# 12장. 클라이언트 관리

> 레디스는 클라이언트의 요청을 어떻게 처리하고 연결을 어떻게 관리할까 ?

## 클라이언트 핸들링

1. TCP 포트 (default)
2. 유닉스 소켓

### 유닉스 소켓

```
unixsocket /tmp/redis.sock
unixsocketperm 777
```

- **멀티플렉싱 방식**
  - 하나의 통신 채널을 통해 여러 데이터 스트림을 전송
  - 하나의 스레드에서 여러 소켓을 감시하고 소켓 이벤트가 발생하는지 지속적 확인 가능
    - 다중 클라이언트 효율적 지원 가능
    - 블로킹 문제 방지
- **논블로킹 I/O**
  - I/O 작업 완료를 대기하지 않고 다른 작업 처리 가능
  - 요청을 비동기적으로 처리하고 다수의 클라이언트 요청을 동시에 처리 가능
- `TCP_NODELAY`
  - 클라이언트 커넥션 생성에 쓰이는 옵션
  - 작은 데이터라도 버퍼링하지 않고 연결 지연을 최소화하여 패킷 전송 시도
  - 작은 데이터조각을 실시간으로 전송해야 하는 경우 사용
  - 지연 시간 최소화를 위해 **기본값은 no**로 동작

- `대규모 트래픽 상황` or `마스터-복제본 간 거리가 먼 경우` yes로 변경하는 게 도움이 될지도 ?

# 클라이언트 버퍼 제한

출력 버퍼 크기에 대한 제한을 둬서 일정 수준 이상으로 증가할 경우 클라이언트 연결을 종료함

## 하드 제한

고정된 제한값, 임계치에 도달하면 레디스는 클라이언트 연결을 가능한 한 빨리 닫는다.

## 소프트 제한

시간에 따라 다름, n초 동안 지속적으로 m보다 큰 출력 버퍼를 유지하면 연결을 닫는다.

## 정책

- **일반 클라이언트**
  - 출력 버퍼 크기 제한 `0`
  - 다음 커맨드를 보내기 전 응답을 기다리기 때문에 연결을 닫아버리면 안됨
- **pub/sub 클라이언트**
  - 기본 하드 제한 `32MB` / 소프트 제한 `60초당 8MB`
  - 빠르게 처리되는 메시지들을 처리하기 위해 더 많은 메모리 공간 필요
- **복제본**
  - 기본 하드 제한 `256MB` / 소프트 제한 `60초당 64MB`
  - 마스터로부터 대량의 데이터를 받아들이는 경우 더 큰 출력 버퍼 필요

# 클라이언트 이빅션

## 7.0 이전

클라이언트 연결과 데이터를 저장하는 데 사용하는 메모리를 통합해 `maxmemory-policy` 설정값으로 메모리 한도 관리

## 7.0 이후

`maxmemory-clients`: 모든 클라이언트 연결이 사용하는 누적 메모리 양 제한

임계치에 도달하면 서버에서 클라이언트 연결을 해제해 메모리를 확보

**가장 많은 메모리를 사용하는 연결부터 해제 시도 → 클라이언트 이빅션**

## Timeout과 TCP Keepalive

특정 시점에서 활동이 없는 클라이언트 정리

### 1. redis.conf 파일에서 timeout 지정

### 2. 레디스 서버가 동작 중일 때 CONFIG 커맨드로 설정

```
> CONFIG SET timeout 60
OK
```

- pub/sub 클라이언트에는 영향을 주지 않는다.

### 3. tcp-keepalive

- 주기적으로 TCP ACK를 보내고 응답이 없는 경우 연결을 끊는 설정
- timeout과 유사하게 클라이언트의 이상 동작 감지 / 네트워크 문제 조기 감지 가능
- 레디스 3.2.0부터 기본값 300 → 5분마다 한 번씩 TCP ACK를 보냄

# 파이프라이닝

네트워크 통신 소요 시간을 줄여 레디스의 성능을 향상시키자 !

## 동작 방식

- 클라이언트가 연속적으로 여러 개의 커맨드를 레디스 서버에 보낼 수 있도록 한다.
- 줄바꿈을 이용해 한 번에 보내거나, 레디스 클라이언트 라이브러리를 이용한다.
- 파이프라이닝에 속한 커맨드가 많아질수록 초당 수행되는 쿼리 수가 선형적으로 증가한다.

## 주의 사항

### 1. 한 번에 너무 많이 보내지 말자.

- 네트워크 대역폭 한계로 속도가 저하된다.
- 클라이언트 쿼리 버퍼 제한에 걸려 오류가 발생한다.

여러 개의 명령을 파이프라이닝으로 처리하려면 명령을 나누어 **배치 형태**로 보내자

### 2. 배치 사이즈는 충분한 테스트를 통해 결정하자

- 너무 많으면 네트워크 대역폭을 너무 많이 차지하거나
- CPU 부하, 클라이언트 버퍼로 인한 메모리 증가로 인해 오히려 성능 문제가 발생한다.

### 3. 레디스 파이프라인은 원자성을 보장하지 않는다.

- 트랜잭션의 개념이 아니기 때문에 일부에 오류가 발생하면 그 커맨드만 수행되지 않는다.

# 클라이언트 사이드 캐싱

버전 6부터 도입된 성능 향상 포인트 !

클라이언트와 레디스 서버 간 통신의 최대 걸림돌? **왕복 시간**

- 클라이언트 측에서 데이터를 로컬에 캐싱하고 필요할 때 반환 → 응답 시간 단축
- 아이템이 자주 변경되지 않는 상황에서 합리적
- **트래킹**: 업데이트 된 데이터를 처리하는 방법(정합성) 고려 필요

## 트래킹

### 기본 모드

레디스 서버가 클라이언트가 액세스한 키를 기억해서 동일한 키가 수정될 때 무효 메시지 전송

- 레디스 서버가 키를 기억해야 해서 메모리가 소모됨
- 정확하게 클라이언트가 갖고 있는 키만 무효 메시지를 보낼 수 있음

### 브로드캐스팅 모드

특정 프리픽스에 대해 접근한 클라이언트만 기억

- 기본 모드보다 메모리 소모가 적음
- 자신이 소유하지 않은 키여도 프리픽스가 일치하면 키가 변경될 때마다 변경 메시지를 수신하여 CPU 자원 소비

**결론!** 자주 요청되지만 드물게 변경되는 키를 효과적으로 캐싱하자. 캐싱 모드 선택은 신중히