Redis: In-memory Key-value Store

팀원: 20192627 이정우
20192641 최은호
20222281 유지현
팀명: 열심히 하조
팀장: 유지현

1. redis-server

1.1. aeMain()

- redis-server는 이벤트 루프의 형식으로 내/외부 요청을 처리한다. ae.c:aeMain() 함수를 중심으로 레디스 서버의 전반적인 동작 과정을 분석하라.
- 처리 과정과 핵심적인 함수 등을 이해하기 쉬운 그림과 함께 설명

redis-server의 핵심 동작은 이벤트 루프를 통해 이루어집니다. ae.c 파일은 "Ansi Event"의 약자로, 이 이벤트 루프를 구현한 파일입니다. aeMain() 함수는 이벤트 루프의 핵심 부분이며, 다양한 이벤트와 상황에 대한 처리를 담당합니다.

aeMain() 함수의 주요 동작과정

이벤트 루프 초기화 및 설정 (aeCreateEventLoop 함수):
aeCreateEventLoop 함수를 호출하여 이벤트 루프를 초기화하고 필요한 설정을 수행합니다. 이 단계에서는 파일 디스크립터, 타이머, 지정된 이벤트 등에 대한 초기화가 이루어집니다.

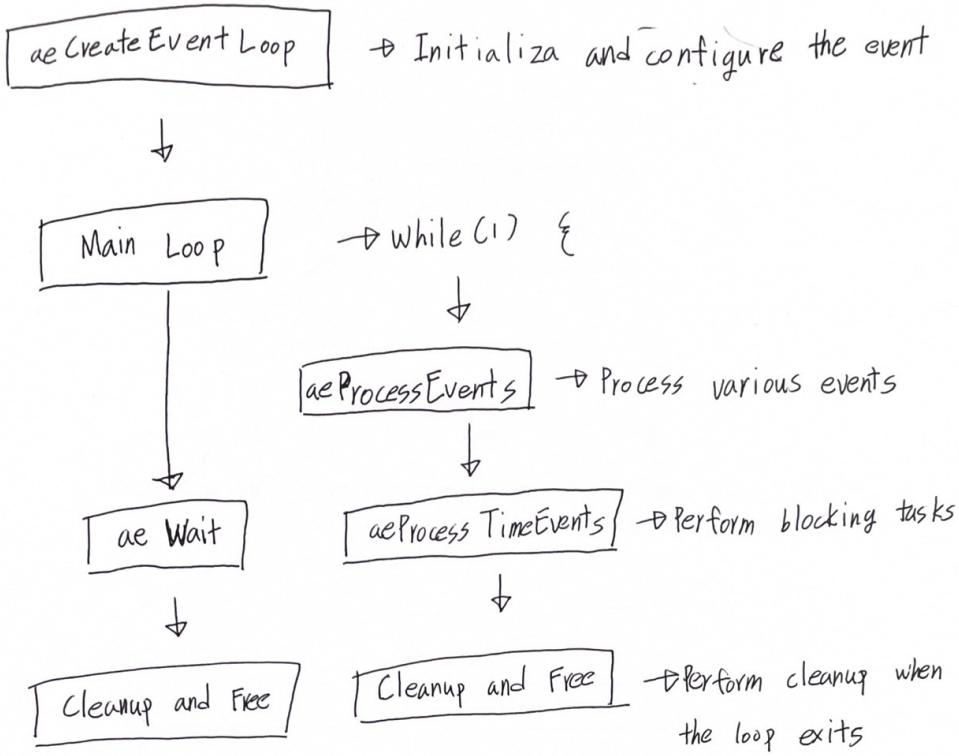
메인 이벤트 루프 (while 루프):
aeMain() 함수는 무한 루프인 while 루프로 진입합니다. 이 루프에서는 다음과 같은 주요 작업이 반복 수행됩니다:

이벤트 처리 (aeProcessEvents 함수):
현재 발생한 이벤트들을 처리하기 위해 aeProcessEvents 함수를 호출합니다. 이 함수는 다양한 이벤트 유형에 대한 처리를 담당하며, 내/외부에서 발생한 다양한 이벤트에 대응합니다.

블로킹된 작업 수행 (aeWait 함수):
블로킹된 작업이 있는 경우 aeWait 함수를 호출하여 해당 작업이 완료될 때까지 기다립니다. 이는 파일 디스크립터나 타이머 이벤트 등이 발생하기를 기다리는 부분입니다.

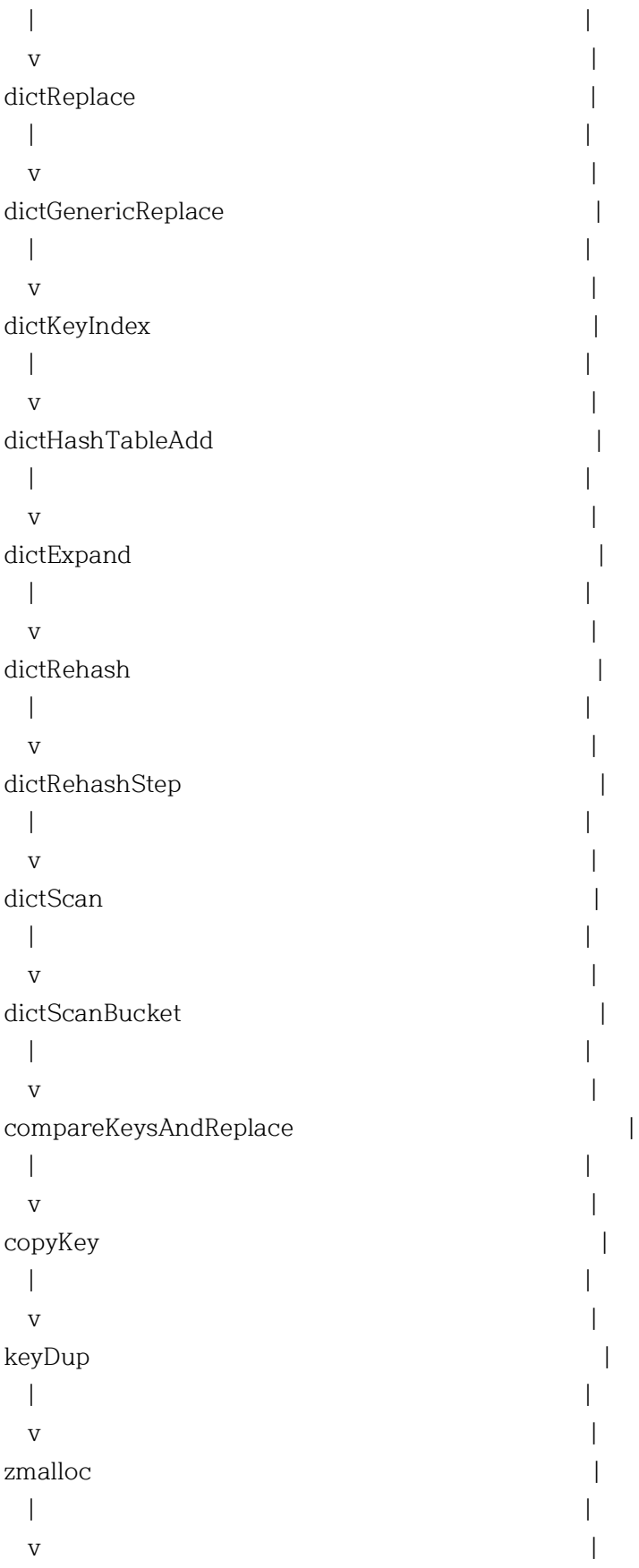
주기적인 작업 수행 (aeProcessTimeEvents 함수):
타이머 이벤트 등을 통해 주기적으로 발생하는 작업들을 처리하기 위해 aeProcessTimeEvents 함수를 호출합니다. 이는 주기적으로 발생하는 작업들을 처리하는 부분입니다.

이벤트 루프 종료:
이벤트 루프가 종료되면, 해당 함수는 정리 작업을 수행하고 메모리를 해제합니다.
이러한 구조를 통해 redis-server는 비동기적으로 다양한 이벤트를 처리하고, 블로킹 작업과 주기적인 작업을 효율적으로 관리할 수 있습니다. 이벤트 루프는 내/외부에서 발생한 다양한 이벤트에 대응하여 레디스 서버가 실제로 동작하고 클라이언트 요청을 처리할 수 있도록 합니다.


aeCreateEventLoop
→ Initializa and configure the event
Main Loop
→ while (1) {
aeProcessEvents
→ Process various events
ae Wait
aeProcess TimeEvents
→ Perform blocking tasks
Cleanup and Free
Cleanup and Free
→ Perform cleanup when the loop exits

1.2. 명령어 처리

아래는 redis-cli에서 SET x 1 명령어를 실행하는 과정에 대한 Call Graph입니다. 이 과정은 클라이언트에서부터 서버로 명령어를 전달하고, 최종적으로 Redis 서버가 In-Memory DB에 값을 저장합니다


dictReplace
dictGenericReplace
dictKeyIndex
dictHashTableAdd
dictExpand
dictRehash
dictRehashStep
dictScan
dictScanBucket
compareKeysAndReplace
copyKey
keyDup
zmalloc

위의 Call Graph에서는 redis-cli에서 SET x 1 명령어가 Redis 서버로 전달되어 처리되는 과정을 보여줍니다. 핵심 단계는 redis-server에서 setCommand 함수를 통해 In-Memory DB에 값을 저장하는 부분입니다. 이를 위해 Redis는 내부적으로 데이터 구조를 유지하고 메모리를 할당하여 데이터를 저장합니다

1.3. pang

순서부터 설명하면

명령어 정의 -> 명령어를 명령 테이블에 등록 -> 명령어의 응답 및 처리 로직 구현

```
// redis/src/server.c

struct serverCommand {
    // ...
};

static struct serverCommand redisCommandTable[] = {
    //
    {"pang", pangCommand, 0, "readonly fast", 0, NULL, 1, 1, 1, 0, 0},
    // ...
};// redis/src/server.c

void pangCommand(client *c) {
    addReply(c, shared.pung);  // "PUNG" 문자열을 클라이언트에게 응답으로 보냄
}
```

2. In-memory Database

2.1. Key-value store

- 레디스 database(key-value store)가 메모리에 저장되는 자료 구조를 분석하라.

레디스(Redis)는 인메모리 데이터 저장소로서 key-value 형태의 데이터를 저장하는데, 이를 가능하게 하는 다양한 자료 구조를 제공합니다. 아래는 레디스에서 사용되는 주요 자료 구조에 대한 간략한 설명입니다.

Strings (문자열): 가장 간단한 형태의 key-value 쌍이며, 이진 데이터나 텍스트 데이터를 저장하는 데 사용됩니다.
예: SET key "value"

Hashes (해시): 여러 필드와 각 필드에 대응하는 값들을 저장하는데 사용됩니다.
예: HSET myhash field1 "value1"

Lists (리스트): 양방향으로 연결된 노드들의 목록으로, 여러 요소들을 저장하고 유지합니다.
예: LPUSH mylist "value"

Sets (집합): 중복되지 않는 요소들을 저장하는데 사용됩니다.
예: SADD myset "value"

Sorted Sets (정렬된 집합): 집합과 유사하지만 각 요소에 대한 순서가 지정되어 있습니다.
예: ZADD myzset 1 "value1"

Bitmaps (비트맵): 비트의 배열로, 특정 오프셋의 비트를 설정하거나 해제하는 데 사용됩니다.
예: SETBIT mybitmap 7 1

HyperLogLog: 대량의 고유한 요소의 개수를 예측하는데 사용됩니다.
예: PFADD myhyperloglog "element"

Geospatial Indexes (지리 공간 인덱스): 위도 및 경도와 같은 지리 정보를 저장하고 조회하는 데 사용됩니다.
예: GEOADD mygeospatial 13.361389 38.115556 "Palermo“

레디스는 이러한 다양한 자료 구조를 조합하여 복잡한 데이터 모델을 만들 수 있습니다. 또한, 레디스는 메모리 기반의 데이터베이스로서 높은 성능을 제공하며, 영속적인 데이터 저장을 위한 기능도 제공합니다.

2.2. Data types

- https://redis.io/docs/data-types/
- Data Type 중 String, List가 어떻게 구현되어 있는지 분석하라.

String 데이터 타입 구현:

Redis의 String 데이터 타입은 가장 기본적인 데이터 타입으로, 바이트 시퀀스를 나타냅니다. 이는 텍스트, 직렬화된 객체, 이진 배열을 포함한 바이트 시퀀스를 저장할 수 있습니다. 이러한 이유로 Strings는 Redis 키와 연결된 가장 간단한 값 유형 중 하나입니다. 주로 캐싱에 사용되지만 카운터 구현 및 비트 연산도 지원합니다.

예를 들어:

```redis
> SET bike:1 Deimos
OK
> GET bike:1
"Deimos"
```

여러 다른 명령어가 문자열에 대한 작업에 사용될 수 있습니다. 예를 들어, `GETSET` 명령어는 키를 새 값으로 설정하고 이전 값을 반환합니다. 이를 사용하면 웹 사이트가 새로운 방문자를 받을 때마다 Redis 키를 INCR을 사용하여 증가시키는 시스템이 있다고 가정해 보겠습니다. 이 정보를 한 시간에 한 번 수집하려면 키를 "0"의 새 값으로 할당하고 이전 값을 다시 읽을 수 있습니다.

```redis
> GETSET total_crashes 0
(integer) 0
> INCR total_crashes
(integer) 1
> INCRBY total_crashes 10
(integer) 11
```

여기서 `INCR` 명령어는 문자열 값을 정수로 구문 분석하고 1씩 증가시키며 최종 값을 새 값으로 설정합니다. 이러한 명령어는 원자적으로 실행되기 때문에 여러 클라이언트가 동일한 키에 대해 `INCR`을 수행해도 경쟁 조건이 발생하지 않습니다.

List 데이터 타입 구현:

Redis의 List 데이터 타입은 삽입 순서대로 정렬된 문자열의 리스트입니다. 주로 스택 및 큐를 구현하거나 백그라운드 워커 시스템을 위한 큐 관리를 구축하는 데 사용됩니다.

기본 명령어:

- `LPUSH`: 리스트의 헤드에 새 요소 추가
- `RPUSH`: 리스트의 테일에 새 요소 추가
- `LPOP`: 리스트의 헤드에서 요소 제거 및 반환
- `RPOP`: 리스트의 테일에서 요소 제거 및 반환
- `LLEN`: 리스트의 길이 반환

예를 들어, 리스트를 큐처럼 다루는 예제:

```redis
> LPUSH bikes:repairs bike:1
(integer) 1
> LPUSH bikes:repairs bike:2
(integer) 2
> RPOP bikes:repairs
"bike:1"
> RPOP bikes:repairs
"bike:2"
```

여기서 `RPUSH`는 리스트의 오른쪽에 요소를 추가하고, `LPUSH`는 리스트의 왼쪽에 요소를 추가합니다. 두 명령 모두 여러 요소를 한 번에 리스트에 푸시할 수 있습니다.

```redis
> RPUSH bikes:repairs bike:1 bike:2 bike:3
(integer) 3
> LPUSH bikes:repairs bike:important_bike bike:very_important_bike
(integer) 5
> LRANGE bikes:repairs 0 -1
1) "very_important_bike"
2) "important_bike"
3) "bike:1"
4) "bike:2"
5) "bike:3"
```

또한 리스트에서 요소를 팝하는 것이 가능하며, 좌우 양쪽에서 요소를 푸시할 수 있습니다.

Blocking 명령어로는 `BRPOP` 및 `BLPOP`이 있습니다. 예를 들어, `BRPOP`은 리스트의 헤드에서 요소를 제거하고 반환하며, 리스트가 비어 있으면 요소가 추가될 때까지 블록됩니다.

```redis
> RPUSH bikes:repairs bike:1 bike:2
(integer) 2
> BRPOP bikes:repairs 1
1) "bikes:repairs"
2) "bike:2"
> BRPOP bikes:repairs 1
1) "bikes:repairs"
2) "bike:1"
> BRPOP bikes:repairs 1
(nil)
(2.01s)
```

이러한 기능은 큐 및 프로세스 간 통신 시스템을 구현하는 데 유용합니다. 이는 생산자/소비자 패턴을 쉽게 구현할 수 있게 해줍니다.

간단한 설명

String:
Redis의 String은 단순한 바이너리 데이터이며, 키와 연관된 값으로 사용됩니다. 아래는 Redis String의 주요 특징과 구현에 대한 간단한 분석입니다:

데이터 저장 형식:

Redis String은 이진 데이터를 저장할 수 있는데, 이는 텍스트 데이터뿐만 아니라 이미지, 직렬화된 객체 등 다양한 형식의 데이터를 저장할 수 있음을 의미합니다.
명령어 예시:

SET key value: 특정 키에 대한 값을 설정합니다.
GET key: 특정 키에 대한 값을 가져옵니다.
INCR key: 숫자 값이 저장된 키에 1을 더합니다.

List:
Redis의 List는 연결 리스트(Linked List) 형태로 구현되어 있습니다. 아래는 Redis List의

주요 특징과 구현에 대한 간단한 분석입니다:

데이터 저장 형식:

List는 여러 요소를 순서대로 저장하는데, 이는 문자열, 숫자, 또는 다른 데이터 타입일 수 있습니다.
각 요소는 더블 링크드 리스트(doubly linked list) 형태로 저장되며, 각 노드는 이전 노드와 다음 노드에 대한 포인터를 가지고 있습니다.
명령어 예시:

LPUSH key element: 리스트의 왼쪽에 요소를 추가합니다.
RPUSH key element: 리스트의 오른쪽에 요소를 추가합니다.
LPOP key: 리스트의 왼쪽에서 요소를 제거하고 반환합니다.
LRANGE key start stop: 리스트의 특정 범위의 요소를 가져옵니다.
구현 세부 사항:
String 구현:

String은 간단한 키-값 매핑으로 구현됩니다.
Redis는 키를 사용하여 빠르게 값을 가져올 수 있는 해시 테이블과 같은 내부 데이터 구조를 사용합니다.
List 구현:

List는 양방향 링크드 리스트로 구현되어 있습니다.
각 노드는 값을 저장하는데, 이 값은 실제 데이터가 됩니다.
특정 인덱스의 요소에 빠르게 접근하기 위해 인덱스에 대한 추가적인 인덱스(인덱스 테이블)도 사용됩니다.

2.3(Bonus). Binary Search Tree

- 레디스의 Data Type에 이진탐색트리를 추가하라. - 이진탐색트리에서 데이터를 검색, 삽입하는 ‘BSGET’, ‘BSSET’ 명령어를 구현한다.

"BSGET" 명령어를 구현하기 위해서는 다음과 같은 절차를 따를 수 있습니다:
1. Redis의 Sorted Set 데이터 타입을 활용하여 이진 탐색 트리를 구현합니다.
2. "BSSET" 명령어를 사용하여 데이터를 Sorted Set에 추가합니다.
3. "BSGET" 명령어를 사용하여 데이터를 검색합니다.

```
-- BSSET 명령어 구현
redis.call('ZADD', 'binary_search_tree', ARGV[1], ARGV[2])

-- BSGET 명령어 구현
local result = redis.call('ZRANGEBYSCORE', 'binary_search_tree', ARGV[1], ARGV[1], 'LIMIT', 0, 1)
```

```
if result[1] == nil then
    return nil
else
    return result[1]
end
```

위의 함수에서는 Redis의 Sorted Set을 이용하여 이진 탐색 트리를 구현합니다.
"BSSET" 명령어는 ZADD명령어를 사용하여 Sorted Set에 데이터를 추가합니다.
"BSGET" 명령어는 ZRANGEBYSCORE명령어를 사용하여 특정 값에 해당하는 데이터를 검색합니다.
이러한 방식으로 "BSSET"과 "BSGET" 명령어를 구현할 수 있습니다.