Asincronía

Asincronía

JS es por naturaleza asíncrono:

- Eventos
- AJAX
- Carga de recursos

Asincronía

¿Qué significa asíncrono?

```
function asincrona() {
  var random = Math.floor(Math.random() * 100);
  setTimeout(function() {
    return random;
  }, random);
}
```

Asincronía

```
function asincrona(callback) {
  var random = Math.floor(Math.random() * 1000);
  setTimeout(function() {
    callback(random);
  }, random);
}

asincrona(function(valor) {
  alert(valor);
});
```

CPS

Los callbacks tienen muchas ventajas

- Muy fáciles de entender e implementar
- Familiares para el programador JavaScript
- Extremadamente flexibles (clausuras, funciones de primer orden, etc, ...)
- Un mecanismo universal de asincronía/continuaciones

Pero...

CPS

```
var fs = require("fs");
fs.exists("./hola.txt", function(exists) {
  if (exists) {
    fs.readFile("./hola.txt", function(err, data) {
      if (err) {
        // MANEJO DE ERROR
      } else {
        fs.writeFile("./copia.txt", data, function(err) {
          if (err) {
            // MANEJO DE ERROR
          } else {
            console.log("OK!");
          }
        })
      }
    })
  } else {
    // MANEJO DE ERROR
  }
});
```

CPS

```
var fs = require("fs");
fs.exists("./hola.txt", function(exists) {
  if (exists) {
    fs.readFile("./hola.txt", function(err, data) {
      if (err) {
        // MANEJO DE ERROR
      } else {
        fs.writeFile("./copia.txt", data, function(err) {
          if (err) {
            // MANEJO DE ERROR
          } else {
            console.log("OK!");
          }
        })
      }
    })
  } else {
    // MANEJO DE ERROR
  }
});
```

CPS

```
var fs = require("fs");
fs.exists("./hola.txt", function(exists) {
  if (exists) {
    fs.readFile("./hola.txt", function(err, data) {
      if (err) {
        // MANEJO DE ERROR
      } else {
        fs.writeFile("./copia.txt", data, function(err) {
          if (err) {
            // MANEJO DE ERROR
          } else {
            console.log("OK!");
          }
        })
      }
    })
    else {
    // MANEJO DE ERROR
  }
});
```

CPS

```
var fs = require("fs");
fs.exists("./hola.txt", function(exists) {
  if (exists) {
    fs.readFile("./hola.txt", function(err, data) {
      if (err) {
        // MANEJO DE ERROR
      } else {
        fs.writeFile("./copia.txt", data, function(err) {
          if (err) {
            // MANEJO DE ERROR
          } else {
            console.log("OK!");
          }
        })
      }
    })
    else {
    // MANEJO DE ERROR
  }
});
```

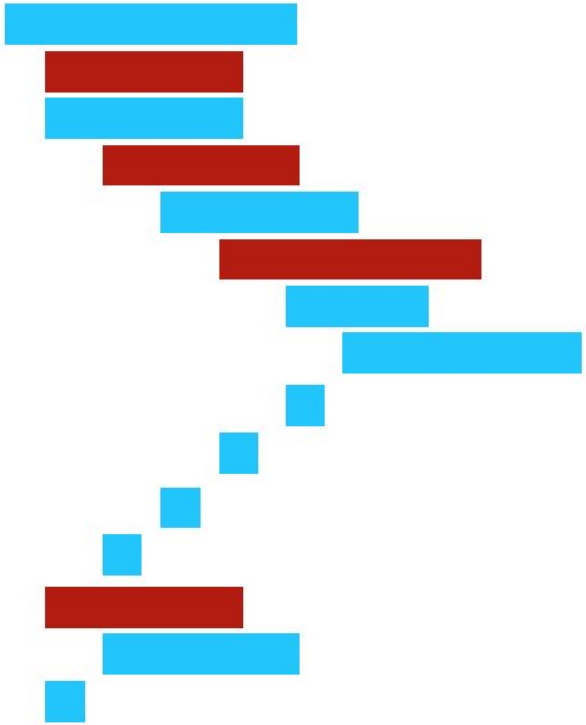
CALLBACK HELL

redradix

CPS

```
var fs = require("fs");
fs.exists("./hola.txt", function(exists) {
  if (exists) {
    fs.readFile("./hola.txt", function(err, data) {
      if (err) {
        // MANEJO DE ERROR
      } else {
        fs.writeFile("./copia.txt", data, function(err) {
          if (err) {
            // MANEJO DE ERROR
          } else {
            console.log("OK!");
          }
        })
      }
    })
  } else {
    // MANEJO DE ERROR
  }
});
```

CPS vs promesas



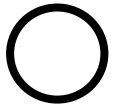
Promesas

Una manera alternativa de modelar asincronía

- Construcción explícita del flujo de ejecución
- Separación en bloques consecutivos
- Manejo de errores más controlado
- Combinación de diferentes flujos asíncronos

Promesas

Una promesa = un flujo de ejecución

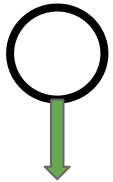


```
promesa.then(function() {
  // bloque
  return readFilePromise("./hola.txt");
})
.then(function(data) {
  // bloque
  return writeFilePromise("./copia.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
});
```

Promesas

Una promesa = un flujo de ejecución

```
promesa.then function() {
  // bloque
  return readFilePromise("./hola.txt");
})
.then(function(data) {
  // bloque
  return writeFilePromise("./copia.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
});
```

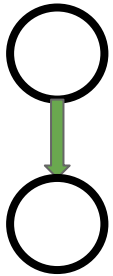


redradix

Promesas

Una promesa = un flujo de ejecución

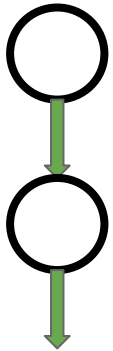
```
promesa.then(function() {
  // bloque
  return readFilePromise("./hola.txt");
})
.then(function(data) {
  // bloque
  return writeFilePromise("./copia.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
});
```



Promesas

Una promesa = un flujo de ejecución

```
promesa.then(function() {
  // bloque
  return readFilePromise("./hola.txt");
})
.then(function(data) {
  // bloque
  return writeFilePromise("./copia.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
});
```

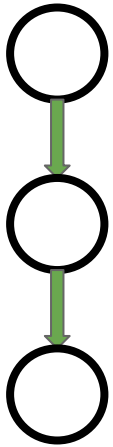


redradix

Promesas

Una promesa = un flujo de ejecución

```
promesa.then(function() {
  // bloque
  return readFilePromise("./hola.txt");
})
.then(function(data) {
  // bloque
  return writeFilePromise("./copia.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
});
```



Promesas

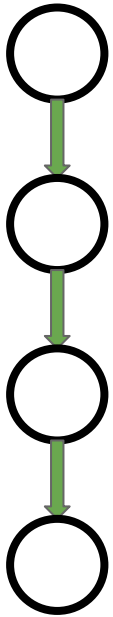
Una promesa = un flujo de ejecución

```
promesa.then(function() {
  // bloque
  return readFilePromise("./hola.txt");
})
.then(function(data) {
  // bloque
  return writeFilePromise("./copia.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
});
```

Promesas

Una promesa = un flujo de ejecución

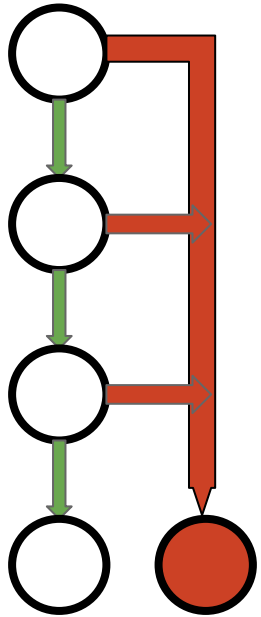
```
promesa.then(function() {
  // bloque
  return readFilePromise("./hola.txt");
})
.then(function(data) {
  // bloque
  return writeFilePromise("./copia.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
});
```



Promesas

Una promesa = un flujo de ejecución

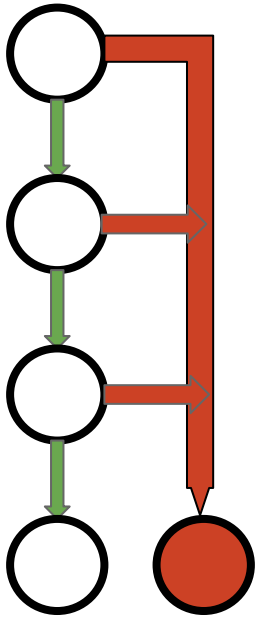
```
promesa.then(function() {
  // bloque
  return readFilePromise("./hola.txt");
})
.then(function(data) {
  // bloque
  return writeFilePromise("./copia.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
});
```



redradix

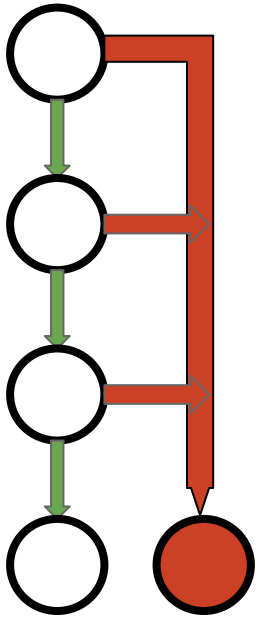
Promesas

Una promesa = un flujo de ejecución



Promesas

¡Pero aún no hemos ejecutado nada! Solamente hemos definido el flujo de ejecución.



Promesas

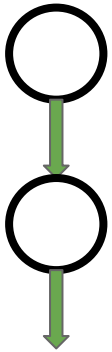
¿Ventajas?

- Código mucho más ordenado y más legible
- Mejor control de errores
- Podemos manipular el flujo
    - Añadir nuevas etapas
    - Devolverlo en funciones
    - Pasarlo como parámetro
- Podemos combinar varios flujos

Promesas

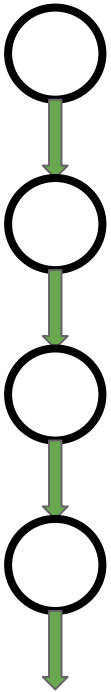
```
function copyFile(from, to) {
  return readFilePromise(from)
  .then(function(data) {
    // bloque
    return writeFilePromise(to);
  });
}
copyFile("./hola.txt", "./copia.txt")
.then(function() {
  return copyFile("./otraCosa.txt", "./copia2.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
  console.log("Oops!");
})
```

Promesas



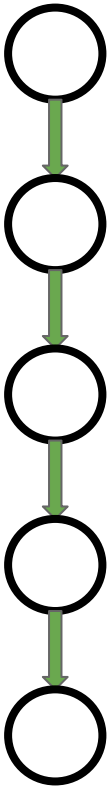
```
function copyFile(from, to) {
  return readFilePromise(from)
  .then(function(data) {
    // bloque
    return writeFilePromise(to);
  });
}
copyFile("./hola.txt", "./copia.txt")
.then(function() {
  return copyFile("./otraCosa.txt", "./copia2.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
  console.log("Oops!");
})
```

Promesas



```
function copyFile(from, to) {
  return readFilePromise(from)
  .then(function(data) {
    // bloque
    return writeFilePromise(to);
  });
}
copyFile("./hola.txt", "./copia.txt")
.then(function() {
  return copyFile("./otraCosa.txt", "./copia2.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
  console.log("Oops!");
})
```

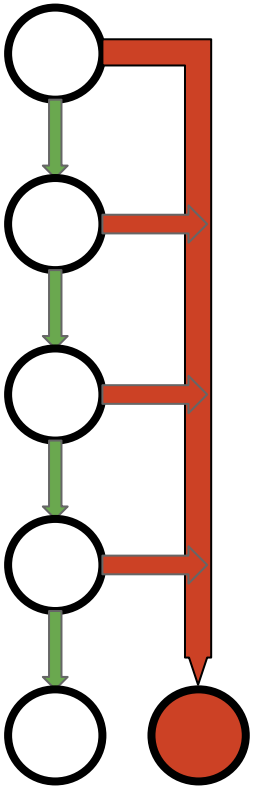
Promesas



```
function copyFile(from, to) {
  return readFilePromise(from)
  .then(function(data) {
    // bloque
    return writeFilePromise(to);
  });
}
copyFile("./hola.txt", "./copia.txt")
.then(function() {
  return copyFile("./otraCosa.txt", "./copia2.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
  console.log("Oops!");
})
```

redradix

Promesas



```
function copyFile(from, to) {
  return readFilePromise(from)
  .then(function(data) {
    // bloque
    return writeFilePromise(to);
  });
}
copyFile("./hola.txt", "./copia.txt")
.then(function() {
  return copyFile("./otraCosa.txt", "./copia2.txt");
})
.then(function() {
  console.log("listo!");
})
.fail(function(err) {
  console.log("Oops!");
})
```

redradix

Promesas

.then(success, [error])

- Concatena bloques
- El nuevo bloque (success)...
  - Sólo se ejecuta si el anterior se ha ejecutado sin errores
  - Recibe como parámetro el resultado del bloque anterior
  - Devuelve el valor que se le pasará el siguiente bloque
    - Si es un dato inmediato, se pasa tal cual
    - Si es una promesa, se resuelve antes de llamar al siguiente bloque

Promesas

`.then(success, [error])`

- El segundo parámetro pone un manejador de error
    - Equivalente a llamar a .fail(error)
- `.then(...)` siempre devuelve una nueva promesa

Promesas

```
var promesa = readFilePromise("./hola.txt");

var promesa2 = promesa.then(function(data) {
  console.log("Contenido del fichero: ", data);
}, function(err) {
  console.log("Ooops!", err);
})
```

Promesas

```
var promesa = readFilePromise("./hola.txt");
```

```
var promesa2 = promesa.then(function(data) {
  console.log("Contenido del fichero: ", data);
}, function(err) {
  console.log("Ooops!", err);
})
```

Promesas

```
var promesa = readFilePromise("./hola.txt");

var promesa2 = promesa.then(function(data) {
  console.log("Contenido del fichero: ", data);
}, function(err) {
  console.log("Ooops!", err);
})
```

Promesas

```
var promesa = readFilePromise("./hola.txt");

var promesa2 = promesa.then(function(data) {
  console.log("Contenido del fichero: ", data);
}, function(err) {
  console.log("Ooops!", err);
})
```

Promesas

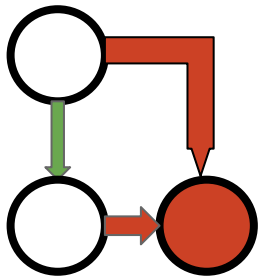
```
var promesa = readFilePromise("./hola.txt");

var promesa2 = promesa.then(function(data) {
  console.log("Contenido del fichero: ", data);
}, function(err) {
  console.log("Ooops!", err);
})
```

Promesas

```
var promesa = readFilePromise("./hola.txt");

var promesa2 = promesa.then(function(data) {
  console.log("Contenido del fichero: ", data);
}, function(err) {
  console.log("Ooops!", err);
})
```

Promesas

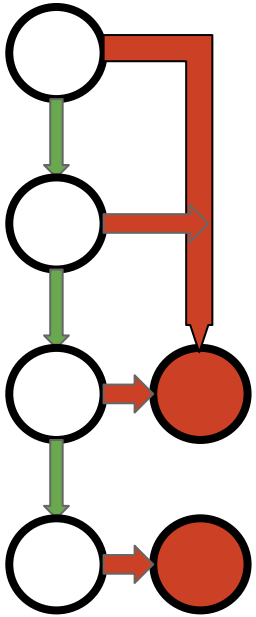
```
var promesa = readFilePromise("./hola.txt");

var promesa2 = promesa.then(function(data) {
  console.log("Contenido del fichero: ", data);
}, function(err) {
  console.log("Ooops!", err);
})
```



redradix

Promesas

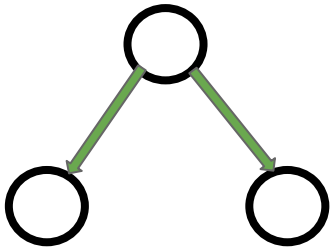
```
var promesa = readFilePromise("./hola.txt");
promesa.then(function(data) {
  return 1;
})
.then(function(uno) {
  return 2;
}, function(err) {
  console.log("Oh, oh...");
})
.then(function(dos) {
  return 3;
})
.fail(function(err) {
  console.log("Oops!");
});
```



redradix

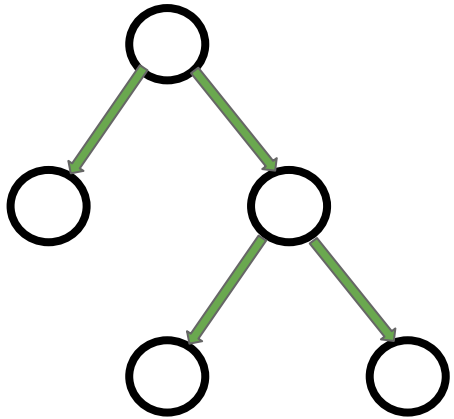
Promesas

```
var promesa = readFilePromise("./hola.txt");

var promesa2 = promesa.then(function(data) {
  return 1;
});

var promesa3 = promesa.then(function(data) {
  return 2;
});
```



Promesas

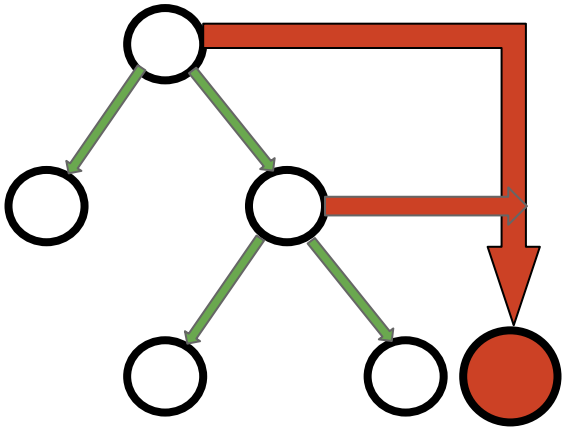
```
var promesa = readFilePromise("./hola.txt");

var promesa2 = promesa.then(function(data) {
  return 1;
});

var promesa3 = promesa.then(function(data) {
  return 2;
});
promesa3.then(function(dos) {
  console.log("Ping!");
});
promesa3.then(function(dos) {
  console.log("Pong!");
});
```



redradix

Promesas

```
var promesa = readFilePromise("./hola.txt");
var promesa2 = promesa.then(function(data) {
  return 1;
});
var promesa3 = promesa.then(function(data) {
  return 2;
});
promesa3.then(function(dos) {
  console.log("Ping!");
});
promesa3.then(function(dos) {
  console.log("Pong!");
});
promesa3.fail(function(err) {
  console.log("Fail!");
});
```



redradix

Promesas en ES6

```
const promise = new Promise((resolve, reject) => {
  const prob = Math.rand()
  if (prob < 0.5) {
    resolve(prob)
  } else {
    reject(prob)
  }
})

promise.then(console.log)
promise.catch(console.error)
```

A trastear

```
var promise = new Promise((resolve, reject) => {
})

promise.then(function() {
  console.log("hola");
})
.then(function() {
  console.log("soy un flujo de ejecución");
})
.then(function() {
  console.log("expresado con promesas");
});
```

Promesas

Una promesa tiene tres estados:

- Pendiente

- Resuelta - cuando llamamos a resolve(valor)

- Rechazada - cuando llamamos a reject(valor)

A trastear

```
var promise = new Promise((resolve, reject) => {
    resolve()
})
promise.then(function() {
  console.log("hola");
})
.then(function() {
  console.log("soy un flujo de ejecución");
})
.then(function() {
  console.log("expresado con promesas");
});
```

A trastear

```
var promise = new Promise((resolve, reject) => {
    reject()
})
promise.then(function() {
  console.log("hola");
})
.then(function() {
  console.log("soy un flujo de ejecución");
})
.then(function() {
  console.log("expresado con promesas");
});
```

A trastear

```
var promise = new Promise((resolve, reject) => {
    lalala
})
promise.then(function() {
  console.log("hola");
})
.then(function() {
  console.log("soy un flujo de ejecución");
})
.then(function() {
  console.log("expresado con promesas");
});
```

A trastear

```
var promise = new Promise((resolve, reject) => {
    lalala
})
promise.then(function() {
  console.log("hola");
})
.then(function() {
  console.log("soy un flujo de ejecución ");
})
.then(function() {
  console.log("expresado con promesas ");
})
.catch(function(err) {
  console.log('algo ha ido mal ', err);
})
```

A trastear++

¿Cómo podemos “promisificar” `fs.readFile` para que devuelva una promesa?

```
fs.readFile('/etc/passwd', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

A trastear++

La el método `fs.stat` obtiene el estado de un fichero (última modificación, creación, tamaño....)

```
fs.stat('/tmp/world', (err, stats) => {
  if (err) throw err;
  console.log(`stats: ${JSON.stringify(stats)}`);
});
```

Intenta “promisificarlo”!

Promesas

La mayoría de funciones asíncronas de nodejs siguen el mismo convenio:

- Reciben un callback como último parámetro

- El callback recibe uno o más parámetros
  - El primero indica si ha habido un error
  - Si el primero es `falsy`, los siguientes son el resultado de la operación

Promesas

Casi todos los métodos de utilidades en node tienen la misma forma:

```
modulo.metodo(parametros, function(err, resultado) => {
  if (err) throw err;
  hacemos lo que sea con resultado;
});
```

A trastear++

Escribe una función `nodefcall` que:

- Reciba una función en “formato node”
- Devuelva una promesa
- Si la función devuelve error, la promesa falla
- Si la función ejecuta bien, resuelve la promesa con todos los valores que recibe el callback excepto el error

A trastear++

```
var fs = require('fs')

function nodefcall(fn) {
  // aquí tu código
}

nodefcall(fs.readFile, './files/uno.txt')
.then((data) => {
  console.log(data.toString())
})
```

redradix

Continuará...

redradix