $300 + 200 = 500$ ms.

Callbacks

En lugar de

```
let argPrefix = getPrefix('ARG')
```

Accedemos a valores via callbacks.

```
let argPrefix = null
getPrefix('ARG', function(error, result) {
  argPrefix = result
})
```

Callbacks – salen del flujo de ejecución

```
getPrefix('ARG', function(error, result) {  
    console.log('Prefijo tel. Argentina: ' + result)  
})  
console.log('después de la invocación')
```

¿En qué orden aparecen los mensajes?

Callbacks – uso ingenuo

Un caso similar, con varias invocaciones.

```
let argPrx = null ; let braPrx = null ; let pryPrx = null
getPrefix('ARG', function(error, result) {
    argPrx = result
})
getPrefix('BRA', function(error, result) {
    braPrx = result
})
getPrefix('PRY', function(error, result) {
    pryPrx = result
})
console.log('Prefijo tel Argentina:  ' + argPrx)
console.log('Prefijo tel Brasil:    ' + braPrx)
console.log('Prefijo tel Paraguay:   ' + pryPrx)
```

Callbacks – corregimos

```
let argPrx = null ; let braPrx = null ; let pryPrx = null
getPrefix('ARG', function(error, result) {
  argPrx = result
  getPrefix('BRA', function(error, result) {
    braPrx = result
    getPrefix('PRY', function(error, result) {
      pryPrx = result
      console.log('Prefijo tel Argentina: ' + argPrx)
      console.log('Prefijo tel Brasil: ' + braPrx)
      console.log('Prefijo tel Paraguay: ' + pryPrx)
    })
  })
})
```

Callbacks – corregimos

```
let argPrx = null ; let braPrx = null ; let pryPrx = null
getPrefix('ARG', function(error, result) {
  argPrx = result
  getPrefix('BRA', function(error, result) {
    braPrx = result
    getPrefix('PRY', function(error, result) {
      pryPrx = result
      console.log('Prefijo tel Argentina: ' + argPrx)
      console.log('Prefijo tel Brasil: ' + braPrx)
      console.log('Prefijo tel Paraguay: ' + pryPrx)
    })
  })
})
```

Bienvenidos al **callback hell** ...

Callbacks – corregimos

```
let argPrx = null ; let braPrx = null ; let pryPrx = null
getPrefix('ARG', function(error, result) {
  argPrx = result
  getPrefix('BRA', function(error, result) {
    braPrx = result
    getPrefix('PRY', function(error, result) {
      pryPrx = result
      console.log('Prefijo tel Argentina: ' + argPrx)
      console.log('Prefijo tel Brasil: ' + braPrx)
      console.log('Prefijo tel Paraguay: ' + pryPrx)
    })
  })
})
```

Bienvenidos al **callback hell** ...

... y las llamadas no dejan de ser **síncronicas**.

Promises (1)

Alternativa a los callbacks.

En lugar de

```
let argPrefix = null
getPrefix('ARG', function(error, result) {
    argPrefix = result
})
```

Hacemos

```
let argPrefix = null
getPrefixPromise('ARG')
    .then(function(result) { argPrefix = result })
```

Promises (2)

Una **Promise** es un **objeto** que encapsula una operación asincrónica.

- ▶ Entiende los mensajes `then` y `catch`. Los parámetros de estos mensajes son **funciones**.
- ▶ Cuando la operación termina, se invoca a la función del `then`, con el resultado como parámetro.
- ▶ Si la operación da error, se invoca a la función del `catch`, con el resultado como parámetro.

```
let argPrefix = null
getPrefixPromise('ARG')
  .then(function(result) { argPrefix = result })
  .catch(function(theError) { console.log(theError) })
```

Promises – salimos del callback hell

```
let argPrx = null ; let braPrx = null ; let pryPrx = null
getPrefixPromise('ARG')
  .then(function(result) {
    argPrx = result
    return getTempPromise('BRA')
  }).then(function(result) {
    braPrx = result
    return getTempPromise('PRY')
  }).then(function(result) {
    pryPrx = result
    console.log('Prefijo tel Argentina:  ' + argPrx)
    console.log('Prefijo tel Brasil:    ' + braPrx)
    console.log('Prefijo tel Paraguay:   ' + pryPrx)
  })
```

De callback a promise

- ▶ Se usa constructor de la clase Promise.
- ▶ El parámetro es una función que llama a la función original.
- ▶ El secreto está en cómo armar el callback.

```
function getPrefixPromise(country) {  
  return new Promise(function(fulfill,reject) {  
    getPrefix(country, function(theError,response,body) {  
      if (theError) { reject(theError) }  
      else { fulfill(body) }  
    })  
  })  
}
```

Las funciones fullfill y reject van a corresponder a los argumentos de la invocación a `.then` y `.catch`.

Promises – paralelizamos operaciones

```
let argPrx = null ; let braPrx = null ; let pryPrx = null
Promise.all([
  getPrefixPromise('ARG'),
  getPrefixPromise('BRA'),
  getPrefixPromise('PRY')])
.spread(function(argPrx, braPrx, pryPrx) {
  console.log('Prefijo tel Argentina: ' + argPrx)
  console.log('Prefijo tel Brasil: ' + braPrx)
  console.log('Prefijo tel Paraguay: ' + pryPrx)
})
```

- ▶ Uso de `Promise.all` para paralelizar requests.
- ▶ Uso de `.spread` en lugar de `.then`.

Promise.all – el manejo de error sigue siendo simple

```
let argPrx = null ; let braPrx = null ; let pryPrx = null
Promise.all([
  getPrefixPromise('ARG'),
  getPrefixPromise('BRA'),
  getPrefixPromise('PRY')])
.spread(function(argPrx, braPrx, pryPrx) {
  console.log('Prefijo tel Argentina: ' + argPrx)
  console.log('Prefijo tel Brasil: ' + braPrx)
  console.log('Prefijo tel Paraguay: ' + pryPrx)
})
.catch(function(theError) {
  ... manejo del error ...
})
```