概念

数据类型

SpringBoot集成redis

Spring Cache集成redis

Redis的发布与订阅

创建一个Redis的消息接收器

```
public class Receiver {
    private static final Logger LOGGER =
LoggerFactory.getLogger(Receiver.class);

    private AtomicInteger counter = new AtomicInteger();

    public void receiveMessage(String message) {
        LOGGER.info("Received <" + message + ">");
        counter.incrementAndGet();
    }

    public int getCount() {
        return counter.get();
    }
}
```

注册监听器并发送消息

Spring Data Redis提供了使用Redis发送和接收消息所需的所有组件。具体来说，您需要配置：

- 连接工厂
- 消息侦听器容器
- Redis模板

可以使用Redis模板发送消息，并在`Receive`消息监听器中注册，以便接收消息。连接工厂将同时注册驱动模板和消息侦听器容器，从而使它们连接到Redis服务器

`RedisConnectionFactory`继承`JedisConnectionFactory`.连接工厂被注入到消息侦听器容器和Redis模板

```
 @Bean
    RedisMessageListenerContainer container(RedisConnectionFactory
connectionFactory,
                                            MessageListenerAdapter
listenerAdapter) {

        RedisMessageListenerContainer container = new
RedisMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.addMessageListener(listenerAdapter, new PatternTopic("chat"));
```

```
        return container;
    }

    @Bean
    MessageListenerAdapter listenerAdapter(Receiver receiver) {
        return new MessageListenerAdapter(receiver, "receiveMessage");
    }

    @Bean
    Receiver receiver() {
        return new Receiver();
    }

    @Bean
    StringRedisTemplate template(RedisConnectionFactory connectionFactory) {
        return new StringRedisTemplate(connectionFactory);
    }
```

`listenerAdapter`方法中定义的Bean在中定义的消息侦听器容器中注册为消息侦听器`container`，并将侦听有关该`chat`主题的消息。由于`Receiver`该类是POJO，因此需要将其包装在实现该`MessageListener`接口的消息侦听器适配器中（要求`addMessageListener()`）。消息侦听器适配器还配置为在消息到达时调用该`receiveMessage()`方法`Receiver`。

连接工厂和消息侦听器容器bean就是您用来侦听消息的全部。要发送消息，您还需要一个Redis模板。在这里，它是一个配置为的bean `StringRedisTemplate`，其实现`RedisTemplate`着重于Redis的常用用法，其中键和值都是`String`实例。

测试

```
@SpringBootTest
@RunWith(SpringJUnit4ClassRunner.class)
public class ReceiverMessageTest {

    private static final Logger LOGGER =
LoggerFactory.getLogger(ReceiverMessageTest.class);

    @Autowired
    Receiver receiver = new Receiver();

    @Autowired
    StringRedisTemplate stringRedisTemplate;

    @Test
    public void test() throws InterruptedException {
        while (receiver.getCount() == 0) {
            try {
                LOGGER.info("Sending message...");
                stringRedisTemplate.convertAndSend("chat", "Hello from Redis!");

                Thread.sleep(500L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

```
}
```

Redis 主从复制

核心：解决数据的冗余备份，不能解决故障的自动转移

从节点只能执行读操作，不能执行写操作

下载配置文件

```
wget http://download.redis.io/redis-stable/redis.conf
```

修改配置文件信息

```
#// 开启远程连接，已踩坑
#bind 127.0.0.1      需注释
protected-mode no
#// 持久化
appendonly yes
#// 设置密码
requirepass password
```

启动容器

```
docker run -d -p 6379:6379 --name master-redis
-v /root/redis/redis.conf:/etc/redis/redis.conf
-v /root/redis/data:/data
redis:latest redis-server
/etc/redis/redis.conf
```

命令解析：

- `-v /root/redis/redis.conf:/etc/redis/redis.conf`

  `-v /root/redis/data:/data`

  把本地的配置文件映射到镜像，修改本地文件，相当于直接修改镜像文件。更改后可以使用 `docker restart 容器id/容器名` 重启服务
- 配置文件中持久化和设置密码也可以在启动容器中设置

  ```
  --appendonly yes
  --requirepass password
  ```

  配置master-redis必须打开持久化，否则master服务挂了之后，从服务仍然可以获取到数据。但是master服务重启后，由于没有持久化，数据会全部清空。从服务会立即同步主服务的数据，从而造成数据丢失

配置从服务(slave-redis)

1. 复制一份redis.conf的文件

```
mv redis.conf redis-salve-1.config
```

2. 启动容器

```
docker run -d -p 6380:6379 --name salve-redis
-v /root/redis/redis.conf:/etc/redis/redis-salve-1.config
redis:latest redis-server
/etc/redis/redis.conf
--appendonly yes
```

3. 查看两个服务的IP

```
docker inspect 容器ID/容器名
```

4. 配置从服务

   进入docker容器内部

   ```
   docker exec -it 容器ID/容器名 redis-cli
   ```

   如果redis设置密码，需验证密码

   ```
   auth 密码
   ```

   查看当前redis角色

   ```
   info replication
   ```

配置

```
slaveof master-redis-IP master-redis-port
```

如果master-redis设置了密码

```
config set masterauth master-password
```

测试

master-redis设值，salve-master会同步更新

salve-redis只能执行读操作，不能写

哨兵机制

解决问题：

- **带有自动的处理故障转移功能的主从式架构**
- 自动完成数据冗余备份

缺点：

- 无法解决现有系统单节点并发压力和物理上限问题

配置

- 在主从复制的基础上搭建哨兵
- 启动哨兵服务，准备哨兵的配置文件

  通过命令获取sentinel匹配

  ```
  wget http://download.redis.io/redis-stable/sentinel.conf
  ```

- 修改配置文件

  ```
  # 让sentinel服务后台运行
  daemonize yes

  # 修改日志文件的路径
  logfile "/var/log/redis/sentinel.log"

  # 修改监控的主redis服务器(必须填写)
  # 最后一个2表示，两台机器判定主被动下线后，就进行failover(故障转移)
  sentinel monitor mymaster 35.236.172.131 6379 2
  ```

缓存穿透、缓存击穿、缓存雪崩

缓存穿透

- 概念

  缓存穿透是指查询一个不存在的缓存key，由于缓存是未命中的时候需要从数据库查询，正常的情况下查不到缓存数据则不写入缓存，就会导致这个不存在的数据每次请求都要去数据库查询，即缓存和数据库中都没有的数据，而用户不断发起请求。造成缓存穿透。

- 解决方案

  1、使用布隆过滤器。系统启动的时候将所有存在的数据哈希到一个足够大的bitmap中，当一个一定不存在的数据请求的时候，会被这个bitmap拦截掉，从而避免了对底层数据库的查询压力。

  2、返回空值。如果一个查询请求查询数据库后返回的数据为空，不管数据不存在还是系统故障，仍然将这个空结果进行缓存，但它的过期时间会很短，比如一分钟，但是此种方法解决不够彻底，这里缓存不存在key的时候一定要设置过期时间，不然当数据库已经新增了这一条记录的时候，这样会导致缓存和数据库不一致的情况

  3、接口层加用户鉴权，参数做数据校验，不合法直接return

缓存击穿

- 概念

  缓存key在某个时间点过期的时候，刚好在这个时间点对这个key有大量的并发请求过来，请求命中缓存失败后会通过db加载数据并写会到缓存，这个时候大并发的请求可能会瞬间把后端db压垮

- 解决方案

  1、加锁的方式读取数据，并写会缓存

  2、设置热点数据永不过期

  3、查询数据库加互斥锁

缓存雪崩

- 概念

  设置缓存时使用了相同的过期时间，导致缓存在某一时刻同时失效，所有的查询都请求到了数据库上，导致应用系统产生各种故障，称之为缓存雪崩。

- 解决方案

  1、限流

  2、设置过期时间为随机值

淘汰策略

持久化

SpringBoot+redis+注解+拦截器实现接口幂等性校验