## 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