

Redis

¿Qué es Redis?

Una base de datos NoSQL...

- Clave/valor
- Extremadamente rápida
- Persistencia y transacciones
- Estructuras de datos
 - Listas
 - Conjuntos
 - Hashes
- Pub/sub

¿Qué es Redis?

Un servidor de estructuras de datos

- Sin tablas, ni queries ni JOINS
- Una manera de pensar muy diferente
- Muchas queries muy rápidas
- Muy fácil de aprender, muy fácil de usar **mal**

¿Qué es Redis?

Redis es una opción **estupenda** para...

- Datos de acceso inmediato
 - Sesiones
 - Cachés
- Escrituras muy rápidas
 - Logs
 - Conteos y estadísticas
- Canal de comunicación pub/sub
 - Comunicación entre procesos (workers, big data)
 - Chats :)

¿Qué NO es Redis?

Redis es una opción **terrible** para...

- Relaciones de datos complejas
- Búsquedas

¿Para qué sirve?

Mi consejo:

- NO bases tu app en redis
- A no ser que sea muy simple o sepas muy bien lo que haces
- Utiliza Redis como complemento a tu BBDD
 - Cachés
 - Estadísticas
 - Workers
 - Sesiones
 - Registros
 - Colas

La idea general

Redis es un gran Hash de claves

- No hay concepto de tabla/colección
- El valor fundamental es la cadena (string)
- Pero una clave puede contener un valor más complejo
 - Hashes (string -> string)
 - Listas (de strings)
 - Sets (de strings)
 - Sets ordenados (de strings)

Requisitos

Necesitas:

- Tener instalado y arrancado redis
- Tener a mano la documentación
 - ➔ <http://redis.io/commands>
- `npm install hiredis redis`
- Como alternativa:
 - ➔ c9.io
 - `nada-nix install redis`
 - `npm install hiredis redis`
 - `redis-server --port 16379 --bind $IP`

Primeros pasos

Primero, necesitas conectarte al servidor

```
var redis = require("redis");  
  
var client = redis.createClient();  
// c9.io:  
// var client = redis.createClient(16379, process.env.IP);
```

Primeros pasos

Ahora puedes mandar comandos a Redis

- `client.nombreDelComando(arg, callback)`
- `client.nombreDelComando(arg1, arg2, callback)`
- callback: `function(err, value) { }`

SET / GET

Guardar y recuperar un valor

```
var redis = require("redis"),
    client = redis.createClient();


client.set("miClave", "miValor", function(err, val) {
  console.log(arguments);
  client.get("miClave", function(err, value) {
    console.log("valor: ", value);
  });
});
```

SET / GET

Guardar y recuperar un valor

```
var redis = require("redis"),
    client = redis.createClient();

client.set("miClave", "miValor", function(err, val) {
  console.log(arguments);
  client.get("miClave", function(err, value) {
    console.log("valor: ", value);
  });
});
```



SET / GET

Guardar y recuperar un valor

```
var redis = require("redis"),  
    Promise = require("bluebird"),  
    client = redis.createClient();
```

```
Promise.promisifyAll(redis);
```

```
client.setAsync("miClave", "miValor")  
  .then(function(val) {  
    console.log(val);  
    return client.GetAsync("miClave");  
  });  
})  
  .then(function(val){ console.log(val) });
```

SET / GET

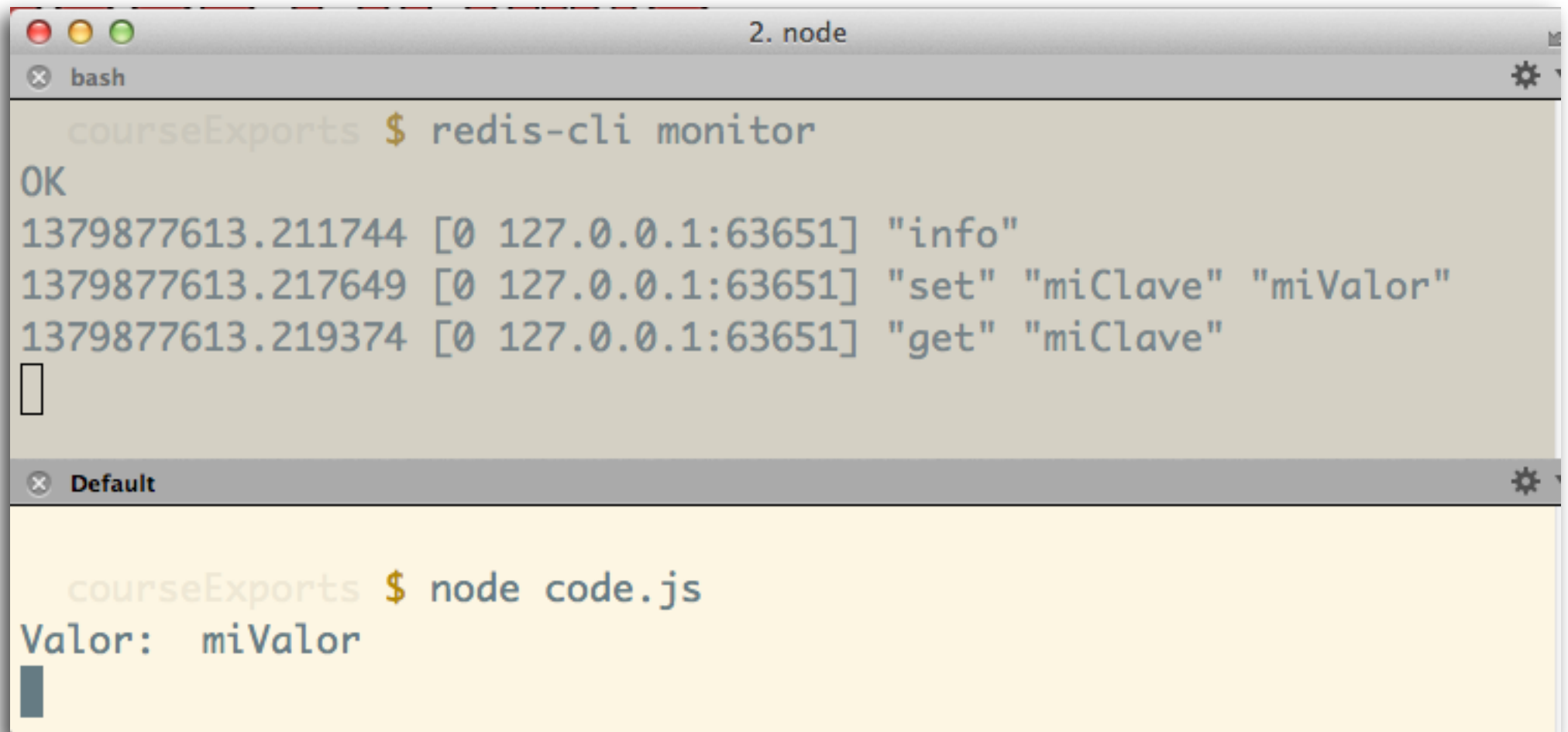
Guardar y recuperar un valor



```
2. redis-cli  
  
courseExports $ redis-cli  
redis 127.0.0.1:6379> SET miClave miValor  
OK  
redis 127.0.0.1:6379> GET miClave  
"miValor"  
redis 127.0.0.1:6379> █
```

Un truco: redis-cli monitor

Para ver qué está pasando en Redis



```
2. node
bash
courseExports $ redis-cli monitor
OK
1379877613.211744 [0 127.0.0.1:63651] "info"
1379877613.217649 [0 127.0.0.1:63651] "set" "miClave" "miValor"
1379877613.219374 [0 127.0.0.1:63651] "get" "miClave"
█

Default
courseExports $ node code.js
Valor:  miValor
█
```

Redis = Strings!!

Cuidado con los valores. Han de ser **siempre strings**.

```
var redis = require("redis"),  
    Promise = require("bluebird"),  
    client = redis.createClient();  
//...
```

```
client.setAsync("miClave", {un: "objeto"})  
  .then(function() {  
    return client.getAsync("miClave");  
  })  
  .then(function(value) {  
    console.log("Valor: ", value); // Valor: [object Object]  
  });
```


Redis = Strings!!

JSON.stringify(obj) -> String

```
var redis = require("redis"),
    client = redis.createClient();
//...
var obj = {propiedad: "valor"};
client.setAsync("miClave", JSON.stringify(obj))
    .then(function() {
        return client.GetAsync("miClave");
    })
    .then(function(value) {
        console.log("Valor: ", value);
        // Valor: {"propiedad": "valor"}
    });
```

Redis = Strings!!

Cuidado que redis.GET nos devuelve también un String, no un objeto

```
var redis = require("redis"),
    Promise = require("bluebird"),
    client = redis.createClient();

//...

var obj = {propiedad: "valor"};
client.setAsync("miClave", JSON.stringify(obj))
    .then(function() {
        return client.GetAsync("miClave");
    })
    .then(function(value) {
        console.log(typeof(value));
        // string
    });
```

Redis = Strings!!

Usamos el inverso de JSON.stringify ->
JSON.parse

```
var redis = require("redis"),  
    Promise = require("bluebird"),  
    client = redis.createClient();  
  
//...  
var obj = {propiedad: "valor"};  
client.setAsync("miClave", JSON.stringify(obj))  
  .then(function() {  
    return client.getAsync("miClave");  
  })  
  .then(JSON.parse)  
  .then(function(valueObj) {  
    console.log(valueObj.propiedad);  
    // string  
  });
```

DEL / EXISTS / TYPE / RENAME

Operaciones con claves

- DEL: borrar una clave
- EXISTS: comprobar si una clave existe
- TYPE: el tipo de valor almacenado en una clave
- RENAME: cambiar el nombre de la calve

DEL / EXISTS / TYPE / RENAME

```
var redis = require("redis"),
    Promise = require("bluebird"),
    client = redis.createClient();

Promise.promisifyAll(client);
client.existsAsync("MiClave")
  .then(function(exists) {
    console.log( exists? "Existe!" : "No existe..." );
    return client.setAsync("MiClave", "MiValor");
  })
  .then(function() {
    return client.renameAsync("MiClave", "MyKey");
  })
  .then(function() {
    return client.typeAsync("MyKey");
  })
  .then(function(type) {
    console.log("MyKey es de tipo", type);
  });
```

Operaciones con cadenas

- APPEND: añade el valor a la cadena
- DECR/INCR: Decrementa/incrementa el valor en 1
- DECRBY/INCRBY: Dec/inc el valor en N
- GETSET: Modifica el valor y devuelve el viejo
- STRLEN: Longitud del valor

Operaciones con cadenas

```
//...promisifyAll
```

```
client.setAsync("miClave", 1)
  .then(function() {
    return client.incrbyAsync("miClave", 10);
  }).then(function() {
    return client.decrAsync("miClave");
  }).then(function() {
    return client.getsetAsync("miClave", "fin");
  }).then(function(valor) {
    console.log("VALOR: ", valor);
    return client.strlenAsync("miClave")
  }).then(function(len) {
    console.log("Len: ", len);
  });
```

Operaciones múltiples

- MGET: Trae el valor de varias claves
- MSET: Modifica el valor de varias claves

```
//promisifyAll...
```

```
client.msetAsync("miClave", 1, "otraClave",  
2)  
.then(function() {  
    return client.mgetAsync("miClave",  
"otraClave");  
})  
.then(function(values) {  
    console.log(values[0], ",", values[1]);  
});
```


Listas

- LPUSH/RPUSH key value [value ...]
- LPOP/RPOP key
- LINDEX key index
- LSET key index value
- LLEN key
- LRANGE key start stop: trae el rango de elementos - si stop es -1 trae **todos**
- LTRIM key start stop: recorta al rango start-stop
- RPOPLPUSH source dest: RPOP sour + LPUSH dest

Listas

//...

```
client.delAsync("miClave")
.then(client.rpushAsync("miClave", 1, 2, 3, 4))
.then(client.lrangeAsync("miClave", 0, 2))
.then(function(values) {
    console.log(values);
    return client.ltrimAsync("miClave", 0, 1);
})
.then(function(){
    return client.llenAsync("miClave");
})
.then(function(len) {
    console.log(len);
});
```

A teclear un poco!

Modifica el ejercicio del blog del tema anterior...

- Para que utilice Redis como BD
- Guardar los posts como objetos JSON
- Guarda los usuarios en claves tipo:
 - “user:admin@asdf.com”: <JSON del usuario>
- Modifica la estrategia de autenticación

Hashes

- `HSET key field value`: Modifica el valor de campo `field` del hash en `value`
- `HGET key field`: Consulta el valor de campo `field` del hash en `value`
- `HEXISTS key field`: Existe el campo `field`?
- `HKEYS/HVALS key`: Todos los campos/valores
- `HGETALL`: Trae el hash entero
- `HINCRBY key field n`: Incrementa el campo en `n`
- `HMGET/HMSET`: Operaciones múltiples

Hashes

```
client.delAsync("miClave").then(function() {  
    return client.hmsetAsync("miClave", "a", 1, "b", 2, "c", 3)  
})  
.then(function() {  
    return client.hincrbyAsync("miClave", "c", 100);  
})  
.then(function() {  
    return client.hgetAllAsync("miClave");  
})  
.then(function(hash) {  
    console.log(hash);  
    // { a: '1', b: '2', c: '103' }  
});
```

Conjuntos

- SADD key member [member ...]: añadir miembros
- SREM key member [member ...]: quitar miembros
- SCARD key: cardinal (número de elementos)
- SDIFF key [key ...] - diferencia de conjuntos
- SINTER key [key ...] - intersección
- SUNION key [key ...] - unión
- SISMEMBER key member: ¿es miembro?
- SMEMBERS key: todos los miembros

Conjuntos

```
//promisifyAll...
```

```
client.saddAsync("miConjunto", 1, 1, 2, 3, 5, 8, 13)
  .then(client.saddAsync("miConjunto2", 1, 3, 5, 7, 9, 11,
13))
  .then(client.sinterAsync("miConjunto", "miConjunto2"))
  .then(function(values) {
    console.log("Intersección:", values);
    return client.sdiffAsync("miConjunto", "miConjunto2");
  })
  .then(function(values) {
    console.log("Diferencia:", values);
  });
```

Conjuntos ordenados

- `ZADD key score member [score member ...]`: añadir miembros con “score” (para ordenar)
- `ZREM key member [member ...]`: quitar miembros
- `ZCARD key`: cardinal (número de elementos)
- `ZRANGE key start stop` - rango ordenado
- `ZREVRANGE key start stop` - rango ordenado inverso
- ...

PUB SUB

- SUBSCRIBE channel
- PUBLISH channel message

PUB SUB

```
var redis = require("redis"),
    client1 = redis.createClient(),
    client2 = redis.createClient(),
    msg_count = 0;

client1.on("subscribe", function (channel, count) {
    //publish stuff when the subscription is ready
    client2.publish("a nice channel", "I am sending a message.");
    client2.publish("a nice channel", "I am sending a second message.");
    client2.publish("a nice channel", "I am sending my last message.");
});

client1.on("message", function (channel, message) {
    console.log("client1 channel " + channel + ": " + message);
    msg_count += 1;
    if (msg_count === 3) {
        client1.unsubscribe();
        client1.end();
        client2.end();
    }
});

client1.subscribe("a nice channel");
```

Ejercicio: acortador de URLs

Escribe un acortador de URLs con Redis

- Registro y login de usuarios utilizando simpleAuth
- Redirección automática de urls
- Estadísticas de visita
 - Cada IP cuenta una sola vez
 - ¿Estadísticas por fecha? ¿Tendencias?
 - ¿Geotracking de visitas (freegeoip.net)?

```
http.get( "http://freegeoip.net/json/83.44.23.171", function(res) {  
  res.on("data", function(data) {  
    console.log(JSON.parse(data));  
  });  
});
```