MongoDB 与 spring 集成

作者：ldk
原文链接：https://ld246.com/article/1516777230758
来源网站：链滴

一、添加maven依赖

pom.xml:

```
<dependency>
   <groupId>org.mongodb</groupId>
   <artifactId>mongo-java-driver</artifactId>
   <version>2.13.0-rc0</version>
</dependency>
<dependency>
   <groupId>org.springframework.data</groupId>
   <artifactId>spring-data-mongodb</artifactId>
   <version>1.7.1.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.data</groupId>
   <artifactId>spring-data-mongodb-cross-store</artifactId>
   <version>1.7.1.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.data</groupId>
   <artifactId>spring-data-mongodb-log4j</artifactId>
   <version>1.7.1.RELEASE</version>
</dependency>
```

二、mongodb-config.xml的配置

在spring-context.xml中引入mongodb-config.xml：

添加引入mongoDB配置文件：

```
<import resource="classpath*:/mongodb-config.xml" />
```

mongodb-config.xml文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:mongo="http://www.springframework.org/schema/data/mongo"
   xsi:schemaLocation="http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.1.xsd
        http://www.springframework.org/schema/data/mongo
        http://www.springframework.org/schema/data/mongo/spring-mongo-1.0.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">

   <!-- 加载mongodb的属性配置文件 -->
   <context:property-placeholder location="classpath:mongodb.properties" />
```

原文链接：MongoDB 与 spring 集成

```
    <!-- 定义mongo对象，对应的是mongodb官方jar包中的Mongo，replica-set设置集群副本的ip
地址和端口 -->
    <mongo:mongo id="mongo" host="${mongo.host}" port="${mongo.port}">
        <mongo:options connections-per-host="${mongo.connectionsPerHost}"
            threads-allowed-to-block-for-connection-multiplier="${mongo.threadsAllowedToBloc
ForConnectionMultiplier}"
            connect-timeout="${mongo.connectTimeout}"
            max-wait-time="${mongo.maxWaitTime}"
            auto-connect-retry="${mongo.autoConnectRetry}"
            socket-keep-alive="${mongo.socketKeepAlive}"
            socket-timeout="${mongo.socketTimeout}"
            slave-ok="${mongo.slaveOk}"
            write-number="1"
            write-timeout="0"
            write-fsync="true" />
    </mongo:mongo>


    <!-- 用户验证，在mongo.conf中配置了auth=true时使用-->
    <bean id="userCredentials" class="org.springframework.data.authentication.UserCredentia
s">
        <constructor-arg name="username" value="${mongo.username}" />
        <constructor-arg name="password" value="${mongo.password}" />
    </bean>


    <!-- mongo的工厂，通过它来取得mongo实例,dbname为mongodb的数据库名，没有的话会自
创建 -->
    <bean id="mongoDbFactory"
        class="org.springframework.data.mongodb.core.SimpleMongoDbFactory">
        <constructor-arg ref="mongo" />
        <constructor-arg value="${mongo.dbname}" />
        <!-- 用户验证，在mongo.conf中配置了auth=true时使用-->
        <constructor-arg ref="userCredentials" />
    </bean>

    <bean id="mappingContext"
        class="org.springframework.data.mongodb.core.mapping.MongoMappingContext" />

    <bean id="defaultMongoTypeMapper"
        class="org.springframework.data.mongodb.core.convert.DefaultMongoTypeMapper">
        <constructor-arg name="typeKey">
            <null />
        </constructor-arg>
    </bean>

    <!-- collection的映射，配置MappingMongoConverter使spring-data-mongodb的java类注解
效 -->
    <bean id="mappingMongoConverter"
        class="org.springframework.data.mongodb.core.convert.MappingMongoConverter">
        <constructor-arg name="mongoDbFactory" ref="mongoDbFactory" />
        <constructor-arg name="mappingContext" ref="mappingContext" />
        <property name="typeMapper" ref="defaultMongoTypeMapper" />
    </bean>
```

```
    <!-- mongodb的主要操作对象，所有对mongodb的增删改查的操作都是通过它完成 -->
    <bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemp
ate">
        <constructor-arg name="mongoDbFactory" ref="mongoDbFactory" />
        <constructor-arg name="mongoConverter" ref="mappingMongoConverter" />
    </bean>
</beans>
```

三、mongodb.properties的配置

```
#数据库名称
mongo.dbname=myFirstDB
#用户名
mongo.username=root
#密码
mongo.password=root
#主机
mongo.host=127.0.0.1
#端口号
mongo.port=27017
#一个线程变为可用的最大阻塞数
mongo.connectionsPerHost=8
#线程队列数,它以上面connectionsPerHost值相乘的结果就是线程队列最大值
mongo.threadsAllowedToBlockForConnectionMultiplier=4
#连接超时时间（毫秒）
mongo.connectTimeout=1500
#最大等待时间
mongo.maxWaitTime=1500
#自动重连
mongo.autoConnectRetry=true
#scoket保持活动
mongo.socketKeepAlive=true
#scoket超时时间
mongo.socketTimeout=1500
#读写分离
mongo.slaveOk=true
```

四、实体类及其DAO操作类（使用MongoTemplate）

实体类UserAddr：

```
import java.io.Serializable;
import java.util.List;
import java.util.Map;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.GeoSpatialIndexed;
import org.springframework.data.mongodb.core.mapping.Document;

/**
 * 用户位置实体类
```

```
 * @author Administrator
 *
 */
@Document(collection="userAddrs")
public class UserAddr implements Serializable, Cloneable {
    @Id
    private String id;  //用户id
    private String name;  //用户名称
    @GeoSpatialIndexed
    private Double[] loc;  //地址经纬度
    private String addrName;

    private Double distance;    //相距

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Double[] getLoc() {
        return loc;
    }
    public void setLoc(Double[] loc) {
        this.loc = loc;
    }
    public String getAddrName() {
        return addrName;
    }
    public void setAddrName(String addrName) {
        this.addrName = addrName;
    }
    public Double getDistance() {
        return distance;
    }
    public void setDistance(Double distance) {
        this.distance = distance;
    }

    /**
     * 根据map初始化对象
     * @param msgMap
     */
    public void initByMap(Map data) {
        String name = data.get("name").toString();
        String addrName = data.get("addrName").toString();
        List<Double> loc = (List<Double>)data.get("loc");
```

```
        this.setName(name);
        this.setAddrName(addrName);
        this.setLoc(new Double[]{loc.get(0), loc.get(1)});
    }
}
```

实体类对应的DAO操作类UserAddrsDao：

```
import java.util.List;
import javax.annotation.Resource;
import org.springframework.data.geo.GeoResult;
import org.springframework.data.geo.GeoResults;
import org.springframework.data.geo.Metrics;
import org.springframework.data.geo.Point;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.NearQuery;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.stereotype.Repository;
import com.google.common.collect.Lists;
import com.ldk.mongodb.entity.UserAddr;

/**
 * mongodb中用户位置操作DAO
 * @author Administrator
 *
 */
@Repository
public class UserAddrsDao {

    @Resource
    private MongoTemplate mongoTemplate;

    /**
     * 插入信息
     * @param userAddr
     */
    public void insert(UserAddr userAddr) {
        mongoTemplate.insert(userAddr);
    }

    /**
     * 根据id获取信息
     * @param id
     * @return
     */
    public UserAddr get(String id) {
        return mongoTemplate.findOne(
                new Query(Criteria.where("id").is(id)),
                UserAddr.class);
    }
```

```
    /**
     * 根据id，更新所有信息（字段为空的将不被更新）
     * @param userAddr
     */
    public void updateInfo(UserAddr userAddr) {
        Update update = new Update();
        if(userAddr.getName() != null) update.set("name", userAddr.getName());
        if(userAddr.getLoc() != null) update.set("loc", userAddr.getLoc());
        if(userAddr.getAddrName() != null) update.set("addrName", userAddr.getAddrName());

        mongoTemplate.upsert(
                new Query(Criteria.where("id").is(userAddr.getId())),
                update,
                UserAddr.class);
    }

    public void createCollection(String name) {
        mongoTemplate.createCollection(name);
    }

    /**
     * 根据id删除信息
     * @param id
     */
    public void remove(String id) {
        mongoTemplate.remove(
                new Query(Criteria.where("id").is(id)),
                UserAddr.class);
    }

    /**
     * 根据中心点坐标，查询范围内的用户地址列表。
     * @param userAddr 用户地址类，其中的loc字段不能为空。
     * @param maxDistance 最大距离，默认为3，单位km
     * @return 返回null则表示loc地址没有输入或输入错误，返回数组则表示查询到的地址列表。
     * @return
     */
    public List<UserAddr> findByLoc(UserAddr userAddr, Double maxDistance) {
        if(userAddr.getLoc() == null || userAddr.getLoc().length != 2) {
            return null;
        }
        Point point = new Point(userAddr.getLoc()[0],userAddr.getLoc()[1]);
        NearQuery near = NearQuery
                .near(point)
                .spherical(true)
                .maxDistance(maxDistance == null ? 3 : maxDistance,Metrics.KILOMETERS) //MILE
以及KILOMETERS自动设置spherical(true)
                .distanceMultiplier(6371);

        List<UserAddr> resultList = Lists.newArrayList();
        GeoResults<UserAddr> result = null;
        try {
            result = mongoTemplate.geoNear(near, UserAddr.class);
        } catch (NullPointerException e) {
```

```
            return resultList;
        }
        for(GeoResult<UserAddr> geoResultResult : result) {
            UserAddr userAddrResult = geoResultResult.getContent();
            userAddrResult.setDistance(geoResultResult.getDistance().getValue());
            resultList.add(userAddrResult);
        }
        return resultList;
    }
}
```

五、测试

```
@Test
public void testAdd() {
    // 添加一百个userAddr
    for (int i = 0; i < 100; i++) {
        UserAddr userAddr = new UserAddr();
        userAddr.setId("" + i);
        userAddr.setName("zcy" + i);
        userAddr.setAddrName("坐标点" + i);
        userAddr.setLoc(new Double[]{121.60225, 31.21410+i*0.01});
        userAddrsDao.insert(userAddr);
    }
}

@Test
public void testQuery() {
    UserAddr userAddr = new UserAddr();
    userAddr.setLoc(new Double[]{121.60225, 31.20410});
    List<UserAddr> userAddrs = userAddrsDao.findByLoc(userAddr, null);
    System.err.println("count ====== "+userAddrs.size());
}
```

六、Spring中映射Mongodb中注解的解释

spring-data-mongodb中的实体映射是通过MappingMongoConverter这个类实现的。它可以通过解把java类转换为mongodb的文档。

它有以下几种注解：

@Id：文档的唯一标识，在mongodb中为ObjectId，它是唯一的，通过时间戳+机器标识+进程ID+增计数器（确保同一秒内产生的Id不会冲突）构成。

@Document：把一个java类声明为mongodb的文档，可以通过collection参数指定这个类对应的文。

@Document(collection="users")：标识实体类在mongodb中对应的集合。

@DBRef：声明类似于关系数据库的关联关系。ps：暂不支持级联的保存功能，当你在本实例中修改DERef对象里面的值时，单独保存本实例并不能保存DERef引用的对象，它要另外保存，如下面例子的erson和Account。

@Indexed：声明该字段需要索引，建索引可以大大的提高查询效率。

@CompoundIndex：复合索引的声明，建复合索引可以有效地提高多字段的查询效率。

@GeoSpatialIndexed ：声明该字段为地理信息的索引。

@Field：代表一个字段，可以不加，不加的话默认以参数名为列名。

@Transient：映射忽略的字段，该字段不会保存到mongodb。

@PersistenceConstructor：声明构造函数，作用是把从数据库取出的数据实例化为对象。该构造函传入的值为从DBObject中取出的数据

参考：https://www.cnblogs.com/byteworld/p/5963112.html