

Redis简介和SpringBoot缓存

Redis简介

□REmote Dictionary Server(Redis) 是一个由Salvatore Sanfilippo写的key-value存储系统。

□Redis是一个开源的使用ANSI C语言编写、遵守BSD协议、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

□它通常被称为数据结构服务器，因为值（value）可以是字符串(String), 哈希(Hash), 列表(list), 集合(sets) 和 有序集合(sorted sets)等类型。

□ Redis 与其他 key - value 缓存产品有以下三个特点：

- Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
- Redis不仅仅支持简单的key-value类型的数据，同时还提供list, set, zset, hash等数据结构的存储。
- Redis支持数据的备份，即master-slave模式的数据备份。

□Redis 优势

- 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s。
- 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。
- 原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。
- 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

使用方法

□将下载文件 Redis-x64-3.2.100.zip 解压即可。

dump.rdb	2020/9/28 16:56	RDB 文件	1 KB
EventLog.dll	2016/7/1 16:27	应用程序扩展	1 KB
Redis on Windows Release Notes.do...	2016/7/1 16:07	Microsoft Word ...	13 KB
Redis on Windows.docx	2016/7/1 16:07	Microsoft Word ...	17 KB
redis.windows.conf	2016/7/1 16:07	CONF 文件	48 KB
redis.windows-service.conf	2016/7/1 16:07	CONF 文件	48 KB
redis-benchmark.exe	2016/7/1 16:28	应用程序	400 KB
redis-benchmark.pdb	2016/7/1 16:28	PDB 文件	4,268 KB
redis-check-aof.exe	2016/7/1 16:28	应用程序	488 KB
redis-check-aof.pdb	2016/7/1 16:28	PDB 文件	5,436 KB
redis-cli.exe	2016/7/1 16:28	应用程序	488 KB
redis-cli.pdb	2016/7/1 16:28	PDB 文件	4,420 KB
redis-server.exe	2016/7/1 16:28	应用程序	1,628 KB
redis-server.pdb	2016/7/1 16:28	PDB 文件	6,916 KB
Windows Service Documentation.docx	2016/7/1 9:17	Microsoft Word ...	14 KB

客户端,可查看redis中存放的数据

双击启动redis服务

常用命令

□keys * : 运行此命令,查看当前数据库中的所有的key

□get key: 根据key取出对应的value

SpringBoot缓存

- **缓存 (cache) , 原始意义是指访问速度比一般随机存取存储器 (RAM) 快的一种高速存储器, 通常它不像系统主存那样使用DRAM技术, 而使用昂贵但较快速的SRAM技术。**
- **缓存的设置是所有现代计算机系统发挥高性能的重要因素之一。**
- 简单的理解:当某一硬件要读取数据时,会首先从缓存汇总查询数据,有则直接执行,不存在时从内存中获取。

Springboot缓存注解

Cache	缓存接口，定义缓存操作。实现有：RedisCache、EhCacheCache、ConcurrentMapCache等
CacheManager	缓存管理器，管理各种缓存（Cache）组件
@Cacheable	主要针对方法配置，能够根据方法的请求参数对其结果进行缓存
@CacheEvict	清空缓存
@CachePut	保证方法被调用，又希望结果被缓存。
@EnableCaching	开启基于注解的缓存
keyGenerator	缓存数据时key生成策略
serialize	缓存数据时value序列化策略

https://blog.csdn.net/qq_38974634

@Cacheable/@CachePut/@CacheEvict 主要的参数		
value	缓存的名称，在 spring 配置文件中定义，必须指定至少一个	例如： @Cacheable(value="mycache") 或者 @Cacheable(value={"cache1","cache2"})
key	缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合	例如： @Cacheable(value="testcache",key="#userName")
condition	缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存/清除缓存，在调用方法之前之后都能判断	例如： @Cacheable(value="testcache",condition="#userName.length()>2")
allEntries (@CacheEvict)	是否清空所有缓存内容，缺省为 false，如果指定为 true，则方法调用后将立即清空所有缓存	例如： @CacheEvict(value="testcache",allEntries=true)
beforeInvocation (@CacheEvict)	是否在方法执行前就清空，缺省为 false，如果指定为 true，则在方法还没有执行的时候就清空缓存，缺省情况下，如果方法执行抛出异常，则不会清空缓存	例如： @CacheEvict(value="testcache", beforeInvocation=true)
unless (@CachePut) (@Cacheable)	用于否决缓存的，不像 condition，该表达式只在方法执行之后判断，此时可以拿到返回值 result 进行判断。条件为 true 不会缓存，false 才缓存	例如： @Cacheable(value="testcache",unless="#result == null") https://blog.csdn.net/qq_38974634

实例

□1.引入依赖

```
<!--redis, 排除默认的lettuce-->
```

```
<dependency>
```

```
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-data-redis</artifactId>
```

```
  <exclusions>
```

```
    <exclusion>
```

```
      <groupId>io.lettuce</groupId>
```

```
      <artifactId>lettuce-core</artifactId>
```

```
    </exclusion>
```

```
  </exclusions>
```

```
</dependency>
```

```
<!--使用jedis作为客户端-->
```

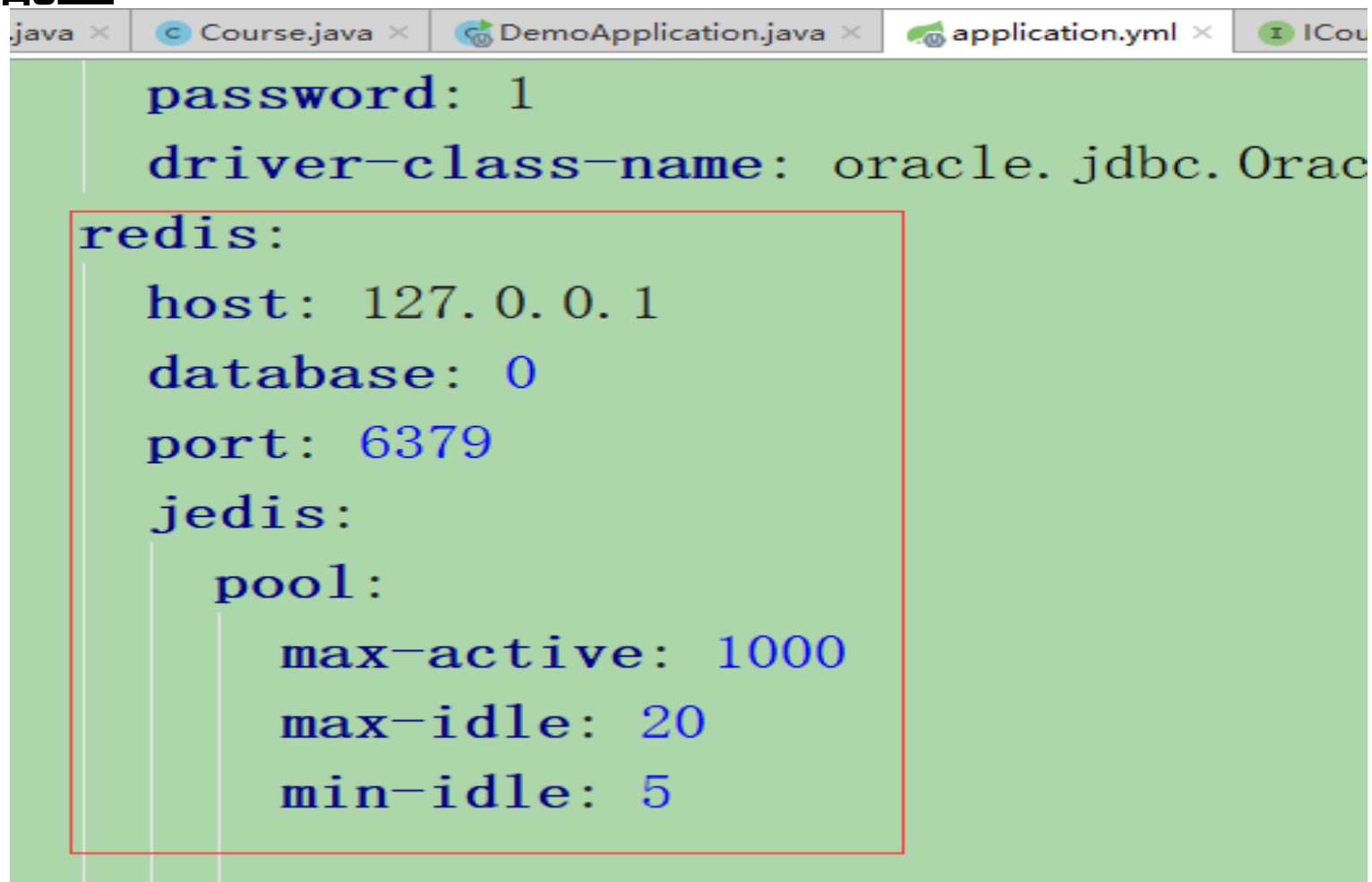
```
<dependency>
```

```
  <groupId>redis.clients</groupId>
```

```
  <artifactId>jedis</artifactId>
```

```
</dependency>
```

□Yml配置



```
password: 1
driver-class-name: oracle.jdbc.OracleDriver
redis:
  host: 127.0.0.1
  database: 0
  port: 6379
  jedis:
    pool:
      max-active: 1000
      max-idle: 20
      min-idle: 5
```

配置缓存管理CacheConfig.java

```
@Configuration
@EnableCaching
public class CacheConfig{

    @Bean
    public CacheManager cacheManager(RedisConnectionFactory redisConnectionFactory) {
        RedisSerializer<String> redisSerializer = new StringRedisSerializer();
        GenericJackson2JsonRedisSerializer genericJackson2JsonRedisSerializer = new
GenericJackson2JsonRedisSerializer();
        RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofMinutes(10))    //设置失效时间
            .serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer
(redisSerializer))
            .serializeValuesWith(RedisSerializationContext.SerializationPair
.fromSerializer(genericJackson2JsonRedisSerializer));
    }
}
```

```
RedisCacheManager redisCacheManager = RedisCacheManager.builder(redisConnectionFactory)
    .cacheDefaults(config)
    .build();

return redisCacheManager;
}
```

@Bean

```
public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
redisConnectionFactory) {
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(redisConnectionFactory);
    GenericJackson2JsonRedisSerializer genericJackson2JsonRedisSerializer = new
GenericJackson2JsonRedisSerializer();
    template.setKeySerializer(new StringRedisSerializer());
}
```

```
template.setValueSerializer(genericJackson2JsonRedisSerializer);  
template.setHashKeySerializer(new StringRedisSerializer());  
template.setHashValueSerializer(genericJackson2JsonRedisSerializer);  
template.afterPropertiesSet();  
return template;  
}  
  
}
```


CourseService

```
@Transactional
```

```
@CachePut(value="courseCache", key="#c.courseName")
```

```
public Course insertCourse(Course c) {  
    id.insertCourse(c);  
    return c;  
}
```

```
@Cacheable(value="courseCache", key="#courseId")
```

```
public Course findCourse(Integer courseId) {  
    // return id.getOne(courseId); //如开启缓存, 不要使用, 会导致缓存无法命中  
    return id.findById(courseId).get();  
}
```

实体类

```
@JsonIgnoreProperties(value = { "hibernateLazyInitializer"})
@Data
@Entity
@Table(name="COURSE") //设置数据库中表名字
public class Course implements Serializable {
    @Id
    @Column(name="COURSE_ID") //数据库实际列名为COURSE_ID
    private Integer courseId;

    @Column(name="COURSE_NAME")
    private String courseName;

    @Column(name="TEACHER_ID")
    private Integer teacherId;

    @Column(name="COURSE_TIME")
```