Spring Boot 入门之缓存和 NoSQL 篇（四）

一、前言

当系统的访问量增大时，相应的数据库的性能就逐渐下降。但是，大多数请求都是在重复的获取相同的数据，如果使用缓存，将结果数据放入其中可以很大程度上减轻数据库的负担，提升系统的响应速度。

本篇将介绍 Spring Boot 中缓存和 NoSQL 的使用。上篇文章《Spring Boot 入门之持久层篇（三）》

二、整合缓存

Spring Boot 针对不同的缓存技术实现了不同的封装，本篇主要介绍 EhCache 和 Redis 缓存。

Spring Boot 提供了以下几个注解实现声明式缓存：

注解	说明
@EnableCaching	开启缓存功能，放在配置类或启动类上
@CacheConfig	缓存配置，设置缓存名称
@Cacheable	执行方法前先查询缓存是否有数据。有则直接返回缓存数据；否则查询数据再将数据放入缓存
@CachePut	执行新增或更新方法后，将数据放入缓存中
@CacheEvict	清除缓存
@Caching	将多个缓存操作重新组合到一个方法中

2.1 EhCache 缓存

2.1.1 添加依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-cache</artifactId>
</dependency>

<dependency>
  <groupId>net.sf.ehcache</groupId>
  <artifactId>ehcache</artifactId>
</dependency>
```

2.1.2 添加配置

1）在 src/main/resources 目录下创建 ehcache.xml 文件，内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd">

    <!-- 磁盘缓存位置 -->
    <diskStore path="java.io.tmpdir/ehcache"/>

    <!-- 默认缓存 -->
    <defaultCache
            maxEntriesLocalHeap="10000"
            eternal="false"
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            maxEntriesLocalDisk="10000000"
            diskExpiryThreadIntervalSeconds="120"
            memoryStoreEvictionPolicy="LRU">
        <persistence strategy="localTempSwap"/>
    </defaultCache>

    <!-- 自定义缓存 -->
    <cache name="department"
           maxElementsInMemory="1000"
           eternal="false"
           timeToIdleSeconds="50"
           timeToLiveSeconds="50"
           overflowToDisk="false"
           memoryStoreEvictionPolicy="LRU"/>
</ehcache>
```

说明：

```
name：Cache 的唯一标识
maxElementsInMemory：内存中允许存储的最大的元素个数
maxElementsOnDisk：硬盘最大缓存个数，0代表无限个
clearOnFlush：内存数量最大时是否清除
eternal：缓存对象是否永久有效，如果是，超时设置将被忽略
overflowToDisk：内存不足（超过 maxElementsInMemory）时，是否启用磁盘缓存
timeToIdleSeconds：设置对象在失效前的允许闲置时间（单位：秒）。仅当
eternal=false对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间无穷大
timeToLiveSeconds：缓存数据的生存时间（TTL），也就是一个元素从构建到消亡的最大时
间间隔值，这只能在元素不是永久驻留时有效，如果该值是0就意味着元素可以停顿无穷长的
时间
diskPersistent：是否将缓存数据持久化到磁盘上，如果为 true，JVM 重启数据依然存
在。默认值是false
diskSpoolBufferSizeMB：这个参数设置DiskStore（磁盘缓存）的缓存区大小。默认是
30MB。每个Cache都应该有自己的一个缓冲区
diskExpiryThreadIntervalSeconds：磁盘失效线程运行时间间隔，默认是120秒
memoryStoreEvictionPolicy：当达到 maxElementsInMemory 限制时，Ehcache 将根据
指定策略清除内存。默认为 LRU（最近最少使用），其他策略有 FIFO（先进先出），
LFU（较少使用）
```

2）application.properties ：

```
# 缓存类型（ehcache、redis）
spring.cache.type=ehcache

# ehcache 配置文件
spring.cache.ehcache.config=classpath:ehcache.xml

# 打印日志，查看 sql
logging.level.com.light.springboot=DEBUG
```

2.1.3 编码

在持久层篇的基础上，结合 Mybatis 测试：

Service 层：

```
@CacheConfig(cacheNames = "department")
@Service
public class DepartmentService {

  @Autowired
  private DepartmentMapper departmentMapper;

  @CachePut(key = "#department.id")
  public Department save(Department department) {
    System.out.println("保存 id=" + department.getId() + " 的数据");
    this.departmentMapper.insert(department);
    return department;
  }

  @CachePut(key = "#department.id")
  public Department update(Department department) {
    System.out.println("修改 id=" + department.getId() + " 的数据");
    this.departmentMapper.update(department);
    return department;
  }

  @Cacheable(key = "#id")
  public Department getDepartmentById(Integer id) {
    System.out.println("获取 id=" + id + " 的数据");
    Department department = this.departmentMapper.getById(id);
    return department;
  }

  @CacheEvict(key = "#id")
  public void delete(Integer id) {
    System.out.println("删除 id=" + id + " 的数据");
    this.departmentMapper.deleteById(id);
  }
}
```

控制层：

```
@Controller
@RequestMapping("department")
@ResponseBody
public class DepartmentController {

  @Autowired
  private DepartmentService departmentService;

  @RequestMapping("save")
  public Map<String,Object> save(Department department) {
    this.departmentService.save(department);

    Map<String,Object> map = new HashMap<String,Object>();
    map.put("code", "200");
    map.put("msg", "保存成功");
    return map;
  }

  @RequestMapping("get/{id}")
  public Map<String,Object> get(@PathVariable("id") Integer id) {
    Department department = this.departmentService.getDepartmentById(id);

    Map<String,Object> map = new HashMap<String,Object>();
    map.put("code", "200");
    map.put("msg", "获取成功");
    map.put("data", department);
    return map;
  }

  @RequestMapping("update")
  public Map<String,Object> update(Department department) {
    this.departmentService.update(department);

    Map<String,Object> map = new HashMap<String,Object>();
    map.put("code", "200");
    map.put("msg", "修改成功");
    return map;
  }

  @RequestMapping("delete/{id}")
```

```
  public Map<String,Object> delete(@PathVariable("id") Integer id) {
    this.departmentService.delete(id);

    Map<String,Object> map = new HashMap<String,Object>();
    map.put("code", "200");
    map.put("msg", "删除成功");
    return map;
  }
}
```

启动类：

添加 @EnableCaching 注解，开启缓存功能。

```
@EnableCaching
@SpringBootApplication
public class SpringbootNosqlApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringbootNosqlApplication.class, args);
  }
}
```

2.1.4 测试说明

由于 ehcache 缓存是存储在应用的内存中，如果使用 junit 测试，方法执行完毕缓存就释放了，无法正常测试缓存效果，因此测试使用发起 http 请求的形式。

1. 发起保存请求：

```
保存 id=2 的数据
2017-12-06 14:50:48.800 DEBUG 680 --- [nio-8081-exec-7]
c.l.s.dao.DepartmentMapper.insert        : ==>  Preparing: insert into
department(id,name,descr) values(?,?,?)
2017-12-06 14:50:48.801 DEBUG 680 --- [nio-8081-exec-7]
c.l.s.dao.DepartmentMapper.insert        : ==> Parameters: 2(Integer),
Ehcache 部门(String), Ehcache(String)
2017-12-06 14:50:48.868 DEBUG 680 --- [nio-8081-exec-7]
c.l.s.dao.DepartmentMapper.insert        : <==    Updates: 1
```

1. 保存成功后，立刻发起查询请求，没有日志打印，但返回对象数据，说明数据是从缓

存中获取。
2. 发起修改请求：

```
修改 id=2 的数据
2017-12-06 14:51:16.588 DEBUG 680 --- [nio-8081-exec-8]
c.l.s.dao.DepartmentMapper.update        : ==>  Preparing: update
department set name = ? , descr = ? where id = ?
2017-12-06 14:51:16.589 DEBUG 680 --- [nio-8081-exec-8]
c.l.s.dao.DepartmentMapper.update        : ==> Parameters: Ehcache 部门
2(String), Ehcache2(String), 2(Integer)
2017-12-06 14:51:16.657 DEBUG 680 --- [nio-8081-exec-8]
c.l.s.dao.DepartmentMapper.update        : <==    Updates: 1
```

1. 修改成功后，立刻发起查询请求，没有日志打印，但返回修改后的对象数据，说明缓存中的数据已经同步。
2. 发起删除请求：

```
删除 id=2 的数据
2017-12-06 14:52:07.572 DEBUG 680 --- [nio-8081-exec-1]
c.l.s.dao.DepartmentMapper.deleteById    : ==>  Preparing: delete from
department where id = ?
2017-12-06 14:52:07.572 DEBUG 680 --- [nio-8081-exec-1]
c.l.s.dao.DepartmentMapper.deleteById    : ==> Parameters: 2(Integer)
2017-12-06 14:52:07.613 DEBUG 680 --- [nio-8081-exec-1]
c.l.s.dao.DepartmentMapper.deleteById    : <==    Updates: 1
```

1. 删除成功后，立刻发起查询请求，控制台打印 sql 语句，说明缓存数据被删除，需要查询数据库。

```
获取 id=2 的数据
2017-12-06 14:52:40.324 DEBUG 680 --- [nio-8081-exec-3]
c.l.s.dao.DepartmentMapper.getById       : ==>  Preparing: select
id,name,descr from department where id = ?
2017-12-06 14:52:40.325 DEBUG 680 --- [nio-8081-exec-3]
c.l.s.dao.DepartmentMapper.getById       : ==> Parameters: 2(Integer)
2017-12-06 14:52:40.328 DEBUG 680 --- [nio-8081-exec-3]
c.l.s.dao.DepartmentMapper.getById       : <==      Total: 0
```

2.2 Redis 缓存

2.2.1 添加依赖

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

2.2.2 添加配置

application.properties：

```
# redis 配置
spring.redis.host=192.168.2.11
spring.redis.port=6379
spring.redis.password=redis123
# 缓存过期时间，单位毫秒
spring.cache.redis.time-to-live=60000

# 缓存类型（ehcache、redis）
spring.cache.type=redis

# 打印日志，查看 sql
logging.level.com.light.springboot=DEBUG
```

注意：**spring.cache.type=redis**，缓存类型设置成 **redis**。

完成上边 2 个步骤后，其他步骤与测试 Ehcache 时的步骤一致。

测试结果也一致，此处省略。

三、整合 **Redis**

上一个小节其实已经介绍了 Spring Boot 整合 Redis 的内容。

在添加 redis 依赖包启动项目后，Spring Boot 会自动配置 RedisCacheManger 和 RedisTemplate 的 Bean。如果开发者不想使用 Spring Boot 写好的 Redis 缓存，而是想使用其 API 自己实现缓存功能、消息队列或分布式锁之类的需求时，可以继续往下浏览。

Spring Data Redis 为我们提供 RedisTemplate 和 StringRedisTemplate 两个模板进行数据操作，它们主要 的访问方法如下：

方法	说明
opsForValue()	操作简单属性的数据
opsForList()	操作含有 list 的数据
opsForSet()	操作含有 set 的数据
opsForZSet()	操作含有 zset 的数据
opsForHash()	操作含有 hash 的数据

3.1 添加依赖

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

3.2 配置连接

```
spring.redis.host=192.168.2.11
spring.redis.port=6379
spring.redis.password=redis123
```

3.3 编码

```
@Component
public class RedisDao {

  @Autowired
  private StringRedisTemplate stringRedisTemplate;

  public void set(String key, String value) {
    this.stringRedisTemplate.opsForValue().set(key, value);
  }

  public String get(String key) {
    return this.stringRedisTemplate.opsForValue().get(key);
  }

  public void delete(String key) {
    this.stringRedisTemplate.delete(key);
  }
}
```

3.4 测试

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class RedisDaoTest {

  @Autowired
  private RedisDao redisDao;

  @Test
  public void testSet() {
    String key = "name";
    String value = "zhangsan";

    this.redisDao.set(key, value);
  }

  @Test
  public void testGet() {
    String key = "name";
    String value = this.redisDao.get(key);
    System.out.println(value);
  }

  @Test
  public void testDelete() {
    String key = "name";
    this.redisDao.delete(key);
  }
}
```

测试结果省略...

四、整合 **MongoDB**

Spring Data MongoDB 提供了 MongoTemplate 模板 和 Repository 让开发者进行数据访问。

4.1 添加依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

4.2 配置连接

```
spring.data.mongodb.host=192.168.2.31
spring.data.mongodb.port=27017
spring.data.mongodb.database=test
```

4.3 编码

4.3.1 使用 MongoTemplate

```
@Component
public class MongodbDao {

    @Autowired
  private MongoTemplate mongoTemplate;

  public void insert(Department department) {
    this.mongoTemplate.insert(department);
  }

  public void deleteById(int id) {
    Criteria criteria = Criteria.where("id").is(id);
    Query query = new Query(criteria);
    this.mongoTemplate.remove(query, Department.class);
  }

  public void update(Department department) {
    Criteria criteria = Criteria.where("id").is(department.getId());
    Query query = new Query(criteria);
    Update update = new Update();
    update.set("descr", department.getDescr());
    this.mongoTemplate.updateMulti(query, update, Department.class);
  }

  public Department getById(int id) {
    Criteria criteria = Criteria.where("id").is(id);
    Query query = new Query(criteria);
    return this.mongoTemplate.findOne(query, Department.class);
  }

  public List<Department> getAll() {
    List<Department> userList =
this.mongoTemplate.findAll(Department.class);
    return userList;
  }

}
```

4.3.2 使用 **Repository**

```
public interface DepartmentRepository extends MongoRepository<Department,
Integer> {


}
```

测试方式与 Redis 测试大同小异，测试结果省略...

五、源码下载

- Spring Boot 入门之缓存和 NoSQL 篇测试源码

六、参考资料

- 官方文档
- ehcache入门基础示例