**Модель баз данных «Ключ-Значение»**

Модель данных – логическое понятие, означающая способ восприятия человеком организации данных (все понятия модель данных сводятся к модели «Сущность-Связь»). Модель хранения данных – способ применяемых СУБД структур данных и принципом манипулирования ими.

Агрегатные СУБД – СУБД типа «ключ-значение», «семейство столбцов», документные СУБД. Оперируют понятием агрегата – коллекция связанных объектов, интерпретируемая как единое целое. Агрегат – естественная единица репликации и фрагментации, поэтому легко используются в кластерных архитектурах. Агрегаты лучше подходят для использования в прикладных программах, так как имеют более близкую структуру объектным моделям клиентских приложений. Базируется на определениях объектно-ориентированной модели программирования и отношениях, прежде всего на агрегации и композиции.

Что выделять в качестве агрегата, зависит от клиентского приложения и основных типов запросов, которые в нем выполняются. Таким образом, база данных представляет собой набор связанных агрегатов.

**Хранилище типа «ключ-значение»** в качестве основы представляет структуру типа простой хеш-таблицы, которая доступ ко всем данным базы осуществляет по первичному ключу. Основные операции – получение значения по ключу, записать значение по ключу, удалить ключ вместе со значением. Значение – объект данных, который записан без детализации его внутренней структуры.

Хранилища «ключ-значение» имеют высокую производительность и легко масштабируются.

Популярные виды СУБД: Riak, Redis, MemcachedDB, BerkleyDB, Amazon DynamoDB. Project Voldemort.

Где удобно использовать СУБД типа «ключ-значение»:

* Хранение информации о сессиях пользователей;
* Хранение профилей пользователей;
* Хранение информации о корзинах товаров и пр.

Не используют СУБД типа «ключ-значение»:

* Нужны частые связи между агрегатами;
* Сложные алгоритмические транзакции;
* Нет механизмов поиска данных по значениям, реализован только поиск по ключу;
* Ограниченность операций сразу над несколькими ключами.

Разберем основные принципы работы на примере СУБД Redis.

* **Redis** предлагает пять фиксированных примитивных структур данных, но **не** дает возможности создавать свои (по принципу таблиц или букетов).
* **Redis** держит все данные в памяти, но имеет гибко настраиваемый механизм сохранения их копии на диск.
* **Redis** реализован в виде однопоточного демона на [C](http://club.cnews.ru/tag/c/), общающегося по собственному текстово-бинарному TCP-протоколу, но без намеков на многопоточность и кластеризацию.

Модель данных у Redis основана на принципе *пар ключ-значение* с возможностью представлять значение как сложную структуру данных с дополнительными операциями:

* **Строка:**в самом случае значением может быть просто последовательность каких-то байт, о содержании которых Redis ничего не знает и знать не должен.
* **Словарь/хэш:** за идентификатором (в языках программирования — *переменной*, а в Redis — *ключом*) стоит не одно значение, а подмножество пар ключ-значение с механизмом доступа, аналогичным основному множеству ключей в Redis.
* **Набор:**используется для хранения *множества* неотсортированных уникальных значений, предоставляя механизмы для работы с этими данными (команды на букву s), в частности стоит обратить внимание пересечение и объединение множеств.
* **Отсортированный набор:**сортировка в наборах осуществляется путем присваивания каждому значению условного ранга (score) в виде целого числа; значения в наборе будут автоматически храниться в порядке от минимального к максимального ранга.
* **Список:** в отличии от отсортированных наборов в списках значения хранятся просто по порядку.

В Redis доступны четыре основных стратегии для сохранения полученных данных:

* Хранить только в памяти: по сути Redis из персистентной базы данных превращается в кэширующий сервер. Производительность в таком режиме максимальна и сравнима с часто используемым в этой роли [memcached](http://club.cnews.ru/unix-way/obzor-memcached/), в некоторых ситуациях его даже превосходит; плюс не забываем про более продвинутую структуру данных. Минус с высоким риском потери данных — очевиден, за производительность надо осознанно платить.
* Периодически сохранять на диск: (по-умолчанию) момент, когда Redis решает сделать очередную копию (snapshot) данных на диске, зависит от времени создания предыдущей и количества измененных ключей. При настройках по-умолчанию это случается раз в несколько минут (1-15).
* Лог транзакций: если потеря нескольких минут изменений данных неприемлема для приложения, есть опция синхронной записи каждого изменения в специальный append-only файл (лог).
* Репликация: в Redis поддержка репликации достаточно примитивна — каждому серверу можно указать мастера (командой [slaveof](http://www.insight-it.ru/goto/http:/redis.io/commands/slaveof) или одноименной строкой в конфигурации), все изменения на мастере будут воспроизводиться и на подчиненном. Это можно использовать для построения более сложных схем сохранности данных, например: мастер работает в режиме «только в памяти», а подчиненный — в режиме «лог транзакций». У каждого сервера может быть только один мастер, но неограниченное количество подчиненных.

Копия данных Redis на диске представляет собой бинарный файл (по-умолчанию dump.rdb).

**Масштабируемость**

Как таковой поддержки масштабируемости в Redis нет, а клиенты умеют подключаться только к одному серверу. Однако имеется ряд приемов для создания масштабируемости клиентскими средствами:

* **Много процессов Redis на одном физическом сервере:**процессы Redis однопоточны и если ваш проект работает на физическом не антикварном оборудовании, то для использования всех ресурсов процессора потребуется запускать несколько процессов Redis на разных портах.
* **Распределение данных по ключам:**естественный хеш на основе ключа. На клиенте держится по соединению на каждый используемый сервер Redis, выбирается нужный и отправляется запрос.
* **Использование репликации:** помимо увеличения максимального объема данных, стоит задумываться и о надежности системы, а также сохранности данных. Как работает репликация мы уже обсуждали, как добавить её к одной из вышеизложенных схем — дело техники. Самый простой вариант — хранить в маршрутизаторе или конфигурационном файле два физических адреса для каждой ВБД, основной и запасной. Запасной же настраивать как подчиненного с логом транзакций, а основной как мастера в режиме кэша или с очень редким созданием копии на диске.

**Дополнительные возможности:**

* **Транзакции:**клиент может отправить команду multi, что означает начало транзакции. Все последующие команды не изменят данные, пока сервер не получит команду exec, либо можно отменить транзакцию командой discard.
* **Публикация и подписка на сообщения:** помимо стандартного использование в роли хранилища данных, Redis может использоваться и как очередь сообщений. Клиент может включить режим ожидания сообщений по определенному ключу/каналу (команды доступа к данным в этом режиме недоступны), а также отправлять произвольные сообщения всем, кто подписан на определенный канал.
* **Мониторинг:**к каждому процессу Redis можно подключиться для просмотра всех поступающих команд или только тех, которые выполнялись больше N микросекунд.

**Базовые операции**

В консоли для общения с Redis используется клиентская программа redis-cli. Рассмотрим базовые операции.  
  
**Задача 1,** создание, выборка, модификация, удаление и базовая информация об объектах.  
Установить и прочитать значения с ключами вида test:1:\*. Узнать тип значения, проверить существование элемента, извлечь все поля записи test:1, удалить поле записи test:1.

redis 127.0.0.1:6379> set test:1:string "my binary safe string"

OK

redis 127.0.0.1:6379> get test:1:string

"my binary safe string"

redis 127.0.0.1:6379> getset test:1:string "other value"

"my binary safe string"

redis 127.0.0.1:6379> type test:1:string

string

redis 127.0.0.1:6379> set test:1:vlaue "487"

OK

redis 127.0.0.1:6379> rename test:1:vlaue test:1:value

OK

redis 127.0.0.1:6379> exists test:1:vlaue

(integer) 0

redis 127.0.0.1:6379> exists test:1:value

(integer) 1

redis 127.0.0.1:6379> keys test:1:\*

1) "test:1:string"

2) "test:1:value"

redis 127.0.0.1:6379> del test:1:value

(integer) 1

redis 127.0.0.1:6379> keys test:1:\*

1) "test:1:string"

Подробности о командах, связанных с этой задачей можно найти в документации в разделах [Keys](http://redis.io/commands#generic) и [Strings](http://redis.io/commands#string).  
  
**Задача 2,** время жизни объекта.

redis 127.0.0.1:6379> ttl test:1:string

(integer) -1

redis 127.0.0.1:6379> expire test:1:string 6000

(integer) 1

redis 127.0.0.1:6379> ttl test:1:string

(integer) 5997

По умолчанию объект не ограничен временем жизни, поэтому TTL возвращает -1, после установки значения командойEXPIRE команда TTL уже возвращает оставшееся количество секунд.  
  
Подробности о командах, связанных с этой задачей можно найти в документации в разделах [Keys](http://redis.io/commands#generic) и [Strings](http://redis.io/commands#string).  
  
**Задача 3,** pipelining, выполнение нескольких команд одним запросом.

PS D:\Programs\Redis\64bit> "set test:1:pvalue 'test'`r`nget test:1:pvalue" | .\redis-cli

OK

"test"

Pipelining удобно использовать для оптимизации массовых вставок.  
  
**Задача 4,** транзакции.  
Для реализации транзакций в Redis используются следующие основные команды:

* MULTI — начать запись команд для транзакции.
* EXEC — выполнить записанные команды.
* DISCARD — удалить все записанные команды.
* WATCH — команда, обеспечивающая поведение типа «check-and-set» (CAS) — транзакция выполняется только в случае, если другие клиенты не изменили значение переменной. Иначе EXEC не выполнит записанные команды.

redis 127.0.0.1:6379> multi

OK

redis 127.0.0.1:6379> set test:1:trValue "1"

QUEUED

redis 127.0.0.1:6379> incr test:1:trValue

QUEUED

redis 127.0.0.1:6379> decr test:1:trValue

QUEUED

redis 127.0.0.1:6379> exec

1) OK

2) (integer) 2

3) (integer) 1

Подробности о командах, связанных с этой задачей можно найти в документации в разделе [Transactions](http://redis.io/topics/transactions).

Строки/числа

**Задача 5,** продемонстрировать основные строковые операции.

redis 127.0.0.1:6379> set test:1:string "hello"

OK

redis 127.0.0.1:6379> append test:1:string " world!"

(integer) 12

redis 127.0.0.1:6379> get test:1:string

"hello world!"

redis 127.0.0.1:6379> strlen test:1:string

(integer) 12

redis 127.0.0.1:6379> getrange test:1:string 6 10

"world"

redis 127.0.0.1:6379> setrange test:1:string 6 "habrahabr!"

(integer) 16

redis 127.0.0.1:6379> get test:1:string

"hello habrahabr!"

**Задача 6,** продемонстрировать операции над числами.

redis 127.0.0.1:6379> set test:1:int 1

OK

redis 127.0.0.1:6379> incr test:1:int

(integer) 2

redis 127.0.0.1:6379> decr test:1:int

(integer) 1

redis 127.0.0.1:6379> incrby test:1:int 20

(integer) 21

redis 127.0.0.1:6379> decrby test:1:int 15

(integer) 6

Подробности о командах, связанных с этими задачами можно найти в документации в разделе [Strings](http://redis.io/commands#string).

Списки

**Задача 7,** создать список, продемонстрировать основные операции над списками.

redis 127.0.0.1:6379> rpush test:1:messages "Hello, world!"

(integer) 1

redis 127.0.0.1:6379> rpush test:1:messages "Hello, user!"

(integer) 2

redis 127.0.0.1:6379> rpush test:1:messages "Wow!"

(integer) 3

redis 127.0.0.1:6379> lrange test:1:messages 0 2

1) "Hello, world!"

2) "Hello, user!"

3) "Wow!"

redis 127.0.0.1:6379> llen test:1:messages

(integer) 3

redis 127.0.0.1:6379> lpop test:1:messages

"Hello, world!"

redis 127.0.0.1:6379> lpop test:1:messages

"Hello, user!"

redis 127.0.0.1:6379> lpop test:1:messages

"Wow!"

redis 127.0.0.1:6379> llen test:1:messages

(integer) 0

Подробности о командах, связанных с этой задачей можно найти в документации в разделе [Lists](http://redis.io/commands#list).

Множества, упорядоченные множества

**Задача 8,** создать множество, продемонстрировать основные операции над множествами.

redis 127.0.0.1:6379> sadd test:1:fruits "banana"

(integer) 1

redis 127.0.0.1:6379> sadd test:1:fruits "apple"

(integer) 1

redis 127.0.0.1:6379> sadd test:1:fruits "strawberry"

(integer) 1

redis 127.0.0.1:6379> sadd test:1:yellowThings "banana"

(integer) 1

redis 127.0.0.1:6379> sadd test:1:yellowThings "apple"

(integer) 1

redis 127.0.0.1:6379> sadd test:1:redThings "strawberry"

(integer) 1

redis 127.0.0.1:6379> scard test:1:fruits

(integer) 3

redis 127.0.0.1:6379> sdiff test:1:fruits test:1:yellowThings

1) "strawberry"

redis 127.0.0.1:6379> sdiffstore test:1:noYellowFruits test:1:fruits test:1:yellowThings

(integer) 1

redis 127.0.0.1:6379> sinter test:1:yellowThings test:1:fruits

1) "banana"

2) "apple"

redis 127.0.0.1:6379> sismember test:1:fruits "banana"

(integer) 1

redis 127.0.0.1:6379> sismember test:1:fruits "tomato"

(integer) 0

redis 127.0.0.1:6379> smembers test:1:noYellowFruits

1) "strawberry"

redis 127.0.0.1:6379> smove test:1:yellowThings test:1:redThings "apple"

(integer) 1

redis 127.0.0.1:6379> smembers test:1:redThings

1) "strawberry"

2) "apple"

redis 127.0.0.1:6379> spop test:1:redThings

"apple"

redis 127.0.0.1:6379> srandmember test:1:fruits

"apple"

redis 127.0.0.1:6379> srem test:1:fruits "banana"

(integer) 1

redis 127.0.0.1:6379> sunion test:1:yellowThings test:1:redThings

1) "strawberry"

2) "banana"

redis 127.0.0.1:6379> sunionstore test:1:allThings test:1:yellowThings test:1:redThings

(integer) 2

redis 127.0.0.1:6379> smembers test:1:allThings

1) "strawberry"

2) "banana"

Подробности о командах, связанных с этой задачей можно найти в документации в разделе [Sets](http://redis.io/commands#set).  
  
**Задача 9,** создать упорядоченное множество и продемонстрировано основные операции над ним.  
В упорядоченном множестве элементы сравниваются по дополнительному параметру «score».

redis 127.0.0.1:6379> zadd hackers 1953 "Richard Stallman"

(integer) 1

redis 127.0.0.1:6379> zadd hackers 1940 "Alan Kay"

(integer) 1

redis 127.0.0.1:6379> zadd hackers 1965 "Yukihiro Matsumoto"

(integer) 1

redis 127.0.0.1:6379> zadd hackers 1916 "Claude Shannon"

(integer) 1

redis 127.0.0.1:6379> zadd hackers 1969 "Linus Torvalds"

(integer) 1

redis 127.0.0.1:6379> zadd hackers 1912 "Alan Turing"

(integer) 1

redis 127.0.0.1:6379> zrange hackers 0 -1

1) "Alan Turing"

2) "Claude Shannon"

3) "Alan Kay"

4) "Richard Stallman"

5) "Yukihiro Matsumoto"

6) "Linus Torvalds"

redis 127.0.0.1:6379> zrevrange hackers 0 -1

1) "Linus Torvalds"

2) "Yukihiro Matsumoto"

3) "Richard Stallman"

4) "Alan Kay"

5) "Claude Shannon"

6) "Alan Turing"

redis 127.0.0.1:6379> zrangebyscore hackers -inf 1950

1) "Alan Turing"

2) "Claude Shannon"

3) "Alan Kay"

redis 127.0.0.1:6379> zremrangebyscore hackers 1940 1960

(integer) 2

redis 127.0.0.1:6379> zrange hackers 0 -1

1) "Alan Turing"

2) "Claude Shannon"

3) "Yukihiro Matsumoto"

4) "Linus Torvalds"

Подробности о командах, связанных с этой задачей можно найти в документации в разделе [Sorted sets](http://redis.io/commands#sorted_set).

Хеш-таблицы

**Задача 10,** Создать хеш-таблицу и продемонстрировать основные операции над хешами.

redis 127.0.0.1:6379> hset users:1 name "Andrew"

(integer) 1

redis 127.0.0.1:6379> hset users:1 email "andrew@example.com"

(integer) 1

redis 127.0.0.1:6379> hkeys users:1

1) "name"

2) "email"

redis 127.0.0.1:6379> hvals users:1

1) "Andrew"

2) "andrew@example.com"

redis 127.0.0.1:6379> hgetall users:1

1) "name"

2) "Andrew"

3) "email"

4) "andrew@example.com"

redis 127.0.0.1:6379> hset users:1 test 1

(integer) 1

redis 127.0.0.1:6379> hincrby users:1 test 123

(integer) 124

redis 127.0.0.1:6379> hvals users:1

1) "Andrew"

2) "andrew@example.com"

3) "124"

redis 127.0.0.1:6379> hdel users:1 test

(integer) 1

redis 127.0.0.1:6379> hvals users:1

1) "Andrew"

2) "andrew@example.com"

Подробности о командах, связанных с этой задачей можно найти в документации в разделе [Hashes](http://redis.io/commands#hash).

Pub/Sub, сообщения в Redis

**Задача 11,** подписаться на сообщения на одном клиенте и отправить сообщение из другого.  
Приведем окна двух клиентов, в первом окне совершается подписка на сообщения и видно отправленное из второго окна сообщение.

redis 127.0.0.1:6379> SUBSCRIBE messages

Reading messages... (press Ctrl-C to quit)

1) "subscribe"

2) "messages"

3) (integer) 1

1) "message"

2) "messages"

3) "Hello world!"

redis 127.0.0.1:6379> PUBLISH messages "Hello world!"

(integer) 1

**Задача 12,** подписаться на сообщения на одном клиенте и отправить сообщение из другого. Подписку осуществить с помощью шаблонов.  
Приведем окна двух клиентов, в первом окне совершается подписка на сообщения и видно отправленное из второго окна сообщение.

redis 127.0.0.1:6379> PSUBSCRIBE news.\*

Reading messages... (press Ctrl-C to quit)

1) "psubscribe"

2) "news.\*"

3) (integer) 1

1) "pmessage"

2) "news.\*"

3) "news.art"

4) "New picture!"

1) "pmessage"

2) "news.\*"

3) "news.cinema"

4) "New movie!"

redis 127.0.0.1:6379> PUBLISH news.art "New picture!"

(integer) 1

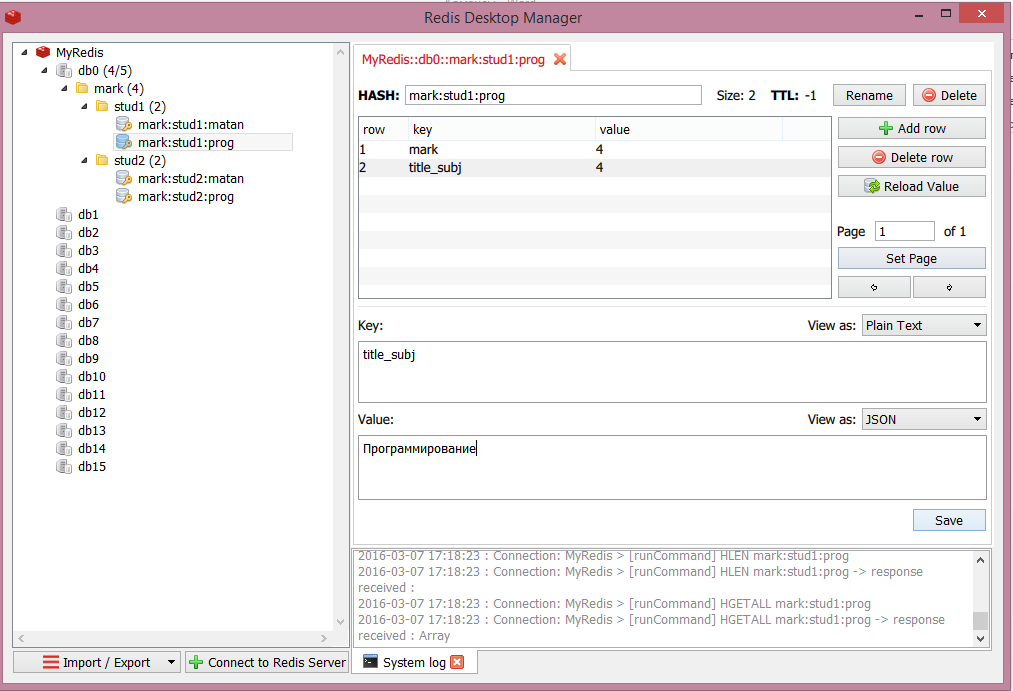
redis 127.0.0.1:6379> PUBLISH news.cinema "New movie!"

(integer) 1

Подробности о командах, связанных с этими задачами можно найти в документации в разделе [Pub/Sub](http://redis.io/commands#pubsub).

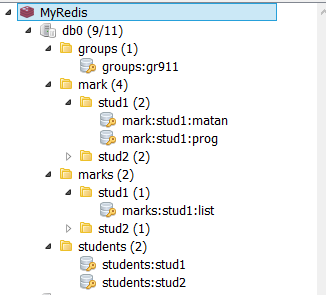
**Разберем на примере некоторые возможности Redis.**

В качестве программной оболочки можно использовать, к примеру, Redis Desktop Manager:



Рассмотрим следующая структуру хранения информации про успеваемость студентов:

* Отдельно хранится информация об оценке как хэш-таблица с ключами title\_subj и mark (ключ формируется по принципу mark:ключ\_студента:ключ\_предмета);
* Оценки группируются в список оценок конкретного студента (marks:ключ\_студента:list);
* Информация о студенте хранится в виде хэш-таблицы с полями FIO, REYTING и marks;
* Студенты объединены в группы – множество.

J

Приведем некоторые команды для добавления, изменения и поиска информации:

* Показать FIO студента с заданным ключом: hmget students:stud1 FIO
* Показать все данные (поля и значения) заданного студента: hgetall students:stud1
* Показать все ключи для заданного студента: hkeys students:stud1
* Показать все значения для заданного студента: hvals students:stud1
* Добавить или изменить свойство в хэш-таблице студента, например, назначить старосту: hset students:stud1 Leader "ДА"
* Зафиксировать сдачу экзамена по алгебре студента Иванова:

hset mark:stud1:alg title\_subj Алгебра

hset mark:stud1:alg mark 4

rpush marks:stud1:list mark:stud1:alg

* Показать все ключи оценок заданного студента, сначала узнать количество, потом печатать диапазон из списка:

llen marks:stud1:list

lrange marks:stud1:list 0 3

* Добавить нового студента в группу:

hset students:stud3 FIO "Кузнецов К.К."

sadd groups:gr911 students:stud3

Показать все ключи студентов группы:

smembers groups:gr911

Для работы с Redis из клиентских приложений существуют специальные библиотеки и провайдеры, например, для Java существует механизм работы через JDBC-драйвер:

|  |
| --- |
|  |
|  | import java.sql.Connection;  import java.sql.DriverManager; |
|  | import java.sql.ResultSet; |
|  | import java.sql.SQLException; |
|  | import java.sql.Statement; |
|  |  |
|  | public class RedisStatementSample { |
|  |  |
|  | public static void main(String[] args) throws SQLException, ClassNotFoundException { |
|  |  |
|  | Class.forName("br.com.svvs.jdbc.redis.RedisDriver"); |
|  |  |
|  | Connection conn = DriverManager.getConnection("jdbc:redis://localhost"); |
|  |  |
|  | Statement statement = conn.createStatement(); |
|  |  |
|  | statement.execute("set my\_first\_key my first value"); |
|  | statement.execute("get my\_first\_key"); |
|  |  |
|  | ResultSet r = statement.getResultSet(); |
|  | while(r.next()) { |
|  | System.out.println(">" + r.getString(0) + "<"); |
|  | } |
|  |  |
|  | conn.commit(); |
|  | conn.close(); |
|  |  |
|  | } |
|  | } |