## Mongodb

### MongoRepository默认查询接口

A、查询所有数据：

        public List<Person> queryAll() throws Exception {

        return personRepository.findAll();

    }

B、查询所有的数据带分页：方法为：Page<?> findAll(Pageable pageable); 该方法中的参数是一个借口，我们需要构造一个子对象，即：PageRequest对象,这个对象中有两个属性，第一个是页码，第二个是页面大小。注意：页码在mongodb中是从0开始的

**public** Page<Person> queryAllByPage(**int** page,**int** rows) **throws** Exception {

        PageRequest pageRequest = **new** PageRequest(page-1,rows);

**return** personRepository.findAll(pageRequest);

    }

C、属性查询：定义方法名的规则为：find + By + 属性名（首字母大写）：

public Person findByName(String name);

D、模糊查询：模糊查询定义方法名的规则为：find + By + 属性名（首字母大写） + Like

       仓库中添加的方法：

        public List<Person> findByNameLike(String name);

        service中的方法：

        在service中直接调用仓库中我们刚才定义的方法即可！

        public List<Person> queryByFirstNameLike(String name) throws Exception {

        return personRepository.findByNameLike(name);

}

E、根据实体类中的属性进行模糊查询带分页：

     仓库中添加的方法：

        public Page<Person> findByNameLike(String name,Pageable pageable);

        在service中对仓库中的方法的调用：

        public List<Person> queryByNameAndPage(int page, int rows, String name)

            throws Exception {

        PageRequest pageRequest = new PageRequest(page-1,rows);

        return personRepository.findByNameLike(name, pageRequest).getContent();

    }

F：根据实体类中的属性进行模糊查询带分页，同时指定返回的键（数据库中的key,实体类中的属性）：

        解释一下什么是指定返回的键：也就是说当我们进行带分页的模糊查询时，不想返回数据库中的所有字段，只是返回一部分字段。例如：只返回Person中的id和name，不返回age.

        若想指定返回的键，我们需要在PersonRepository中添加方法，同时使用注解@Query。

        仓库中定义的方法：

        @Query(value="{'name':?0}",fields="{'name':1}")

    public Page<Person> findByNameLike(String name,Pageable pageable);

        其中value是查询的条件，？0这个是占位符，对应着方法中参数中的第一个参数，如果对应的是第二个参数则为？1。fields是我们指定的返回字段，其中id是自动返回的，不用我们指定，bson中{'name':1}的1代表true，也就是代表返回的意思。

        在service中对仓库中的方法的调用：

        public List<Person> queryByNameAndPage(int page, int rows, String name)

            throws Exception {

        PageRequest pageRequest = new PageRequest(page-1,rows);

        return personRepository.findByNameLike(name, pageRequest).getContent();

    }

G：需要查询所有数据，同时指定返回的键

        当我们需要查询所有数据，同时需要指定返回的键时，则不能使用仓库中自带的findAll（）方法了。我们可以查询所有id不为空的数据，同时指定返回的键。当我们需要根据一个key且该key不为空进行查询，方法名的定义规则为：find + By + 属性名（首字母大写） + NotNull。

  仓库中定义的方法：

        @Query(value="{'\_id':{'$ne':null}}",fields="{'name':1}")

    public Page<Person> findByIdNotNull(Pageable pageable);

service中调用仓库中的方法：

        public List<Person> queryAll(int page, int rows) throws Exception {

        PageRequest pageRequest = new PageRequest(page-1,rows);

        return personRepository.findByIdNotNull(pageRequest).getContent();

    }

F、其他：MongoDB的其他查询不一一列举，但将java中的仓库定义的方法名的规则列举如下，使用时将仓库中方法上的注解@Query中的value进行适当泰欧正即可。

        GreaterThan(大于)

        方法名举例：findByAgeGreaterThan(int age)

        query中的value举例：{"age" : {"$gt" : age}}

        LessThan（小于）

        方法名举例：findByAgeLessThan(int age)

        query中的value举例：{"age" : {"$lt" : age}}

        Between（在...之间）

        方法名举例：findByAgeBetween(int from, int to)

        query中的value举例：{"age" : {"$gt" : from, "$lt" : to}}

        Not（不包含）

        方法名举例：findByNameNot(String name)

        query中的value举例：{"age" : {"$ne" : name}}

        Near（查询地理位置相近的）

        方法名举例：findByLocationNear(Point point)

        query中的value举例：{"location" : {"$near" : [x,y]}}

### MongoRepository插入

insert:当主键"\_id"在集合中存在时，不做任何处理。 抛异常

　　save:当主键"\_id"在集合中存在时，进行更新。 数据整体都会更新 ，新数据会替换掉原数据 ID 以外的所有数据。如ID 不存在就新增一条数据

      save 方法 需要遍历列表，一个个插入， 而 insert 方法 是直接批量插入

　　那么

　　　　Springboot-mongodb   MongoRepository接口   并未提供 insert 方法 ，只提供了 save 方法  。 而  数据比较多 想使用 insert  批量插入 提高速度  怎么办

第一种  使用 原生 MongoTemplate 类  进行处理  想怎么样就 怎么样 。  比如  ID 自增

@Autowired

private MongoTemplate mongoTemplate;

public void insertList(List<ThothOrder> t)

{

mongoTemplate.insertAll(t);

}

第二种   看 MongoRepository  接口 的具体实现类   SimpleMongoRepository<T, ID extends Serializable>   save 方法到底怎么写的。如果批量save，只要有一个存在，就遍历save，否则就批量insertAll

public <S extends T> List<S> save(Iterable<S> entities)

### SpringData原始操作

<https://blog.csdn.net/zhouzhiwengang/article/details/53099382>

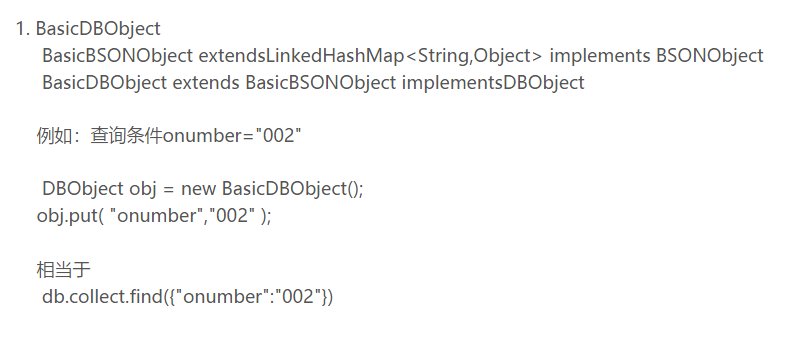
**org.springframework.data.mongodb.core.query**



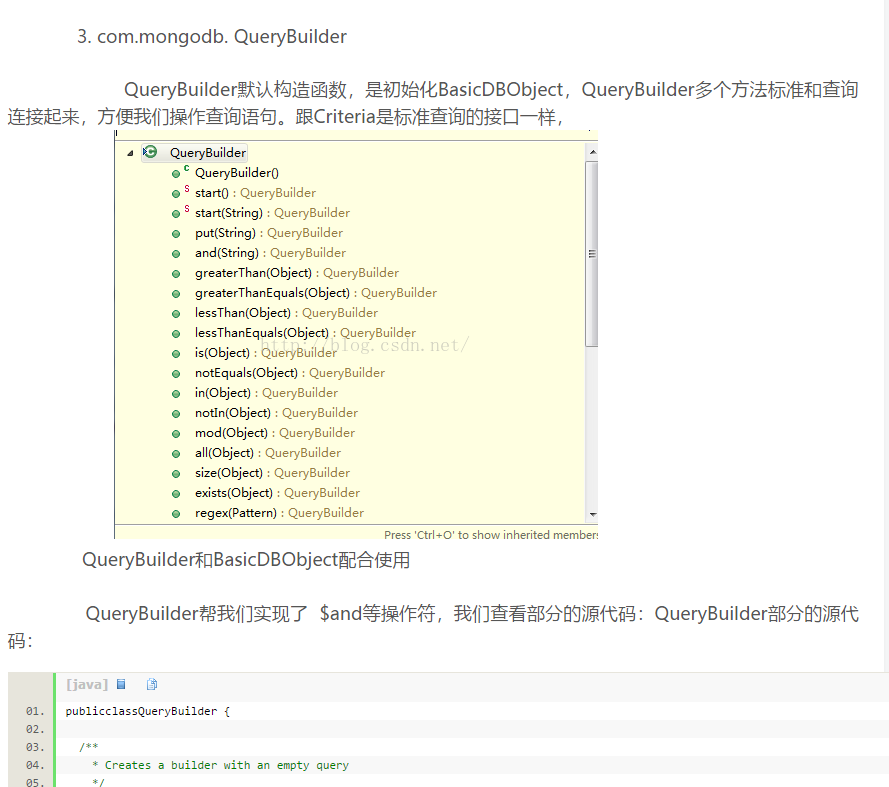


**子类 org.springframework.data.mongodb.core.query.BasicQuery**









### mongoTemplate原始操作

mongoTemplate.findOne(query,entityClass)

1. Queryquery=newQuery(Criteria.where("onumber").is("002"));
2. Ordersorder=ordersDao.findOne(query,collectionName);
3. System.out.println(JSONObject.fromObject(order));

指定返回的字段

1. QueryBuilder queryBuilder = **new** QueryBuilder();
2. queryBuilder.or(**new** BasicDBObject("onumber", "002"), **new** BasicDBObject("cname","zcy1"));
3. BasicDBObject fieldsObject=**new** BasicDBObject();
4. fieldsObject.put("onumber", 1);
5. fieldsObject.put("cname", 1);
6. uery query=**new** BasicQuery(queryBuilder.get(),fieldsObject);

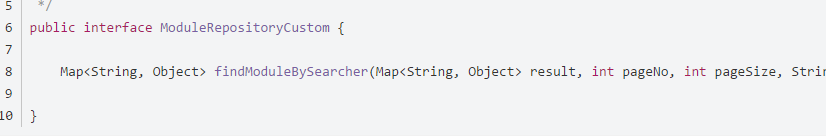
https://blog.csdn.net/qq\_27093465/article/details/51578369

### 自定义接口（增强）

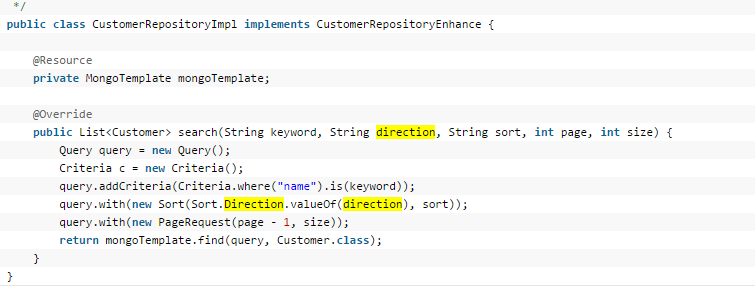
1. 原始接口：



1. 定义自定义接口声明



1. 实现自定义接口





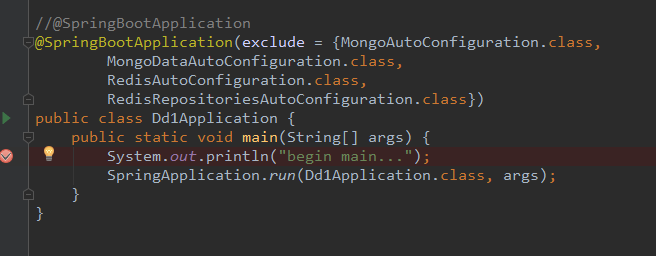
1. 组合接口



1. service使用



## 启动

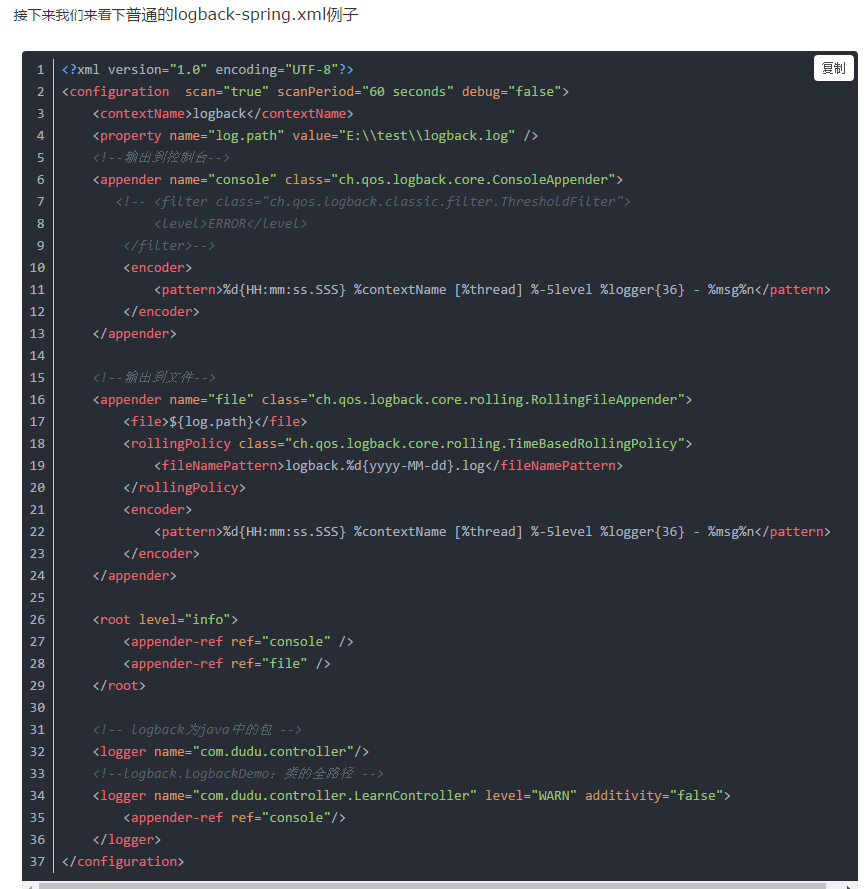




## 3、spring boot日志

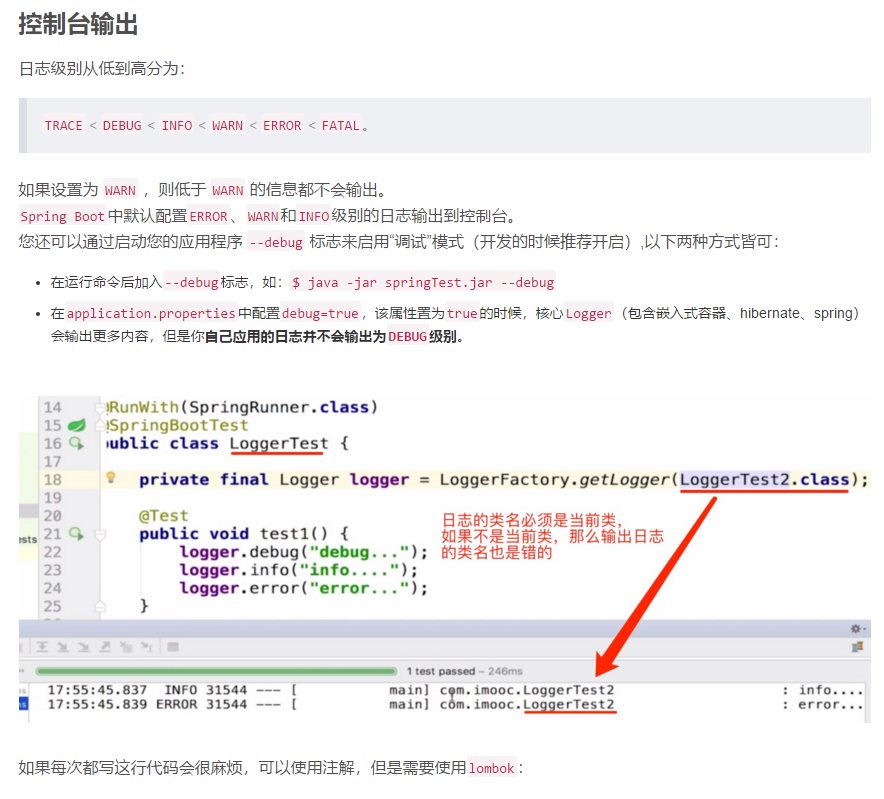
日志配置：<https://blog.csdn.net/inke88/article/details/75007649>

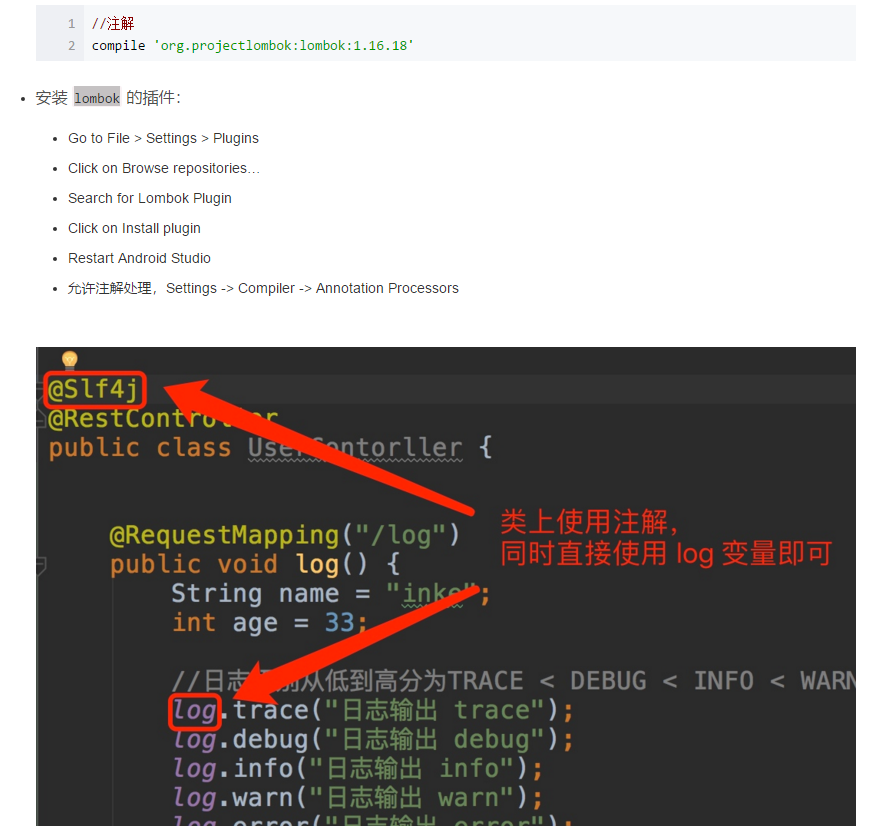
<https://blog.csdn.net/gwd1154978352/article/details/78344091>



默认日志 Logback：

默认情况下，Spring Boot会用Logback来记录日志，并用INFO级别输出到控制台。在运行应用程序和其他例子时，你应该已经看到很多INFO级别的日志了。















日志自定义输出到mongodb：

https://www.cnblogs.com/yeyinfu/p/7316994.html





## 4、spring boot test

<https://www.jianshu.com/p/72b19e24a602>













https://blog.csdn.net/congzi0424/article/details/51518819每一个小类又分三种，这三种有什么区别？

\* 第一种和第二种的首个参数都是用String表示一个URI。但它们的最后一个参数分别是Object[]和Map

\* 第三种的首个参数使用java.net.URI表示一个URI。且只有两个参数

  这是因为，String类型的URI支持占位符。比如：

restTemplate.getForObject("http://example.com/hotels/{hotel}/bookings/{booking}",String.class,"42", "21");

那么最终访问的URI为：<http://example.com/hotels/42/bookings/21>

  但是String有一个小缺陷：String形式的URI会被URL编码两次（URL encode请自行百度），这就要求服务器在获取URI中的参数时主动进行一次解码，但如果服务的提供者不这么做呢？

  这时就需要使用不会使用任何编码的java.net.URI

PS：参数‘Class<T>  responseType’定义了返回数据的类型。

（3）Exchange

  与其它接口的不同：

>允许调用者指定HTTP请求的方法（GET,POST,PUT等）

>可以在请求中增加body以及头信息，其内容通过参数‘HttpEntity<?>requestEntity’描述

>exchange支持‘含参数的类型’（即泛型类）作为返回类型，该特性通过‘ParameterizedTypeReference<T>responseType’描述。比如：

1. List<String> a = new ArrayList<String>();
2. System.out.println(a.getClass());
3. System.out.println(a.getClass().getGenericSuperclass());
4. ParameterizedTypeReference pt = new ParameterizedTypeReference<ArrayList<String>>() {};
5. System.out.println(pt.getType());

得到的结果是：

1. class java.util.ArrayList
2. java.util.AbstractList<E>
3. java.util.ArrayList<java.lang.String>

\* 这是因为ParameterizedTypeReference<ArrayList<String>>并不根据实参而是使用getGenericSuperclass()方法获取其父类的类型（注意这里的new有花括号，是ParameterizedTypeReference的子类），父类的类型通过**[java](https://blog.csdn.net/congzi0424/article/details/51518819" \t "_blank).**[**lang**](https://blog.csdn.net/congzi0424/article/details/51518819)**.**[**reflect**](https://blog.csdn.net/congzi0424/article/details/51518819)**.Type**描述，然后通过Type的getActualTypeArguments()获得了父类的实参类型，注意得到的Type类，并不是class类。  
  
（4）excute

所有的get、post、delete、put、options、head、exchange方法最终调用的都是excute方法。举个栗子:

1. @Override
2. public <T> T getForObject(String url, Class<T> responseType, Object... urlVariables) throws RestClientException {
3. RequestCallback requestCallback = acceptHeaderRequestCallback(responseType);
4. HttpMessageConverterExtractor<T> responseExtractor =
5. <span style="white-space:pre"> </span>new HttpMessageConverterExtractor<T>(responseType, getMessageConverters(), logger);
6. return execute(url, HttpMethod.GET, requestCallback, responseExtractor, urlVariables);
7. }

Excute方法只是将String格式的URI转成了java.net.URI，之后调用了doExecute方法。整个调用过程

.doExcute

6.1 定义

protected <T> T doExecute(URI url, HttpMethod method, RequestCallback requestCallback,ResponseExtractor<T> responseExtractor) throws RestClientException {…}

这里需要了解两个类： RequestCallback &ResponseExtractor

## 5、JMS

## 6、mysql使用

## 7、redis使用

## 8、cache使用

## 9、自定义配置类

## 10、session

## 11、security

## 12、ldap

## 13、JMX

## 14、elasticsearch

## 15、hbase

## 16、mapreduce