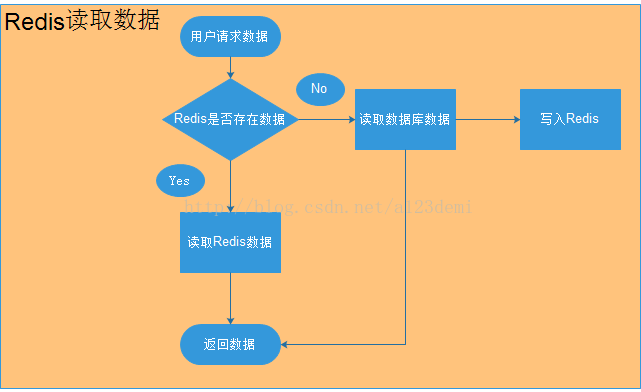
# 常用命令

|  |
| --- |
| 1）连接操作命令  quit：关闭连接（connection）  auth：简单密码认证  help cmd： 查看cmd帮助，例如：help quit  2）持久化  save：将数据同步保存到磁盘  bgsave：将数据异步保存到磁盘  lastsave：返回上次成功将数据保存到磁盘的Unix时戳  shundown：将数据同步保存到磁盘，然后关闭服务  3）远程服务控制  info：提供服务器的信息和统计  monitor：实时转储收到的请求  slaveof：改变复制策略设置  config：在运行时配置Redis服务器  4）对value操作的命令  exists(key)：确认一个key是否存在  del(key)：删除一个key  type(key)：返回值的类型  keys(pattern)：返回满足给定pattern的所有key  randomkey：随机返回key空间的一个  keyrename(oldname, newname)：重命名key  dbsize：返回当前数据库中key的数目  expire：设定一个key的活动时间（s）  ttl：获得一个key的活动时间  select(index)：按索引查询  move(key, dbindex)：移动当前数据库中的key到dbindex数据库  flushdb：删除当前选择数据库中的所有key  flushall：删除所有数据库中的所有key  5）String  set(key, value)：给数据库中名称为key的string赋予值value  get(key)：返回数据库中名称为key的string的value  getset(key, value)：给名称为key的string赋予上一次的value  mget(key1, key2,…, key N)：返回库中多个string的value  setnx(key, value)：添加string，名称为key，值为value  setex(key, time, value)：向库中添加string，设定过期时间time  mset(key N, value N)：批量设置多个string的值  msetnx(key N, value N)：如果所有名称为key i的string都不存在  incr(key)：名称为key的string增1操作  incrby(key, integer)：名称为key的string增加integer  decr(key)：名称为key的string减1操作  decrby(key, integer)：名称为key的string减少integer  append(key, value)：名称为key的string的值附加value  substr(key, start, end)：返回名称为key的string的value的子串  6）List  rpush(key, value)：在名称为key的list尾添加一个值为value的元素  lpush(key, value)：在名称为key的list头添加一个值为value的 元素  llen(key)：返回名称为key的list的长度  lrange(key, start, end)：返回名称为key的list中start至end之间的元素  ltrim(key, start, end)：截取名称为key的list  lindex(key, index)：返回名称为key的list中index位置的元素  lset(key, index, value)：给名称为key的list中index位置的元素赋值  lrem(key, count, value)：删除count个key的list中值为value的元素  lpop(key)：返回并删除名称为key的list中的首元素  rpop(key)：返回并删除名称为key的list中的尾元素  blpop(key1, key2,… key N, timeout)：lpop命令的block版本。  brpop(key1, key2,… key N, timeout)：rpop的block版本。  rpoplpush(srckey, dstkey)：返回并删除名称为srckey的list的尾元素，并将该元素添加到名称为dstkey的list的头部    7）Set  sadd(key, member)：向名称为key的set中添加元素member  srem(key, member) ：删除名称为key的set中的元素member  spop(key) ：随机返回并删除名称为key的set中一个元素  smove(srckey, dstkey, member) ：移到集合元素  scard(key) ：返回名称为key的set的基数  sismember(key, member) ：member是否是名称为key的set的元素  sinter(key1, key2,…key N) ：求交集  sinterstore(dstkey, (keys)) ：求交集并将交集保存到dstkey的集合  sunion(key1, (keys)) ：求并集  sunionstore(dstkey, (keys)) ：求并集并将并集保存到dstkey的集合  sdiff(key1, (keys)) ：求差集  sdiffstore(dstkey, (keys)) ：求差集并将差集保存到dstkey的集合  smembers(key) ：返回名称为key的set的所有元素  srandmember(key) ：随机返回名称为key的set的一个元素  8）Hash  hset(key, field, value)：向名称为key的hash中添加元素field  hget(key, field)：返回名称为key的hash中field对应的value  hmget(key, (fields))：返回名称为key的hash中field i对应的value  hmset(key, (fields))：向名称为key的hash中添加元素field  hincrby(key, field, integer)：将名称为key的hash中field的value增加integer  hexists(key, field)：名称为key的hash中是否存在键为field的域  hdel(key, field)：删除名称为key的hash中键为field的域  hlen(key)：返回名称为key的hash中元素个数  hkeys(key)：返回名称为key的hash中所有键  hvals(key)：返回名称为key的hash中所有键对应的value  hgetall(key)：返回名称为key的hash中所有的键（field）及其对应的value |

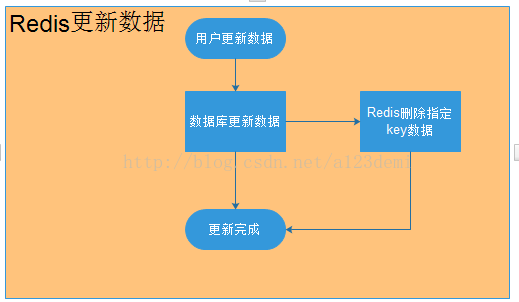
# Redis缓存策略

Redis和数据库结合使用，使用策略如下：

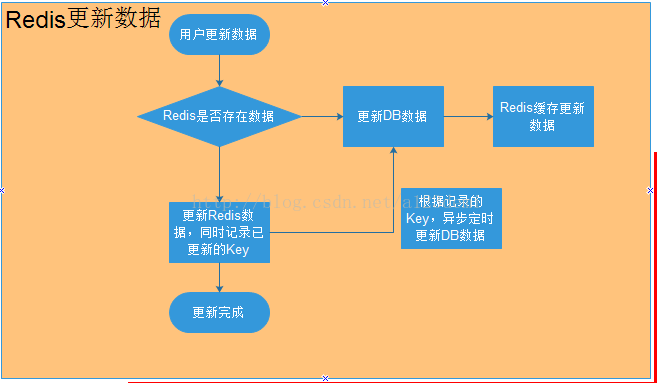
2.1  读取数据



2.2  更新数据—常用数据，变动性不强，并发不高



2.3  更新数据—并发较高



# Redis依赖

|  |
| --- |
| <!-- redis -->  <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-redis</artifactId>  <version>1.3.0.RELEASE</version>  </dependency>  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.6.2</version>  </dependency> |

# JedisUtil



# redis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">  <!-- 缓存的层级 -->  <context:component-scan base-package="com.lpf.springmvc" />  <!-- 引入redis配置 -->  <!-- <context:property-placeholder location="redis.properties" ignore-unresolvable="true" /> -->  <!-- Redis 配置 -->  <bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">  <property name="maxTotal" value="${redis.pool.maxTotal}" />  <property name="maxIdle" value="${redis.pool.maxIdle}" />  <property name="maxWaitMillis" value="${redis.pool.maxWaitMillis}" />  <property name="testOnBorrow" value="${redis.pool.testOnBorrow}" />  </bean>  <bean id="jedisPool" class="redis.clients.jedis.JedisPool">  <constructor-arg name="poolConfig" ref="jedisPoolConfig" />  <constructor-arg name="host" value="${redis.host}" />  <constructor-arg name="port" value="${redis.port}" type="int" />  <constructor-arg name="timeout" value="${redis.pool.maxWaitMillis}" type="int" />  <constructor-arg name="password" value="#{'${redis.password}'!=''?'${redis.password}':null}" />  <constructor-arg name="database" value="${redis.db.index}" type="int" />  </bean>  <!-- JedisCluster 集群高可用配置 -->  <!--  <bean id=*"jedisCluster"* class=*"redis.clients.jedis.JedisCluster"*>  <constructor-arg index=*"0"*>  <set>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg index=*"0"* value=*"${redis.ip1}"* />  <constructor-arg index=*"1"* value=*"${redis.port1}"* type=*"int"* />  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg index=*"0"* value=*"${redis.ip2}"* />  <constructor-arg index=*"1"* value=*"${redis.port2}"* type=*"int"* />  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg index=*"0"* value=*"${redis.ip3}"* />  <constructor-arg index=*"1"* value=*"${redis.port3}"* type=*"int"* />  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg index=*"0"* value=*"${redis.ip4}"* />  <constructor-arg index=*"1"* value=*"${redis.port4}"* type=*"int"* />  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg index=*"0"* value=*"${redis.ip5}"* />  <constructor-arg index=*"1"* value=*"${redis.port5}"* type=*"int"* />  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg index=*"0"* value=*"${redis.ip6}"* />  <constructor-arg index=*"1"* value=*"${redis.port6}"* type=*"int"* />  </bean>  </set>  </constructor-arg>  <constructor-arg index=*"1"* value=*"2000"* type=*"int"*></constructor-arg>  <constructor-arg index=*"2"* value=*"100"* type=*"int"*></constructor-arg>  <constructor-arg index=*"3"* ref=*"jedisPoolConfig"*></constructor-arg>  </bean>  -->  <!--redis Sentinel主从高可用方案配置 -->  <! --  <bean id=*"sentinelConfiguration"* class=*"org.springframework.data.redis.connection.RedisSentinelConfiguration"*>  <property name=*"master"*>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <property name=*"name"* value=*"master-1"*></property>  </bean>  </property>  <property name=*"sentinels"*>  <set>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <constructor-arg name=*"host"* value=*"${sentinel1.ip}"*></constructor-arg>  <constructor-arg name=*"port"* value=*"${sentinel1.port}"*></constructor-arg>  </bean>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <constructor-arg name=*"host"* value=*"${sentinel2.ip}"*></constructor-arg>  <constructor-arg name=*"port"* value=*"${sentinel2.port}"*></constructor-arg>  </bean>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <constructor-arg name=*"host"* value=*"${sentinel3.ip}"*></constructor-arg>  <constructor-arg name=*"port"* value=*"${sentinel3.port}"*></constructor-arg>  </bean>  </set>  </property>  </bean>  <bean id=*"jedisConnectionFactory"* class=*"org.springframework.data.redis.connection.jedis.JedisConnectionFactory"* p:use-pool=*"true"*>  <property name=*"password"* value=*"${redis.pass}"* />  <property name=*"poolConfig"*>  <ref bean=*"jedisPoolConfig"* />  </property>  <constructor-arg name=*"sentinelConfig"* ref=*"sentinelConfiguration"* />  </bean>  -->  <!-- redis单节点数据库连接配置 -->  <bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">  <property name="hostName" value="${redis.host}" />  <property name="port" value="${redis.port}" />  <property name="password" value="${redis.password}" />  <property name="poolConfig" ref="jedisPoolConfig" />  </bean>  <!-- redisTemplate配置，redisTemplate是对Jedis的对redis操作的扩展，有更多的操作，封装使操作更便捷 -->  <bean id="redisTemplate" class="org.springframework.data.redis.core.StringRedisTemplate">  <property name="connectionFactory" ref="jedisConnectionFactory" />  </bean>  </beans> |

# redis.properties

|  |
| --- |
| redis.pool.maxTotal=105  redis.pool.maxIdle=10  redis.pool.maxWaitMillis=5000  redis.pool.testOnBorrow=true  redis.host=127.0.0.1  redis.port=6379  redis.password=  redis.db.index=0 |

# ****application.xml配置****

|  |
| --- |
| <!-- redis相关的注入 -->  <bean id="redisDao" class="com.lpf.springmvc.dao.RedisDaoImpl"></bean>  <import resource="redis/redis-config.xml" /> |

# RedisPool的使用

#### dao

redis dao

RedisDao.java

|  |
| --- |
| @Repository  public interface RedisDao  {  String get(String key);  } |

RedisDaoImpl.java

|  |
| --- |
| public class RedisDaoImpl implements RedisDao  {  @Resource  JedisPool jedisPool;  public String get(String key)  {  // TODO Auto-generated method stub  JedisUtil jedisUtil = new JedisUtil(jedisPool);  JedisUtil.Strings strings = jedisUtil.new Strings();  System.out.println("cache: " + strings.get(key));  return strings.get(key);  }  } |

#### service

UserInfoService.java

|  |
| --- |
| public interface UserInfoService {  /\*\*  \* 根据redis的key获取redis的一个文本型value  \* @param key  \* @return  \*/  public String getValueByRedisKey(String key);  } |

UserInfoServiceImpl.java

|  |
| --- |
| @Service  public class UserInfoServiceImpl implements UserInfoService {  @Resource  RedisDao redisDao;  @Override  public String getValueByRedisKey(String key) {  return redisDao.get(key);  }  } |

#### 测试

SpringTestCase.java

|  |
| --- |
| //指定bean注入的配置文件  @ContextConfiguration(locations = { "classpath:application.xml" })  // 使用标准的JUnit @RunWith注释来告诉JUnit使用Spring TestRunner  @RunWith(SpringJUnit4ClassRunner.class)  public abstract class SpringTestCase extends AbstractJUnit4SpringContextTests {  protected Logger logger = LoggerFactory.getLogger(getClass());  } |

UserInfoTest.java

|  |
| --- |
| public class UserInfoTest extends SpringTestCase {  @Resource  UserInfoService usersInfoService;  @Test  public void getValueByRedisKey(){  String va =usersInfoService.getValueByRedisKey("name");  logger.debug("key为name的redis数据:"+va);  }  } |

#### 测试结果

