

# Системы хранения и передачи данных: Кэширование, Redis/Memcached



Иван  
Богданов



## Иван Богданов

Технический менеджер, Яндекс.Облако



[Иван Богданов](#)



# План занятия

1. [Что такое кэширование?](#)
2. [Какие проблемы призвано решить кэширование?](#)
3. [Алгоритмы кэширования](#)
4. [Memcached](#)
5. [Redis](#)
6. [Redis Sentinel и Cluster](#)
7. [Итоги](#)
8. [Домашнее задание](#)



## Вопросы с прошлой лекции

**Вопрос:** Какие типы СУБД соответствуют требованиям ACID, а какие BASE?



## Вопросы с прошлой лекции

**Вопрос:** Какие типы СУБД соответствуют требованиям ACID, а какие BASE?

**Ответ:** Реляционные — ACID, NoSQL — BASE.



## Вопросы с прошлой лекции

**Вопрос:** Какие архитектурные многопользовательские модели вы помните с прошлого занятия?



## Вопросы с прошлой лекции

**Вопрос:** Какие архитектурные многопользовательские модели вы помните с прошлого занятия?

**Ответ:** Централизованные, файл-серверные, клиент-серверные.



# Что такое кэширование?



---

## Что такое кэширование?

**Кэширование** — это способ оптимизации работы приложения, при котором данные временно кладутся в промежуточный буфер с быстрым доступом.


Выбор буфера для кэша зависит от решаемой задачи и может быть как памятью сервера, так и файлом, базой данных или любой другой сущностью, куда можно положить данные.

Обычно данные в кэше — это данные, к которым наиболее часто осуществляется запрос.

---

## Важно учесть при работе с кэшем

- **Инвалидация.** Если данные изменяемые (например, кэш ответов из базы данных), то их нужно будет инвалидировать при изменении в источнике;
- **TTL (time-to-live).** Кэш не может жить вечно, и нужно быть готовым к тому, что он «протухнет» или «потеряется в любой момент»;
- Кэшировать данные нужно **только при необходимости**, так как это усложняет архитектуру и может добавить проблем в эксплуатации.



**Какие проблемы призвано  
решить кэширование?**

---

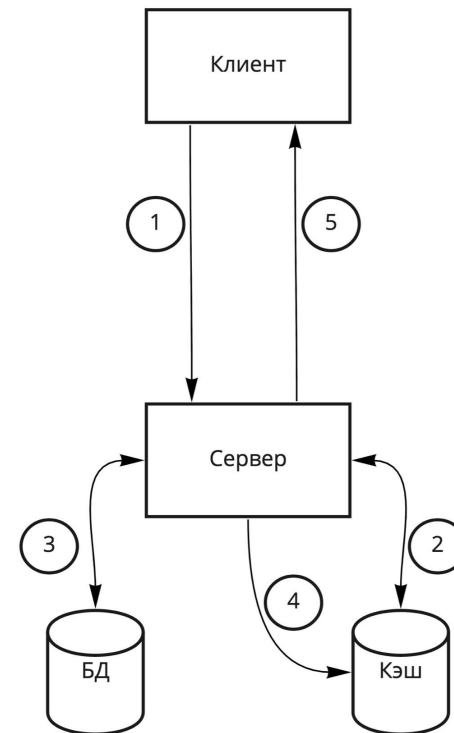
## Какие проблемы призвано решить кэширование?

- **Повышение производительности** достигается за счет складывания в кэш данных, к которым чаще всего происходит обращение;
- **Увеличение скорости ответа;**
- **Экономия ресурсов** базы данных, например, применяя кэширование тяжелых запросов;
- **Сглаживание бустов трафика.** Например, во время черной пятницы онлайн-магазины используют кэш, чтобы переживать резкое увеличение трафика.

# Какие проблемы призвано решить кэширование?

Понятие кэширования очень растяжимое:

- DNS;
- CDN;
- Кэш процессора;
- Мобильное приложение;
- Кэш внутри сервиса.





# Алгоритмы кэширования



## Самый оптимальный алгоритм

Самая оптимальная стратегия кэширования — это подход, при котором мы кэшируем только то, что будем использовать в будущем и убираем из кэша то, что использовать не будем.

Однако, чтобы добиться этого, нужно уметь предсказывать будущее.

## LRU (least recently used)

**LRU** (*вытеснение давно неиспользуемых*) — это алгоритм, при котором вытесняются значения, которые дольше всего не запрашивались. Соответственно, необходимо хранить время последнего запроса к значению. И как только число закэшированных значений превосходит  $N$ , необходимо вытеснить из кеша значение, которое дольше всего не запрашивалось.



## LFU (Least-Frequently Used)

**LFU** (наименее часто используемый) — из кэша вытесняются элементы, которые реже всего используются. Для этого у каждого элемента в кэше ведется счетчик обращения к нему.

В таком виде алгоритм может давать сбои и вытеснять только что добавленные элементы, к которым еще ни разу не было повторного обращения.

Редко можно встретить реальное использование.



# Memcached



# Memcached

Бесплатная высокопроизводительная система кэширования с открытым исходным кодом. Универсальная по своей природе, но предназначенная для ускорения работы динамических веб-приложений за счет снижения нагрузки на базу данных.

**Memcached** — это in-memory хранилище значений ключей для небольших фрагментов произвольных данных (строк, объектов) из результатов вызовов баз данных, API или рендеринга страниц.



# Memcached

Был написан в LiveJournal Брэдом Фицпатриком и выпущен в OSS в 2003 году. С тех пор оброс большим комьюнити.

Является простым и популярным решением для in-memory кэширования даже в наши дни.

Из коробки умеет шардировать ключи с помощью клиентской библиотеки.

Работает на порту 11211.

Работает с кэшем по алгоритму LRU с TTL.

---

# Memcached

База сохраняет свою простоту, но:

- **Нет неймспейсов.** Чтобы отделить ключи одного приложения от другого, нужно либо использовать некрасивые префиксы, либо запускать отдельный инстанс memcached;
- **Нельзя флашнуть сабсет ключей.** Из-за того что ключи никак не сгруппированы, нельзя удалить часть и нельзя запросить часть для удаления;
- **In-memory.** Рестарт базы удалит все данные;
- **Нет репликации** из коробки;
- **Нет типизации.** Все данные — это строки.

# Memcached

Установка memcached на сервер с Debian 10 с помощью apt:

```
$ sudo apt update && apt install memcached
```

Убеждаемся, что база запустилась:

```
$ systemctl status memcached
● memcached.service - memcached daemon
   Loaded: loaded (/lib/systemd/system/memcached.service; enabled; vendor
  preset: enabled)
   Active: active (running) since Sun 2021-08-15 21:10:46 UTC; 1s ago
     Docs: man:memcached(1)
    Main PID: 1591 (memcached)
```

Если клиентом memcached будет не localhost, то переопределяем IP-адрес, который слушает база в конфиге /etc/memcached.conf:

```
# Specify which IP address to listen on. The default is to listen on all IP
addresses
# This parameter is one of the only security measures that memcached has, so
make sure
# it's listening on a firewalled interface.
-l 127.0.0.1
```

# Memcached

Интерфейс управления — это порт, например, добавление ключа с ttl 20 секунд:

```
# telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
add key 0 20 5
value
STORED
get key
VALUE key 0 5
value
END
```



## Memcached и масштабирование

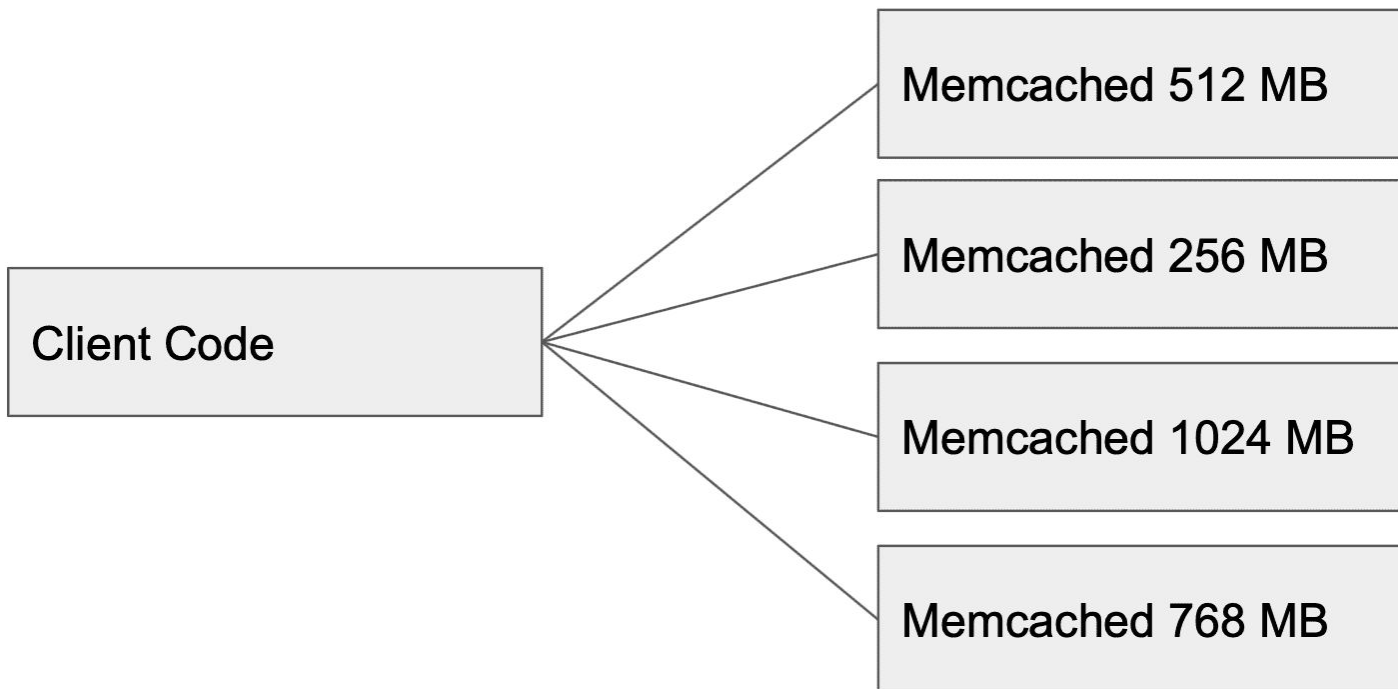
Масштабирование memcached сложная тема, оно, вроде, есть, но его нет. Масштабироваться можно через добавление нод в кластер. Они могут отличаться настройками и ничего друг про друга не знают.

Вся балансировка чтения/записи происходит на клиенте.



# Memcached и масштабирование

Как схематично выглядит балансировка на клиенте:



# Memcached и масштабирование

**Вопрос:** Если балансировка происходит на клиенте, то как же memcached распределяет ключи по нодам?

**Ответ:** Хеширование ключа, и взятие остатка от деления на количество серверов:

```
$server_target = inthash($key) % $servercount;
```

Отсюда банальная проблема - при изменении servercount мы не сможем ни читать ни писать.

Есть возможность писать во все ноды параллельно, но проблему это не решает.

# Memcached и масштабирование

Запустим memcached в docker-compose, попробуем записать данные и убить один инстанс:

```
version: '3.7'

services:
  memcached1:
    image: 'memcached:latest'
    networks:
      memcached-net:
        ipv4_address: 10.5.0.2

  memcached2:
    image: 'memcached:latest'
    networks:
      memcached-net:
        ipv4_address: 10.5.0.3

networks:
  memcached-net:
    driver: bridge
    ipam:
      config:
        - subnet: 10.5.0.0/16
```



## Memcached и масштабирование

Если все-таки нужно настроить репликацию в memcached, придется брать стороннее решение — [mcrouter](#).

Он был написал в Facebook, чтобы решить проблемы масштабирования.

---

# Memcached и масштабирование

Среди прочего, он умеет:

- реплицировать запись;
- понимать, что данных нет в ноде, или она недоступна и читать с соседней (fallback);
- автоматическая проверка живости нод memcached.

# Memcached и масштабирование

Пишем конфиг mcrouter:

```
{
  "pools": {
    "A": {
      "servers": [
        "10.5.0.2:11211",
        "10.5.0.3:11211"
      ]
    }
  },
  "route": {
    "type": "OperationSelectorRoute",
    "operation_policies": {
      "delete": "AllSyncRoute|Pool|A",
      "get": "FailoverRoute|Pool|A",
      "gets": "FailoverRoute|Pool|A",
      "set": "AllSyncRoute|Pool|A"
    }
  }
}
```

# Memcached и масштабирование

Описание применяемых в конфиге опций:

- AllSyncRoute — один и тот же запрос отправляется всем нодам. Клиенту отвечаем худшим ответом;
- FailoverRoute — если первая нода из пула ответила ошибкой, продолжаем обходить ноды.

Больше описания есть в [wiki mcrouter](#).

# Memcached и масштабирование

Пишем конфиг mcrouter:

```
...
  mcrouter:
    image: 'mcrouter/mcrouter:latest'
    volumes:
      - /root/config.json:/etc/config.json
    networks:
      memcached-net:
        ipv4_address: 10.5.0.4
    command: "-f /etc/config.json -p 11211"
...
```





# Redis

# Redis



**Redis** (*remote dictionary server*) — это in-memory хранилище структур данных с открытым исходным кодом, используемое в качестве базы данных, кэша и очереди сообщений.

## Select a ranking

- Complete ranking
- Relational DBMS
- Key-value stores
- Document stores
- Time Series DBMS
- Graph DBMS
- Object oriented DBMS
- Search engines
- RDF stores
- Wide column stores
- Multivalued DBMS
- Native XML DBMS
- Spatial DBMS
- Event Stores
- Content stores
- Navigational DBMS

## Special reports

- Ranking by database model
- Open source vs. commercial

## Featured Products

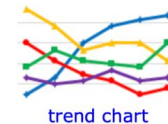
[Ranking](#) > Complete Ranking

[RSS](#) [RSS Feed](#)

## DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the [method](#) of calculating the scores.



373 systems in ranking, August 2021

Rank			DBMS	Database Model	Score		
Aug 2021	Jul 2021	Aug 2020			Aug 2021	Jul 2021	Aug 2020
1.	1.	1.	Oracle +	Relational, Multi-model i	1269.26	+6.59	-85.90
2.	2.	2.	MySQL +	Relational, Multi-model i	1238.22	+9.84	-23.36
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model i	973.35	-8.61	-102.53
4.	4.	4.	PostgreSQL +	Relational, Multi-model i	577.05	-0.10	+40.28
5.	5.	5.	MongoDB +	Document, Multi-model i	496.54	+0.38	+52.98
6.	6.	7.	Redis +	Key-value, Multi-model i	169.88	+1.58	+17.01
7.	7.	6.	IBM Db2	Relational, Multi-model i	165.46	+0.31	+3.01
8.	8.	8.	Elasticsearch	Search engine, Multi-model i	157.08	+1.32	+4.76
9.	9.	9.	SQLite +	Relational	129.81	-0.39	+3.00
10.	11.	10.	Microsoft Access	Relational	114.84	+1.39	-5.02
11.	10.	11.	Cassandra +	Wide column	113.66	-0.35	-6.18
12.	12.	12.	MariaDB +	Relational, Multi-model i	98.98	+0.99	+8.06

---

# Redis

Отличительные характеристики:

- **Есть неймспейсы.** На старте базы их 16, но это значение меняется в конфиге;
- **In-memory.** Но данные персистентно хранятся на диске;
- **Репликация** из коробки;
- **Типизация.** Есть поддержка разных типов данных;
- **Lua-скриптинг.** Позволяет писать свои хранимые процедуры;
- **Транзакции.**

---

# Redis

Политики вытеснения работают только тогда, когда у redis закончилось место в памяти:

- **noeviction**: возвращает ошибки, когда достигнут предел памяти и клиент пытается выполнить команды, которые могут привести к увеличению объема используемой памяти;
- **allkeys-lru**: ключи удаляются по алгоритму LRU;
- **volatile-lru**: удаляются только ключи с истекшим TTL;
- **allkeys-random**: случайное удаление ключей;
- ...

---

# Redis

- **allkeys-random**: случайное удаление ключей;
- **volatile-random**: случайное удаление ключей с истекшим TTL;
- **volatile-ttl**: удаление ключей с истекшим TTL и с наиболее коротким TTL;
- **volatile-lfu**: удаляются ключи с истекшим TTL по LFU;
- **allkeys-lfu**: все ключи удаляются по LFU.

# Redis

Установка redis на сервер с Debian 10 с помощью apt:

```
$ sudo apt update && apt install redis
```

Убеждаемся, что база запустилась:

```
$ systemctl status redis
• redis-server.service - Advanced key-value store
  Loaded: loaded (/lib/systemd/system/redis-server.service; enabled; vendor
  preset: enabled)
  Active: active (running) since Tue 2021-08-17 06:22:23 UTC; 24min ago
```

Идем в redis-cli после установки:

```
# redis-cli
127.0.0.1:6379> INFO
# Server
redis_version:5.0.3
```

# Redis

Немного примеров команд:

```
$ redis-cli
127.0.0.1:6379> incr mycounter
(integer) 1
127.0.0.1:6379> incr mycounter
(integer) 2
127.0.0.1:6379> get mycounter
"2"
127.0.0.1:6379> set testkey hello
OK
127.0.0.1:6379> get testkey
"hello"
127.0.0.1:6379> set newtestkey "will expire in a 5 sec" EX 5
OK
127.0.0.1:6379> get newtestkey
"will expire in a 5 sec"
127.0.0.1:6379> get newtestkey
(nil)
127.0.0.1:6379> flushall
OK
127.0.0.1:6379> ping
PONG
```

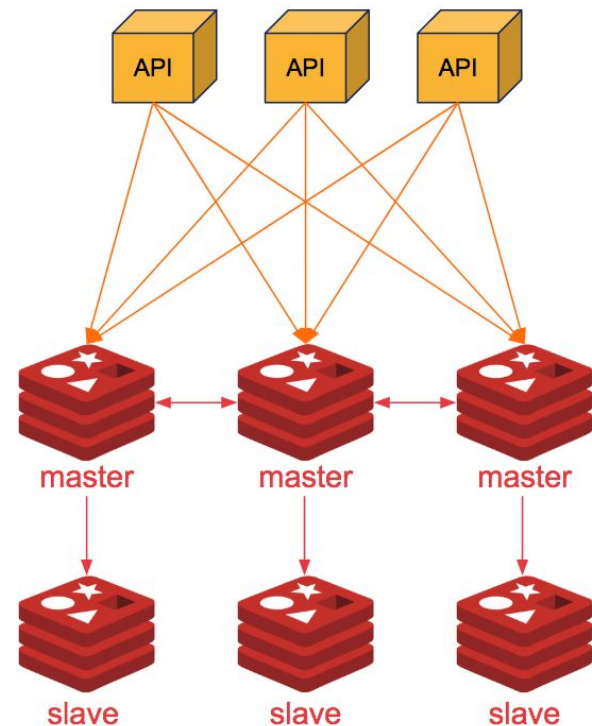


# Redis Sentinel и Cluster



# Redis Cluster

**Cluster** предоставляет удобный способ запуска множества отказоустойчивых нод redis и умеет автоматически шардировать данные. Каждый шард кластера состоит из мастера и набора реплик.



Источник

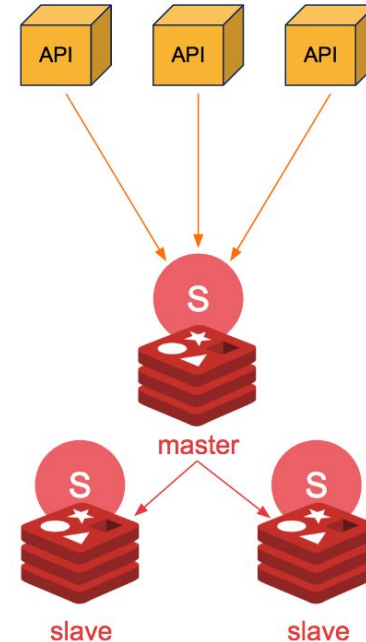
---

## Важные особенности cluster

- Для минимального запуска требуется 6 нод: 3 master и 3 slave;
- Несколько master-нод, у каждой может быть до 1000 слейвов;
- Поддерживает шардинг, репликацию, переключение мастера и синхронизацию данных.

# Redis Sentinel

**Sentinel** был добавлен в Redis v.2.4, и является сервисом мониторинга состояния мастер и слейв нод.



Источник

---

## Важные особенности sentinel

- Может работать как отдельный демон или как redis в режиме sentinel;
- В случае выхода из строя мастера, переключает его на один из слейвов;
- Для работы в идеале требуется минимум 3 ноды, чтобы собирать *кворум* на переключение мастера;
- Клиенты подключаются к sentinel, чтобы узнать, кто сейчас мастер в кластере.

# Redis Sentinel

Настроим redis sentinel. Подготовим окружение:

```
version: '3.7'

services:
  redis1:
    image: 'redis:latest'
    networks:
      redis-net:
        ipv4_address: 10.6.0.2

  redis2:
    image: 'redis:latest'
    networks:
      redis-net:
        ipv4_address: 10.6.0.3

  redis3:
    image: 'redis:latest'
    networks:
      redis-net:
        ipv4_address: 10.6.0.4

networks:
  redis-net:
    driver: bridge
    ipam:
      config:
        - subnet: 10.6.0.0/16
```

---

# Redis Sentinel

Настраиваем репликацию:

```
10.6.0.4:6379> SLAVEOF 10.6.0.2 6379
OK
10.6.0.4:6379> info replication
# Replication
role:slave
master_host:10.6.0.2
master_port:6379
master_link_status:up
master_last_io_seconds_ago:1
master_sync_in_progress:0
slave_read_repl_offset:0
slave_repl_offset:0
slave_priority:100
slave_read_only:1
replica_announced:1
connected_slaves:0
master_failover_state:no-failover
```

# Redis Sentinel

Настраиваем sentinel. Создаем конфиг sentinel.conf:

```
$ cat sentinel.conf
sentinel monitor mymaster 10.6.0.2 6379 1
sentinel down-after-milliseconds mymaster 5000
sentinel failover-timeout mymaster 10000
```

Запускаем redis на хост-машине в режиме sentinel:

```
$ redis-server sentinel.conf --sentinel
```



# Redis vs Memcached

Вопрос: когда использовать redis, а когда memcached?





## Redis vs Memcached

Вопрос: когда использовать redis, а когда memcached?

Ответ: Memcached имеет смысл использовать только, когда нужна максимальная простота и у вас мало серверов, мало запросов и они точно не будут расти.



## Redis vs Memcached

Вопрос: когда использовать redis, а когда memcached?

Ответ: Memcached имеет смысл использовать только, когда нужна максимальная простота и у вас мало серверов, мало запросов и они точно не будут расти.

Но даже в этом случае с задачей лучше справится redis.



# Итоги

---

# Итоги

## Сегодня мы:

- узнали, что такое кэширование и какие проблемы оно решает;
- какие бывают алгоритмы кэширования;
- что такое memcached и redis и как они работают.





# Домашнее задание



## Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

---

## Дополнительные материалы

- [Кэширование;](#)
- [Про LRU и Pseudo LRU;](#)
- [Сравнение redis и memcached;](#)
- [Репликация redis;](#)
- [Mcrouter.](#)
- [Еще одно сравнение redis и memcached.](#)

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Иван Богданов**