# 依赖相同包不同版本问题

Pom.xml中

依赖zkclient：zookeeper客户端

依赖dubbo-spring-boot-starter：dubbo服务

这两个依赖，都同时引入了zookeeper的依赖，而且是不同版本的两个zookeeper包

问题：

导致启动应用时报java.lang.NoSuchMethodError:org.apache.zookeeper.ZooKeeper.

解决：

1. Maven Project中点击show Dependencies的图标
2. 在依赖树结构目录中找到zookeeper，有多个不同版本，把不需要的右键选择excluse
3. 查看Pom.xml对应的zkclient和dubbo-spring-boot-starter依赖，确认<exclusion>掉了一个版本

但不能全部都<exclusion>掉

1. Maven Project更新一下依赖，再重新启动应用

# 接口打包

插件：maven-jar-plugin

1、在pom.xml 文件的<plugins>中增加<plugin>导入maven-jar-plugin插件

2、在插件的<configuration>配置 <includes>指定打包目录或者文件

在<excludes>指定打包不包含的目录或者文件

3、Maven Project更新一下依赖导入插件

4、插件显示名称：jar，选择jar目录下的jar:jar打包

5、在项目的target目录下生成打包后的jar文件

6、如果是本地调式，直接导入本地项目、如果是远程或者线上使用，要上传到maven私服maven仓库

# RocketMq

下载配置参考<http://rocketmq.apache.org/release_notes/release-notes-4.3.2/>

启动和控制台参考：<https://jingyan.baidu.com/article/219f4bf7bf0f26de452d3840.html>

Broker 启动后1分钟后失效：https://blog.csdn.net/jpf254/article/details/80748021

start mqbroker.cmd autoCreateTopicEnable =true -c D:\util\rocketmq-all-4.3.2-bin-release\rocketmq-all-4.3.2-bin-release\conf\2m-noslave\broker-a.properties

start mqbroker.cmd -n localhost:9876 autoCreateTopicEnable=true

# Maven

参考：http://www.cnblogs.com/wkrbky/p/6350334.html?utm\_source=itdadao&utm\_medium=referral

# RabbitMq

erlang 下载：<https://www.erlang.org/downloads>

RabbitMQ下载：<http://www.rabbitmq.com/>

激活Rabbit MQ's Management Plugin：

\sbin rabbitmq-plugins.bat enable rabbitmq\_management

重启rabbitmq服务，管理员运行cmd输入命令:

Net stop rabbitmq && net start rabbitmq

访问：用户名：guest，密码：guest

<http://localhost:15672>

发送消息先发送给ExChange，然后再分发给相应关系的Queue

ExChange四种类型：fanout、direct、topic、headers

Fanout：发送给与交换机绑定的所有队列

Direct：发送给同一个交换机下的拥有相应RoutingKey的队列（精确）

Topic：发送给同一个交换机下的拥有相应RoutingKey的队列（表达式）

Headers：发送给同一个交换机下的拥有相应headers的队列

# Redis

官网下载：<https://github.com/MicrosoftArchive/redis/releases>

启动：主目录下双击redis-server.exe、redis-cli.exe

Key——value

EX：有效期

NX：键不存在时设置键

组合命令：SET key value EX 5 NX

多端口启动：

复制redis.windows.conf，改名。（注意：不是复制redis.windows-service.conf）

启动：cmd到redis目录命令：redis-server 改后名称.conf，看到redis日志说明启动成功

客户端访问：cmd到redis目录命令：redis-cli.exe  -h localhost -p 新端口号

访问list数据：lrange listName 0 -1(第一个到最后一个)

StringRedisTemplate extends RedisTemplate

两者区别：

1、数据不互通

2、RedisTemplate采用JDK的序列化策略

StringRedisTemplate采用String的序列化策略（redis查看数据是String类型）

# Nginx

官网下载：<http://nginx.org/en/download.html>

主目录下双击nginx.exe就可启动

Cmd命令：

启动：主目录下：start nginx 或者nginx.exe

停止：nginx –s stop 或者任务管理器结束进程

可看是否已启动：tasklist /fi “imagename eq nginx.exe”

修改配置：\conf目录下nginx.conf，不修改也能启动、浏览器访问localhost，可看到欢迎页面

实现负载均衡

http：upstream配置服务器集群

http：server：location 配置nignx代理转发到服务器集群，可使用正则表达式匹配请求

#Tomcat服务器集群

upstream tomcat\_servers{

server localhost:8080;

server localhost:8081;

}

location /girl {

proxy\_set\_header Host $http\_host;

proxy\_pass http://tomcat\_servers;

}

# Zookeeper

官网下载：http://mirror.bit.edu.cn/apache/zookeeper/

下载后\conf目录下复制zoo\_sample.cfg，改名为zoo.cfg（单服务器不用使用，集群需要另外配置）

进入\bin目录下，启动zkServer.cmd服务器，zkCli.cmd客户端，就可使用

# Dubbo

官网下载：<http://dubbo.apache.org/zh-cn/docs/user/quick-start.html>

下载demo:https://github.com/apache/incubator-dubbo.git

下载Zookeeper：<http://archive.apache.org/dist/zookeeper/zookeeper-3.3.3/zookeeper-3.3.3.tar.gz>

修改配置：修改dubbo 配置提供者和消费者的Zookeeper地址，Zookeeper不需要集群时不用修改Zookeeper的配置

管理平台war包和Tomcat：工具的分享链接：https://pan.baidu.com/s/1dFRTXYD 密码：bdk8

1、将dubbo-admin-2.5.4.war放入tomcat的webapps目录下启动注册中心，启动提供者、消费者

2、进入tomcat的bin目录运行startup.bat生成解压文件dubbo-admin

3、修改zookeeper配置：进入dubbo-admin/WEB-INF/目录下面修改dubbo-properties

4、重新启动tomcat

访问地址：<http://localhost:8080/dubbo-admin/>

输入用户名root密码admin\_dubbo即可登录成功

配置

1. 接口打jar包，提供者和消费者导入
2. Xml配置：application、registry、提供者（protocol、service）、消费者（reference）
3. 使配置文件生效：Application的main方法所在类增加

@DubboComponentScan

@ImportResource({"classpath:/application-dubbo.xml" })

1. 增加依赖dubbo-spring-boot-starter
2. 提供者实现接口注解不能用@Service、消费者使用接口@Reference(check = false)

例子：

spa\_manger： 提供者：微信菜单生成接口

Dubbo：服务管理平台，可以查看到注册的提供者和消费者状态

authorization\_service：消费者：授权成功后调用微信菜单生成接口

实现不同服务之间的同步通信

Dubbo是什么：分布式服务框架，提供RPC远程服务调用解决方案，SOA服务治理方案

Dubbo有什么特点：

远程通信：透明化远程方法调用,支持多协议、默认tcp

集群容错：软负载均衡、失效容错，地址路由，动态配置等集群支持

自动发现：基于注册中心目录服务，便服务的消费方能动态的查找服务提供方，支持平滑减少或者增加机器

注册中心：默认采用Zookeeper

为什么要使用dubbo：

MVC, RPC,SOA演进

Dubbo和spring cloud、ice的区别

Zookeeper是什么：注册中心，管理服务提供者Url地址，服务消费者可订阅服务，可通过

Dubbo文档

http://dubbo.apache.org/zh-cn/docs/user/quick-start.html

# 常用类

类：ArrayList、HashMap、Math、Date、Calendar、HSSFWorkbook、Object、String、System

包：java lang、java util、javax.transaction、javax.persistence、org.slf4j; java.io

接口：List、Map、Set、Iterator、Comparable、Runnable

# 集合

Collection实现comparable/comparator接口

实现了List接口（List接口继承了Collection接口）、Iterator遍历元素、get(index)

ArrayList ：单列，有序，可重复，非线程安全，存储空间增加一次0.5倍，继承AbstractList抽象类

Vector ：单列，有序，可重复，线程安全，存储空间增加一次1倍，继承AbstractList抽象类

LinkedList：双向链表实现存储、非线程安全，增删改快，查慢（前向或后向遍历）

实现了set接口（set接口继承了Collection接口）

HashSet ：单列，有序，不可重复，非线程安全，继承Abstractset抽象类、Iterator遍历元素、按照hashcode值的某种运算方式进行存储

LinkedHashSet：单列，有序，不可重复、按插入的顺序存储

TreeSet：，继承Abstractset抽象类、基于TreeMap实现、自然排序 或者 根据创建TreeSet 时提供的 Comparator 进行排序、实现NavigableSet接口（NavigableSet继承SortedSet，SortedSet继承Set）

实现了Map接口、get(key)

HashMap：键值对，无序，键不重复，非线程安全，键值可为null，继承AbstractMap，containsvalue和containsKey

Hashtable：键值对，无序，键不重复，线程安全，键值不可为null，继承Dictionary，contains

# Thread.sleep

不用import、创建对象、实现Runnable接口也能使用

public static void sleep()：静态方法直接通过类使用

Theard和String类一样都在java.lang包下，这个包下的类都会自动引入不需要显示引包

会自动获取到当前线程

# 事件机制：监听器

定义事件Event继承 ApplicationEvent

定义监听器Listenert实现接口 ApplicationListener

实现ApplicationListener接口中的方法onApplicationEvent

SmartApplicationListener extends ApplicationListener<ApplicationEvent>, Ordered

两者区别：

SmartApplicationListener，可排序，新增加两个方法可判断EventType和SourceType

# MySql主从数据库

分解数据库执行的压力，提高响应速度

主数据库：增、删、改操作

从数据库：查操作，定时任务执行

主数据库增加用户：用户权限能读取主数据库的二进制操作日志

从数据库通过主数据库的操作日志同步数据

主从数据库分别安装在不同的服务器上，用硬件换执行速度

# 前端传数组参数

1. 参数增加@RequestBody @Valid注解
2. model创建接收的参数为List类型
3. postman中，
   1. Headers ：Content-Type = application/json
   2. body：raw：JOSN（application/json）

写入参数：

{

"settingsList": [{

"commissionType": "1",

"commissionTime": "1",

"leftInterval": "0",

"rightInterval": "1000",

"commissionAmount": "500"

}]

}

# @PostConstruct和@PreDestroy

@PostConstruct修饰的方法会在服务器启动时依赖关系注入完成之后运行，只会被服务器执行一次。@PostConstruct修饰的方法在构造函数之后、init()方法之前执行。@PreDestroy修饰的方法在destroy()方法之后执行

两者都用来修饰非静态的void（）方法

# @Value("${xx.xx}")

[使用@Value("${xx.xx}")](mailto:使用@Value(%22$%7bxx.xx%7d%22))注解的变量，变量的值取配置文件中xx.xx的值

# 事务提交后执行

如果对数据库的操作使用了事务，希望在整个事务成功提交后再再执行一些操作，则可以通过以下模块注册同步操作

TransactionSynchronizationManager.registerSynchronization

实现TransactionSynchronizationAdapter抽象类中的afterCommit方法，在方法中实现需要事务成功提交后同步执行的事情

# [类名.方法调用与@Autowired](mailto:类名.方法调用与@Autowired)注入bean后调用的区别

类名.方法：静态方法或者静态变量是类的属性，不是实例对象的属性，所以在调用时需要通过类名调用，所有所有该类的实例都共享，全局性作用域

@Autowired：实例对象的属性，需要依赖注入后通过具体的实例对象调用，不同的实例属性值不同，实例的生命周期由spring框架管理

# 研发规范记录

1. 数据库建表空字符指明DEFAULT ‘’ not NULL
2. 修改数据库的接口，尽量指明用POST方式
3. controller,层尽量简洁，业务逻辑实现尽量放到service层，数据库操作尽量放到Dao层
4. 金额字段以分为单位，int类型，前端后端交互、数据库保存都以分为单位

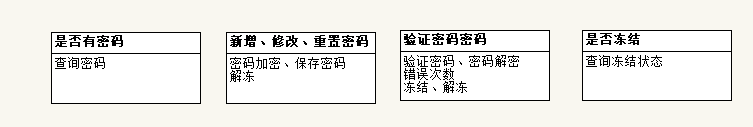
# 密码

## 流程图



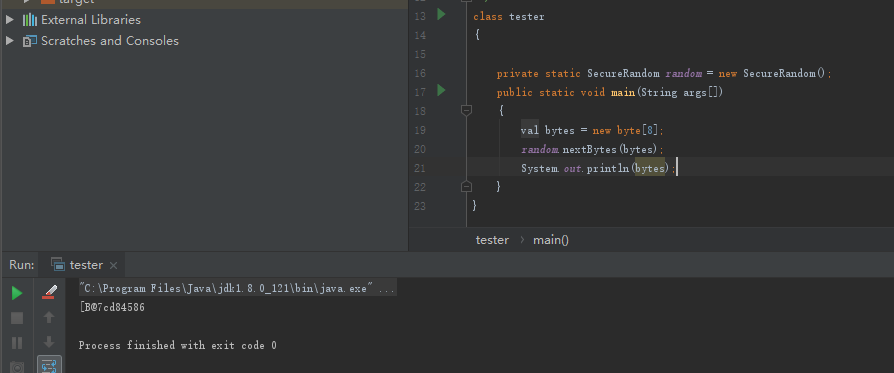
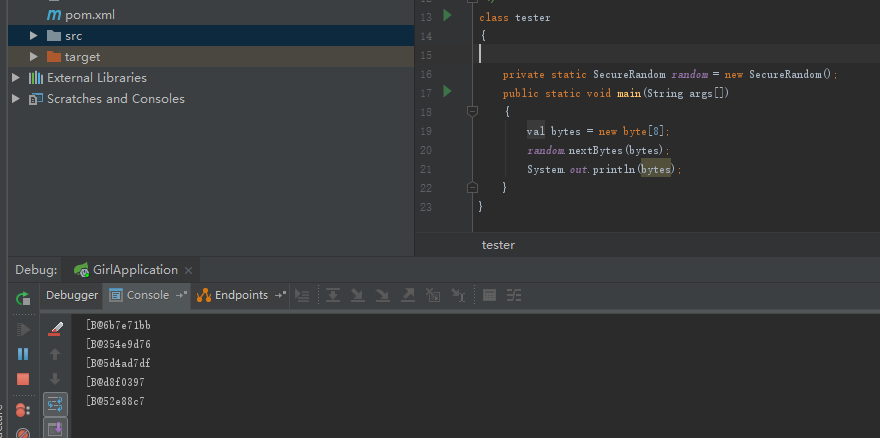
## 模块

主要分拆成这几个模块来设计接口或者函数



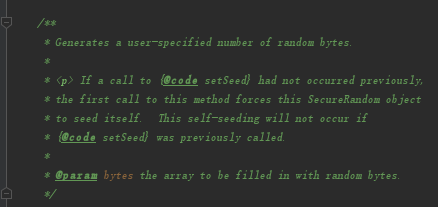
## 加密解密

### 疑問1

  
关于我们加密用的那个随机数  
截图1、接口调用，每次调用输出都是不同的  
截图2、main函数运行，每次运行的输出都是相同的

截图2这种方式一起想不明白，网上查过，也看了点源码，也没得出确切的答案

我猜想是：main函数运行每次使用的种子都是相同的，而通过接口调用时种子更新了



源码中的注解，看起来run main函数时JDK不认为是a call

### 疑問2

为什么多次sha-1 hash哈希计算都要复位摘要

### 加密

1、通过java.security.SecureRandom的nextBytes方法得到一个伪随机salt

2、再通过org.apache.commons.codec.binary.Hex#encodeHexString把伪随机salt进行Hex编码，得到可视化字符串再保存

3、用这个伪随机数salt经过多次 sha-1 hash，得到加密后的密码再保存

1. 创建sha-1消息摘要对象：java.security.MessageDigest#getInstance("SHA-1")
2. 使用伪随机数更新摘要：java.security.MessageDigest#update(salt)
3. 完成哈希计算，在调用完成后，摘要被重置：java.security.MessageDigest#digest(password.getBytes())

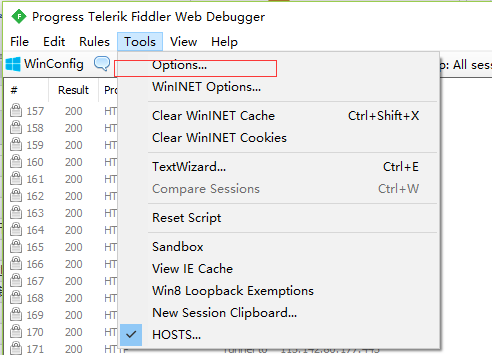
### 验证密码

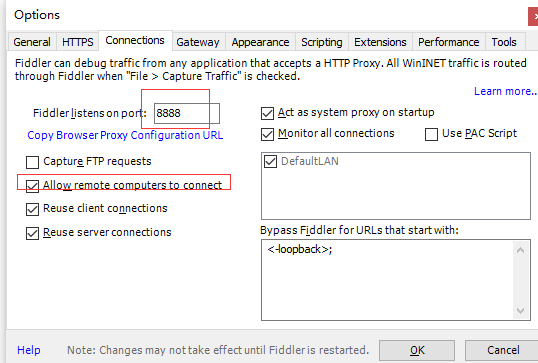
1. 用之前保存的Hex编码后的salt，通过org.apache.commons.codec.binary.Hex#decodeHex进行Hex解码得到伪随机salt

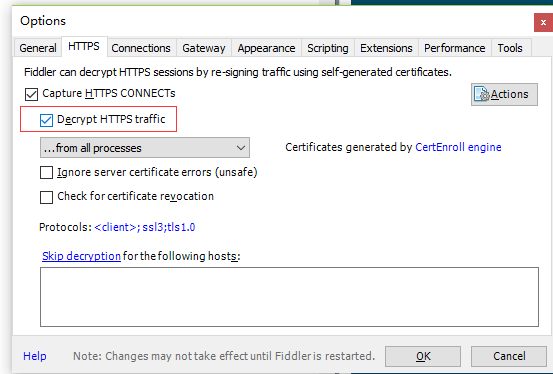
2、再用这个伪随机数把要验证的密码字符串用相同的加密方法加密后，与保存的正确密码进行比较

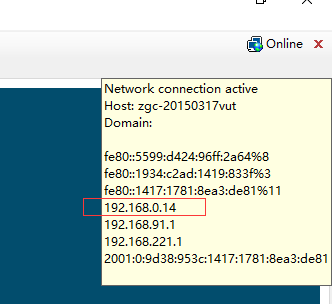
# Fiddler 手机抓包

下载并安装Fiddler 4

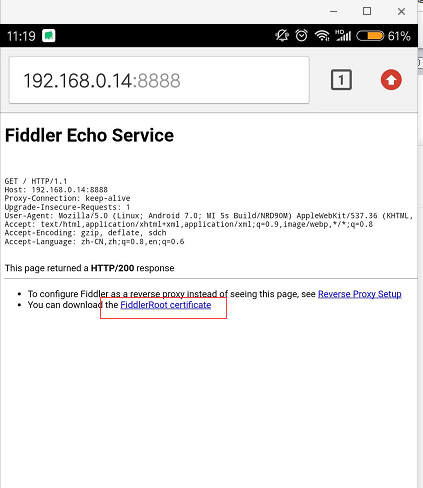


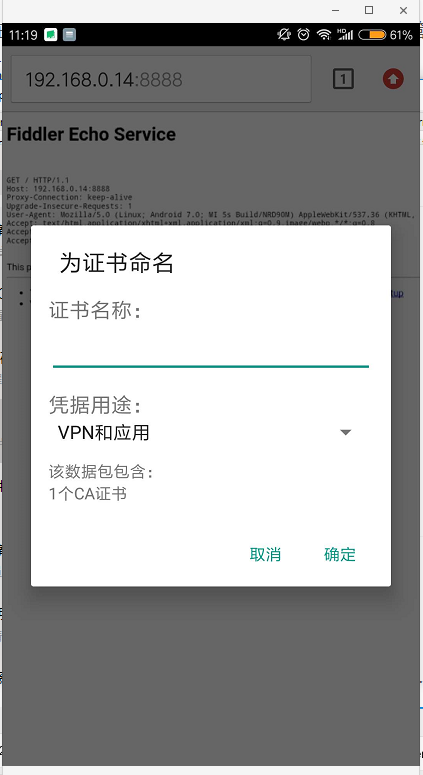


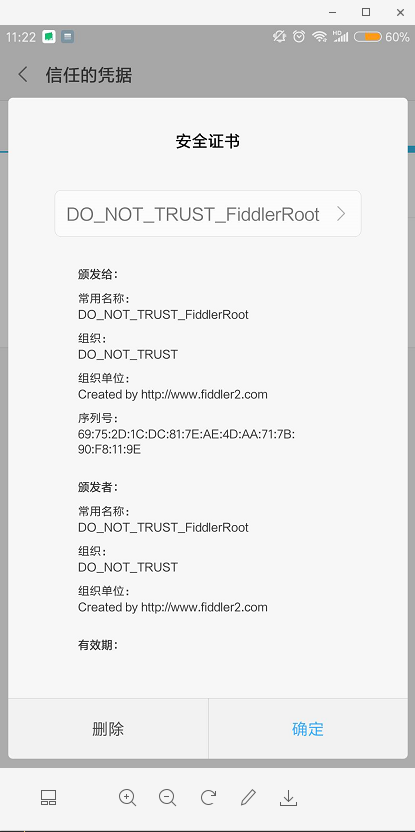












# 多数据源配置

实现同一个项目以不同的方式链接使用多个数据库，如果只使用一个数据源，很多配置都可以省略，因为spring boot构架会有一些默认的配置可以直接使用，但是想要使用多个数据源时，就需要更多的配置来加以区分来告诉spring boot应该使用哪个数据源，事务管理、session管理等对应的是哪个数据源

## 1、pom.xml配置文件

在dependencies标签对中增加两个dependency标签对配置数据库的依赖包，

### 支持jdbc、Hibernate数据库链接

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
</dependency>

### 支持mybatis数据库链接

还需要增加两个dependency标签对配置

<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>1.3.2</version>  
</dependency>

## 2、application.properties配置文件

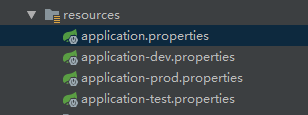
### profiles配置

如果使用profiles配置，可启用不同环境的配置文件，如：开发环境，测试环境，生产环境

假如现在处于开发阶段，那么在application.properties配置文件中增加

#环境配置  
spring.profiles.active=dev

不同环境的配置文件命名示例：



以上配置表明，目前生效的配置文件是application.properties和application-dev.properties两个文件

### 数据库链接参数配置

数据库参数配置（jdbc、Hibernate、mybatis，三种方式的配置都是一样的

只是定义的名称要有所区分如dataSource1，dataSource2……，该名称会在Java配置文件定义其他bean时被引用）

dataSource1.datasource.driverClassName=com.mysql.jdbc.Driver  
dataSource1.datasource.jdbc-url=jdbc:mysql://域名:端口号/数据库名?useUnicode=true&useSSL=false  
dataSource1.datasource.username=数据库用户名  
dataSource1.datasource.password=数据库密码

## 3、Java配置文件

选择使用Java配置文件来配置，主要是因为项目代码是使用java语言开发，使用Java配置文件更容易理解

这一步主要的作用是定义各种bean时，定义不同的bean名称，并且在需要传入数据库的方法中，在与第2步数据库链接参数配置的名称一一对应

### Jdbc

#### 注解

@Configuration

什么是Java配置文件，就是创建的java类，增加了@Configuration注解，这样spring boot框架扫描bean的时候，才会把这样的Java类文件当作是配置文件，然后是在类里面可以随意的手动增加一些bean

#### dataSource

DataSource（jdbc、Hibernate、mybatis，三种方式的DataSource配置都是一样的

只是定义的名称要有所区分，可以与第2步数据库链接参数配置的名称，如dataSource1，dataSource2……）

@Bean(name = "dataSource1")//因为是多数据源，所以需要取别名  
 @Qualifier("dataSource1")   
 @ConfigurationProperties(prefix = " dataSource1.datasource")//第2步数据库链接参数配置的名称  
 public DataSource dataSource1 (){//定义数据源bean  
 return DataSourceBuilder.*create*().build();  
 }

#### Template

这一步关键的生成一个重新命名并配置好数据源的JdbcTemplate

1. 重新命名：在方法前面增加@Bean

@Bean(name = "jdbcTemplate")

1. 定义该方法时要传入带了别名的数据源为参数

public JdbcTemplate jdbcTemplate(@Qualifier("dataSource1") DataSource dataSource)

1. 生成的JdbcTemplate并把数据源set进来，然后返回JdbcTemplate

return new JdbcTemplate(dataSource);

#### 使用

##### 直接使用

引入JdbcTemplate就可以调用Template里封装的查询方法了,按封装方法的要求传入参数就可以了

@Autowired  
private JdbcTemplate jdbcTemplate;

JdbcTemplate自带查询抽象方法如下：

public <T> List<T> query(String sql, @Nullable Object[] args, RowMapper<T> rowMapper)

实际调用如下：

事先写好String类型的sql，和要输入的参数列表params，以及返回结果列表集

BeanPropertyRowMapper是RowMapper的实现类

List<mapper> list = jdbcTemplate.query(sql, params.toArray(), new BeanPropertyRowMapper<>( mapper.class));

##### b、封装成通用工具

当然了，如果调用JdbcTemplate的抽象方法之前想多做一些事情，比如打印日志，参数转换等，可以按自己的想法，再封装多一个通用工具

另创建一个类当作通用工具类，引入JdbcTemplate再调用JdbcTemplate里封装的查询方法

@SuppressWarnings("unchecked")  
public <T> List<T> querySpaDataList(String sql, Object[] params, Class<T> clazz) {  
 printSqlToConsole(sql, params);  
 return jdbcTemplate.query(sql, params, new BeanPropertyRowMapper<T>(clazz));  
}

以上封装的方法增加了查询语句打印，结果类的参数的做了转换，在调用时第三个参数可直接传结果类class

实际调用如下：

List<mapper> list = jdbcTemplate.query(sql, params.toArray(),mapper.class);

### mybatis

#### 注解

@Configuration：同JDBC

@ MapperScan：在配置类上增加实体类扫描注解

@MapperScan(basePackages = {"com.myboot.mapper"}, sqlSessionTemplateRef = "sqlSessionTemplate")

basePackages：数据表映射接口所在路径

sqlSessionTemplateRef：SqlSessionTemplate的bean名称（要与后面讲到的SqlSessionTemplate配置的名称一致）

#### dataSource

同jdbc,但名称要有所区分，

@Bean(name = " dataSource2")

#### SqlSessionFactory

这一步关键的生成一个配置好数据源和XML文件路径的SqlSessionFactory

1. 重新命名：在方法前面增加@Bean

@Bean(name = "sqlSessionFactory")

1. 定义该方法时要传入带了别名的数据源为参数

public SqlSessionFactory sqlSessionFactory(@Qualifier("dataSource2") DataSource dataSource) throws Exception

1. 在方法内生成一个默认的SqlSessionFactory，并把数据源set进来

SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  
bean.setDataSource(dataSource);

1. 添加XML目录

ResourcePatternResolver resolver = new PathMatchingResourcePatternResolver();

bean.setMapperLocations(resolver.getResources("classpath:mappings/\*.xml"));

1. 记得关键语句要捕抓异常，不然intellij IDEA会给你一个错误提示

#### SqlSessionTemplate

这一步关键的生成一个配置好SqlSessionFactory的SqlSessionTemplate

1. 重新命名：在方法前面增加@Bean

@Bean(name = "sqlSessionTemplate")

1. 定义该方法时要传入带了别名的SqlSessionFactory为参数（要与上一步SqlSessionFactory配置的名称一致）

public SqlSessionTemplate sqlSessionTemplate(@Qualifier("sqlSessionFactory") SqlSessionFactory sqlSessionFactory) throws Exception

1. 在方法内生成一个默认的SqlSessionTemplate，并把SqlSessionFactory set进来

然后返回这个template

SqlSessionTemplate template = new SqlSessionTemplate(sqlSessionFactory);

#### DataSourceTransactionManager

如果你对数据源的操作只有查询，没有增加、删除、修改的操作，那这一步可以不需要

这一步的配置与SqlSessionFactory的配置也有相同之处，那就是生成一个配置好数据源DataSourceTransactionManager

1. 重新命名：在方法前面增加@Bean

@Bean(name = "transactionManager")

1. 定义该方法时要传入带了别名的数据源为参数

public DataSourceTransactionManager transactionManager(@Qualifier("dataSource2") DataSource dataSource) throws Exception

1. 在方法内生成一个默认的DataSourceTransactionManager，并把数据源set进来

并返回DataSourceTransactionManager

return new DataSourceTransactionManager(dataSource);

##### 使用

本项目中每一个数据表对应一个xml文件，每一个xml文件对应一个数据表映射接口，xml文件和对应的接口分别放到两个不同的目录下

xml文件："classpath:mappings/\*.xml"

接口："com.myboot.mapper"

使用时引入接口，通过接口调用xml里的sql语句执行数据库操作

1. xml文件中必须定义所对应的数据表映射接口的全路径

<mapper namespace="com.myboot.mapper.Table1Mapper " >

如定义一个查询方法（resultMap=" Table1"，这个要在xml中先定义一个名为Table1的resultMap）

<select id="selectByDate" resultMap=" Table1" parameterType="java.util.HashMap">  
 SELECT \*  
 FROM ` Table1`  
 WHERE 1=1  
 <if test="startDate != null and startDate != ''">  
 and stat\_date &gt;= #{startDate}  
 </if>  
 <if test="endDate != null and endDate != ''">  
 and stat\_date &lt;= #{endDate}  
 </if>  
 ORDER BY stat\_date DESC  
</select>

1. 定义接口

注解：数据访问Dao注解

@Mapper  
@Repository("Table1Mapper ")

Xml的接口（要先写一个Table1Stat，Table1的实体类）

List< Table1Stat> selectByDate(@Param("startDate") String startDate,  
 @Param("endDate") String endDate);

1. 调用接口

引入接口

@Autowired  
private Table1Mapper table1Mapper;

通过table1Mapper调用

table1Mapper.selectByDate(startDate, endDate);

### hibernate

#### 注解

@Configuration：同JDBC

@EnableTransactionManagement，事务管理

@ EnableJpaRepositories：在配置类上增加实体类扫描注解

@EnableJpaRepositories(entityManagerFactoryRef = "girlEntityManagerFactory", transactionManagerRef = "girlTransactionManager", basePackages = "com.myboot.repository")

basePackages：数据表映射接口所在路径

entityManagerFactoryRef：entityManagerFactory的bean名称（要与后面讲到的LocalContainerEntityManagerFactoryBean配置的名称一致）

transactionManagerRef：事务管理的bean名称（要与后面讲到的PlatformTransactionManager配置的名称一致）

#### dataSource

同jdbc,但名称要有所区分，

@Bean(name = " dataSource3")

#### LocalContainerEntityManagerFactoryBean

这一步关键通过EntityManagerFactoryBuilder生成一个配置好数据源、实体类等属性的EntityManagerFactory

1. 重新命名：在方法前面增加@Bean

@Bean(name = "girlEntityManagerFactory")

1. 定义该方法时要传入带了别名的数据源和EntityManagerFactoryBuilder为参数

public LocalContainerEntityManagerFactoryBean girlEntityManagerFactory (@Qualifier("dataSource3") DataSource dataSource,EntityManagerFactoryBuilder builder)

1. 在方法内通过EntityManagerFactoryBuilder 把属性set进来

然后返回这个LocalContainerEntityManagerFactoryBean

return builder  
 .dataSource(dataSource)  
 .properties(getVendorProperties())  
 .packages("com.myboot.entity")  
 .persistenceUnit("primaryPersistenceUnit")  
 .build();

getVendorProperties()这个方法要先定义好

@Autowired  
private JpaProperties jpaProperties;

private Map<String, Object> getVendorProperties() {  
 return jpaProperties.getHibernateProperties(new HibernateSettings());  
}

#### PlatformTransactionManager

如果你对数据源的操作只有查询，没有增加、删除、修改的操作，那这一步可以不需要

这一步的配置与SqlSessionFactory的配置也有相同之处，那就是生成一个配置好数据源DataSourceTransactionManager

1. 重新命名：在方法前面增加@Bean

@Bean(name = "girlTransactionManager")

1. 定义该方法时要传入带了别名的数据源为参数

public PlatformTransactionManager girlTransactionManager(@Qualifier("dataSource3")DataSource dataSource, EntityManagerFactoryBuilder builder)

1. 在方法内生成一个默认的DataSourceTransactionManager，并把数据源set进来

并返回TransactionManager

return new JpaTransactionManager(girlEntityManagerFactory(dataSource, builder).getObject());

##### 使用

本项目中每一个数据表对应一个实体类，每一个实体类对应一个数据表映射接口，实体类和对应的接口分别放到两个不同的目录下

实体类："com.myboot.entity"

接口："com.myboot.repository"

使用时引入接口，通过接口调用xml里的sql语句执行数据库操作

1. 实体类中必须定义所对应的数据表，并增加实体类注解@Entity

@Entity  
@Table(name = "girl")

1. 定义接口(其中Girl就是实体类，JpaRepository是spring boot框架提供的数据库访问接口)

public interface GirlDao extends JpaRepository<Girl, Integer>

1. 数据库执行接口

@Query("select g from Girl g where g.createTime　>= startDate and g.createTime < endDate ")

List< Girl > selectByDate(String startDate, String endDate);

1. 调用接口

引入接口

@Autowired  
GirlDao girlDao;

通过girlDao调用

girlDao.selectByDate(startDate, endDate);

girlDao.findAll();//JpaRepository接口自带的查询方法

# RocketMq

## 使用

定义Event，继承ApplicationEvent抽象类，实现Serializable

定义Listener，实现ApplicationListener接口中的方法onApplicationEvent(E event);

## 例子：

spa\_manger：提供者，发送日志send message topic ，发送成功日志send message suc

RocketMq-Console-Ng：消息管理平台/消息队列，按topic，和发送时间查找消息

push\_service：消费者，执行发送

实现不同服务之间的异步通信 可以订阅？

## 其他：

控制台rocketmq-console-ng

<https://blog.csdn.net/jayjjb/article/details/72674738>

使用

<https://blog.csdn.net/sunliyang1992/article/details/80257908>

<https://blog.csdn.net/chen19920219/article/details/79317692>

rocketmq如何使用：

1.下载源码包，进行初始化，编译，这个过程很简单基本按照官网的步骤就行了。

2.启动，这个时候我们要了解rocketmq的运行机制，首先rocketmq也分为注册中心服务和消息的发送服务，分别为namesrv和broker

Namesrv：用于服务中心的管理，监控broker的运行情况，同步topic客服端的心跳检测等

Broker：进行topic同步信息，在发给订阅了topic的客服端

推送方式：客服端pull主动拉去，服务端push主动发送，namesrv可以进行分布式，但是没有主从，所有的topic都在nanmesrv上面都有存储，broker可以搭建主从，通过commitLog的Offset进行同步，主能进行读写，从只能进行读

启动namesrv命令：./mqnamesrv -n IP:9876 &

启动broker命令：./mqbroker -c 配置文件 -n ip:9876 autoCreateTopicEnable=true

# 短信供应商

# 命令

## MySQL

Show full processList：查看链接进程

EXPLAIN：选择更好的索引和写出更优化的查询语句

## 在日志文件内查找

本地日志目录：C:\Users\Administrator\.IntelliJIdea2018.1\system\tomcat\Unnamed\_local-spa-manager\_2\logs

Less 日志文件名：

查找内容按/：

n：向后找下一处， N： 向上找上一处

g：日志文件开始处， G： 日志文件结束处

按Ctrl+B：屏幕往后移动一页。[常用]

按Ctrl+F：屏幕往前移动一页。[常用]

## 在日志文件所在目录下查找

找内容：grep “查找的内容 ” .log文件：在某份日志文件里查找内容

找文件：find –name “\*.2018-05-1[4,5,6]\*” ：在目录下按日志文件名模糊查找文件

找运行中的应用/进程/内容：ps -ef | grep 进程名/内容：查看服务或者进程是否在运行