

# 【Redis相關】 RedisSDK

```
import org.apache.commons.beanutils.BeanUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.dao.DataAccessException;
import org.springframework.data.redis.connection.RedisConnection;
import org.springframework.data.redis.core.RedisCallback;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.util.CollectionUtils;

import java.io.Serializable;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.TimeUnit;

/**
 * @date 2018/9/25 15:07
 * @description:
 */
public class RedisSDK {

    /**
     * 日誌
     */
    public static final Logger LOG = LoggerFactory.getLogger(RedisSDK.class);

    private RedisTemplate<String, Object> redisTemplate;

    private static final String TEMP_STR1 = ",value:";

    public void setRedisTemplate(RedisTemplate<String, Object> redisTemplate) {
        this.redisTemplate = redisTemplate;
    }

    //=====common=====

    /**
     * 指定緩存失效時間
     *
     * @param key 鍵
     * @param time 時間(秒)
     * @return
     */
    public boolean expire(String key, long time) {
        try {
            if (time > 0) {
                redisTemplate.expire(key, time, TimeUnit.SECONDS);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 根據key 獲取過期時間
     *
     * @param key 鍵 不能為null
     * @return 時間(秒) 返回0代表為永久有效
     */
    public long getExpire(String key) {
        return redisTemplate.getExpire(key, TimeUnit.SECONDS);
    }
}
```

```

/**
 * 判斷key是否存在
 *
 * @param key 鍵
 * @return true 存在 false不存在
 */
public boolean hasKey(String key) {
    try {
        return redisTemplate.hasKey(key);
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * 刪除緩存
 *
 * @param key 可以傳一個值 或多個
 */
@SuppressWarnings("unchecked")
public void del(String... key) {
    if (key != null && key.length > 0) {
        if (key.length == 1) {
            redisTemplate.delete(key[0]);
        } else {
            redisTemplate.delete(CollectionUtils.arrayToList(key));
        }
    }
}

//=====操作String=====

/**
 * 普通緩存獲取
 *
 * @param key 鍵
 * @return 值
 */
public Object get(String key) {
    return key == null ? null : redisTemplate.opsForValue().get(key);
}

public String getString(String key) {
    Object result = get(key);
    return result == null ? null : result.toString();
}

/**
 * 普通緩存放入
 *
 * @param key 鍵
 * @param value 值
 * @return true成功 false失敗
 */
public boolean set(String key, Object value) {
    try {
        redisTemplate.opsForValue().set(key, value);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**

```

```

* 判斷是否存在
* @param key
* @return
*/
public boolean exists(String key) {
    try {
        return redisTemplate.execute(new RedisCallback<Boolean>() {
            @Override
            public Boolean doInRedis(RedisConnection redisConnection) throws DataAccessException {
                return redisConnection.exists(key.getBytes());
            }
        });
    } catch (Exception e) {
        LOG.error("", e);
    }
    return false;
}

/**
 * 普通緩存放入並設置時間
 *
 * @param key 鍵
 * @param value 值
 * @param time 時間(秒) time要大於0 如果time小於等於0 將設置無限期
 * @return true成功 false 失敗
 */
public boolean set(String key, Object value, long time) {
    try {
        if (time > 0) {
            redisTemplate.opsForValue().set(key, value, time, TimeUnit.SECONDS);
        } else {
            set(key, value);
        }
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * 遞增
 *
 * @param key 鍵
 * @param by 要增加幾(大於0)
 * @return
 */
public long incr(String key, long by) {
    if (by < 0) {
        throw new RuntimeException("遞增因子必須大於0");
    }
    return redisTemplate.opsForValue().increment(key, by);
}

/**
 * 遞減
 *
 * @param key 鍵
 * @param by 要減少幾(小於0)
 * @return
 */
public long decr(String key, long by) {
    if (by < 0) {
        throw new RuntimeException("遞減因子必須大於0");
    }
    return redisTemplate.opsForValue().increment(key, -by);
}

```

```
//=====操作Map=====
```

```
/**
 * HashGet
 *
 * @param key 鍵 不能為null
 * @param item 項 不能為null
 * @return 值
 */
public Object hget(String key, String item) {
    return redisTemplate.opsForHash().get(key, item);
}
```

```
/**
 * 獲取hashKey對應的所有鍵值
 *
 * @param key 鍵
 * @return 對應的多個鍵值
 */
public Map<Object, Object> hmget(String key) {
    return redisTemplate.opsForHash().entries(key);
}
```

```
/**
 * HashSet
 *
 * @param key 鍵
 * @param map 對應多個鍵值
 * @return true 成功 false 失敗
 */
public boolean hmset(String key, Map<String, Object> map) {
    try {
        redisTemplate.opsForHash().putAll(key, map);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

```
/**
 * HashSet 並設置時間
 *
 * @param key 鍵
 * @param map 對應多個鍵值
 * @param time 時間(秒)
 * @return true成功 false失敗
 */
public boolean hmset(String key, Map<String, Object> map, long time) {
    try {
        redisTemplate.opsForHash().putAll(key, map);
        if (time > 0) {
            expire(key, time);
        }
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

```
/**
 * 向一張hash表中放入數據,如果不存在將創建
 *
 * @param key 鍵
 * @param item 項
 * @param value 值
 * @return true 成功 false失敗
 */
```

```

public boolean hset(String key, String item, Object value) {
    try {
        redisTemplate.opsForHash().put(key, item, value);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * 向一張hash表中放入數據,如果不存在將創建
 *
 * @param key 鍵
 * @param item 項
 * @param value 值
 * @param time 時間(秒) 注意:如果已存在的hash表有時間,這裡將會替換原有的時間
 * @return true 成功 false失敗
 */
public boolean hset(String key, String item, Object value, long time) {
    try {
        redisTemplate.opsForHash().put(key, item, value);
        if (time > 0) {
            expire(key, time);
        }
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * 刪除hash表中的值
 *
 * @param key 鍵 不能為null
 * @param item 項 可以使多個 不能為null
 */
public void hdel(String key, Object... item) {
    redisTemplate.opsForHash().delete(key, item);
}

/**
 * 判斷hash表中是否有該項的值
 *
 * @param key 鍵 不能為null
 * @param item 項 不能為null
 * @return true 存在 false不存在
 */
public boolean hHasKey(String key, String item) {
    return redisTemplate.opsForHash().hasKey(key, item);
}

/**
 * hash遞增 如果不存在,就會創建一個 並把新增後的值返回
 *
 * @param key 鍵
 * @param item 項
 * @param by 要增加幾(大於0)
 * @return
 */
public double hincr(String key, String item, double by) {
    return redisTemplate.opsForHash().increment(key, item, by);
}

/**
 * hash遞減
 *

```

```

* @param key 鍵
* @param item 項
* @param by 要減少記(小於0)
* @return
*/
public double hincr(String key, String item, double by) {
    return redisTemplate.opsForHash().increment(key, item, -by);
}

/**
* @param key
* @return
*/
public <T> T hgetAll(String key, Class<T> clazz) {
    try {
        Map<Object, Object> values = hmget(key);
        LOG.info("redis獲取值,key:" + key + TEMP_STR1 + JSONObject.toJSONString(values));
        String json = JSONObject.toJSONString(values);
        Map<String, String> all = (Map) JSONObject.parse(json);
        T t = clazz.newInstance();
        BeanUtils.populate(t, all);
        return t;
    } catch (Exception e) {
        LOG.info("redis獲取值,key:" + key + "失敗");
    }
    return null;
}

//=====操作set=====

/**
* 根據key獲取Set中的所有值
*
* @param key 鍵
* @return
*/
public Set<Object> sGet(String key) {
    try {
        return redisTemplate.opsForSet().members(key);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

/**
* 根據value從一個set中查詢,是否存在
*
* @param key 鍵
* @param value 值
* @return true 存在 false不存在
*/
public boolean sHasKey(String key, Object value) {
    try {
        return redisTemplate.opsForSet().isMember(key, value);
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
* 將數據放入set緩存
*
* @param key 鍵
* @param values 值 可以是多個
* @return 成功個數
*/

```

```

public long sSet(String key, Object... values) {
    try {
        return redisTemplate.opsForSet().add(key, values);
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

/**
 * 將set數據放入緩存
 *
 * @param key 鍵
 * @param time 時間(秒)
 * @param values 值 可以是多個
 * @return 成功個數
 */
public long sSetAndTime(String key, long time, Object... values) {
    try {
        Long count = redisTemplate.opsForSet().add(key, values);
        if (time > 0) {
            expire(key, time);
        }
        return count == null ? 0L : count;
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

/**
 * 獲取set緩存的長度
 *
 * @param key 鍵
 * @return
 */
public long sGetSetSize(String key) {
    try {
        Long size = redisTemplate.opsForSet().size(key);
        return size == null ? 0L : size;
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

/**
 * 移除值為value的
 *
 * @param key 鍵
 * @param values 值 可以是多個
 * @return 移除的個數
 */
public long setRemove(String key, Object... values) {
    try {
        Long count = redisTemplate.opsForSet().remove(key, values);
        return count == null ? 0L : count;
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

//=====操作list=====

/**
 * 獲取list緩存的內容

```

```

*
* @param key 鍵
* @param start 開始
* @param end 結束 0 到 -1代表所有值
* @return
*/
public List<Object> IGet(String key, long start, long end) {
    try {
        return redisTemplate.opsForList().range(key, start, end);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * 獲取list緩存的長度
 *
 * @param key 鍵
 * @return
 */
public long IGetListSize(String key) {
    try {
        return redisTemplate.opsForList().size(key);
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

/**
 * 通過索引 獲取list中的值
 *
 * @param key 鍵
 * @param index 索引 index>=0時, 0 表頭,1 第二個元素,依次類推;index<0時,-1,表尾,-2倒數第二個元素,依次類推
 * @return
 */
public Object IGetIndex(String key, long index) {
    try {
        return redisTemplate.opsForList().index(key, index);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * 將list放入緩存
 *
 * @param key 鍵
 * @param value 值
 * @return
 */
public boolean ISet(String key, Object value) {
    try {
        redisTemplate.opsForList().rightPush(key, value);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * 將list放入緩存
 *
 * @param key 鍵
 * @param value 值
 * @param time 時間(秒)

```

```

    * @return
    */
    public boolean lSet(String key, Object value, long time) {
        try {
            redisTemplate.opsForList().rightPush(key, value);
            if (time > 0) {
                expire(key, time);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 將list放入緩存
     *
     * @param key 鍵
     * @param value 值
     * @return
     */
    public boolean lSet(String key, List<Object> value) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 將list放入緩存
     *
     * @param key 鍵
     * @param value 值
     * @param time 時間(秒)
     * @return
     */
    public boolean lSet(String key, List<Object> value, long time) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            if (time > 0) {
                expire(key, time);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 根據索引修改list中的某條數據
     *
     * @param key 鍵
     * @param index 索引
     * @param value 值
     * @return
     */
    public boolean lUpdateIndex(String key, long index, Object value) {
        try {
            redisTemplate.opsForList().set(key, index, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

```

```

    }
}

/**
 * 移除N個值為value
 *
 * @param key 鍵
 * @param count 移除多少個
 * @param value 值
 * @return 移除的個數
 */
public long lRemove(String key, long count, Object value) {
    try {
        Long remove = redisTemplate.opsForList().remove(key, count, value);
        return remove;
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

/**
 * 發佈消息
 *
 * @param channel
 * @param message
 */
public void sendMessage(String channel, Serializable message) {
    redisTemplate.convertAndSend(channel, message);
}

}

```

🕒修訂版本 #1

★由 treeman 建立於 27 🕒🕒🕒🕒 2023 05:13:30

🔧由 treeman 更新於 26 🕒@🕒🕒 2024 18:15:31