1.postgresql返回主键自增

```
<insert id="insertByUser"
parameterType="com.precision.common.entity.basicandauth.MyUser">
        <selectKey resultType="java.lang.Integer" order="AFTER"
keyProperty="id">
            SELECT currval('basic_user_id_seq')
        </selectKey>
        insert into public.basic_user (user_name, pass_word,
        sex, img_url, birthday,
        phone, email, status,
        create_time, update_time)
        values (#{userName,jdbcType=VARCHAR}, #{passWord,jdbcType=VARCHAR},
        #{sex,jdbcType=SMALLINT}, #{imgUrl,jdbcType=VARCHAR}, #
{birthday,jdbcType=DATE},
        #{phone,jdbcType=VARCHAR}, #{email,jdbcType=VARCHAR}, #
{status,jdbcType=SMALLINT},
        #{createTime,jdbcType=TIMESTAMP}, #{updateTime,jdbcType=TIMESTAMP})
    </insert>
```

注意

postgre设置主键自增时，直接在创建表时，将字段设置为serial，会自动生成序列

使用`SELECT currval('basic_user_id_seq')`，不是`nextval('basic_user_id_seq'::regclass)`

不然主键会每次自增两个1 3 5 7 9

2.字段列表中的“id”列不明确

一般出现在检查联合查询时，两张表同时有id字段，不能直接指定返回id，应该具体到那张表的那个id。

```
<sql id="Base_Column_List">
        p.id, parent_id, method, zuul_prefix, service_prefix, uri, name
    </sql>

    <select id="findByRoleId" parameterType="java.lang.Integer"
resultMap="BaseResultMap">
        select
        <include refid="Base_Column_List" />
        from public.basic_role_permission rp,public.basic_permission p
        where rp.permission_id = p.id and rp.role_id = #
{roleId,jdbcType=INTEGER}
    </select>
```

3.不要在sql中写*

防止sql注入，不要在mapper映射sql时使用select *，应该使用resultMap指定

```
<resultMap id="BaseResultMap"
type="com.precision.common.entity.basicandauth.Permission">
        <id column="id" jdbcType="INTEGER" property="id" />
        <result column="parent_id" jdbcType="INTEGER" property="parentId" />
        <result column="method" jdbcType="VARCHAR" property="method" />
        <result column="zuul_prefix" jdbcType="VARCHAR" property="zuulPrefix" />
        <result column="service_prefix" jdbcType="VARCHAR"
property="servicePrefix" />
        <result column="uri" jdbcType="VARCHAR" property="uri" />
        <result column="name" jdbcType="VARCHAR" property="name" />
    </resultMap>
```

4.加盐密码比较

BCryptPasswordEncoder 判断密码是否相同

加密

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
BCryptPasswordEncoder encode = new BCryptPasswordEncoder();
encode.encode(password);
```

比较

```
matches(CharSequence rawPassword, String encodedPassword)
```

需要通过自带的方法 matches 将未经过加密的密码和已经过加密的密码传进去进行判断，返回布尔值。

举例

```
public class BCryptPasswordEncoderTest {
    public static void main(String[] args) {
        String pass = "admin";
        BCryptPasswordEncoder bcryptPasswordEncoder = new
BCryptPasswordEncoder();
        String hashPass = bcryptPasswordEncoder.encode(pass);
        System.out.println(hashPass);

        boolean flag = bcryptPasswordEncoder.matches("admin",hashPass);
        System.out.println(flag);
    }}
```

可以看到，每次输出的hashPass 都不一样，但是最终的flag都为 true,即匹配成功。

查看代码，可以看到，其实每次的随机盐，都保存在hashPass中。在进行matchs进行比较时，调用 BCrypt 的String hashpw(String password, String salt)方法。两个参数即"admin"和 hashPass。

```
//******BCrypt.java******salt即取出要比较的DB中的密码*******
real_salt = salt.substring(off + 3, off + 25);
try {
// ************************************************
    passwordb = (password + (minor >= 'a' ? "\000" : "")).getBytes("UTF-8");
}
catch (UnsupportedEncodingException uee) {}
saltb = decode_base64(real_salt, BCRYPT_SALT_LEN);
B = new BCrypt();
hashed = B.crypt_raw(passwordb, saltb, rounds);
```

假定一次hashPass为：$2a$10$AxafsyVqK51p.s9WAEYWYeIY9TKEoG83LTEOSB3KUkoLtGsBKhCwe

随机盐即为 AxafsyVqK51p.s9WAEYWYe（salt = BCrypt.*gensalt*();中有描述），可见，随机盐（AxafsyVqK51p.s9WAEYWYe），会在比较的时候，重新被取出。

即，加密的hashPass中，前部分已经包含了盐信息。

5.Spring Boot 2.2.x Junit找不到包

使用 maven 创建了一个 parent 项目 A，其 pom.xml 继承 parent 为 spring-boot-starter-parent 2.1.10。

然后创建 module 项目 B，使用 spring initializr 构建项目，用的是 IDEA，当时没有选 Spring Boot 版本，结果默认使用的是 2.2.1。 创建成功之后的pom.xml如下 Spring Boot 2.2 之后的 pom.xml。

修改项目 B 的 pom 的 parent 为 A，结果测试类报错，找不到 org.junit.jupiter.api.Test

spring boot 2.2 之前使用的是 Junit4 而后续的使用的是Junit5，导致缺少包。

解决方案:
将父工程 A 的 parent 升级为 spring-boot-starter-parent 2.2.1，如果使用了依赖管理 dependencyManagement，需要把里面的 spring-boot-starter-test 版本号改为 与 parent 对应的 2.2.1。

当然，也可以直接指定 module工程B 的 spring-boot-starter-test 版本号改为 与 parent 对应的 2.2.1

2.2之前

```
package com.example.demo1;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class Demo1ApplicationTests {

    @Test
    public void contextLoads() {
    }
```

```
}
```

2.2之后

```
package com.example.demo;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class DemoApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

2.2之前pom

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.10.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

2.2之后pom

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.1.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

官方文档： 25.Testing

Spring Boot provides a number of utilities and annotations to help when testing your application. Test support is provided by two modules: `spring-boot-test` contains core items, and `spring-boot-test-autoconfigure` supports auto-configuration for tests.

Most developers use the `spring-boot-starter-test` "Starter", which imports both Spring Boot test modules as well as JUnit Jupiter, AssertJ, Hamcrest, and a number of other useful libraries.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

25.测试

Spring Boot提供了许多实用程序和注释，可以在测试应用程序时提供帮助。测试支持由两个模块提供：spring-boot-test包含核心项，并spring-boot-test-autoconfigure支持测试的自动配置。大多数开发人员都使用spring-boot-starter-test"入门程序"，该程序同时导入Spring Boot测试模块以及JUnit Jupiter，AssertJ，Hamcrest和许多其他有用的库。

启动程序还带来了老式引擎，因此您可以运行JUnit 4和JUnit 5测试。如果已将测试迁移到JUnit 5，则应排除对JUnit 4的支持，如以下示例所示：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

25.3。测试Spring Boot应用程序

Spring Boot应用程序是Spring ApplicationContext，因此除了对普通Spring上下文进行正常测试以外，无需执行任何其他特殊操作即可对其进行测试。

默认情况下，Spring Boot的外部属性，日志记录和其他功能仅在SpringApplication用于创建时才安装在上下文中。
Spring Boot提供了一个@SpringBootTest注释，spring-test @ContextConfiguration当您需要Spring Boot功能时，它可以用作标准注释的替代。注释通过创建ApplicationContext在测试中使用过的来SpringApplication起作用。除了@SpringBootTest提供许多其他注释外，还用于测试应用程序的更多特定部分。

如果使用的是JUnit 4，请不要忘记也将其添加@RunWith(SpringRunner.class)到测试中，否则注释将被忽略。如果您使用的是JUnit 5，则无需添加等价项@ExtendWith(SpringExtension.class)，@SpringBootTest并且其他@...Test注释已经在其中进行了注释。

6.Hbase关闭一直等待

今天关闭HBase时，输入stop-hbase.sh一直处于等待状态

解决方法：

先输入：hbase-daemon.sh stop master

再输入：stop-hbase.sh就可以关闭HBase集群了。