一、功能说明

该工具包包含spring-boot-starter-data-Redis的集成, 并在该基础之上封装了分布式ID生成器、分布式锁等分布式场景下常见的解决方案。

背景

- 在微服务横行的当下, 分布式业务并发风险剧增;
- 急需一套相应的分布式场景解决方案保证高并发下的业务安全。

二、工具使用

- 添加依赖: pom.xml

```
<dependency>
    <groupId>cn.gmlee</groupId>
    <artifactId>tools-redis</artifactId>
    <version>3.0.0</version>
</dependency>
```

- 启用工具

> 在启动类上或其他配置类上扫描包: cn.gmlee.tools

```
@SpringBootApplication
@ComponentScan("cn.gmlee.tools")
public class XxxApp {
    public static void main(String[] args) {
        SpringApplication.run(XxxApp.class, args);
    }
}
```

- 依赖注入

```
// @Autowired
// RedisUtil redisUtil;
@Autowired
RedisClient redisClient;
@Autowired
RedisId redisId;
@Autowired
RedisLock redisLock;
```

- 工具说明
- RedisUtil: 用于同时获得: RedisClient、RedisId、RedisLock

> 如只需要以下某 1 个, 建议直接注入其对象即可

- RedisClient: 用于对Redis进行基本的KV操作
- RedisId: 用于生成分布式场景下全局唯一的标识

- RedisLock: 用于分布式场景下全局加锁操作

三、工具原理

逻辑

- 通过利用Redis单线程执行的便利设计的RedisId和RedisLock
- RedisId在服务端存储了lastId, 当客户端获取Id时, Redis服务器将lastId自增指定步长(比如: +1)
- RedisLock在服务端存储一个指定过期时间的key, 其value也存过期时间

> 其他节点或线程获取锁时: 检测key是否存在, 存在继续监测value是否过期
>
> 未过期则睡眠与过期时间相同的时间, 继续获取锁
>
> 成功获取锁则执行加锁代码, 否则继续睡眠/取锁操作

代码

1. 调用RedisId.generate(String key) 可获得分布式ID
2. 调用RedisId.generateToday(String key, Integer length) 可获得当天指定长度的分布式ID
3. 调用RedisId.generate(String key, int increment) 可获得指定步长的分布式ID
4. 调用RedisLock.lock(String key, long expire, Runnable run) 可在分布式场景下安全执行一段代码
5. 调用RedisLock.lock(String key, long expire, Callable call) 同上,并且获得代码执行的返回值

数据

- 工具产生的数据, 均存在于Redis中

四、使用示例

- 客户端

```
// 获取登陆用户信息
Login<U, S, D, C> login = redisClient.get(TOKEN_PREFIX.concat(token));
// 设置登陆用户信息
redisClient.set(AuthController.TOKEN_PREFIX.concat(token), login,
expireSecond);
// 设置新值获取旧值
String old = redisClient.getAndSet(key, value);
// 追加过期时间
redisClient.addExpire(key, 1000L);
// 获取过期时间
Long expire = redisClient.getExpire(key);
```

- 分布式锁

```
Object ret = redisLock.lock(appId, 5000L, () -> {
    // 省略业务代码: 括号内部所有代码将加全局锁执行
    // 返回值可省略: 不需要返回值可去除返回和接收`ret`;
    return ret;
});
```

- 分布式ID

```
// 获取订单编号
long sn = redisId.generate(REDISID_PRIFEX);
```

- 工具包

```
// 获取Redis客户端: 可以借助泛型之力去除强转风险, 正所谓强扭的瓜不甜...
RedisClient<String, Login> redisClient = redisUtil.rc(String.class,
Login.class);
// 获取RedisId工具
RedisId redisId = redisUtil.ri();
// 获取RedisLock工具
RedisLock redisLock = redisUtil.rl();
```