Redis 安装和使用

redis 是一个 key value 式的缓存数据库， 他的数据是存储在 内存里的，
所以速度特别快

1. 有的数据库支持 SQL查询
2. 有的数据库 支持 JSON 特定类型查询
3. 有的数据库 支持 Key value Redis 就是 k v 类型的数据库
    a. 缓存 前端查询
    b. 热点数据缓存
    c. 门户网站
4. 很多开发工具都是基于 Linux 开发的， window下面一般用于开发， 所以这些工具天生对 Linux 环境十分友好， 所以我们使用 Docker 安装， 使用会比较顺畅
5. Redis 是一个服务
    a. 默认端口号 6379
    b. 默认不允许外网连接 127.0.0.1 访问
    c. 默认没有密码
6. 能用 docker 安装的使用 docker 安装
7. 当然也可以使用安装包安装Redis
8. Redis是一个单线程运行的缓存数据库
    a. redis 系统里面所有的KEY 都是字符串类型
    b. Redis的数据类型
        i. String字符串
            1. 可以是字符串、整数或浮点数 对整个字符串或字符串的一部分进行操作；对整数或浮点数进行自增或自减操作；
        ii. List列表
            1. 一个链表，链表上的每个节点都包含一个字符串 对链表的两端进行push和pop操作，读取单个或多个元素；根据值查找或删除元素；
        iii. Set集合
            1. 包含字符串的无序集合 字符串的集合，包含基础的方法有看是否存在添加、获取、删除；还包含计算交集、并集、差集等
        iv. Hash散列
            1. 包含键值对的无序散列表 包含方法有添加、获取、删除单个元素
        v. Zset有序集合
            1. 和散列一样，用于存储键值对 字符串成员与浮点数分数之间的有序映射；元素的排列顺序由分数的大小决定；包含方法有添加、获取、删除单个元素以及根据分值范围或成员来获取元素
9. **String字符串**

a. set (key) (value)：设置键值对
b. setnx (key) (value)：防止覆盖，设置键值对，
    i. 如果key不存在就设置，返回1
    ii. ##### 如果key已经存在就不设置，返回0 #####
    iii. 一般用来做分布式锁
c. get(key)：获取key对应的value
d. getset (key) (value) ： 先get再set，返回旧值，如果没有旧值返回nil
e. append (key) (value)：向指定的key的value后追加字符串
f. del (key) ： 删除key
g. strlen (key)：获取key对应值的字符串长度

10. 基本数据类型数据加减
    a. incr (key) value + 1
    b. decr (key) ： value - 1
    c. incrby (key) (number)：value + number
    d. decrby (key) (number)：value - number

11. 设置键值过期时间
    a. setex (key) (seconds) expire：设置键过期时间
    b. ttl (key) 查看key剩余存活时间

12. 批量操作
    a. mset (key1) (value1) (key2) (value2)：用于同时设置一个或多个 key-value 对
    b. mget (key1) (key2) ： 返回所有(一个或多个)给定 key 的值（如果某个key不存在，不存在的key返回null)
    c. msetnx(key1) (value1) (key2) (value2)：当所有 key 都成功设置，返回 1 。 如果有一个key设置失败，所有的key设置都会失败，返回 0 。原子操作

13. **List 列表**
    a. RPUSH 将给定值推入到列表右端 RPUSH key value
    b. LPUSH 将给定值推入到列表左端 LPUSH key value
    c. RPOP 从列表的右端弹出一个值，并返回被弹出的值 RPOP key
    d. LPOP 从列表的左端弹出一个值，并返回被弹出的值 LPOP key
    e. LRANGE 获取列表在给定范围上的所有值 LRANGE key 0 -1
    f. LSET 对指定下标进行更新 lset key index value
    g. Ltrim key 对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区 间之内的元素都将被删除。
    h. LINDEX 通过索引获取列表中的元素。你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。 LINEX key index

14. **Set集合**
    a. SADD 向集合添加一个或多个成员 SADD key value

b. SCARD 获取集合的成员数 SCARD key

c. SMEMBERS 返回集合中的所有成员 SMEMBER key

d. SISMEMBER 判断 member 元素是否是集合 key 的成员 SISMEMBER key member

e. SRANDMEMBER 随机返回值

f. SPOP 用于移除集合中的指定 key 的一个或多个随机元素

g. SMOVE 从一个集合中移动到另一个集合中

i. - 差集： sdiff

ii. - 交集： sinter

iii. - 并集： sunion

15. **Hash散列**

a. HSET 添加键值对 HSET hash-key sub-key1 value1

b. HGET 获取指定散列键的值 HGET hash-key key1

c. HGETALL 获取散列中包含的所有键值对 HGETALL hash-key

d. HDEL 如果给定键存在于散列中，那么就移除这个键 HDEL hash-key sub-key1

e. HLEN 获取哈希表中字段的数量。

f. hexists 查看哈希表的指定字段是否存在。

g. hkeys 获取哈希表中的所有Key

h. hvals 返回哈希表所的值。

i. hsetnx 为哈希表中不存在的的字段赋值

16. **Zset有序集合 做数据排行会用到 冰城萌娃活动**

a. *zadd 将一个或多个成员元素及其分数值加入到有序集当中。*

b. *zrange 返回有序集中，指定区间内的成员*

c. zrangebyscore 递增排序

d. ZREVRANGE 递减排序

e. zrem 删除一个元素

f. zcard 命令用于计算集合中元素的数量。

g. zcount 计算有序集合中指定分数区间的成员数量。

h. zrank 返回有序集中指定成员的排名。其中有序集成员按分数值递增(从小到大)顺序排列。

Redis 实战应用

Spring-security 基于Spring的企业应用系统提供声明式的安全访问控制解决方案的安全框架

可以使用 spring.security.user.name=admin 和 spring.security.user.password=123456 指定密码

需求依赖

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

自定义redis序列化配置

```java
@Configuration
public class RedisConfig {

    @Bean
    @SuppressWarnings("all")
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory ){
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(redisConnectionFactory);

        GenericJackson2JsonRedisSerializer genericJackson2JsonRedisSerializer = new GenericJackson2JsonRedisSerializer();

        redisTemplate.setKeySerializer( RedisSerializer.string() );
        redisTemplate.setHashKeySerializer( RedisSerializer.string() );

        redisTemplate.setValueSerializer( genericJackson2JsonRedisSerializer );
        redisTemplate.setHashValueSerializer( genericJackson2JsonRedisSerializer );

        return redisTemplate;
    }

}
```

JWTUtils

```java
public class JWTUtils {
    public static String getToken(String secret, Long id){
        Algorithm algorithm = Algorithm.HMAC256(secret);
        Map<String, Object> header = new HashMap<>();
        header.put("typ", "JWT");
        header.put("alg", "HS256");
        return JWT.create()
                .withHeader(header)
                .withClaim("ID", id)
                .sign(algorithm);
    }

    public static Long getDesc(String token){
        return JWT.decode(token).getClaim("ID").asLong();
    }

    public static boolean verify(String secret, String token) throws Exception {
        Algorithm algorithm = Algorithm.HMAC256(secret);
        JWTVerifier build = JWT.require(algorithm).build();
        try {
            build.verify(token);
        }catch (SignatureVerificationException e){
            throw new Exception("无效签名! ");
        }catch (TokenExpiredException e){
            throw new Exception( "token过期! ");
        }catch (AlgorithmMismatchException e){
            throw new Exception( "算法不一致! ");
        }catch (Exception e){
            throw new Exception( "token无效");
        }
        return true;
    }
}
```

SecurityConfig

```java
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class SecurityConfig {
    @Autowired
    SpringSecurityFilter springSecurityFilter;
    @Bean
    public SecurityFilterChain getSecurityFilterChain(HttpSecurity httpSecurity) throws Exception {
        HttpSecurity http = httpSecurity
                .csrf()
                .disable()
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.NEVER)
                .and()
                .authorizeRequests()
                .antMatchers("/user/login").anonymous()
                .mvcMatchers("/user/logout").permitAll()
                .anyRequest().authenticated()
                .and();

        http.addFilterBefore( springSecurityFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
}
```

SpringSecurityFilter

```java
@Component
public class SpringSecurityFilter extends OncePerRequestFilter {

    @Value("${jwt.secret}")
    private String secret;

    @Resource
    RedisTemplate redisTemplate;
```

```
9
10      @Override
11
12      protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
    response, FilterChain filterChain) throws ServletException, IOException {
13
14          String token = request.getHeader("token");
15          if ( !StringUtils.hasText(token) ) {
16              token = request.getParameter("token");
17          }
18          if ( !StringUtils.hasText(token) ) {
19              filterChain.doFilter( request, response );
20              return;
21          }
22          Long uid = null;
23          try {
24              JWTUtils.verify( secret, token );
25              uid = JWTUtils.getDesc(token);
26          } catch (Exception e) {
27              System.out.println( e.getMessage() );
28          }
29
30          UserDao o = (UserDao) redisTemplate.opsForValue().get("token:" + uid);
31
32          if ( uid != null ) {
33              UsernamePasswordAuthenticationToken userToken = new
    UsernamePasswordAuthenticationToken(uid, null, null);
34              SecurityContextHolder.getContext().setAuthentication( userToken );
35          } else {
36              SecurityContextHolder.clearContext();
37          }
38          filterChain.doFilter( request, response );
39      }
40
41  }
```

UserDetails

```
1  @Data
```

```
public class UserDao implements UserDetails {

    private String password;

    private String username;

    @Override
    @JsonIgnore
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return null;
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return username;
    }

    @Override
    @JsonIgnore
    public boolean isAccountNonExpired() {
        return false;
    }

    @Override
    @JsonIgnore
    public boolean isAccountNonLocked() {
        return false;
    }

    @Override
    @JsonIgnore
    public boolean isCredentialsNonExpired() {
        return false;
    }

```

```
42      @Override
43      @JsonIgnore
44      public boolean isEnabled() {
45          return false;
46      }
47  }
48
```