

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", redis.print);  
})
```

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

```
Q.ninvoke(client, "set", "miClave", "miValor")  
  .then(function() {  
    return Q.ninvoke(client, "get", "miClave");  
  })  
  .then(function(value) {  
    console.log("Valor: ", value);  
  })  
  .done();
```

SET / GET

Guardar y recuperar un valor

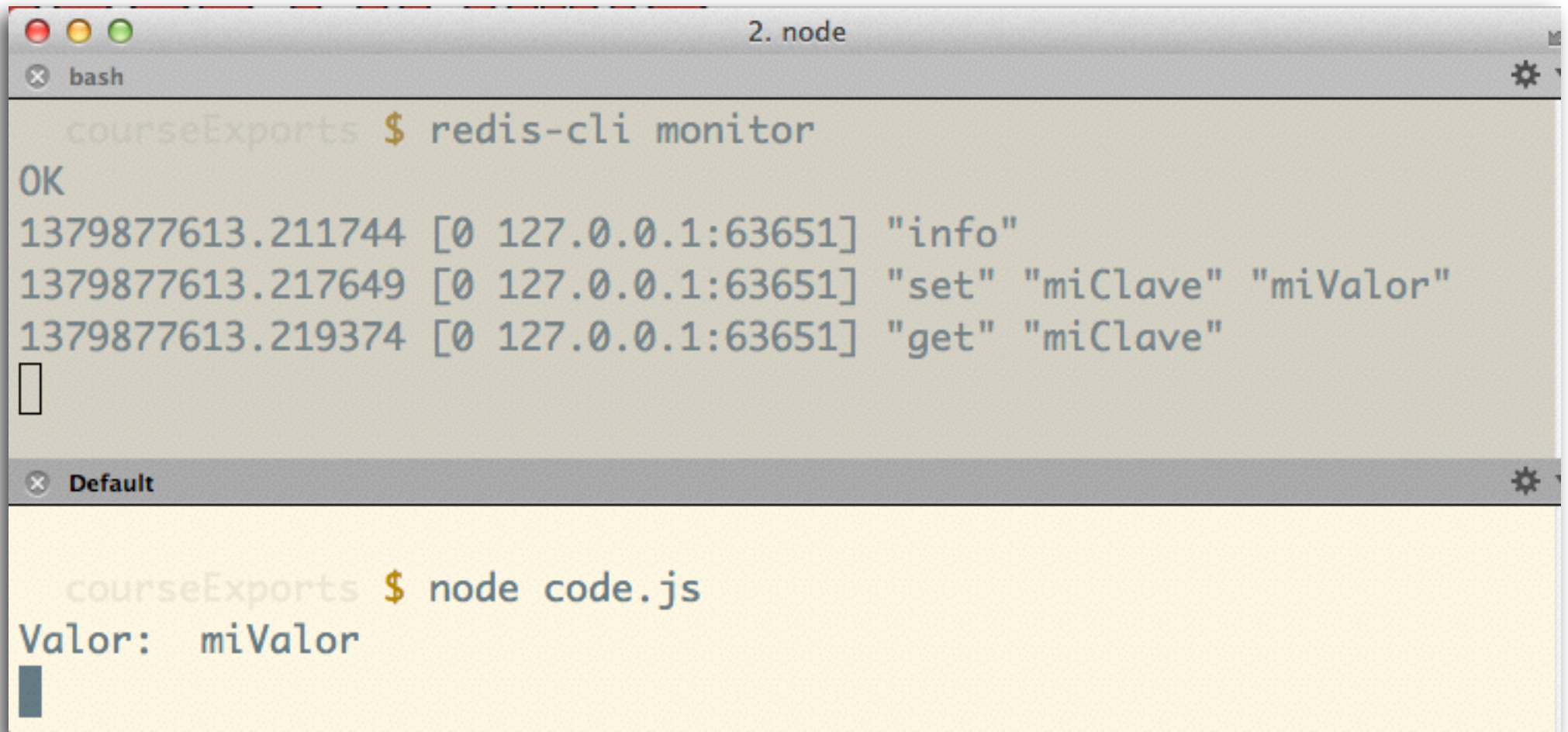
A screenshot of a terminal window titled "2. redis-cli". The terminal shows a sequence of commands and responses. First, the user runs "courseExports \$ redis-cli". Then, they enter "redis 127.0.0.1:6379> SET miClave miValor", which returns "OK". Next, they enter "redis 127.0.0.1:6379> GET miClave", which returns "\"miValor\"". Finally, they enter "redis 127.0.0.1:6379>" followed by a cursor, indicating the prompt is ready for the next command.

```
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



The image shows a screenshot of a terminal window with two tabs. The top tab is titled '2. node' and contains a 'bash' shell. The user has entered the command 'redis-cli monitor'. The output shows three Redis commands being received: 'info', 'set' (with keys 'miClave' and value 'miValor'), and 'get' (for 'miClave'). The bottom tab is titled 'Default' and shows the user running 'node code.js'. The output of this command is 'Valor: miValor'.

```
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"
█

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

Redis = Strings!!

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

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

Q.ninvoke(client, "set", "miClave", {un: "objeto"})
  .then(function() {
    return Q.ninvoke(client, "get", "miClave");
  })
  .then(function(value) {
    console.log("Valor: ", value); // Valor: [object Object]
  })
  .done();
```

Redis = Strings!!

```
var obj = {propiedad: "valor"};

Q.ninvoke(client, "set", "miClave", JSON.stringify(obj))
  .then(function() {
    return Q.ninvoke(client, "get", "miClave");
  })
  .then(function(value) {
    console.log("Valor: ", value);
    // Valor: {"propiedad": "valor"}
  })
  .done();
```

Redis = Strings!!

```
var serializable = {  
  toString: function() { return JSON.stringify(this); }  
};
```

```
var obj = Object.create(serializable, {  
  propiedad: {  
    enumerable: true,  
    value: "valor"  
  }  
});
```

```
Q.ninvoke(client, "set", "miClave", obj)  
  .then(function() {  
    return Q.ninvoke(client, "get", "miClave");  
  })  
  .then(function(value) {  
    console.log("Valor: ", value); // Valor: {"propiedad": "valor"}  
  })  
  .done();
```

Redis = Strings!!

```
var serializable = {  
  toString: function() { return JSON.stringify(this); }  
};
```

```
var obj = Object.create(serializable);  
obj.propiedad = "valor";
```

```
Q.ninvoke(client, "set", "miClave", obj)  
  .then(function() {  
    return Q.ninvoke(client, "get", "miClave");  
  })  
  .then(function(value) {  
    console.log("Valor: ", value);  
    // Valor: {"propiedad": "valor"}  
  })  
  .done();
```

Redis = Strings!!

Una solución más radical:

```
Object.prototype.toString = function() {  
    return JSON.stringify(this);  
};
```

```
var obj = {propiedad: "valor"};
```

```
Q.ninvoke(client, "set", "miClave", obj)  
  .then(function() {  
    return Q.ninvoke(client, "get", "miClave");  
  })  
  .then(function(value) {  
    console.log("Valor: ", value);  
    // Valor: {"propiedad": "valor"}  
  })  
  .done();
```

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"),
    Q = require("q"),
    client = redis.createClient();

Q.ninvoke(client, "exists", "MiClave")
  .then(function(exists) {
    console.log( exists? "Existe!" : "No existe..." );
    return Q.ninvoke(client, "set", "MiClave", "MiValor");
  })
  .then(function() {
    return Q.ninvoke(client, "rename", "MiClave", "MyKey");
  })
  .then(function() {
    return Q.ninvoke(client, "type", "MyKey");
  })
  .then(function(type) {
    console.log("MyKey es de tipo", type);
  })
  .done();
```


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

```
var op = Q.ninvoke.bind(Q, client);

op("set", "miClave", 1)
  .then(function() {
    return op("incrby", "miClave", 10);
  }).then(function() {
    return op("decr", "miClave");
  }).then(function() {
    return op("getset", "miClave", "fin");
  }).then(function(valor) {
    console.log("VALOR: ", valor);
    return op("strlen", "miClave");
  }).then(function(len) {
    console.log("Len: ", len);
  })
  .done();
```

Operaciones múltiples

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

```
var op = Q.ninvoke.bind(Q, client);

op("mset", "miClave", 1, "otraClave", 2)
  .then(function() {
    return op("mget", "miClave", "otraClave");
  })
  .then(function(values) {
    console.log(values[0], ",", values[1]);
  })
  .done();
```

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
- LTRIM key start stop: limita al rango start-stop
- RPOPLPUSH source dest: RPOP sour + LPUSH dest

Listas

```
var op = Q.ninvoke.bind(Q, client),
    makeOp = function() {
        var args = arguments;
        return function() { return op.apply(null, args); }
    };
```

```
op("del", "miClave")
  .then(makeOp("rpush", "miClave", 1, 2, 3, 4))
  .then(makeOp("lrange", "miClave", 0, 2))
  .then(function(values) {
    console.log(values);
    return op("ltrim", "miClave", 0, 1);
  })
  .then(makeOp("llen", "miClave"))
  .then(function(len) {
    console.log(len);
  })
  .done();
```

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

```
op("del", "miClave").then(function() {  
    return op("hmsset", "miClave", "a", 1, "b", 2, "c", 3);  
})  
.then(function() {  
    return op("hincrby", "miClave", "c", 100);  
})  
.then(function() {  
    return op("hgetall", "miClave");  
})  
.then(function(hash) {  
    console.log(hash);  
    // { a: '1', b: '2', c: '103' }  
})  
.done()
```


Conjuntos

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

Conjuntos

```
var op = Q.ninvoke.bind(Q, client),
    makeOp = function() {
        var args = arguments;
        return function() { return op.apply(null, args); }
    };

op("sadd", "miConjunto", 1, 1, 2, 3, 5, 8, 13)
  .then(makeOp("sadd", "miConjunto2", 1, 3, 5, 7, 9, 11, 13))
  .then(makeOp("sinter", "miConjunto", "miConjunto2"))
  .then(function(values) {
    console.log("Intersección:", values);
    return op("sdiff", "miConjunto", "miConjunto2");
  })
  .then(function(values) {
    console.log("Diferencia:", values);
  })
  .done();
```

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));  
  });  
});
```