*Tutorial baseado no livro*

*“Armazenando dados  com Redis”,*
*Rodrigo Lazoti*

# BANCO DE DADOS II

*Bancos de Dados Chave-Valor*

# AGENDA

- Bancos de dados Chave-Valor
- Tutorial REDIS
- Exercícios

# BANCO DE DADOS CHAVE-VALOR

- Sistemas gerenciadores NoSQL simples e velozes.
- Podem se espalhar por vários servidores de forma transparente
- Úteis para armazenamento de estruturas de dados garantindo a escalabilidade, problemas de baixa complexidade

# BANCO DE DADOS CHAVE-VALOR - APLICAÇÕES

- Armazenamento de dados de sessão
- Perfil de usuário sem necessidade de esquema
- Preferências de usuário
- Carrinhos de compras
- **Memória de apps**

# BANCO DE DADOS CHAVE-VALOR - ONDE NÃO USAR

- Necessidade identificar relações entre os dados
- Operações de múltiplas chaves
- Requisitos de negócio mudam frequentemente

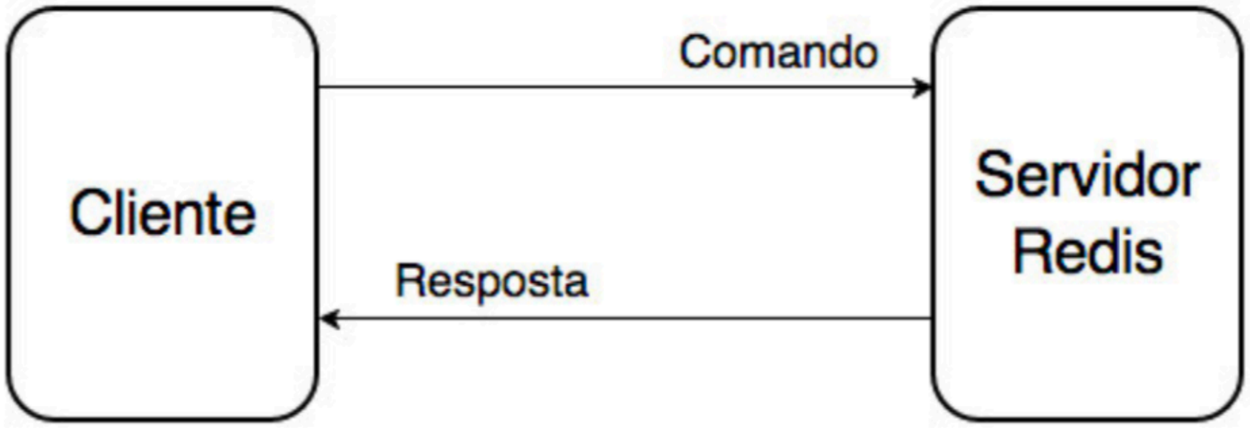
# TUTORIAL REDIS


**Redis: *REmote DIctionary Server***

- Banco de dados *NoSQL, open-source sob licença BSD*
- Armazena seus dados em memória, embora seja possível persistir os dados fisicamente.
  - Extremamente rápido em operações de IO
- Todos os comandos executados no Redis são atômicos
  - Redis é executado como uma aplicação *single-thread* (enquanto um comando está sendo executado, nenhum outro comando será executado).
- Armazena os dados na forma de chave-valor
  - O valor contido na chave de um registro suporta diferentes formatos que podem ser strings, hashes, lists, sets e sets ordenados

# TUTORIAL REDIS


- O Redis é um servidor TCP que faz uso do modelo cliente-servidor.
  - Uma requisição feita por um cliente ao servidor é seguida das seguintes etapas:
    - O cliente envia um comando ao servidor e fica aguardando uma resposta do servidor (geralmente bloqueando a conexão) através de uma conexão estabelecida via socket;
    - O servidor processa o comando e envia a resposta de volta ao cliente.



# TUTORIAL REDIS


- O Redis oferece uma interface de linha de comando (CLI)
  - É a forma mais rápida e fácil de executar comandos Redis, mas normalmente a comunicação é feita via aplicação
  - Suporte a várias linguagens de programação: http://redis.io/clients
  - Redis em JAVA:

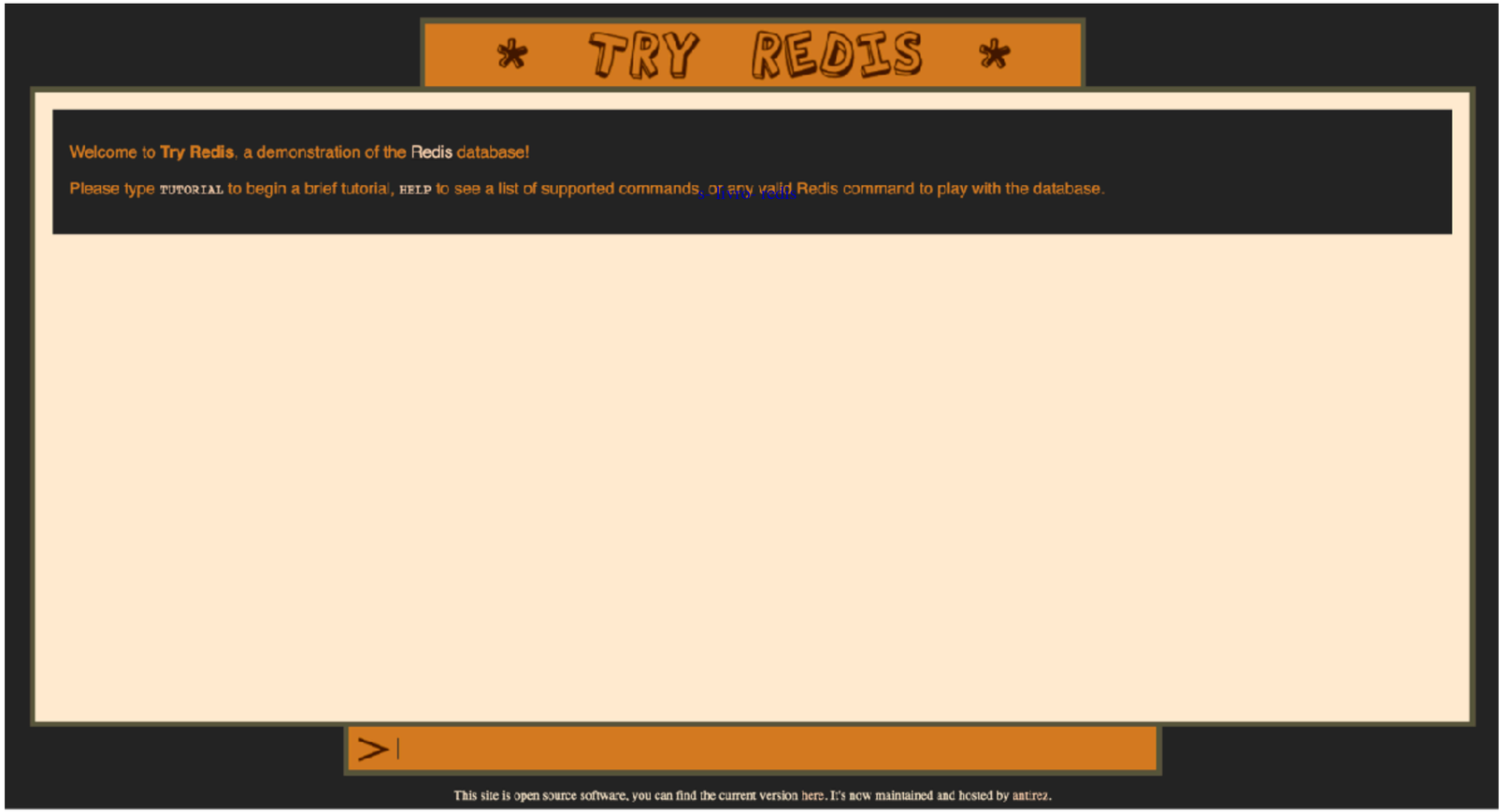
configuração:

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.4.2</version>
</dependency>
```

programando com o Redis:

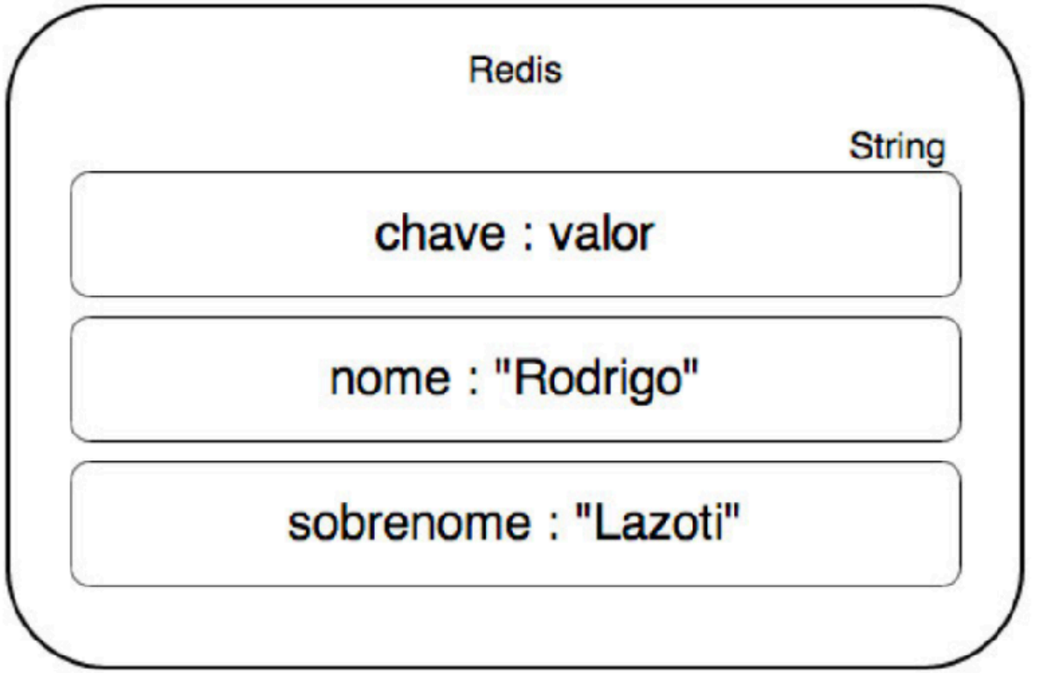
```
Jedis jedis = new Jedis("localhost");
String resultado = jedis.echo("ola redis!");
System.out.println(resultado);
```


- http://try.redis.io - Aplicação web
  - Disponibiliza um cliente desenvolvida em Ruby para testes do Redis



# REDIS - CACHE DE DADOS COM STRINGS

- String é o tipo de dado mais comum disponível no Redis
- Um valor do tipo string pode conter um tamanho de no máximo 512 Megabytes
  - Pode armazenar um texto, um JSON, objetos serializados ou até mesmo os dados binários de uma imagem



# REDIS – OPERAÇÕES DE LEITURA E ESCRITA


- Cache: repositório de dados em memória
- Comando GET/SET - Ler e armazenar dados no REDIS

```java
public class ArmazenarUltimoUsuarioLogado {
  public static void main(String[] args) {

    Jedis jedis = new Jedis("localhost");
    String resultado =
jedis.set("ultimo_usuario_logado", "Tony Stark");

    System.out.println(resultado);
  }

}
```

```java
public class ObterUltimoUsuarioLogado {
  public static void main(String[] args) {

    Jedis jedis = new Jedis("localhost");
    String valor = jedis.get("ultimo_usuario_logado");

    System.out.println(valor);
  }
}
```

# REDIS - CHAVES NO REDIS


## tipo-de-objeto:identificador:nome-campo

- Exemplo: armazenando sorteios da loteria no Redis

```
String dataDoSorteio1 = "04-09-2013";
String numerosDoSorteio1 = "10, 11, 18, 42, 55, 56";
String chave1 = String.format("resultado:%s:megasena", dataDoSorteio1);
…
Jedis jedis = new Jedis("localhost");
String resultado = jedis.mset(
    chave1, numerosDoSorteio1,
    chave2, numerosDoSorteio2,
    chave3, numerosDoSorteio3,
    chave4, numerosDoSorteio4
);
```

- MSET vs SET: MSET aceita vários conjuntos de chave-valor como parâmetro, SET aceita apenas um único conjunto de chave-valor.

**"resultado:04-09-2013:megasena"**

# REDIS - COMANDO KEYS


- Retorna as chaves que satisfazem a condição parametrizada

KEYS resultado:0?-??-????:megasena
KEYS resultado:0*-??-????:megasena
KEYS resultado:0?-*-????:megasena
KEYS resultado:0*-*-????:megasena
KEYS resultado:0?-??-*:megasena
KEYS resultado:0*-??-*:megasena
KEYS resultado:0?-*:megasena
KEYS resultado:0*-*:megasena

Exemplo: FiltrarHistoricoDaMegaSena

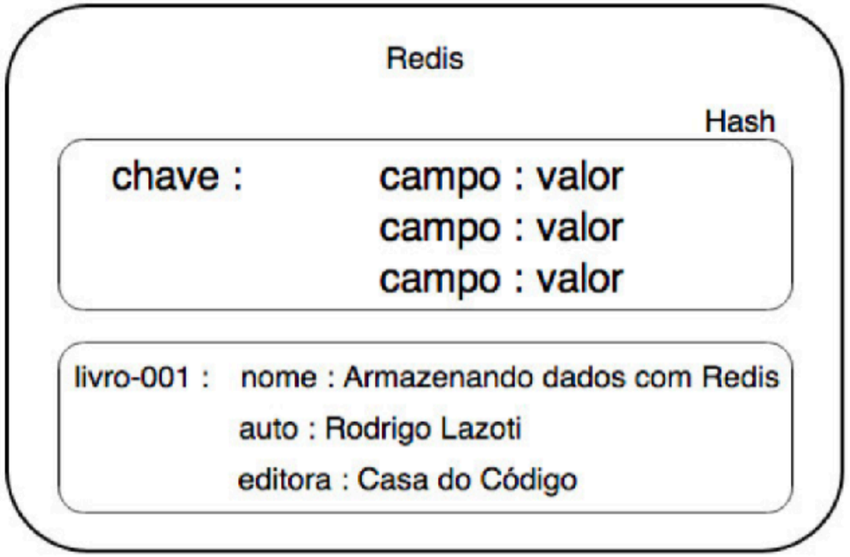
- MGET vs GET: MGET retorna vários conjuntos de chave-valor
- STRLEN

# COMANDOS PARA STRINGS


- **APPEND chave valor** : adiciona o valor a uma chave existente, ou cria uma nova chave (caso esta ainda não exista) com seu respectivo valor;
- **DEL chave [chave ...]** : remove a(s) chave(s) informada(s) e seu(s) respectivo(s) valor(es);
- **GET chave** : retorna o valor correspondente à chave informada;
- **GETRANGE chave inicio fim** : retorna uma parte da string armazenada conforme a chave informada;
- **MGET chave [chave ...]** : retorna os valores correspondentes às chaves informadas;
- **MSET chave valor [chave valor ...]**: armazena um ou mais conjuntos de chave valor. Caso uma chave informada já exista, seu valor será sobrescrito pelo novo;
- **SET chave valor**: armazena a chave e seu respectivo valor. Caso já exista uma chave definida, seu valor é sobrescrito;
- **STRLEN chave**: retorna o tamanho da string armazenada conforme a chave informada.

# HASHES


- Um hash é um map que contém campos e valores do tipo string
- Um hash pode armazenar 4 bilhões de pares campo-valor
- Exemplo: armazenando os ganhadores da loteria com hashes: hset/hget ArmazenarDadosDaMegaSena & ObterDadosDaMegaSena



# COMANDOS PARA HASHES


- **HDEL** ***chave campo*** **[*campo ...*]** : remove o(s) campo(s) e seu(s) respectivo(s) valor(es) do hash informado;
- **HEXISTS** ***chave campo*** : determina se um hash e seu campo existem;
- **HGET** ***chave campo*** : retorna o valor do campo associado ao hash informado;
- **HLEN** ***hash*** : retorna a quantidade de campos que um hash possui;
- **HMGET** ***chave campo*** **[*campo ...*]** : retorna os valores de todos os campos informados que são associados a um hash;
- **HMSET** ***chave campo valor*** **[*campo valor ...*]** : define múltiplos campos e valores em um hash;
- **HSET** ***chave campo valor*** : armazena um hash com o campo e seu respectivo valor. Caso o hash e o campo já existam, o valor é sobrescrito.

# EXPIRANDO CHAVES AUTOMATICAMENTE


- Quando uma aplicação web é distribuída para mais de um servidor, as informações de sessão ficam isoladas em um deles. É preciso então replicar os dados e garantir a atualização. Solução: gerenciar a *validade* dos dados
- Para definir o tempo de expiração, o Redis dispõe do comando TTL

- Exemplos: ArmazenarSessaoDoUsuario, DefinirTempoExpiracaoDeSessaoDoUsuario

# COMANDOS DE EXPIRAÇÃO DE DADOS


- **EXPIRE *chave tempo*** : define um tempo (em segundos) de expiração para uma chave;
- **PERSIST *chave :*** remove o tempo de expiração de uma chave;
- **PEXPIRE *chave tempo*** : define um tempo (em milissegundos) de expiração para uma chave;
- **PTTL *chave*** : retorna o tempo (em milissegundos) de vida restante para expiração da chave;
- **TTL *chave* :** retorna o tempo (em segundos) de vida restante para expiração da chave.

# ESTATÍSTICAS DE PÁGINAS VISITADAS


- Estrutura da chave: pagina:[url da pagina]:[data]
- Como as operações são atômicas, entre as operações de GET e SET poderia haver alguma inconsistência
- Comando INCR: incrementa o valor de uma chave
- Exemplo: GerarEstatisticaDePaginasVisitadas
- Comandos para Incremento/decremento:
  - **INCR *chave*** : incrementa (adiciona 1) ao valor (número inteiro) da chave;
  - **INCRBY *chave incremento*** : incrementa ou decrementa o valor (número inteiro) da chave conforme o valor do incremento;
  - **INCRBYFLOAT *chave incremento*** : incrementa ou decrementa o valor (número de ponto flutuante) da chave conforme o valor do incremento.

# BITMAPS


- Array de bits, um recurso importante para armazenar grandes volumes de dados
- Cálculos efetuados com comandos de BITMAP são extremamente rápidos
  - Bitmaps são utilizados para gerar estatísticas em tempo real
- Comando SETBIT: para definir ou remover um bit de um *offset* em um valor armazenado em uma chave
- Comando GETBIT: recupera o valor do bit armazenado para um determinado offset
- Exemplos: ArmazenarAcessosDosUsuariosComBitmap & ObterDadosAcessoPorDataComBitmap

# BITMAPS


- **BITCOUNT**: retorna o número de bita definidos no valor de uma chave - soma a quantidade de offsets que tiveram o bit definido como 1 em um bitmap
- No caso do acesso, se quiséssemos verificar quantos usuários estiveram na página no dia 01 e no dia 02, poderíamos comparar as ocorrências dos dias:
  - **BITOP**: realiza operações binárias entre múltiplas chaves e o resultado dessa operação é armazenada em uma nova chave (AND, OR, XOR)
  - **BITOP AND** ***acessos_dias_01_e_02 acesso:01/01/2014 acesso:02/01/2014***

    (integer) 6
  - **BITCOUNT**: retorna o número de ocorrências da operação
    - BITCOUNT acessos_dias_01_e_02

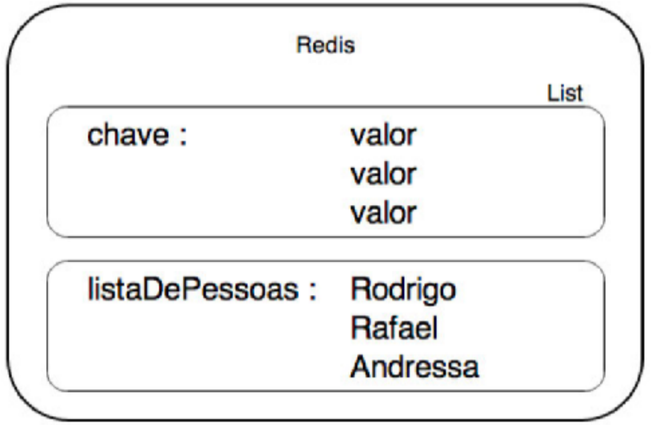
    (integer) 2

# COMANDOS BITMAPS


- **BITCOUNT chave** : retorna a quantidade de bits definidos em uma chave;
- **BITOP operador chave-resultado chave [chave...]** : realiza uma operação lógica entre diversas chaves e armazena seu resultado em uma chave definida;
- **GETBIT chave offset** : retorna o valor do bit de um offset armazenado em uma chave;
- **SETBIT chave offset valor** : define o valor do bit de um offset de uma chave

# TRABALHANDO COM LISTAS


- **List**: estrutura de dados organizadas pela ordem de inserção de cada item
  - Tamanho máximo: 4.294.967.295 elementos
- Exemplo: Exibir as últimas páginas visitadas do blog
  - **ArmazenarUltimasPaginasVisitadas.class**
  - **LimitandoUltimasPaginasVisitadas.class**




- **LPUSH chave valor [valor ...]** : adiciona um ou mais valores ao topo (head) da lista definida pela chave;
- **LLEN chave** : retorna a quantidade de itens armazenados em uma lista;
- **LRANGE chave inicio fim** : retorna um range de itens armazenados em uma lista, os valores inicio e fim são índices iniciados em 0;
- **LTRIM chave inicio fim** : apara a lista deixando apenas os itens definidos entre os índices de inicio e fim.

# CRIANDO UMA FILA DE MENSAGENS


- Útil para executar tarefas em segundo plano, enviar ou receber mensagens entre aplicações de forma assíncrona
- Exemplo: envio de e-mails para clientes (cadastro de usuário depende de um e-mail de confirmação)
- Fila **FIFO** (first in, first out)
- Exemplo: ArmazenarItemNaFila.java e ConsumirItemDaFila.java
- Comandos: LPOP remover e retornar o item do topo

# EXERCÍCIOS


- Criar um programa de carrinho de compras (dados do cliente e os itens comprados)
- Operações de inserção de novos itens e lista de itens