凌云壮志，‘锋’起云涌

记录每一次感动

数据库多表连接的查询

2018年7月8日 未分类

连接查询是关系数据库中最主要的查询，主要包括 自连接，内连接，外连接和交叉连接。通过连接运算符可以实现多个表查询。连接是关系数据库模型的主要特点，也是区别于其他类型数据库管理系统的一个标志。

内连接的连接查询结果集中仅包含满足条件的行，内连接是SQL Server却省的连接方式，可以把INNERJOIN简写成JOIN，根据所使用比较方式不同，内连接又分为等值连接，，自然连接和不等连接三种；

交叉连接的连接查询结果集中包含两个表中所有行的组合；

外连接的连接查询结果集中即包含那些满足条件的行，还包括其中某个表的全部行，有三种形式的外连接：左外连接，右外连接，全外连接。

一 交叉连接

交叉连接即笛卡尔乘积，是指两个关系中所有元组的任意组合。一般情况下，交叉查询是没有实际意义的。

二 内连接

内连接是一种最常用的连接类型。内连接查询实际上是一种任意条件的查询。使用内连接时，如果两个表的相关字段满足连接条件，就从这两个表中提取数据并组合成新的记录，也就是在内连接查询中，只有满足条件的元组才能出现在结果关系中。

例如：要查询每个已经选课的学生的情况，查询语句为

SELECT*

FROM学生表**INNER JOIN**选课表**ON**学生表.学号=选课表.学号

分类：

根据比较方式分为：

1）等值连接：在连接条件中使用等于号(=)运算符比较被连接列的列值，其查询结果中列出被连接表中的所有列，包括其中的重复列。

2）不等连接：在连接条件使用除等于运算符以外的其它比较运算符比较被连接的列的列值。这些运算符包括>、>=、<=、<、!>、!<和<>。

3）自然连接：在连接条件中使用等于(=)运算符比较被连接列的列值，但它使用选择列表指出查询结果集合中所包括的列，并删除连接表中的重复列。

方法重载与方法重写

2018年7月6日 未分类

一直容易搞混重载与重写的一些区别。

首先是方法重载:

判断方法重载的依据:

1 必须是在同一个类中。

2 方法名相同。

3 方法参数的个数，顺序或类型不同。

4 与方法的修饰符或返回值没有关系

也就是说方法重载的前提是 方法名相同，参数列表不同 。

与方法的修饰符或返回值没有关系。

```
public void f() {

}

public int f() {
return 3;
}
```

这种情况是通不过编译的。

方法重写是重写父类方法:

方法名称，参数列表，返回类型完全相同。

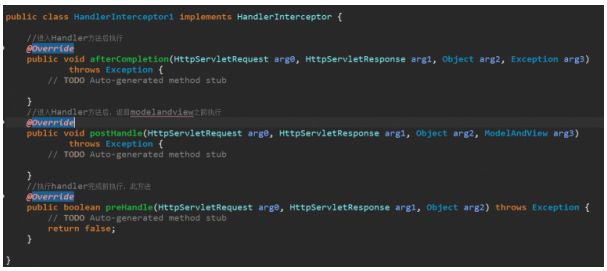
被重写的方法不能有更严格的访问权限。

拦截器

2018年6月30日 SpringMvc

定义拦截器

实现HandlerInterceptor接口



```
public class HandlerInterceptor1 implements HandlerInterceptor {

    //进入Handler方法后执行
    @Override
    public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)
            throws Exception {
        // TODO Auto-generated method stub

    }
    //进入Handler方法后，返回modelandview之前执行
    @Override
    public void postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)
            throws Exception {
        // TODO Auto-generated method stub

    }
    //执行handler完成前执行，此方法
    @Override
    public boolean preHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2) throws Exception {
        // TODO Auto-generated method stub
        return false;
    }

}
```

public class HandlerInterceptor1 implements HandlerInterceptor {

//进入Handler方法后执行
@Override
public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)
throws Exception {
// TODO Auto-generated method stub

}
//进入Handler方法后，返回modelandview之前执行
@Override
public void postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)
throws Exception {
// TODO Auto-generated method stub

}
//执行handler完成前执行，此方法
@Override
public boolean preHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2) throws Exception {
// TODO Auto-generated method stub

return false;
}

}

拦截器配置

针对handlermapping进行配置

springmvc拦截器针对HandlerMapping进行拦截配置。 如果在某个HandlerMapping中配置拦截，经过该HandlerMapping映射成功的handler最终使用该拦截器。

(不推荐使用)

<bean class=”org.springframework.web.servlet.Handler.BeanNameUrlHandlerMapping”>

<property name=”interceptors”>

<list>

<ref bean=”handlerInterceptor1″ />

<ref bean=”handlerInterceptor2″ />

</list>

</property>

</bean>

<bean id=”handlerInterceptor1″ class=”com.itcast.filter.HandlerInterceptor1″/>

<bean id=”handlerInterceptor2″ class=”com.itcast.filter.HandlerInterceptor2″/>

针对全局进行配置

springmvc配置类似全局的拦截器,springmvc框架将配置的类似的全局的拦截器注入到每个handlerMapping中。

在springmvc.xml文件中添加

```
<!-- 拦截器 -->
<mvc:interceptors>
   <mvc:interceptor>
      <mvc:mapping path="/**"/>
      <bean calss="com.itcast.filter.HandlerInterceptor1"/>
   </mvc:interceptor>
</mvc:interceptors>
```

<!– 拦截器 –>
<mvc:interceptors>
<mvc:interceptor>
<mvc:mapping path=”/**”/>
<bean calss=”com.itcast.filter.HandlerInterceptor1″/>
</mvc:interceptor>
</mvc:interceptors>

拦截器测试

全局配置两个拦截器

两个拦截器都放行

H1 …preHandle

H2…preHandle

H2…postHandle

H1…postHandle

H2…afterCompletion

H1…afterCompletion

根据测试结果总结!

preHandler方法按顺序执行

postHandler和afterCompletion安拦截器配置的逆向顺序执行

拦截器1放行，拦截器2不放行

H1…preHandle

H2…preHandle

H1…afterCompletion

对拦截器应用

比如:统一日志处理拦截器，需要该拦截器preHandle一定要放行，且将它放在拦截器链接中第一个位置。

比如:登录认证拦截器，放在拦截器链接中第一个位置。权限校验拦截器，放在登录认证拦截器后。(因为登录后才能认证)。

实现登录认证

1 用户请求url

2 拦截器进行拦截校验

如果请求的url是公开地址(无需登录即可访问的url)放行。

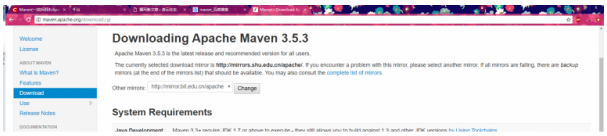
如果用户session不存在跳转到登录页面

如果用户session存在放行，继续操作

MAVEN的使用

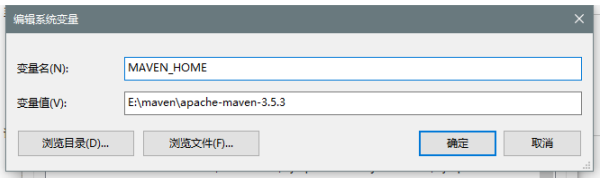
2018年6月13日 未分类

下载:

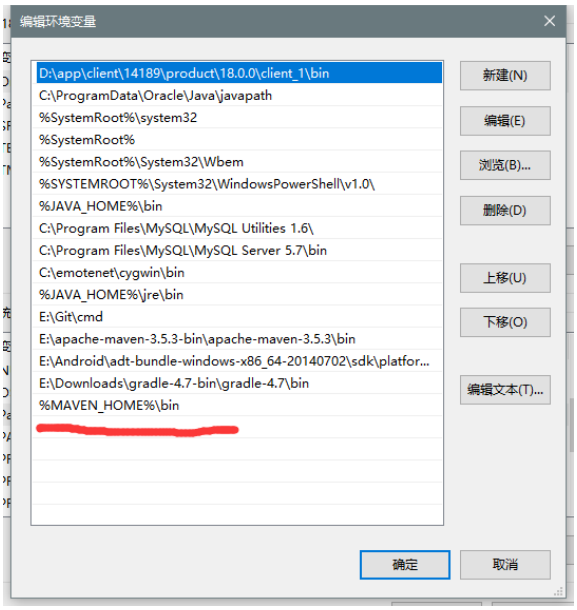


配置环境变量:

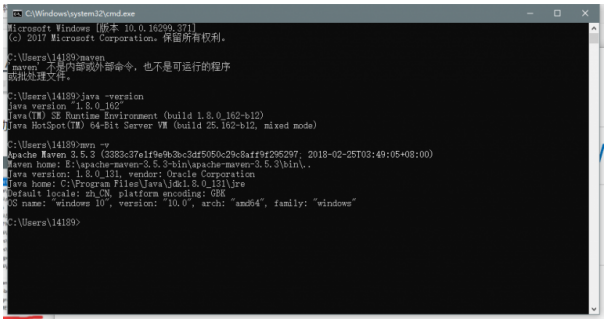
新建



path中添加



检验配置是否成功



配置本地仓库：

conf/setting.xml

```
  | Default: ${user.home}/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  -->
 <localRepository>M:\repository</localRepository>
  <!-- interactiveMode
   | This will determine whether maven prompts you when it needs
   | maven will use a sensible default value, perhaps based on s
```

Springmvc-Validation 检验

2018年6月10日 SpringMvc

在实际开发项目中，为了系统了稳定性，鲁棒性，安全性等等等等。我们需要对数据进行检验剔除，非法的，或格式错误的数据。

一般有两种选择，可以在前端进行校验，或者在后端进行校验。

然而对于安全性较高要求建议在服务器进行检验。所以今天我主要记录分享后端校验的方法。

在后端进行检验，也分多种情况。controller层,service层，和dao层。

控制层:

检验页面请求的参数的合法性，在服务器控制层controller检验，不区分客户端类型(浏览器，手机客户端)。

业务层(service层)：这也是使用最多的，主要校验关键业务参数，仅限于service接口中使用的参数。

持久层:一般不校验。

校验思路:

页面提交请求的参数，请求到controller方法中，使用validation进行校验。如果校验出错，将错误信息展示到页面。

是使用validation检验，要先引入三个JAR包。

配置校验器

将校验器注入到处理器适配器中

举个在Controller中使用校验的栗子!

首先确认你要检验的对象，这里以Items为例

在Items类中

```
public class Items {
    private Integer id;
```

@Size是字符长度检验，在本例中允许商品名称在1~30字符之间。如果违法这个规定的话，将会输出输出错误配置文件当中的信息。

@Null为非空检验

在controller中，对需要进行校验的对象加上@Validated注解，BindingResult 接收校验出错的信息。

分组检验

什么是分组检验呢？为啥需要分组校验？

假设有这样一种需求

一个pojo被多个controller所共用,当不同的controller方法对同一个pojo进行检验。

不同的controller对pojo检验需求是不同的。比如name长度，有的的contoller可以是1~30字符，有controller允许0个字符。

在这种情况下,就需要定义多个分组检验。

其实就是定义不同的java接口，通过接口来区分不同的分组。

public interface ValidGroup1{

//接口中不需要定义任何方法，仅是对不同的校验规则进行分组
}

@Size(min=1,….,group={ValidatedGroup1.class})

在controller使用指定分组的校验

@Validated(value{ValidGroup1.class})

这样不同的controller通过不同的分组，使用不同的检验。

记录一个不愉快的中午

2018年6月8日 未分类

伤心：

心想今天咸鱼了一上午，中午得学点东西，不然就找不到女朋友了。想了一会，就复习下昨天学的图片上传吧。然后信心满满的开工了。

写好表单，添加file。配好controller心想能在controller里输出传进来的文件名就算成功了。结果我的悲惨剧情又一次拉开了帷幕。

一来就给我报nullPingException异常。

难道是属性名和controller的形参名不一致???

检查发现一致呀。

很郁闷。

没办法呀只能百度看看啦

说是要在springmvc配置个什么bean.

我发现我配了啊

那是什么问题呢难道是文件没传进去？？？

那就测下文件传进去了吧

于是把form的提交方式改为get

通过url发现传进去了啊。

肿么办，苍天啊饶了我吧。

继续百度，

说让加这句话那就加呗

加了之后又又又报错了

这又是什么鬼啊

然后百度说要在form里加上

enctype=”multipart/form-data”

那就加呗，加了还是一样啊，心灰意冷了。。

心想不做程序猿了，还是回家卖煎饼吧

回来看到一篇文章说上传文件只能用post请求方式，突然又觉的有戏了。

于是把之前的get请求方式改成post请求方式，发现居然好了，哈哈。

哈哈终于输出来了。。

居然中文又乱码。^ _^，不管了

SpringMvc—参数绑定

2018年6月7日 SpringMvc

参数绑定

参数绑定，简单来说就是客户端发送请求，而请求中包含一些数据，那么这些数据怎么到达Controller？这在实际项目开发中也是用到的最多的，那么SpringMvc的参数绑定是怎么实现的呢?下面我们来详细的讲解。

1.SpringMvc参数绑定

在SpringMvc中，提交请求的数据是通过方法形参来接收的。从客户端请求的key/value数据，经过参数绑定，将key/value数据绑定到Controller的形参上，然后在Controller就可以直接使用该形参。

参数绑定组件

在SpringMvc早期版本使用PropertyEditor(只能将字符串转成java对象)后期使用converter(进行任意类型的转换)。

SpringMvc提供了很多converter(转换器)在特殊情况下需要自定义converter

比如对日期数据绑定需要自定义converter

2.默认支持的类型

SpringMvc有支持的默认参数类型。我们直接在形参上给出这些默认类型的声明，就能直接使用。如下:

1)HttpServletRequest对象

2)HttpServletResponse对象

3)HttpSession对象

4)Model/ModelMap对象

这里我们重点说一下Model/ModelMap,ModelMap是Model接口的一个实现类，作用是将Model数据填充到request域，即使使用Model接口，其内部绑定还是由ModelMap来实现。

3.基本数据类型的绑定

哪些是基本数据类型，我们这里重新总结一下:

1) byte 占用一个字节，取值范围为 -128~127,默认是”\u0000″,标示空

2)short 占用两个字节，取值范围为-32768~32767

3)int 占用四个字节，取值范围为，-2147483648~2147483647

4)long 占用八个字节 ，对long类型变量赋值必须加上”L”或”l”,否则不认为是long型

5)float 占用四个字节，对 float 型进行赋值的时候必须加上“F”或“f”，如果不加，会产生编译错误，因为系统自动将其定义为 double 型变量。double转换为float类型数据会损失精度。float a = 12.23产生编译错误的，float a = 12是正确的

6)double 占用八个字节，对double型变量赋值的时候最好加上”D”或”d”,但加不加不是硬性规定。

7)char 占用两个字节，在定义字符型变量时，要用单括号括起来

8)boolean 只有两个值”true”和”false”,默认值false，不能用0或非0来代替，这点和c语言不同

4.包装数据类型绑定

举个栗子:

只要注意: form 里的name 和Controller的参数名保持一致就可。

如果不想一致怎么办??

@RequestParam(value=”usename”,required=false) String usename_C ;

加上@RequestParam 注解。只要value值和form的name保持一致。后面的属性名可以自定义。

5.pojo(实体类)类型的绑定

举个栗子

只要pojo类中有 form 的name属性值，则会自动完成映射。

这样会有一个新问题如果类似下面的情况，绑定结果会是怎样的呢？？

测试一下:

可见，两个参数都正常的绑定了。

其实从框架设计上这样的结果也是应该的！！！

今天先到这，明天更新

SpringMVC+HibernateValidator，配置在properties文件中的错误信息回显前端页面出现中文乱码

2018年6月6日 SpringMvc

问题描述:

后台在springMVC中使用hibernate-validator做参数校验的时候，用properties文件配置了校验失败的错误信息。发现回显给前端页面的时候中文错误信息显示乱码。

.properties文件内容如下

待校验的属性如下:

前端界面如下，为了检测用户名为空提交:

返回结果如下：

发现中文乱码

解决办法：

先检查ecplise编码

检查发现所有地方均设为UTF-8,无解很头疼，没办法只能接着百度。后来发现

经过潜(搜)心(索)研(谷)究(歌)，发现问题就出现在spring-mvc.xml的配置。看起来貌似没有问题：

```
<!-- 资源文件编码格式 -->
<property name="fileEncodings" value="UTF-8"/>
```

后来搜到一篇提到过乱码，处理方式是Controller中获取错误信息从IOS-8895-1转UTF-8。但【鲁迅眉头一皱，发现事情并不靠谱.jpg】,框架出了这么久，怎么可能还存在这种需要自己手动转码的问题。
再后来搜到一篇靠谱的文章，发现用的是这个配置：

<property name=“defaultEncoding” value=“UTF-8”/>

于是加上这个配置就OK了

成功后的界面