## Springboot传参方式:

注解

@RequestParam

这个注解用来绑定单个请求数据，既可以是url中的参数，也可以是表单提交的参数和上传的文件。

不能处理JSON格式的请求

@PathVariable

这个注解可以将URL中的占位符参数绑定到控制器处理方法的入参

@RequestBody

这个注解是传JSON对象用的，不能通过Form表单、URL传参。

Form-data

普通表单

传送form表单数据，可以不用注解，直接传参，参数名字要一样。

## 数据库表记录操作：

删除记录：删除选中数据记录，保留所有索引的分配页

清空表：清除表中所有数据记录，清空表命令会保留所有索引的分配页

截断表：truncate 命令则仅仅删除了表中所有的数据行。表的结构和所有的索引仍然继续存在，直到

你输入删除表的命令（如上所述）。绑定到列上的规则、默认值、约束仍然继续绑定，并且触发器也

仍然起作用。截断表命令还会回收所有索引的分配页

删除表：表删除包括表的定义和关联对象（规则、索引、约、触发器、主键，等）。很明显，一旦表

被删除，那么表中包含的所有的数据行都会被一同删除

## hibernate的配置属性:

spring.jpa.properties.hibernate.hbm2ddl.auto是hibernate的配置属性，其主要作用是：自动创建、更新、验证数据库表结构。

该参数的几种配置如下：

create：每次加载hibernate时都会删除上一次的生成的表，然后根据你的model类再重新来生成新表，哪怕两次没有任何改变也要这样执行，这就是导致数据库表数据丢失的一个重要原因。

create-drop：每次加载hibernate时根据model类生成表，但是sessionFactory一关闭,表就自动删除。

update：最常用的属性，第一次加载hibernate时根据model类会自动建立起表的结构（前提是先建立好数据库），以后加载hibernate时根据model类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行。要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等应用第一次运行起来后才会。

validate：每次加载hibernate时，验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值。

## spring中遇到的applicationContext.xml (系统找不到指定的文件。)

错误提示，如下：

Exception in thread "main" org.springframework.beans.factory.BeanDefinitionStoreException: IOException parsing XML document from file [D:\eclipse\_Indigo\workspace\spring\_PointcutAdviser\applicationContext.xml]; nested exception is java.io.FileNotFoundException: applicationContext.xml (系统找不到指定的文件。)  
Caused by: java.io.FileNotFoundException: applicationContext.xml (系统找不到指定的文件。)  
 at java.io.FileInputStream.open(Native Method)  
 at java.io.FileInputStream.<init>(Unknown Source)  
 at org.springframework.core.io.FileSystemResource.getInputStream(FileSystemResource.java:85)  
 at org.springframework.beans.factory.xml.XmlBeanDefinitionReader.loadBeanDefinitions(XmlBeanDefinitionReader.java:307)  
 at org.springframework.beans.factory.xml.XmlBeanDefinitionReader.loadBeanDefinitions(XmlBeanDefinitionReader.java:290)  
 at org.springframework.beans.factory.support.AbstractBeanDefinitionReader.loadBeanDefinitions(AbstractBeanDefinitionReader.java:131)  
 at org.springframework.beans.factory.support.AbstractBeanDefinitionReader.loadBeanDefinitions(AbstractBeanDefinitionReader.java:147)  
 at org.springframework.beans.factory.support.AbstractBeanDefinitionReader.loadBeanDefinitions(AbstractBeanDefinitionReader.java:173)  
 at org.springframework.context.support.AbstractXmlApplicationContext.loadBeanDefinitions(AbstractXmlApplicationContext.java:112)  
 at org.springframework.context.support.AbstractXmlApplicationContext.loadBeanDefinitions(AbstractXmlApplicationContext.java:79)  
 at org.springframework.context.support.AbstractRefreshableApplicationContext.refreshBeanFactory(AbstractRefreshableApplicationContext.java:101)  
 at org.springframework.context.support.AbstractApplicationContext.obtainFreshBeanFactory(AbstractApplicationContext.java:394)  
 at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:324)  
 at org.springframework.context.support.FileSystemXmlApplicationContext.<init>(FileSystemXmlApplicationContext.java:124)  
 at org.springframework.context.support.FileSystemXmlApplicationContext.<init>(FileSystemXmlApplicationContext.java:68)  
 at test.main(test.java:12)

找到test.java:12，该语句为：ApplicationContext ac=new FileSystemXmlApplicationContext("applicationContext.xml");

解决办法，有两种，如下：

第一种：通过FileSystemXmlApplicationContext读取配置文件时，需要使用绝对路径。因此，只需要将上述语句改为如下情形：

ApplicationContext ac=new FileSystemXmlApplicationContext("D:\\eclipse\_Indigo\\workspace\\spring\_PointcutAdviser\\src\\applicationContext.xml");

这里要注意路径的写法，在电脑中显示的路径是“D:\eclipse\_Indigo\workspace\spring\_PointcutAdviser\src”，这里必须改写，否则无效。

第二种： 采用这条语句ApplicationContext ac=new ClassPathXmlApplicationContext("applicationContext.xml");

ClassPathXmlApplicationContext是通过相对路径来读取配置文件的。

## [Spring中引入其他配置文件](https://www.cnblogs.com/LiuChunfu/p/5605473.html)

1. 引入其他 模块XML

在Spring的配置文件，有时候为了分模块的更加清晰的进行相关实体类的配置。

比如现在有一个job-timer.xml的配置

在Spring的整体的配置文件中使用 <import resource="classpath\*:/spring/job-timer.xml" />引入。

1. 引入properties文件

方法1：

<!--引入数据库配置信息 -->

<context:property-placeholder location="classpath\*:properties/db.properties" />

方法2：

情况1

配置一个：

<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="location" value="classpath\*:db/jdbc.properties" />

</bean>

情况2

配置多个：

<bean id="propertyConfigure" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="locations">

<list>

<value>classpath:/opt/demo/config/demo-db.properties</value>

<value>classpath:/opt/demo/config/demo-db2.properties</value>

</list>

</property>

</bean>

这些properties中就是key-value的键值对，使用的时候可以使用${xxx} 获取。

## Hibernate映射文件(Entity.hbm.xml)

hbm全称:Hibernate Mapping(Hibernate映射)

<?xml version="1.0"?>

<!--头文件-->

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<!--这里的package属性写的是实体类包-->

<hibernate-mapping package="com.me.relation.entity">

<!--class标签中的name为实体对应的类名,table标签为创建的表名,如果不写默认就是类名首字母小写(student)-->

<class name="Student" table="student">

<!--

以下所有标签,name为对应实体中属性名,column属性可有可无,有的话表示自己定义在表中的字段名,不写的话默认与name一致(与实体中的属性名一致)

-->

<!--必须有的主键标签-->

<id name="id">

<!--主键生成策略,这里表示自增长-->

<generator class="identity"></generator>

</id>

<property name="joinTime" column="join\_time"></property>

<property name="name"></property>

<property name="sex"></property>

<property name="age"></property>

</class>

</hibernate-mapping>

**主键生成策略详解:**

increment， 数字类型的自增。（用于单线程，或线程安全的）   
原理:当前表最大的id值,+1,然后插入时加进去

identity，数字类型的自增。依赖于数据库底层的实现。   
原理:数据库底层主键的auto\_increament

uuid，生成32位的16进制的一串没有任何意义的唯一编码。   
原理:框架生成,然后插入时加进去

guid ,用书mySql或者Sql service数据中生成字符串类型的唯一标识。   
原理:用数据库自带的uuid函数

native: 依赖于数据库底层的实现。数据库默认的主键生成策略是什么，他就是什么。   
原理:数据库默认(auto\_increament)

assigned：手动分配。   
原理:手动加

foreign : 使用外键做主键。（比较少见，一般只会一对一当中出现）   
原理:获取外键的值,插入到表中

## eclipse 执行springboot App.java报Unable to start embedded container

main] ERROR org.springframework.boot.SpringApplication - Application startup failed  
org.springframework.context.ApplicationContextException: Unable to start embedded container; nested exception is org.springframework.context.ApplicationContextException: Unable to start EmbeddedWebApplicationContext due to missing EmbeddedServletContainerFactory bean.  
    at org.springframework.boot.context.embedded.EmbeddedWebApplicationContext.onRefresh(EmbeddedWebApplicationContext.java:137)  
    at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:536)  
    at org.springframework.boot.context.embedded.EmbeddedWebApplicationContext.refresh(EmbeddedWebApplicationContext.java:122)  
    at org.springframework.boot.SpringApplication.refresh(SpringApplication.java:737)  
    at org.springframework.boot.SpringApplication.refreshContext(SpringApplication.java:370)  
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:314)  
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:1162)  
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:1151)  
    at com.dispatchCenter.main.DispatchCenterApplication.main(DispatchCenterApplication.java:9)  
Caused by: org.springframework.context.ApplicationContextException: Unable to start EmbeddedWebApplicationContext due to missing EmbeddedServletContainerFactory bean.  
    at org.springframework.boot.context.embedded.EmbeddedWebApplicationContext.getEmbeddedServletContainerFactory(EmbeddedWebApplicationContext.java:189)  
    at org.springframework.boot.context.embedded.EmbeddedWebApplicationContext.createEmbeddedServletContainer(EmbeddedWebApplicationContext.java:162)  
    at org.springframework.boot.context.embedded.EmbeddedWebApplicationContext.onRefresh(EmbeddedWebApplicationContext.java:134)

... 8 common frames omitted

**原因：启动类中少写了注解**  
    @EnableEurekaServe(不用)

@SpringBootApplication

## Java连接Mysql数据库警告：Establishing SSL connection without server's identity verification is not recommend

Java使用mysql-jdbc连接MySQL出现如下警告：

Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.

原因是MySQL在高版本需要指明是否进行SSL连接。解决方案如下：

在mysql连接字符串url中加入ssl=true或者false即可，如下所示。

url=jdbc:mysql://127.0.0.1:3306/framework?characterEncoding=utf8&useSSL=true

## A query was run and no Result Maps were found

使用mybatis时出现异常问题：

有如下的错误

Error querying database.  Cause: org.apache.ibatis.executor.ExecutorException: A query was run and no Result Maps were found for the Mapped Statement

（后面内容省略）

网上说是oracle不允许插入空值。

但是我没有进行插入操作的，只是进行了查询。

后来看看mybatis的配置文件，漏了一项：resultType属性值

原来的配置是：<select id="countSiteInv"  parameterType="Map">

修改之后的配置<select id="countSiteInv" resultType="CamelCaseKeyMap" parameterType="Map">

mybatis中不管什么查询，都必须返回resultType或者resultMap的值，否则就会报错的。

## Jsp的九大对象，七大动作，三大指令

jsp九大内置对象:  
1>out 向客户端输出数据,字节流.如out.print(" dgaweyr");  
   
2>request 接收客户端的http请求.  
String getParameter(String name):得到表单参数名name的值.  
String[] getParameterValues(String name):(得到String[]复选框时常用).  
setAttribute(String name,Object obj):设置属性名为name,属性值为obj.  
getAttribute(String name);得到属性值.  
   
3>response:封装jsp产生的回应,然后发送到客户端以响应客户的请求.重定向跳转任意界面.(服务器跳转)  
addCookie(Cookie cookie):  
sendRedirect("/wel.jsp"):跳转到指定页面  
   
4>session:用于保存用户信息,跟踪用户行为,当前打开的浏览器内,多个页面共享数据. session对象指的是客户端与服务器的一次会话,从客户连到服务器的一个WebApplication开始,直到客户端与服务器断开连接为止.它是HttpSession类的实例.  
setAttribute(String name,Object obj):设置属性名为name,属性值为obj.  
getAttribute(String name):得到属性值.  
   
5>application对象:实现了用户间数据的共享,可存放全局变量.它开始于服务器的启动,直到服务器的关闭,在此期间,此对象将一直存在;这样在用户的前后连接或不同用户之间的连接中,可以对此对象的同一属性进行操作;在任何地方对此对象属性的操作,都将影响到其他用户对此的访问.服务器的启动和关闭决定了application对象的生命.它是ServletContext类的实例.  
   
session,application,request的区别:  
一个项目中session尽量少用几个,因为过多的session会影响程序的执行效率.它主要用于保存登录信息(用户信息,权限,资源)即频繁使用的信息.  
application: 用于多个浏览器之间共享数据,多个用户共享该对象,可以做计数器.它的用法与session完全一样.  
数据范围:  
application(服务器关闭时失效)>session(浏览器关闭时失效)>request(只能用于两个跳转页面之间)  
   
6>page对象代表jsp这个实体本身,即当前页面有效.相当于java中的this.  
数据范围:page<session<application  
   
7>.exception:代表运行时的异常.  
在会发生异常的页面加入指令:<%@ page errorPage="处理错误的页面.jsp"%>  
在处理异常的页面写入:<%@ page isErrorPage="true"%>  
   
8>.pageContext对象 pageContext对象提供了对JSP页面内所有的对象及名字空间的访问,也就是说他可以访问到本页所在的SESSION,也可以取本页面所在的application的某一属性值,他相当于页面中所有功能的集大成者,它的本类名也叫pageContext.  
   
9>.config jsp对应的servlet的配置,可以得到web.xml中的初使化参数.  
   
jsp七大动作:  
一:include 动态包含(分别编译):  
用jsp:include动作实现<jsp: include page="included.jsp" flush="true" />  
它总是会检查所含文件中的变化，适合用于包含动态页面，并且可以带参数。flush属性: 用true ，表示页面可刷新。默认为false;  
   
二:useBean动作(jsp页面使用javaBean的第二种方式):  
<jsp:useBean id="对象名" class="包名.类名" scope="作用范围(request/page/application/session)"/>  
作用域默认为page(本页面有效).  
三:getProperty动作(name为useBean动作中的id).  
从对象中取出属性值：<jsp:getProperty name="javaBean对象" property="javaBean对象属性名" />  
四:setProperty动作(name为useBean动作中的id):  
为对象设置属性值：<jsp:setProperty name="javaBean对象" property="javaBean对象属性名" value=http://www.hake.cc/kf/201109/"值"/>  
为对象设置属性值:<jsp:setProperty property="javaBean对象属性名" name="javaBean对象" param="username"/>  
(param="username" 相当于 value=http://www.hake.cc/kf/201109/<%=request.getParameter("username")%>)  
   
五:param动作(传递参数)):  
到达跳转页面可以通过 request.getParameter(“参数名”)方式取出参数值  
<jsp:include page="转向页面的url" >  
           <jsp:param   name="参数名1" value=http://www.hake.cc/kf/201109/"参数值1">

<jsp:param   name="参数名2" value=http://www.hake.cc/kf/201109/"参数值2">  
           ...........  
 </jsp:include>  
或:  
<jsp:forward page="转向页面的url" >  
           <jsp:param   name="参数名1" value=http://www.hake.cc/kf/201109/"参数值1">  
           <jsp:param   name="参数名2" value=http://www.hake.cc/kf/201109/"参数值2">  
           ...........  
</jsp:forward>  
   
六:forward动作:  
跳转页面:<jsp:forward page="login.jsp" />  
   
七:plugin动作:<jsp:plugin>:用于指定在客户端运行的插件  
   
JSP三大指令之:  
一:page指令:  
1.指定页面编码.例:  
<%@ page language="java" contentType="text/html;charset=gbk" pageEncoding="gbk" %>  
2.导入包,例:  
<%@ page import="java.util.\*,java.text.\*" %>.  
二:include 指令  
静态包含(统一编译):<%@ include file="included.jsp"%>  
   
三：taglib  
   
三．Jsp中的静态包含与动态包含  
动态include(<jsp: include page="included.jsp"/>)  
静态include(<%@ include file="included.jsp"%>)  
1. 静态include的结果是把其他jsp引入当前jsp,两者合为一体,可以达到数据的共享即可以说是统一编译的,而  
动态include的结构是两者独立的,直到输出时才合并即为分别编译的.  
2.动态include的jsp文件独立性很强,是一个单独的jsp文件,需要使用的对象,页面设置,都由自己创建,而静态include纯粹是把代码写在外面的一种共享方法,所有的变量都是可以和include它的主文件共享,两者高度紧密结合,不能 有变量同名的冲突.而页面设置也可以借用主文件的.  
3.动态包含总是检查被包含页面的变化,静态包含不一定检查被包含页面的变化.  
4.动态包含可带参数,静态包含不能带参数.如(<jsp: include page="included.jsp">放入参数</jsp:include>);

## Mysql索引最左匹配原则

组合索引可以这样理解，比如（a,b,c），abc都是排好序的，在任意一段a的下面b都是排好序的，任何一段b下面c都是排好序的

(0) select \* from mytable where a=3 and b=5 and c=4;

abc三个索引都在where条件里面用到了，而且都发挥了作用

(1) select \* from mytable where c=4 and b=6 and a=3;

这条语句列出来只想说明 mysql没有那么笨，where里面的条件顺序在查询之前会被mysql自动优化，效果跟上一句一样

(2) select \* from mytable where a=3 and c=7;

a用到索引，b没有用，所以c是没有用到索引效果的

(3) select \* from mytable where a=3 and b>7 and c=3;

a用到了，b也用到了，c没有用到，这个地方b是范围值，也算断点，只不过自身用到了索引

(4) select \* from mytable where b=3 and c=4;

因为a索引没有使用，所以这里 bc都没有用上索引效果

(5) select \* from mytable where a>4 and b=7 and c=9;

a用到了 b没有使用，c没有使用

(6) select \* from mytable where a=3 order by b;

a用到了索引，b在结果排序中也用到了索引的效果，前面说了，a下面任意一段的b是排好序的

(7) select \* from mytable where a=3 order by c;

a用到了索引，但是这个地方c没有发挥排序效果，因为中间断点了，使用 explain 可以看到 filesort

(8) select \* from mytable where b=3 order by a;

b没有用到索引，排序中a也没有发挥索引效果