Базы Данных

Семинар 14

Описание семинара: Redis. Практика

С одной стороны есть традиционные ACID реляционные базы данных такие как MySQL, PostgreSQL, Oracle и др. Они надежны и стабильны. Сама аббревиатура ACID описывает требования к транзакционной системе (Atomicity — Атомарность, Consistency — Согласованность, Isolation — Изолированность, Durability — Долговечность). Их основная задаче не просто хранить данные, а хранить с максимальной надежностью. Но их основной недостаток, они очень медленные.

С другой стороны есть очень быстрые хранилища в памяти типа ключ-значение, например memcached. Они очень быстрые за счет предельной простоты и отказа от надежности. Несложно привести несколько примеров задач где нужна производительность даже за счет надежности:
— система мониторинга с n-ым количеством датчиков, которые должны постоянно отправлять данные.
— система логирования действий пользователя на каком нибудь сайте
— контроль трафика сети в реалтайм
— хранение сессий пользователей
и т.д.

Redis относится к третьему типу хранилищей. Когда нужно быстрая обработка данных и при этом сохранялось бы определенный уровень надежности и возможности масштабирования.

Redis (REmote DIctionary Server) — это не реляционная структура данных в памяти, используемая в качестве базы данных. Данные хранятся в виде пары ключ-значение. И при этом хранилище умеет масштабироваться путем репликации между серверами. Redis сохраняет все данные в памяти, что позволяет сделать доступ к данным максимально быстрым по сравнению с другими базами данных. Почему Redis известен своей исключительной высокой производительностью даже среди других key-value хранилищ.

Redis позволяет нам хранить данные в высокоуровневых структурах данных, такие как строки, хэши, списки, наборы. Это дает нам больше гибкости в отношении типа и объема информации, которую мы можем хранить в хранилище данных Redis.

Он также довольно дружелюбен для разработчиков, поскольку поддерживает большинство языков высокого уровня, таких как Python, JavaScript, Java, C / C ++ и PHP.

Установка Redis

Redis распространяется по лицензии BSD. Написан на ANSI C, поэтому легок и не имеет внешних зависимостей. Официальной версии для Window нет, но есть версия под Open Server. Но его можно запустить под Docker .

Для дальнейшего изучения Redis нам необходимо скачать и установить сервер Redis. Можно использовать инструкции с официальной веб-страницы. Так же под MacOS можно использовать Homebrew, а для Linux что типа apt. Для запуска Redis используется команда redis-server. Установка сервера Redis:

```
$ redis-server /usr/local/etc/redis.conf
```

У Redis есть Redis-CLI (Common Line Interface), который можно использовать для взаимодействия с данными напрямую на сервере Redis.

Чтобы проверить правильность установки Redis, запустите redis-cli, а затем введите команду ping в появившейся подсказке:

```
$ redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

Если сервер отвечает ответом PONG значит он готов к работе. По умолчанию сервер Redis работает на порту 6379, что видно в нашем приглашении. Чтобы выйти из redis-cli используйте команду quit.

Для вывода помощи по списку команд в консоли можно использовать команду HELP @string. Для вывода помощи по конкретной команде можно использовать команду HELP APPEND Например:

```
127.0.0.1:6379> HELP APPEND

APPEND key value
summary: Append a value to a key
since: 2.0.0
group: string
```

Далее приведен краткий список самых необходимых команд. Для изучения полного списка команд, если в этом есть необходимость, обратитесь к официальной документации.

Прежде чем начать использовать Redis в любом языке программирования нужно узнать о базовых командах и структурах используемых в Redis.

Команды

Прежде всего Redis это хранилище типа ключ: значение. И самые первые команды которую все изучают, это команды SET и GET:

SET - Команда используется для установки ключа и его значения, с дополнительными необязательными параметрами для указания срока действия записи значения ключа. Давайте установим ключ foo со значением «hello world». Параметр EX указывает время жизни объекта в секундах, PX в милисекундах:

```
127.0.0.1:6379> SET foo "hello world"
OK
127.0.0.1:6379> SET foo1 "hello world" ex 5
OK
```

GET - Команда используется для получения значения, связанного с ключом. Если запись значения ключа превысила срок действия, будет возвращено nil:

По умолчанию все значение в Redis сохраняются как строки.

```
127.0.0.1:6379> GET foo
"hello world"
# если истечет время жизни записи
127.0.0.1:6379> GET foo
(nil)
```

EXISTS - Эта команда проверяет, существует ли что то с данным ключом. Она возвращает 1 если объект существует или 0 если нет. Boolean типа в Redis нет.

```
127.0.0.1:6379> EXISTS foo
(integer) 1
```

FLUSHALL - Эта команда полностью удаляет все данные в текущем сеансе.

GETSET - Команда возвращает текущее значение и устанавливает новое. Используется для атомарного управления данными.

```
127.0.0.1:6379> SET foo "hello"
OK
127.0.0.1:6379> GETSET foo "world"
"hello"
127.0.0.1:6379> GET foo
"world"
```

DEL - Команда удаляет ключ и соответствующее значение:

```
127.0.0.1:6379> DEL foo
(integer) 1
```

APPEND - Команда добавляем в соотвествующий ключ дополнительное значение. Возвращает количество символов итогового значения.

```
127.0.0.1:6379> SET foo "hello"
OK
127.0.0.1:6379> APPEND foo " world"
(integer) 11
127.0.0.1:6379> GET foo
"hello world"
```

KEYS - Возвращает все ключи из базы по указанному шаблону. Есть предостережение что в реальных приложения эту команду лучше не использовать из-за того что она очень медленная.

```
127.0.0.1:6379> KEYS *
1) "boo"
2) "foo"
```

INCR / DECR - Инкремент / декримент. Если значение ключа integer (хотя в базе храниться все равно строка) можно увеличить или уменьшить значение на 1. Если использовать команду INCR с несуществующем значением то создаться новый ключ со значением 1.

```
127.0.0.1:6379> SET foo 1
OK
127.0.0.1:6379> INCR foo
(integer) 2
127.0.0.1:6379> INCR foo
(integer) 3
127.0.0.1:6379> DECR foo
(integer) 2
```

TTL - Когда ключ установлен с истечением срока действия (например SET foo EX 10), эту команду можно использовать для просмотра оставшегося времени:

```
127.0.0.1:6379> SET foo 10 EX 10
OK
127.0.0.1:6379> TTL foo
(integer) 6
127.0.0.1:6379> TTL foo
(integer) 5
127.0.0.1:6379> TTL foo
(integer) 2
127.0.0.1:6379> TTL foo
(integer) -2
```

PERSIST - Если мы передумаем об истечении срока действия ключа, мы можем использовать эту команду, чтобы удалить период истечения срока действия:

```
127.0.0.1:6379> PERSIST foo
(integer) 1
127.0.0.1:6379> TTL foo
(integer) -1
127.0.0.1:6379> GET foo
"bar"
```

RENAME - Эта команда используется для переименования ключей на нашем сервере Redis:

```
127.0.0.1:6379> RENAME foo foo2
OK
127.0.0.1:6379> GET foo
(nil)
127.0.0.1:6379> GET foo2
"bar"
```

Комплексные типы данных

Хеш таблицы

Redis позволяет в качестве значения так же использовать ключ: значение. Что по сути будет почти аналогией объектов из JavaScript или словари в Python. Для записи объекта используется команда HSET в следующем формате HSET имя_ключа имя_атрибута значение. Для чтения объекта используется команда HGET в формате HGET имя_ключа имя_атрибута. Команда HGETALL используется для получения

```
127.0.0.1:6379> HSET obj attr1 value1
(integer) 1
127.0.0.1:6379> HSET obj attr2 value2
(integer) 1
127.0.0.1:6379> HGET obj attr1
"value1"
127.0.0.1:6379> HGETALL obj
1) "attr1"
2) "value1"
3) "attr2"
4) "value2"
```

Множества

Не упорядоченная коллекция уникальных элементов. Аналог set в Python. Для добавление нового элемента во множество используется команда SADD. Для получения все элементов используется команда SMEMBERS. SUNION используется для объединение множеств. SDIFF используется для вычитания из первого множества второго. SINTER возвращает общие элементы указаных множеств. SPOP удаляет и возвращает случайный элемент множества.

```
127.0.0.1:6379> SADD objects obj1
(integer) 1
127.0.0.1:6379> SADD objects obj2
(integer) 1
127.0.0.1:6379> SADD objects obj3
(integer) 1
127.0.0.1:6379> SADD objects obj1
(integer) 0
127.0.0.1:6379> SMEMBERS objects
1) "obj2"
2) "obj3"
3) "obj1"
```

Упорядоченные множества

Упорядоченная коллекция уникальных элементов. Для добавление нового элемента в упорядоченное множество используется команда ZADD. Формат ZADD имя_ключа порядковое_число_упорядочивания_множества значение
Команда ZRANGE возвращает срез данных множества

```
127.0.0.1:6379> ZADD objects 1500 value1
(integer) 1
127.0.0.1:6379> ZADD objects 1600 value2
(integer) 1
127.0.0.1:6379> ZADD objects 1700 value3
(integer) 1
127.0.0.1:6379> ZRANGE objects 0 -1
1) "value1"
2) "value2"
3) "value3"
```

Списки

Список это последовательность значений упорядоченных по порядку их создания. Аналог двух-стороннего стека (то есть стека в который можно добавлять с двух сторон). Списки обычно применяются для создания очередей.
Команда добавления элемента в список LPUSH
LRANGE — возвращает срез списка с левой стороны. Формат LRANGE имя_списка индекс_начало индекс_конца Указание -1 означает до конца списка.
LPOP — вернуть одно значение с левой стороны и удалить его из списка. Формат LPOP имя_списка. Аналогичные команды RPUSH, RRANGE, RPOP только для правой стороны.

```
127.0.0.1:6379> LPUSH obj1 value1
(integer) 1
127.0.0.1:6379> LPUSH obj1 value2
(integer) 2
127.0.0.1:6379> LPUSH obj1 value3
(integer) 3
127.0.0.1:6379> LRANGE obj1 0 -1
1) "value3"
2) "value2"
3) "value1"
127.0.0.1:6379>
```

Транзакции в Redis

Обычное определение транзакций для реляционных баз данных означает следующее: транзакции это группа команд с базой данных, которые должны либо полностью выполнится или в случае возникновение ошибки вернуть состояние базы данных в исходное состояние. В

Redis то же есть такое понятие как транзакции. Но означает немного другое. Транзакции в Redis это просто последовательное выполнение ранее записаных команд без возможности полноценного возвращения исходного состояния в случае ошибки исполнения.

С помощью команды MULTI можно начать запись команд. Далее введенные команды не исполняются а записываются в буфер. Это будет происходит до ввода команды на исполнения транзакции EXEC. Далее все ранее введенные команды будут исполнены один за другим. Команда DISCARD отмена записи команд транзакций. Если возникнет ошибка в процессе ввода команд вся транзакция не будет выполнена.

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> SET obj1 value1
QUEUED
127.0.0.1:6379> SET obj2 value2
QUEUED
127.0.0.1:6379> EXEC
1) OK
2) OK
127.0.0.1:6379> GET obj1
"value1"
```

Механиз подписок PUS-SUB

Одно из основных преимуществ Redis от других key-value хранилищ заключается в том, что в Redis есть механизм подписок. То есть Redis можно использовать как сервер сообщений.

Одни клиенты подписываются на определенные каналы используя команду SUBSCRIBE имя_канала

Другие клиенты могут отправлять сообщения в этот канал используя команду PUBLISH имя_канала значение

Допустим один клиент подписывается на канал

```
SUBSCRIBE channel
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "channel"
3) (integer) 1
```

Другой клиент что то отправляет в этот канал

```
PUBLISH channel "hello world"
(integer) 1
```

И в этот момент первый клиент получит это сообщение

```
SUBSCRIBE channel
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "channel"
3) (integer) 1
1) "message"
2) "channel"
3) "hello world"
```

Основы применение Redis в Python

Redis очень широко применяется в современной разработке ПО. Библиотеки поддержки есть для любого языка программирования.

Его используют в:

- Кеширование данных
- Чаты и системы обмена сообщенийДиспетчеризация любых данных
- Различные очереди задач
- Мониторинг различных данных

Кратко рассмотрим использование Redis в Python. Для этого первым делом загрузим библиотеку поддержки:

```
#!/usr/bin/env python
import redis
```

Далее подключимся к серверу

```
client = redis.Redis(host='127.0.0.1')
```

И далее можно уже начать попробовать использовать все ранее рассмотренные команды. Надеюсь они будут понятны без дополнительных пояснений:

```
client.set('Language', 'Python')
print client.get('Language')
```

```
from time import sleep
client.set('Language', 'Python', ex=5)
print client.get('Language')
sleep(3)
print client.ttl('Language')
```

```
# Пример использования множеств
client.sadd('pythonlist', 'value1', 'value2', 'value3')
print client.smembers('pythonlist')
client.sadd('redislist', 'value1', 'value5', 'value6', 'value7',
'value8')
print "sinter", client.sinter('pythonlist', 'redislist')
print "sunion", client.sunion('pythonlist', 'redislist')
print client.scard('pythonlist')
```

```
# Пример использования хеш таблиц
client.hset('Person', 'Name', 'Person1')
client.hset('Person', 'Health', '600')
client.hset('Person', 'Mana', '200')
print client.hgetall('Hero')
```

Заключение

Redis — это мощный и быстрый вариант хранения данных, который при правильном использовании может принести много преимуществ. Он не имеет крутой кривой обучения, поэтому с ним легко начать работать. Также поставляется с удобным инструментом CLI, который помогает нам взаимодействовать с ним с помощью простых и интуитивно понятных команд.