# 1 实战

## 1.1 文件上传

### 1.1.1 spring mvc 单文件上传

jsp文件如下：

|  |
| --- |
| <form action="服务器后台 url " method="post"  enctype="multipart/form-data">  <input type="file" name="img"><br />   <input type="submit" name="提交"> </form> |

java 的 controller如下：

@Controller  
@RequestMapping("/test")  
public class MyController {  
 @RequestMapping(value = "/upload, method = RequestMethod.POST)  
 // 这里的MultipartFile对象变量名跟表单中的file类型的input标签的name相同，所以框架会自动用MultipartFile对象来接收上传过来的文件，当然也可以使用@RequestParam("img")指定其对应的参数名称  
 public String upload(MultipartFile img, HttpSession session)  
     throws Exception {  
   // 如果没有文件上传，MultipartFile也不会为null，可以通过调用getSize()方法获取文件的大小来判断是否有上传文件  
   if (img.getSize() > 0) {  
     // 得到项目在服务器的真实根路径，如：/home/tomcat/webapp/项目名/images  
     String path = session.getServletContext().getRealPath("images");  
     // 得到文件的原始名称，如：美女.png  
     String fileName = img.getOriginalFilename();  
     // 通过文件的原始名称，可以对上传文件类型做限制，如：只能上传jpg和png的图片文件  
     if (fileName.endsWith("jpg") || fileName.endsWith("png")) {  
       File file = new File(path, fileName);  
       img.transferTo(file);  
       return "/success.jsp";  
    }  
  }  
   return "/error.jsp";  
}  
}

spring 配置：

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xsi:schemaLocation="  
   http://www.springframework.org/schema/beans  
   http://www.springframework.org/schema/beans/spring-beans.xsd  
   http://www.springframework.org/schema/tx  
   http://www.springframework.org/schema/tx/spring-tx.xsd  
   http://www.springframework.org/schema/mvc  
   http://www.springframework.org/schema/mvc/spring-mvc.xsd  
   http://www.springframework.org/schema/aop  
   http://www.springframework.org/schema/aop/spring-aop.xsd  
   http://www.springframework.org/schema/context  
   http://www.springframework.org/schema/context/spring-context.xsd">  
​  
...  
​  
 <!-- 注意：CommonsMultipartResolver的id是固定不变的，一定是multipartResolver，不可修改 -->  
 <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  
   <!-- 如果上传后出现文件名中文乱码可以使用该属性解决 -->  
   <property name="defaultEncoding" value="utf-8"/>  
   <!-- 单位是字节，不设置默认不限制总的上传文件大小，这里设置总的上传文件大小不超过1M（1\*1024\*1024） -->  
   <property name="maxUploadSize" value="1048576"/>  
   <!-- 跟maxUploadSize差不多，不过maxUploadSizePerFile是限制每个上传文件的大小，而maxUploadSize是限制总的上传文件大小 -->  
   <property name="maxUploadSizePerFile" value="1048576"/>  
   <!-- 设定文件上传时写入内存的最大值，如果小于这个参数不会生成临时文件，默认为10240 -->  
   <property name="maxInMemorySize" value="40960"></property>  
   <!-- 上传文件的临时路径,超过上述值后，会生成临时文件 -->  
   <property name="uploadTempDir" value="fileUpload/temp"></property>  
   <!-- 延迟文件解析 -->  
   <property name="resolveLazily" value="true"/>  
 </bean>  
​  
 <!-- 设置一个简单的异常解析器，当文件上传超过大小限制时跳转 -->  
 <bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">  
   <property name="defaultErrorView" value="/error.jsp"/>  
 </bean>  
</beans>

### 1.1.2 多文件上传

<form action="${pageContext.request.contextPath}/test/upload.do" method="post" enctype="multipart/form-data">  
file 1 : <input type="file" name="imgs"><br />   
file 2 : <input type="file" name="imgs"><br />   
file 3 : <input type="file" name="imgs"><br />   
 <input type="submit" name="提交">  
</form>

**2、控制器**

控制器中的处理方法使用MultipartFile[]数组作为接收参数，并不能直接使用，需要校正参数，具体说明请看代码注释。

// 这里的MultipartFile[] imgs表示前端页面上传过来的多个文件，imgs对应页面中多个file类型的input标签的name，但框架只会将一个文件封装进一个MultipartFile对象，  
 // 并不会将多个文件封装进一个MultipartFile[]数组，直接使用会报[Lorg.springframework.web.multipart.MultipartFile;.<init>()错误，  
 // 所以需要用@RequestParam校正参数（参数名与MultipartFile对象名一致），当然也可以这么写：@RequestParam("imgs") MultipartFile[] files。  
 public String upload(@RequestParam MultipartFile[] imgs, HttpSession session) throws Exception

### 1.1.3 文件上传综合

当然，项目开发中，场景可能并不是这么简单，上述的多文件上传是一个个文件选择后一起上传（即多个name相同的input标签），那要是我项目中只要一个input标签就可以一次性多个文件呢？又或者一个页面中既要一个个选择的多文件上传，又要一次性选择的多文件上传，还要有单文件上传呢？没问题，MultipartFile[]通吃，代码也很easy，下面直接上代码。

<form action="${pageContext.request.contextPath}/test/upload.do" method="post" enctype="multipart/form-data">  
​  
一次选择多个文件的多文件上传 : <br />   
 <input type="file" name="imgs1" multiple><br /> <br />   
​  
一次选择一个文件的多文件上传 : <br />   
 <input type="file" name="imgs2"><br />   
 <input type="file" name="imgs2"><br /><br />   
​  
单文件上传 : <br />   
 <input type="file" name="imgs3"><br /><br />   
 <input type="submit" name="提交">  
</form>

@Controller  
@RequestMapping("/test")  
public class MyController {  
​  
 @RequestMapping(value = "/upload.do", method = RequestMethod.POST)  
 public String upload(@RequestParam MultipartFile[] imgs1,@RequestParam MultipartFile[] imgs2,@RequestParam MultipartFile[] imgs3, HttpSession session)  
     throws Exception {  
   String path = session.getServletContext().getRealPath("images");  
   for (MultipartFile img : imgs1) {  
     uploadFile(path, img);  
  }  
   for (MultipartFile img : imgs2) {  
     uploadFile(path, img);  
  }  
   for (MultipartFile img : imgs3) {  
     uploadFile(path, img);  
  }  
   return "/success.jsp";  
}  
​  
 private void uploadFile(String path, MultipartFile img) throws IOException {  
   if (img.getSize() > 0) {  
     String fileName = img.getOriginalFilename();  
     if (fileName.endsWith("jpg") || fileName.endsWith("png")) {  
       File file = new File(path, fileName);  
       img.transferTo(file);  
    }  
  }  
}  
}

MultipartFile[]就是如此强大，不管单个多个，逻辑处理一样，所以建议在项目开发中使用MultipartFile[]作为文件的接收参数。

### 1.1.4 拓展

MultipartFile类常用的一些方法

String getContentType()//获取文件MIME类型  
InputStream getInputStream()//获取文件流  
String getName() //获取表单中文件组件的名字  
String getOriginalFilename() //获取上传文件的原名  
long getSize() //获取文件的字节大小，单位byte  
boolean isEmpty() //是否为空  
void transferTo(File dest)

CommonsMultipartResolver的属性

defaultEncoding：表示用来解析request请求的默认编码格式，当没有指定的时候根据Servlet规范会使用默认值ISO-8859-1。当request自己指明了它的编码格式的时候就会忽略这里指定的defaultEncoding。  
uploadTempDir：设置上传文件时的临时目录，默认是Servlet容器的临时目录。  
maxUploadSize：设置允许上传的总的最大文件大小，以字节为单位计算。当设为-1时表示无限制，默认是-1。  
maxUploadSizePerFile：跟maxUploadSize差不多，不过maxUploadSizePerFile是限制每个上传文件的大小，而maxUploadSize是限制总的上传文件大小。  
maxInMemorySize：设置在文件上传时允许写到内存中的最大值，以字节为单位计算，默认是10240。  
resolveLazily：为true时，启用推迟文件解析，以便在UploadAction中捕获文件大小异常。

**注意**

1. 在开发过程中，建议把配置文件中的异常解析器（SimpleMappingExceptionResolver）先注释掉，方便我们查看错误。
2. 有时候上传出错，是因为我们在配置文件中限制了上传文件的大小，你可以不加这个限制，但个人建议这个限制最好还是加上，具体文件大小限制请根据公司项目情况而定。
3. SpringMVC中使用MultipartFile接收上传文件需要依赖两个jar包，分别是：commons-fileupload-1.3.3.jar、

### 1.1.5 MIME类型 /maim/

| **扩展名** | **类型/子类型** |
| --- | --- |
|  | application/octet-stream |
| 323 | text/h323 |
| acx | application/internet-property-stream |
| ai | application/postscript |
| aif | audio/x-aiff |
| aifc | audio/x-aiff |
| aiff | audio/x-aiff |
| asf | video/x-ms-asf |
| asr | video/x-ms-asf |
| asx | video/x-ms-asf |
| au | audio/basic |
| avi | video/x-msvideo |
| axs | application/olescript |
| bas | text/plain |
| bcpio | application/x-bcpio |
| bin | application/octet-stream |
| bmp | image/bmp |
| c | text/plain |
| cat | application/vnd.ms-pkiseccat |
| cdf | application/x-cdf |
| cer | application/x-x509-ca-cert |
| class | application/octet-stream |
| clp | application/x-msclip |
| cmx | image/x-cmx |
| cod | image/cis-cod |
| cpio | application/x-cpio |
| crd | application/x-mscardfile |
| crl | application/pkix-crl |
| crt | application/x-x509-ca-cert |
| csh | application/x-csh |
| css | text/css |
| dcr | application/x-director |
| der | application/x-x509-ca-cert |
| dir | application/x-director |
| dll | application/x-msdownload |
| dms | application/octet-stream |
| doc | application/msword |
| dot | application/msword |
| dvi | application/x-dvi |
| dxr | application/x-director |
| eps | application/postscript |
| etx | text/x-setext |
| evy | application/envoy |
| exe | application/octet-stream |
| fif | application/fractals |
| flr | x-world/x-vrml |
| gif | image/gif |
| gtar | application/x-gtar |
| gz | application/x-gzip |
| h | text/plain |
| hdf | application/x-hdf |
| hlp | application/winhlp |
| hqx | application/mac-binhex40 |
| hta | application/hta |
| htc | text/x-component |
| htm | text/html |
| html | text/html |
| htt | text/webviewhtml |
| ico | image/x-icon |
| ief | image/ief |
| iii | application/x-iphone |
| ins | application/x-internet-signup |
| isp | application/x-internet-signup |
| jfif | image/pipeg |
| jpe | image/jpeg |
| jpeg | image/jpeg |
| jpg | image/jpeg |
| js | application/x-javascript |
| latex | application/x-latex |
| lha | application/octet-stream |
| lsf | video/x-la-asf |
| lsx | video/x-la-asf |
| lzh | application/octet-stream |
| m13 | application/x-msmediaview |
| m14 | application/x-msmediaview |
| m3u | audio/x-mpegurl |
| man | application/x-troff-man |
| mdb | application/x-msaccess |
| me | application/x-troff-me |
| mht | message/rfc822 |
| mhtml | message/rfc822 |
| mid | audio/mid |
| mny | application/x-msmoney |
| mov | video/quicktime |
| movie | video/x-sgi-movie |
| mp2 | video/mpeg |
| mp3 | audio/mpeg |
| mpa | video/mpeg |
| mpe | video/mpeg |
| mpeg | video/mpeg |
| mpg | video/mpeg |
| mpp | application/vnd.ms-project |
| mpv2 | video/mpeg |
| ms | application/x-troff-ms |
| mvb | application/x-msmediaview |
| nws | message/rfc822 |
| oda | application/oda |
| p10 | application/pkcs10 |
| p12 | application/x-pkcs12 |
| p7b | application/x-pkcs7-certificates |
| p7c | application/x-pkcs7-mime |
| p7m | application/x-pkcs7-mime |
| p7r | application/x-pkcs7-certreqresp |
| p7s | application/x-pkcs7-signature |
| pbm | image/x-portable-bitmap |
| pdf | application/pdf |
| pfx | application/x-pkcs12 |
| pgm | image/x-portable-graymap |
| pko | application/ynd.ms-pkipko |
| pma | application/x-perfmon |
| pmc | application/x-perfmon |
| pml | application/x-perfmon |
| pmr | application/x-perfmon |
| pmw | application/x-perfmon |
| pnm | image/x-portable-anymap |
| pot, | application/vnd.ms-powerpoint |
| ppm | image/x-portable-pixmap |
| pps | application/vnd.ms-powerpoint |
| ppt | application/vnd.ms-powerpoint |
| prf | application/pics-rules |
| ps | application/postscript |
| pub | application/x-mspublisher |
| qt | video/quicktime |
| ra | audio/x-pn-realaudio |
| ram | audio/x-pn-realaudio |
| ras | image/x-cmu-raster |
| rgb | image/x-rgb |
| rmi | audio/mid |
| roff | application/x-troff |
| rtf | application/rtf |
| rtx | text/richtext |
| scd | application/x-msschedule |
| sct | text/scriptlet |
| setpay | application/set-payment-initiation |
| setreg | application/set-registration-initiation |
| sh | application/x-sh |
| shar | application/x-shar |
| sit | application/x-stuffit |
| snd | audio/basic |
| spc | application/x-pkcs7-certificates |
| spl | application/futuresplash |
| src | application/x-wais-source |
| sst | application/vnd.ms-pkicertstore |
| stl | application/vnd.ms-pkistl |
| stm | text/html |
| svg | image/svg+xml |
| sv4cpio | application/x-sv4cpio |
| sv4crc | application/x-sv4crc |
| swf | application/x-shockwave-flash |
| t | application/x-troff |
| tar | application/x-tar |
| tcl | application/x-tcl |
| tex | application/x-tex |
| texi | application/x-texinfo |
| texinfo | application/x-texinfo |
| tgz | application/x-compressed |
| tif | image/tiff |
| tiff | image/tiff |
| tr | application/x-troff |
| trm | application/x-msterminal |
| tsv | text/tab-separated-values |
| txt | text/plain |
| uls | text/iuls |
| ustar | application/x-ustar |
| vcf | text/x-vcard |
| vrml | x-world/x-vrml |
| wav | audio/x-wav |
| wcm | application/vnd.ms-works |
| wdb | application/vnd.ms-works |
| wks | application/vnd.ms-works |
| wmf | application/x-msmetafile |
| wps | application/vnd.ms-works |
| wri | application/x-mswrite |
| wrl | x-world/x-vrml |
| wrz | x-world/x-vrml |
| xaf | x-world/x-vrml |
| xbm | image/x-xbitmap |
| xla | application/vnd.ms-excel |
| xlc | application/vnd.ms-excel |
| xlm | application/vnd.ms-excel |
| xls | application/vnd.ms-excel |
| xlt | application/vnd.ms-excel |
| xlw | application/vnd.ms-excel |
| xof | x-world/x-vrml |
| xpm | image/x-xpixmap |
| xwd | image/x-xwindowdump |
| z | application/x-compress |
| zip | application/zip |

## 1.2 日志

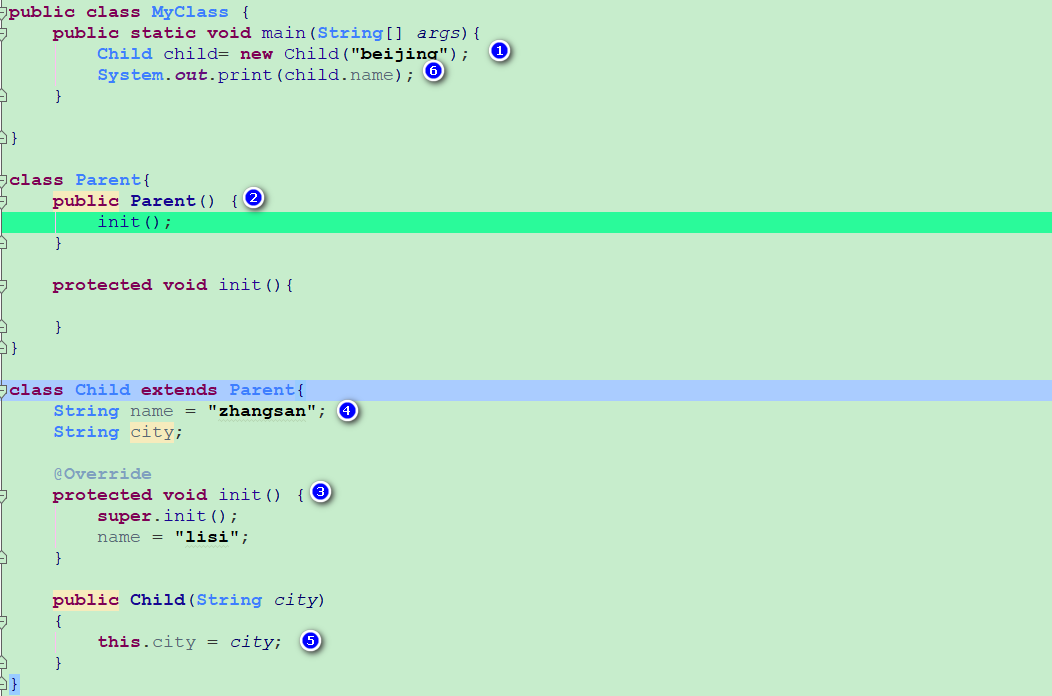
%n - 换行  
%m - 日志内容  
%p - 日志级别(FATAL,   ERROR,   WARN,   INFO,   DEBUG   or   custom)  
%r - 程序启动到现在的毫秒数  
%% - percent   sign   in   output  
%t - 当前线程名  
%d   -  日期和时间,  
常用的格式有 %d{DATE}, %d{ABSOLUTE}, %d{HH:mm:ss,SSS}, %d{ddMMyyyy HH:mm:ss,SSS}。。。  
%l - 同 %F%L%C%M  
%F - java源文件名  
%L - java源码行数  
%c 输出所属类的全名，可写为 %c{Num} ,Num类名输出的范围 如："com.sun.aaa.classB",%C{2}将使日志输出输出范围为：aaa.classB  
%M - java方法名  
%n - 换行  
%m - 日志内容  
%p - 日志级别(FATAL,  ERROR,  WARN,  INFO,  DEBUG  or  custom)  
%r - 程序启动到现在的毫秒数  
%l：输出日志事件的发生位置，相当于%c.%M(%F:%L)的组合，包括类全名、方法、文件名以及在代码中的行数。例  
%x：输出和当前线程相关联的NDC(嵌套诊断环境)，尤其用到像java servlets这样的多客户多线程的应用中。  
​  
常用的格式有 %d{DATE}, %d{ABSOLUTE}, %d{HH:mm:ss,SSS}, %d{ddMMyyyy HH:mm:ss,SSS}。。。  
​  
示例：  
[%d{HH\:mm\:ss\:SSS}][%p] (%c\:%L) - %m%n  
​  
输出格式为：[08:58:59:412][INFO] (com.soon.action:35) - 服务器启动  
[%d{ISO8601}] %-5p %-x %X{user} %n     [%l]-%m%n  
[2018-11-16 10:20:58,432] ERROR     
    审批同意与否不能为空, task id is: 201811160011  
​  
log4j.appender.CONSOLE.layout.ConversionPattern = %d{ABSOLUTE} %5p %c{1}:%L [%t:%r]- %m%n

## 1.3 打包命令

jar -cvfM0 catalog20181215.war ./catalog

## 1.4 构造过程

默认先调用父类无参构造函数构造出父类，在父类构造时 init 函数重写，调用子类 init 函数，注意这里虽然是父类的构造函数，但是调用的是子类 init。父类构造完毕，开始子类构造，对于有默认值的属性，先处理默认值，再执行构造方法内的代码。



## Excel工具

POI提供了HSSF、XSSF以及SXSSF三种方式操作Excel。

**HSSF：**Excel97-2003版本，扩展名为.xls。一个sheet最大行数**65536**，最大列数256。

**XSSF：**Excel2007版本开始，扩展名为.xlsx。一个sheet最大行数**1048576**，最大列数16384。

**SXSSF：**是在XSSF基础上，POI3.8版本开始提供的**支持低内存占用**的操作方式，扩展名为.xlsx。

Excel版本兼容性是**向下兼容**。

三种类的接口及方法：

HSSF：HSSFWorkbook、HSSFSheet、HSSFRow、HSSFCell……

XSSF：XSSFWorkbook、XSSFSheet、XSSFRow、XSSFCell……

SXSSF：SXSSFWorkbook、Sheet、Row、Cell……

很显然，**SXSSF比较特殊**，除了workBook带前缀外，其余均无前缀。而HSSF和XSSF都带了各自的前缀。所以设置格式等在相互之间切换还是很简单的。

**很重要的一点**，SXSSF之所以是一种低内存操作方式，是因为他的构造方法：

SXSSFWorkbook w3= new SXSSFWorkbook(100);

这个100，你可以理解为POI操作时，内存中最多只有100行数据，当超过这个数据时，就将内存之前的数据删除，并且会在硬盘中生成临时文件。从而保证了低内存消耗。当然，你也可以将这个数字调大一点。

## Mybatis

Collection 中 column 的 key 是参数，值是列名字，不是相应的属性名

|  |
| --- |
| <resultMap id="findCountryCityAddressMap" type="map">  <result column ="country\_id" property ="country"/>  <result column="user\_id" property="userId" jdbcType="INTEGER" />  <result column=" city\_id " property="theme" jdbcType="VARCHAR" />  <collection property="cityList"  column="{cityId=city\_id,adr= country\_id, dis= user\_id }"  ofType="map"  javaType="java.util.List" select="ref\_1"/>  </resultMap>  <select id=" ref\_1" parameterType="java.util.Map" resultMap="selectAddressByCityIdMap">  SELECT \* WHERE ci.city\_id = #{cityId} and ads.address RegExp #{adr} and ads.district Regexp #{dis}  </ select> |

## alibaba.druid 连接池配置

## ****DRUID介绍****

    DRUID是阿里巴巴开源平台上一个数据库连接池实现，它结合了C3P0、DBCP、PROXOOL等DB池的优点，同时加入了日志监控，可以很好的监控DB池连接和SQL的执行情况，可以说是针对监控而生的DB连接池(据说是目前最好的连接池,不知道速度有没有BoneCP快)。

## ****配置参数****

和其它连接池一样DRUID的DataSource类为：com.alibaba.druid.pool.DruidDataSource，基本配置参数如下：

|  |  |  |
| --- | --- | --- |
| 配置 | 缺省值 | 说明 |
| name |  | 配置这个属性的意义在于，如果存在多个数据源，监控的时候可以通过名字来区分开来。  如果没有配置，将会生成一个名字，格式是："DataSource-" + System.identityHashCode(this) |
| jdbcUrl |  | 连接数据库的url，不同数据库不一样。例如：  mysql : jdbc:mysql://10.20.153.104:3306/druid2  oracle : jdbc:oracle:thin:@10.20.149.85:1521:ocnauto |
| username |  | 连接数据库的用户名 |
| password |  | 连接数据库的密码。如果你不希望密码直接写在配置文件中，可以使用ConfigFilter。详细看这里：https://github.com/alibaba/druid/wiki/%E4%BD%BF%E7%94%A8ConfigFilter |
| driverClassName | 根据url自动识别 | 这一项可配可不配，如果不配置druid会根据url自动识别dbType，然后选择相应的driverClassName(建议配置下) |
| initialSize | 0 | 初始化时建立物理连接的个数。初始化发生在显示调用init方法，或者第一次getConnection时 |
| maxActive | 8 | 最大连接池数量 |
| maxIdle | 8 | 已经不再使用，配置了也没效果 |
| minIdle |  | 最小连接池数量 |
| maxWait |  | 获取连接时最大等待时间，单位毫秒。配置了maxWait之后，缺省启用公平锁，并发效率会有所下降，如果需要可以通过配置useUnfairLock属性为true使用非公平锁。 |
| poolPreparedStatements | false | 是否缓存preparedStatement，也就是PSCache。PSCache对支持游标的数据库性能提升巨大，比如说oracle。在mysql下建议关闭。 |
| maxOpenPreparedStatements | -1 | 要启用PSCache，必须配置大于0，当大于0时，poolPreparedStatements自动触发修改为true。在Druid中，不会存在Oracle下PSCache占用内存过多的问题，可以把这个数值配置大一些，比如说100 |
| validationQuery |  | 用来检测连接是否有效的sql，要求是一个查询语句。如果validationQuery为null，testOnBorrow、testOnReturn、testWhileIdle都不会其作用。 |
| testOnBorrow | true | 申请连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能。 |
| testOnReturn | false | 归还连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能 |
| testWhileIdle | false | 建议配置为true，不影响性能，并且保证安全性。申请连接的时候检测，如果空闲时间大于timeBetweenEvictionRunsMillis，执行validationQuery检测连接是否有效。 |
| timeBetweenEvictionRunsMillis |  | 有两个含义：  1) Destroy线程会检测连接的间隔时间2) testWhileIdle的判断依据，详细看testWhileIdle属性的说明 |
| numTestsPerEvictionRun |  | 不再使用，一个DruidDataSource只支持一个EvictionRun |
| minEvictableIdleTimeMillis |  |  |
| connectionInitSqls |  | 物理连接初始化的时候执行的sql |
| exceptionSorter | 根据dbType自动识别 | 当数据库抛出一些不可恢复的异常时，抛弃连接 |
| filters |  | 属性类型是字符串，通过别名的方式配置扩展插件，常用的插件有：  监控统计用的filter:stat日志用的filter:log4j防御sql注入的filter:wall |
| proxyFilters |  | 类型是List<com.alibaba.druid.filter.Filter>，如果同时配置了filters和proxyFilters，是组合关系，并非替换关系 |

* 使用方法

DB数据源的使用方法也就是2种，一种是在代码中写死通过NEW操作符创建DataSSource，然后set一些连接属性，这里不在累述;另外一种是基于SPRING的配置方法，然后让SPRING的Context自动加载配置（以下配置文件默认都在项目根目录下conf文件夹中）

1、属性文件:application.properties(DataSource连接参数)

jdbc.driverClassName=com.mysql.jdbc.Driver   
jdbc.url=jdbc:mysql://127.0.0.1:3306/test   
jdbc.username=root   
jdbc.password=1qaz!QAZ

2、SPRING配置文件：spring-base.xml

<?xml version="1.0" encoding="UTF-8"?>   
<beans xmlns=" <http://www.springframework.org/schema/beans>"   
 xmlns:xsi=" <http://www.w3.org/2001/XMLSchema-instance>" xmlns:batch=" <http://www.springframework.org/schema/batch>"   
 xsi:schemaLocation=" <http://www.springframework.org/schema/beans>   
           <http://www.springframework.org/schema/beans/spring-beans-4.0.xsd>">

 <bean id="propertyConfigure"   
  class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">   
  <property name="locations">   
   <list>   
    <value>./conf/application.properties</value>   
   </list>   
  </property>   
 </bean>

 <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"   
  init-method="init" destroy-method="close">   
  <property name="driverClassName" value="${jdbc.driverClassName}" />   
  <property name="url" value="${jdbc.url}" />   
  <property name="username" value="${jdbc.username}" />   
  <property name="password" value="${jdbc.password}" />   
  <!-- 配置初始化大小、最小、最大 -->   
  <property name="initialSize" value="1" />   
  <property name="minIdle" value="1" />   
  <property name="maxActive" value="10" />

  <!-- 配置获取连接等待超时的时间 -->   
  <property name="maxWait" value="10000" />

  <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->   
  <property name="timeBetweenEvictionRunsMillis" value="60000" />

  <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->   
  <property name="minEvictableIdleTimeMillis" value="300000" />

  <property name="testWhileIdle" value="true" />

  <!-- 这里建议配置为TRUE，防止取到的连接不可用 -->   
  <property name="testOnBorrow" value="true" />   
  <property name="testOnReturn" value="false" />

  <!-- 打开PSCache，并且指定每个连接上PSCache的大小 -->   
  <property name="poolPreparedStatements" value="true" />   
  <property name="maxPoolPreparedStatementPerConnectionSize"   
   value="20" />

  <!-- 这里配置提交方式，默认就是TRUE，可以不用配置 -->

  <property name="defaultAutoCommit" value="true" />

  <!-- 验证连接有效与否的SQL，不同的数据配置不同 -->   
  <property name="validationQuery" value="select 1 " />   
  <property name="filters" value="stat" />   
  <property name="proxyFilters">   
   <list>   
    <ref bean="logFilter" />   
   </list>   
  </property>   
 </bean>

 <bean id="logFilter" class="com.alibaba.druid.filter.logging.Slf4jLogFilter">   
  <property name="statementExecutableSqlLogEnable" value="false" />   
 </bean>

</beans>

上面红色标注部分为监控DB池连接执行监控，后面在做详细说明.

* 监控方式

1、WEB方式监控配置

<servlet>   
     <servlet-name>DruidStatView</servlet-name>   
     <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class>   
 </servlet>   
 <servlet-mapping>   
     <servlet-name>DruidStatView</servlet-name>   
     <url-pattern>/druid/\*</url-pattern>   
 </servlet-mapping>   
 <filter>   
  <filter-name>druidWebStatFilter</filter-name>   
  <filter-class>com.alibaba.druid.support.http.WebStatFilter</filter-class>   
  <init-param>   
   <param-name>exclusions</param-name>   
   <param-value>/public/\*,\*.js,\*.css,/druid\*,\*.jsp,\*.swf</param-value>   
  </init-param>   
  <init-param>   
   <param-name>principalSessionName</param-name>   
   <param-value>sessionInfo</param-value>   
  </init-param>   
  <init-param>   
   <param-name>profileEnable</param-name>   
   <param-value>true</param-value>   
  </init-param>   
 </filter>   
 <filter-mapping>   
  <filter-name>druidWebStatFilter</filter-name>   
  <url-pattern>/\*</url-pattern>   
 </filter-mapping>

把上面servlet配置添加到项目web.xml即可。然后运行Tomcat，浏览器输入 [http://IP:PROT/druid](http://ipprot/)

就可以打开Druid的监控页面了.

2、日志文件监控

Druid提供了多种日志文件监控 commons-logging、log4j等，这里我们主要使用slf4j和logback来进行日志监控配置。

首先要引入slf4j和logback相关的jar文件（从Maven公共仓库下载 <http://search.maven.org/>）

<slf4j.version>1.7.7</slf4j.version>   
<logback.version>1.1.2</logback.version>

<dependency>   
    <groupId>org.slf4j</groupId>   
    <artifactId>slf4j-api</artifactId>   
    <version>${slf4j.version}</version>   
 </dependency>   
<dependency>   
    <groupId>ch.qos.logback</groupId>   
    <artifactId>logback-access</artifactId>   
    <version>${logback.version}</version>   
</dependency>   
<dependency>   
    <groupId>ch.qos.logback</groupId>   
    <artifactId>logback-core</artifactId>   
    <version>${logback.version}</version>   
</dependency>   
<dependency>   
    <groupId>ch.qos.logback</groupId>   
    <artifactId>logback-classic</artifactId>   
    <version>${logback.version}</version>   
</dependency>

接下配置logback的配置文件(./conf/logback.xml)

<configuration>

 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">   
  <layout class="ch.qos.logback.classic.PatternLayout">   
   <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n   
   </Pattern>   
  </layout>   
 </appender>

 <appender name="FILE" class="ch.qos.logback.core.FileAppender">   
  <file>./logs/druid\_info.log</file>   
  <layout class="ch.qos.logback.classic.PatternLayout">   
   <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</Pattern>   
  </layout>   
  <filter class="ch.qos.logback.classic.filter.ThresholdFilter">   
   <level>debug</level>   
  </filter>   
 </appender>

 <root level="DEBUG">   
  <appender-ref ref="FILE" />   
 </root>   
</configuration>

最后就是写一个测试类进行测试

public class TestMain {

 public static void loadLoggerContext() {   
  System.getProperties().put("logback.configurationFile", "./conf/logback.xml");   
  LoggerContext lc = (LoggerContext) LoggerFactory.getILoggerFactory();   
  StatusPrinter.setPrintStream(System.err);   
  StatusPrinter.print(lc);   
 }

 public static void main(String[] args) {   
  try {   
   loadLoggerContext();   
   FileSystemXmlApplicationContext context = new FileSystemXmlApplicationContext("./conf/spring-base.xml");   
  
  } catch (Exception e) {   
   System.out.println(e);   
  }   
 }   
}

## 生成 xml 文件

# ava生成xml文件的四种方式

2017年10月28日 19:33:56 [泪水三千](https://me.csdn.net/qq_39237801) 阅读数：9368 标签： [Java](http://so.csdn.net/so/search/s.do?q=Java&t=blog)[xml](http://so.csdn.net/so/search/s.do?q=xml&t=blog) 更多

个人分类： [Java](https://blog.csdn.net/qq_39237801/article/category/6982160)

 版权声明：本文为博主原创文章，转载请注明出处，谢谢了。 https://blog.csdn.net/qq\_39237801/article/details/78378486

#绪论  
上一篇文章写了xml文件4种读取方式，本篇文章将简介使用四种方式对xml进行写入。

#一、Dom写入

import java.io.File;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.junit.Test;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

public class DomXml {

@Test

public void test(){

Long start = System.currentTimeMillis();

createXml();

System.out.println("运行时间："+ (System.currentTimeMillis() - start));

}

/\*\*

\* 生成xml方法

\*/

public static void createXml(){

try {

// 创建解析器工厂

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder db = factory.newDocumentBuilder();

Document document = db.newDocument();

// 不显示standalone="no"

document.setXmlStandalone(true);

Element bookstore = document.createElement("bookstore");

// 向bookstore根节点中添加子节点book

Element book = document.createElement("book");

Element name = document.createElement("name");

// 不显示内容 name.setNodeValue("不好使");

name.setTextContent("雷神");

book.appendChild(name);

// 为book节点添加属性

book.setAttribute("id", "1");

// 将book节点添加到bookstore根节点中

bookstore.appendChild(book);

// 将bookstore节点（已包含book）添加到dom树中

document.appendChild(bookstore);

// 创建TransformerFactory对象

TransformerFactory tff = TransformerFactory.newInstance();

// 创建 Transformer对象

Transformer tf = tff.newTransformer();

// 输出内容是否使用换行

tf.setOutputProperty(OutputKeys.INDENT, "yes");

// 创建xml文件并写入内容

tf.transform(new DOMSource(document), new StreamResult(new File("book1.xml")));

System.out.println("生成book1.xml成功");

} catch (Exception e) {

e.printStackTrace();

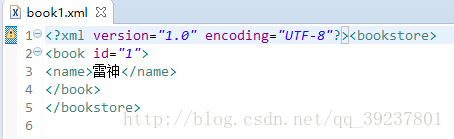
System.out.println("生成book1.xml失败");

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66

生成结果：  


#二、Dom4j写入

import java.io.File;

import java.io.FileOutputStream;

import org.dom4j.Document;

import org.dom4j.DocumentHelper;

import org.dom4j.Element;

import org.dom4j.io.OutputFormat;

import org.dom4j.io.XMLWriter;

import org.junit.Test;

public class Dom4jXml {

@Test

public void test(){

Long start = System.currentTimeMillis();

createXml();

System.out.println("运行时间："+ (System.currentTimeMillis() - start));

}

/\*\*

\* 生成xml方法

\*/

public static void createXml(){

try {

// 1、创建document对象

Document document = DocumentHelper.createDocument();

// 2、创建根节点rss

Element rss = document.addElement("rss");

// 3、向rss节点添加version属性

rss.addAttribute("version", "2.0");

// 4、生成子节点及子节点内容

Element channel = rss.addElement("channel");

Element title = channel.addElement("title");

title.setText("国内最新新闻");

// 5、设置生成xml的格式

OutputFormat format = OutputFormat.createPrettyPrint();

// 设置编码格式

format.setEncoding("UTF-8");

// 6、生成xml文件

File file = new File("rss.xml");

XMLWriter writer = new XMLWriter(new FileOutputStream(file), format);

// 设置是否转义，默认使用转义字符

writer.setEscapeText(false);

writer.write(document);

writer.close();

System.out.println("生成rss.xml成功");

} catch (Exception e) {

e.printStackTrace();

System.out.println("生成rss.xml失败");

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55

生成结果：  


#三、jDom写入

import java.io.File;

import java.io.FileOutputStream;

import org.jdom.Document;

import org.jdom.Element;

import org.jdom.output.Format;

import org.jdom.output.XMLOutputter;

import org.junit.Test;

public class JDomXml {

@Test

public void test(){

Long start = System.currentTimeMillis();

createXml();

System.out.println("运行时间："+ (System.currentTimeMillis() - start));

}

/\*\*

\* 生成xml方法

\*/

public static void createXml(){

try {

// 1、生成一个根节点

Element rss = new Element("rss");

// 2、为节点添加属性

rss.setAttribute("version", "2.0");

// 3、生成一个document对象

Document document = new Document(rss);

Element channel = new Element("channel");

rss.addContent(channel);

Element title = new Element("title");

title.setText("国内最新新闻");

channel.addContent(title);

Format format = Format.getCompactFormat();

// 设置换行Tab或空格

format.setIndent(" ");

format.setEncoding("UTF-8");

// 4、创建XMLOutputter的对象

XMLOutputter outputer = new XMLOutputter(format);

// 5、利用outputer将document转换成xml文档

File file = new File("rssNew.xml");

outputer.output(document, new FileOutputStream(file));

System.out.println("生成rssNew.xml成功");

} catch (Exception e) {

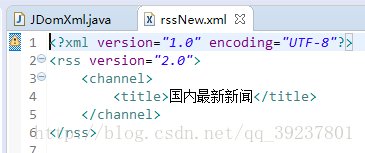
System.out.println("生成rssNew.xml失败");

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55

生成结果：  


#四、Sax写入

import java.io.File;

import java.io.FileOutputStream;

import java.util.ArrayList;

import java.util.List;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Result;

import javax.xml.transform.Transformer;

import javax.xml.transform.sax.SAXTransformerFactory;

import javax.xml.transform.sax.TransformerHandler;

import javax.xml.transform.stream.StreamResult;

import org.junit.Test;

import org.xml.sax.helpers.AttributesImpl;

public class SaxXml {

@Test

public void test(){

List<Book> bList = new ArrayList<Book>();

Book b = new Book();

b.setName("冰与火之歌");

b.setAuthor("乔治马丁");

b.setId("1");

b.setLanguage("English");

b.setPrice("86");

b.setYear("2014");

bList.add(b);

Long start = System.currentTimeMillis();

createXml(bList);

System.out.println("运行时间："+ (System.currentTimeMillis() - start));

}

// 生成xml

public static void createXml(List<Book> bList){

// 1、创建一个SAXTransformerFactory类的对象

SAXTransformerFactory tff = (SAXTransformerFactory) SAXTransformerFactory.newInstance();

try {

// 2、通过SAXTransformerFactory创建一个TransformerHandler的对象

TransformerHandler handler = tff.newTransformerHandler();

// 3、通过handler创建一个Transformer对象

Transformer tr = handler.getTransformer();

// 4、通过Transformer对象对生成的xml文件进行设置

// 设置编码方式

tr.setOutputProperty(OutputKeys.ENCODING, "UTF-8");

// 设置是否换行

tr.setOutputProperty(OutputKeys.INDENT, "yes");

// 5、创建一个Result对象

File f = new File("src/newbooks.xml");

// 判断文件是否存在

if(!f.exists()){

f.createNewFile();

}

Result result = new StreamResult(new FileOutputStream(f));

// 6、使RESULT与handler关联

handler.setResult(result);

// 打开document

handler.startDocument();

AttributesImpl attr = new AttributesImpl();

handler.startElement("", "", "bookstore", attr);

attr.clear();

for (Book book : bList) {

attr.clear();

attr.addAttribute("", "", "id", "", book.getId());

handler.startElement("", "", "book", attr);

// 创建name

attr.clear();

handler.startElement("", "", "name", attr);

handler.characters(book.getName().toCharArray(), 0, book.getName().length());

handler.endElement("", "", "name");

// 创建year

attr.clear();

handler.startElement("", "", "year", attr);

handler.characters(book.getYear().toCharArray(), 0, book.getYear().length());

handler.endElement("", "", "year");

// 创建author

if(book.getAuthor() != null && !"".equals(book.getAuthor().trim())){

attr.clear();

handler.startElement("", "", "author", attr);

handler.characters(book.getAuthor().toCharArray(), 0, book.getAuthor().length());

handler.endElement("", "", "author");

}

// 创建price

if(book.getPrice() != null && !"".equals(book.getPrice().trim())){

attr.clear();

handler.startElement("", "", "price", attr);

handler.characters(book.getPrice().toCharArray(), 0, book.getPrice().length());

handler.endElement("", "", "price");

}

// 创建language

if(book.getLanguage() != null && !"".equals(book.getLanguage().trim())){

attr.clear();

handler.startElement("", "", "language", attr);

handler.characters(book.getLanguage().toCharArray(), 0, book.getLanguage().length());

handler.endElement("", "", "language");

}

handler.endElement("", "", "book");

}

handler.endElement("", "", "bookstore");

// 关闭document

handler.endDocument();

System.out.println("生成newbooks.xml成功");

} catch (Exception e) {

e.printStackTrace();

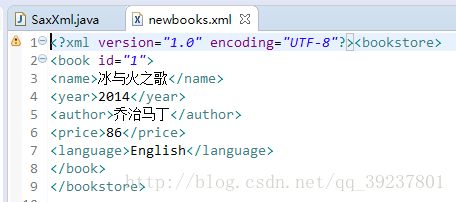
System.out.println("生成newbooks.xml失败");

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119

生成结果：  


#五、其它

/\*\*

\* 四种方式总结

\*

\* dom基于tree，sax基于事件，JDOM和DOM4J基于底层API

\*

\* dom生成的内容会保存到内存中，方便删除和修改

\* sax不能修改已经生成的标签

\*

\* 添加JUnit在Libraries选择Add Library中的JUnit即可

\*

\*/

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

（若有什么错误，请留言指正，3Q）