Architecture of Enterprise Applications 6
Memory Caching

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

- Contents
  - MemCached
  - Redis
  - Caching Samples in Spring Web Applications

- Objectives
  - 能够根据业务需求，设计并实现基于分布式缓存的数据访问方案，实现数据访问性能的优化

MemCached

- Memcached
  - Free & open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

- Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

- Memcached is simple yet powerful.
  - Its simple design promotes quick deployment, ease of development, and solves many problems facing large data caches.
  - Its API is available for most popular languages.

- At heart it is a simple Key/Value store.

MemCached

- Installing memcached with Homebrew and Lunchy

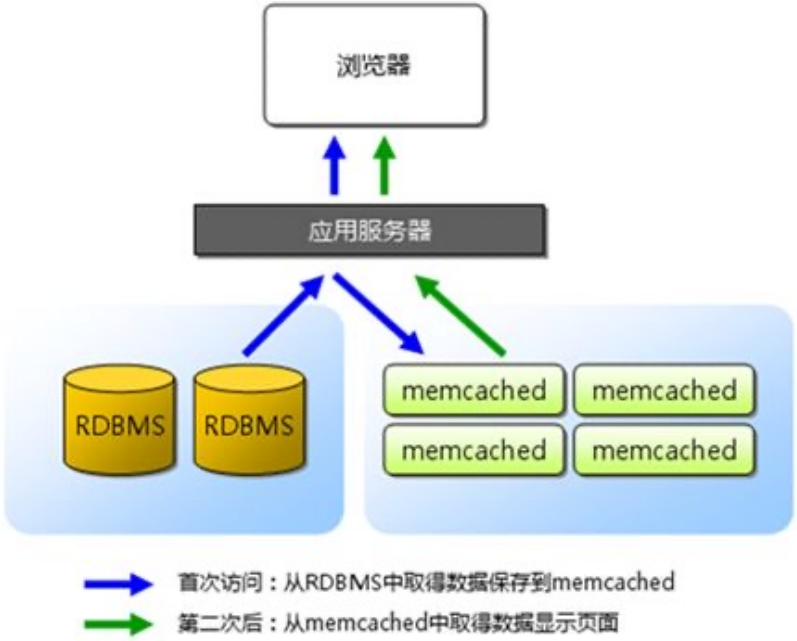
- Step 1 — Install Homebrew
  - $ ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)”

- Step 2 — Install memcached
  - $ brew install Memcached

- Step 3 — Install Lunchy
  - $ gem install lunchy

- Step 4 — Start/Stop memcached
  - $ mkdir ~/Library/LaunchAgents
  - $ cp /usr/local/Cellar/memcached/$version/homebrew.mxcl.memcached.plist ~/Library/LaunchAgents/
  - $ lunchy start memcached
  - $ lunchy stop memcached

MemCached - Principle



MemCached - Example

- LocalSSMConfiguration.java

```
@Configuration
public class LocalSSMConfiguration extends AbstractSSMConfiguration {
   @Bean
   @Override
   public CacheFactory defaultMemcachedClient() {
      final CacheConfiguration conf = new CacheConfiguration();
      conf.setConsistentHashing(true);
      final CacheFactory cf = new CacheFactory();
      cf.setCacheClientFactory(new com.google.code.ssm.providers.xmemcached.MemcacheClientFactoryImpl());
      cf.setAddressProvider(new DefaultAddressProvider("127.0.0.1:11211"));
      cf.setConfiguration(conf);
      return cf;
   }
}
```

MemCached - Example

- Person.java

```
@Entity
@Table(name = "persons", schema = "ormsample")
@JsonIgnoreProperties(value = {"handler","hibernateLazyInitializer","fieldHandler"})
@JsonIdentityInfo(
      generator = ObjectIdGenerators.PropertyGenerator.class,
      property = "personId")
public class Person implements Serializable {
   private int personId;
   private int age;
   private String firstname;
   private String lastname;

   @Id
   @Column(name = "id")
   public int getPersonId() {
      return personId;
   }
   .....
```

MemCached - Example

- PersonRepository.java

```
public interface PersonRepository extends JpaRepository<Person, Integer>{}
```

- PersonDao.java

```
public interface PersonDao {
   Person findOne(Integer id);
}
```

- PersonDaoImpl.java

```
@Repository
public class PersonDaoImpl implements PersonDao {
   @Autowired
   private PersonRepository personRepository;
   @Override
   @ReadThroughSingleCache(namespace = "Alpha", expiration = 3600)
   public Person findOne(@ParameterValueKeyProvider Integer id) {
      return personRepository.getOne(id);
   }
}
```

MemCached - Example

- PersonService.java

```
public interface PersonService {
   Event findPersonById(Integer id);
}
```

- PersonServiceImpl.java

```
@Transactional
@Service
public class PersonServiceImpl implements PersonService {

   @Autowired
   private PersonDao personDao;

   @Override
   public Person findPersonById(Integer id){
      return personDao.findOne(id);
   }
}
```

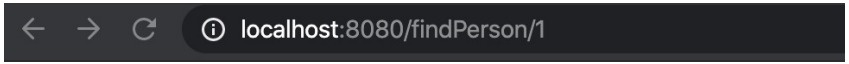
MemCached - Example

- pom.xml

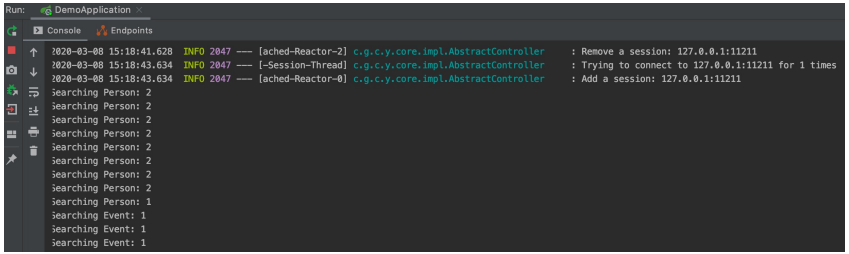
```
<!-- https://mvnrepository.com/artifact/com.googlecode.xmemcached/xmemcached --
>
<dependency>
  <groupId>com.google.code.simple-spring-memcached</groupId>
  <artifactId>xmemcached-provider</artifactId>
  <version>4.1.3</version>
</dependency>
```

MemCached - Example

- Start Memcached
  - $lunchy start Memcached

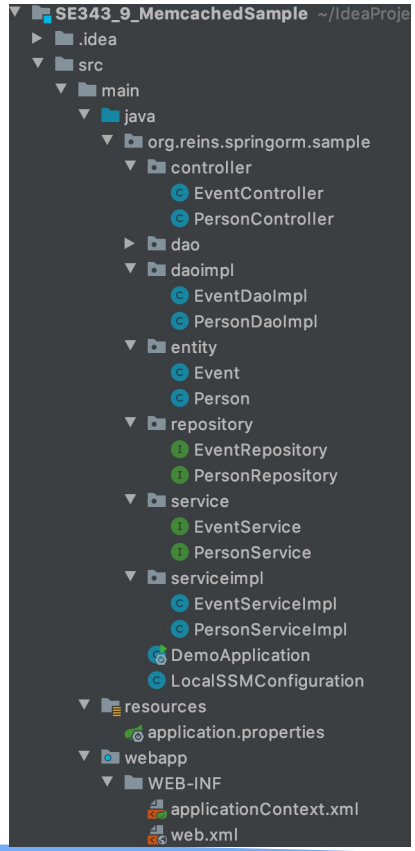


{"personId":1,"age":54,"firstname":"Cao","lastname":"Cao"}



  - $telnet localhost 11211

```
stats cachedump 12 2
ITEM Alpha:1 [1056 b; 1583655630 s]
ITEM Alpha:2 [1056 b; 1583655525 s]
```



MemCached - Storage Commands

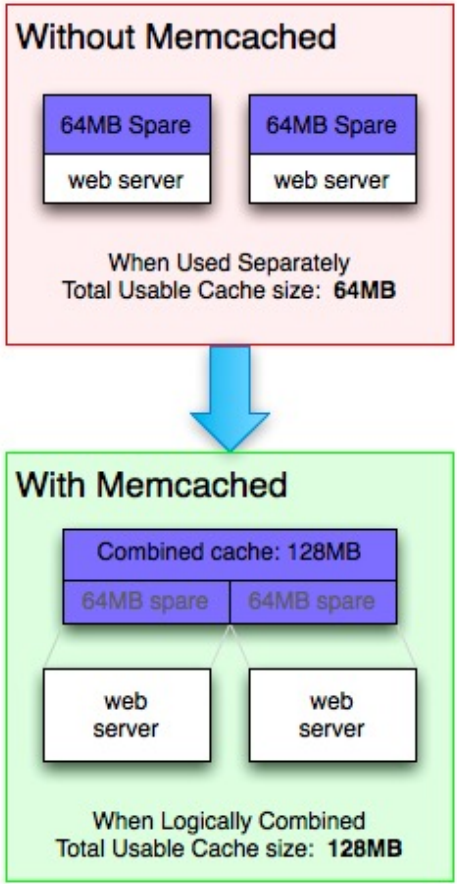
- **set**
  - Most common command. Store this data, possibly overwriting any existing data. New items are at the top of the LRU.
- **add**
  - Store this data, only if it does not already exist. New items are at the top of the LRU. If an item already exists and an add fails, it promotes the item to the front of the LRU anyway.
- **replace**
  - Store this data, but only if the data already exists. Almost never used, and exists for protocol completeness (set, add, replace, etc)
- **append**
  - Add this data after the last byte in an existing item. This does not allow you to extend past the item limit. Useful for managing lists.
- **prepend**
  - Same as append, but adding new data before existing data.
- **cas**
  - Check And Set (or Compare And Swap). An operation that stores data, but only if no one else has updated the data since you read it last. Useful for resolving race conditions on updating cache data.

MemCached - Retrieval Commands

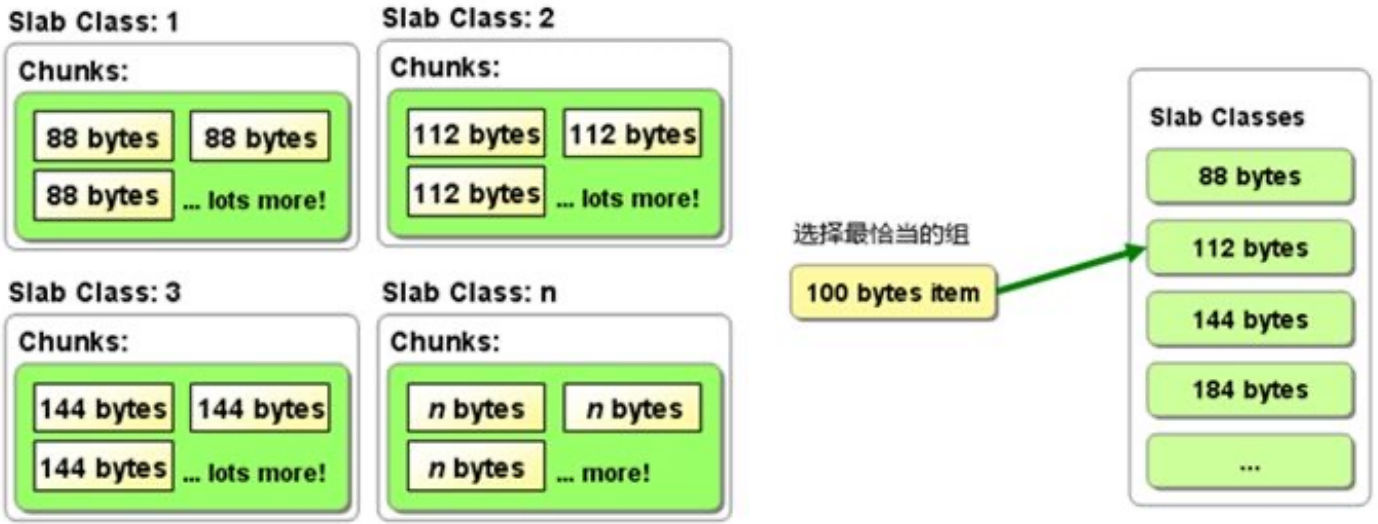
- **get**
  - Command for retrieving data. Takes one or more keys and returns all found items.
- **gets**
  - An alternative get command for using with CAS. Returns a CAS identifier (a unique 64bit number) with the item. Return this value with the cas command. If the item's CAS value has changed since you gets'ed it, it will not be stored.
- **delete**
  - Removes an item from the cache, if it exists.
- **incr/decr**
  - Increment and Decrement. If an item stored is the string representation of a 64bit integer, you may run incr or decr commands to modify that number. You may only incr by positive values, or decr by positive values. They does not accept negative values.
  - If a value does not already exist, incr/decr will fail.

MemCached

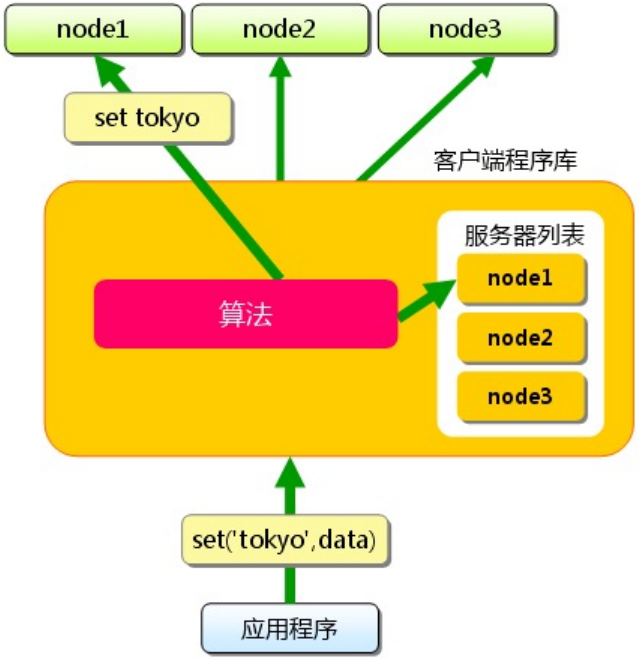
- memcached also allows you to make better use of your memory.
  - Each node is completely independent (top).
  - Each node can make use of memory from other nodes (bottom)

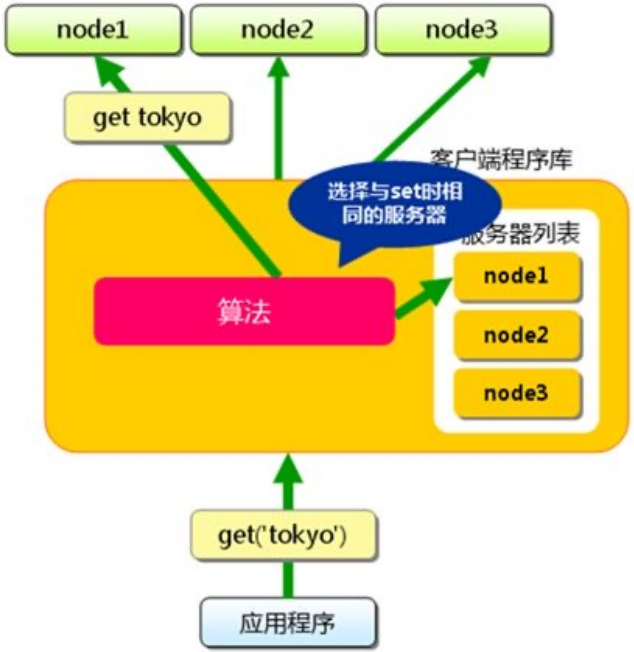


MemCached - Memory Management

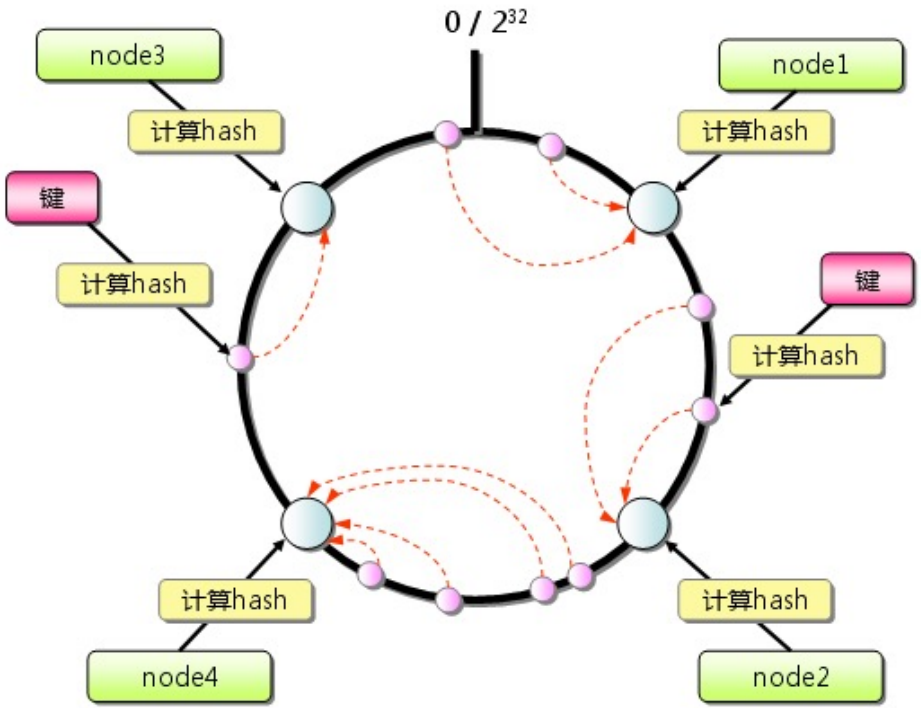


MemCached - Distribution

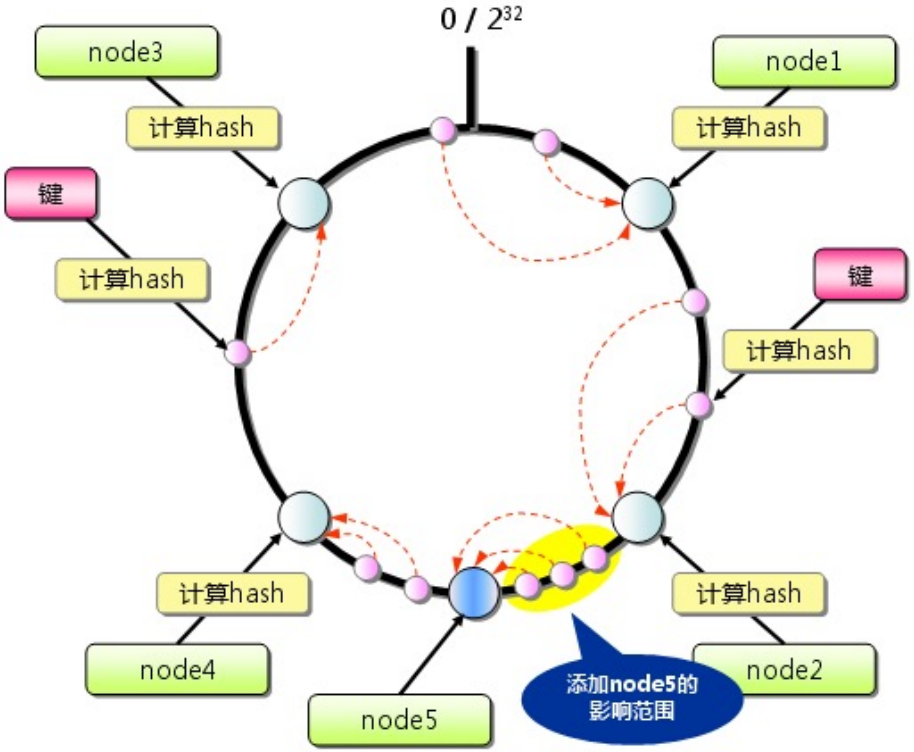




MemCached - Distribution



MemCached - Distribution



Redis

- Redis is what is called a key-value store,
  - often referred to as a NoSQL database.
  - The essence of a key-value store is the ability to store some data, called a value, inside a key.

- Redis is an open source, BSD licensed, advanced key-value store.
  - It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets.

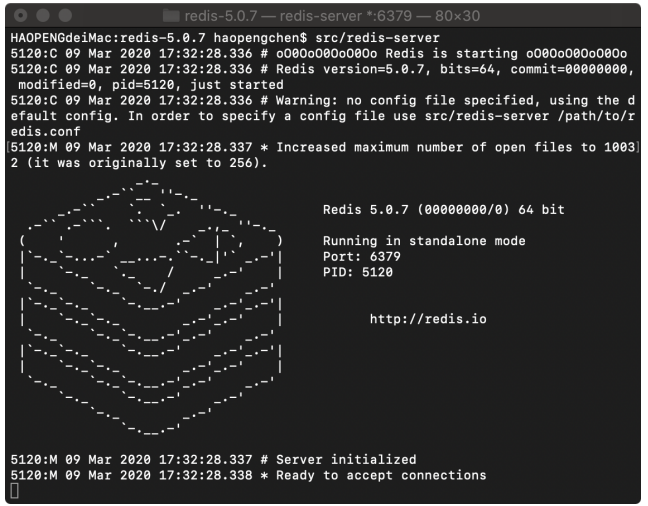
- In order to achieve its outstanding performance, Redis works with an in-memory dataset.
  - Depending on your use case, you can persist it either by dumping the dataset to disk every once in a while, or by appending each command to a log.

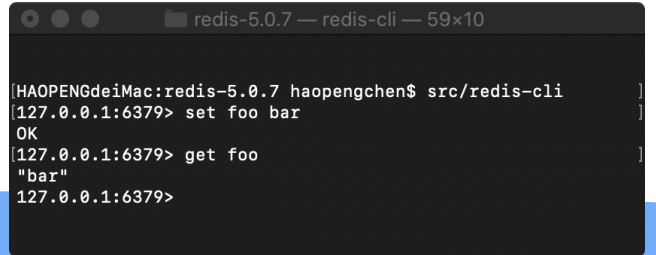
- Redis also supports trivial-to-setup master-slave replication,
  - with very fast non-blocking first synchronization, auto-reconnection on net split and so forth.

Redis

- Download, extract and compile Redis with:
  - `$ wget http://download.redis.io/releases/redis-5.0.7.tar.gz`
  - `$ tar xzf redis-5.0.7.tar.gz`
  - `$ cd redis-5.0.7`
  - `$ make`

- Run `redis server`
  - `$ src/redis-server`
  - `redis-server.exe`
- Run `redis Client`
  - `$ src/redis-cli`
  - `redis> set foo bar`
  - `OK`
  - `redis> get foo`
  - `"bar"`



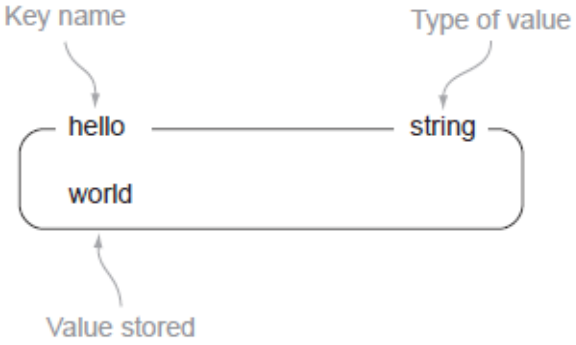


Redis - structure in Redis

Structure type	What it contains	Structure read/write ability
STRING	Strings, integers, or floating-point values	Operate on the whole string, parts, increment/decrement the integers and floats
LIST	Linked list of strings	Push or pop items from both ends, trim based on offsets, read individual or multiple items, find or remove items by value
SET	Unordered collection of unique strings	Add, fetch, or remove individual items, check membership, intersect, union, difference, fetch random items
HASH	Unordered hash table of keys to values	Add, fetch, or remove individual items, fetch the whole hash
ZSET (sorted set)	Ordered mapping of string members to floating-point scores, ordered by score	Add, fetch, or remove individual values, fetch items based on score ranges or member value

Redis - String

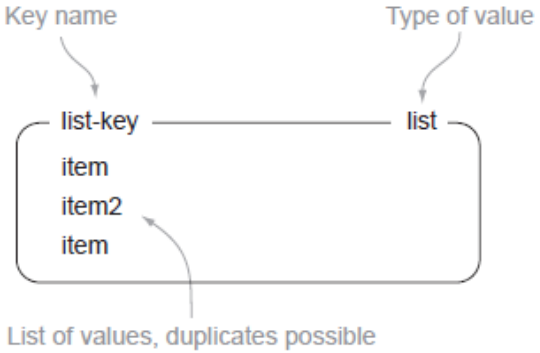
- String in Redis



Command	What it does
GET	Fetches the data stored at the given key
SET	Sets the value stored at the given key
DEL	Deletes the value stored at the given key (works for all types)

Redis - List

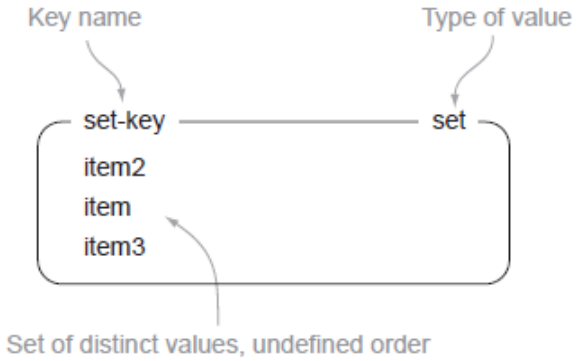
- List in Redis



Command	What it does
RPUSH	Pushes the value onto the right end of the list
LRANGE	Fetches a range of values from the list
LINDEX	Fetches an item at a given position in the list
LPOP	Pops the value from the left end of the list and returns it

Redis - Set

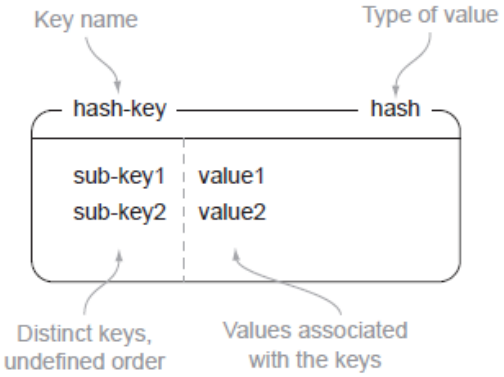
- Set in Redis



Command	What it does
SADD	Adds the item to the set
SMEMBERS	Returns the entire set of items
SISMEMBER	Checks if an item is in the set
SREM	Removes the item from the set, if it exists

Redis - Hash

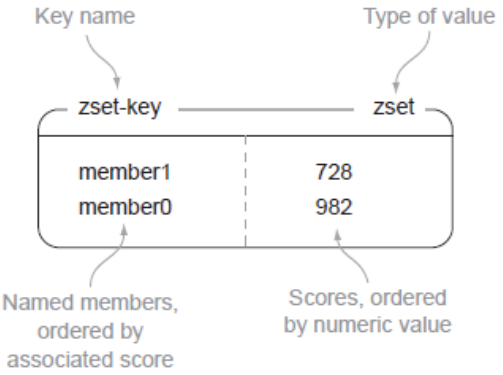
- Hash in Redis



Command	What it does
HSET	Stores the value at the key in the hash
HGET	Fetches the value at the given hash key
HGETALL	Fetches the entire hash
HDEL	Removes a key from the hash, if it exists

Redis - Sorted Set

- Sorted set in Redis



Command	What it does
ZADD	Adds member with the given score to the ZSET
ZRANGE	Fetches the items in the ZSET from their positions in sorted order
ZRANGEBYSCORE	Fetches items in the ZSET based on a range of scores
ZREM	Removes the item from the ZSET, if it exists

Redis

Name	Type	Data storage options	Query types	Additional features
Redis	In-memory non-relational database	Strings, lists, sets, hashes, sorted sets	Commands for each data type for common access patterns, with bulk operations, and partial transaction support	Publish/Subscribe, master/slave replication, disk persistence, scripting (stored procedures)
memcached	In-memory key-value cache	Mapping of keys to values	Commands for create, read, update, delete, and a few others	Multithreaded server for additional performance
MySQL	Relational database	Databases of tables of rows, views over tables, spatial and third-party extensions	`SELECT`, `INSERT`, `UPDATE`, `DELETE`, functions, stored procedures	ACID compliant (with InnoDB), master/slave and master/master replication
PostgreSQL	Relational database	Databases of tables of rows, views over tables, spatial and third-party extensions, customizable types	`SELECT`, `INSERT`, `UPDATE`, `DELETE`, built-in functions, custom stored procedures	ACID compliant, master/slave replication, multi-master replication (third party)
MongoDB	On-disk non-relational document store	Databases of tables of schema-less BSON documents	Commands for create, read, update, delete, conditional queries, and more	Supports map-reduce operations, master/slave replication, sharding, spatial indexes

Redis - Example

- RedisConfig.java

```
@Configuration
@EnableCaching
public class RedisConfig extends CachingConfigurerSupport {

   @Bean
   public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {
      RedisTemplate<String, Object> template = new RedisTemplate<String, Object>();
      template.setConnectionFactory(factory);
      Jackson2JsonRedisSerializer jacksonSeial = new Jackson2JsonRedisSerializer(Object.class);
      ObjectMapper om = new ObjectMapper();
      om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
      om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
      jacksonSeial.setObjectMapper(om);
      template.setValueSerializer(jacksonSeial);
      template.setKeySerializer(new StringRedisSerializer());
      template.setHashKeySerializer(new StringRedisSerializer());
      template.setHashValueSerializer(jacksonSeial);
      template.afterPropertiesSet();
      return template;
   }
```

Redis - Example

- application.properties

```
# Redis数据库索引（默认为0）
spring.redis.database=0
# Redis服务器地址
spring.redis.host=localhost
# Redis服务器连接端口
spring.redis.port=6379
# Redis服务器连接密码（默认为空）
spring.redis.password=
#连接池最大连接数（使用负值表示没有限制）
spring.redis.jedis.pool.max-active=8
# 连接池最大阻塞等待时间（使用负值表示没有限制）
spring.redis.jedis.pool.max-wait=-1
# 连接池中的最大空闲连接
spring.redis.jedis.pool.max-idle=8
# 连接池中的最小空闲连接
spring.redis.jedis.pool.min-idle=0
# 连接超时时间（毫秒）
spring.redis.timeout=300
```

Redis - Example

- RedisConfig.java

```
@Configuration
@EnableCaching
public class RedisConfig extends CachingConfigurerSupport {
   @Bean
   public HashOperations<String, String, Object> hashOperations(RedisTemplate<String, Object> redisTemplate) {
      return redisTemplate.opsForHash();
   }
   @Bean
   public ValueOperations<String, Object> valueOperations(RedisTemplate<String, Object> redisTemplate) {
      return redisTemplate.opsForValue();
   }

   @Bean
   public ListOperations<String, Object> listOperations(RedisTemplate<String, Object> redisTemplate) {
      return redisTemplate.opsForList();
   }
```

Redis - Example

- RedisConfig.java

```
@Configuration
@EnableCaching
public class RedisConfig extends CachingConfigurerSupport {
   @Bean
   public SetOperations<String, Object> setOperations(RedisTemplate<String, Object> redisTemplate) {
      return redisTemplate.opsForSet();
   }

   @Bean
   public ZSetOperations<String, Object> zSetOperations(RedisTemplate<String, Object> redisTemplate) {
      return redisTemplate.opsForZSet();
   }
}
```

Redis - Example

- PersonDaoImpl.java

```
@Repository
public class PersonDaoImpl implements PersonDao {
    @Autowired
    private PersonRepository personRepository;
    @Autowired
    RedisUtil redisUtil;

    @Override
    public Person findOne(Integer id) {
        Person person = null;
        Object p = redisUtil.get("user" + id);
        if (p == null) {
            person = personRepository.getOne(id);
            redisUtil.set("user" + id, JSONArray.toJSON(person));
        } else {
            person = JSONArray.parseObject(p.toString(), Person.class);
        }
        return person;
    }
}
```

Redis - Example

{"personId":1,"age":54,"firstname":"Cao","lastname":"Cao"}

```
2020-03-10 09:00:54.236  INFO 921 --- [nio-8080-exec-3] io.lettuce.core.EpollProvider           : Starting without optional epoll library
2020-03-10 09:00:54.237  INFO 921 --- [nio-8080-exec-3] io.lettuce.core.KqueueProvider          : Starting without optional kqueue library
Person: 1 is not in Redis
Searching Person: 1 in DB
Searching Person: 1
Searching Person: 1 in Redis
Person: 1 is in Redis
```

Messaging with Redis

- MessagingRedisApplication.java

```
@SpringBootApplication
public class MessagingRedisApplication {

  private static final Logger LOGGER = LoggerFactory.getLogger(MessagingRedisApplication.class);

  @Bean
  RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory,
      MessageListenerAdapter listenerAdapter) {

    RedisMessageListenerContainer container = new RedisMessageListenerContainer();
    container.setConnectionFactory(connectionFactory);
    container.addMessageListener(listenerAdapter, new PatternTopic("chat"));

    return container;
  }
```

Messaging with Redis

- MessagingRedisApplication.java

```
@Bean
  MessageListenerAdapter listenerAdapter(Receiver receiver) {
    return new MessageListenerAdapter(receiver, "receiveMessage");
  }

  @Bean
  Receiver receiver() {
    return new Receiver();
  }

  @Bean
  StringRedisTemplate template(RedisConnectionFactory connectionFactory) {
    return new StringRedisTemplate(connectionFactory);
  }

  public static void main(String[] args) throws InterruptedException {
    ApplicationContext ctx = SpringApplication.run(MessagingRedisApplication.class, args);
  }
}
```

Messaging with Redis

- Receiver.java

```
package com.example.messagingredis;

import java.util.concurrent.atomic.AtomicInteger;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Receiver {
  private static final Logger LOGGER = LoggerFactory.getLogger(Receiver.class);

  private AtomicInteger counter = new AtomicInteger();

  public void receiveMessage(String message) {
    LOGGER.info("Received <" + message + ">");
    counter.incrementAndGet();
    LOGGER.info("counter <" + counter.get() + ">");
  }

  public int getCount() {
    return counter.get();
  }
}
```

Messaging with Redis

- MsgController.java

```
package com.example.messagingredis;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.context.WebApplicationContext;

@RestController
public class MsgController {
   @Autowired
   WebApplicationContext applicationContext;

   @GetMapping(value = "/msg")
   public void findOne( ) {
      StringRedisTemplate template = applicationContext.getBean(StringRedisTemplate.class);

      // Send a message with a POJO - the template reuse the message converter
      System.out.println("Sending an email message.");
      template.convertAndSend("chat", "Hello from Redis in HTTP!");
   };
}
```

Messaging with Redis

REliable, INtelligent & Scalable Systems

- pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

```
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
c.e.m.MessagingRedisApplication          : Started MessagingRedisApplication in 1.904 seconds (JVM running for 2.296)
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms

com.example.messagingredis.Receiver      : Received <Hello from Redis in HTTP!>
com.example.messagingredis.Receiver      : counter <1>

com.example.messagingredis.Receiver      : Received <Hello from Redis in HTTP!>
com.example.messagingredis.Receiver      : counter <2>
```

作业二

- 请你在大二开发的E-Book系统的基础上，完成下列任务：
  1. 仿照课件中给出的例子，在你的E-Book的首页上增加一个访问量统计功能，通过多线程控制，确保计数值准确，不会出现因多用户同时访问而统计不准确的情况。
  2. 在你的项目中增加Redis或Memcached缓存，如果你无法将所有数据的持久化操作都通过缓存来实现，那么你至少应该将书籍的持久化操作改写为通过缓存来执行。

  – 请将你编写的相关代码整体压缩后上传，请勿压缩整个工程提交。

- 评分标准：
  1. 多线程控制下的计数值能够在多用户访问时正确计数 (2分)
  2. 缓存功能实现正确，读写操作合理(3分)

References

- MemCached
  - https://memcached.org/
- MemCached
  - https://github.com/memcached/memcached/wiki
- 分布式缓存-Memcached
  - http://blog.sina.com.cn/s/blog_493a845501013ei0.html
- HomeBrew
  - https://brew.sh
- Installing memcached on Mac with Homebrew and Lunchy
  - https://gist.github.com/tomysmile/ba6c0ba4488ea51e6423d492985a7953
- Memcached结合Spring
  - https://www.jianshu.com/p/56d9d79d75b3
- Accessing Memcached from the command line
  - http://www.alphadevx.com/a/90-Accessing-Memcached-from-the-command-line

References

- Redis
  - https://redis.io
- Spring Data Redis
  - https://docs.spring.io/spring-data/redis/docs/2.2.5.RELEASE/reference/html/#get-started
  - https://github.com/ZhangZiSheng001/spring-data-projects/tree/master/spring-data-redis-demo
- Retwis
  - http://retwis.redis.io
  - https://github.com/spring-projects/spring-data-keyvalue-examples
  - https://docs.spring.io/spring-data/data-keyvalue/examples/retwisj/current/
- Jedis
  - https://github.com/xetorthio/jedis
- SpringBoot整合Redis实战项目
  - https://www.cnblogs.com/david1216/p/11473764.html
- Redis in Action
  - JOSIAH L. CARLSON, Manning Publications Co, 2013
- Messaging with Redis
  - https://spring.io/guides/gs/messaging-redis/

Thank You!