# 数据源，连接池，数据库连接之间的关系

数据源所依附的对象是mysql这个软件，只要安装有这个软件就有数据源，不管你有没有创建自己的数据库实例，数据源都是存在的而且一般一个数据库软件都会只有一个数据源。

数据源包含两个部分：连接池和连接管理。

        接下来就讲讲这个连接池，在你把mysql安装好了之后，环境变量也配好了，那么这个数据源这两个部分也就存在了，在这个连接池里面它包含了一定数量的Connection对象，即数据库连接。但此时这些数据库连接Connection还是原生的。并没有指向具体的那一个数据库实例，例如samples。当你使用DriverManager.getConnection(“jdbc:mysql;//127.0.0.1:3306/samples”,root,123456)方法从连接池里面获取一个原生Connection之后，这个原生的Connection就会利用该方法参数进行初始化修饰。例如上面这个方参数：“jdbc:mysql;//127.0.0.1:3306/samples”就是原生Connection指向具体数据库实例的Url，root就是所指向具体数据库实例samples的用户名，123456就是密码。至此一个指向mysql软件里的数据库实例samples的Connection获取到了，此时数据库连接池就少了一个原生的Connection。当获取到的这个指向samples的Connection使用完了，就对其进行还原，并重新释放到连接池中。而这一切的获取和释放操作都是又连接管理来进行管理的。

# 我的第一个springboot程序

### 1.在idea中创建maven项目

例如springbootDemo

<groupId>com.springboot</groupId>  
<artifactId>springbootDemo</artifactId>

### 2.在pom.xml中配置

<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
</dependencies>

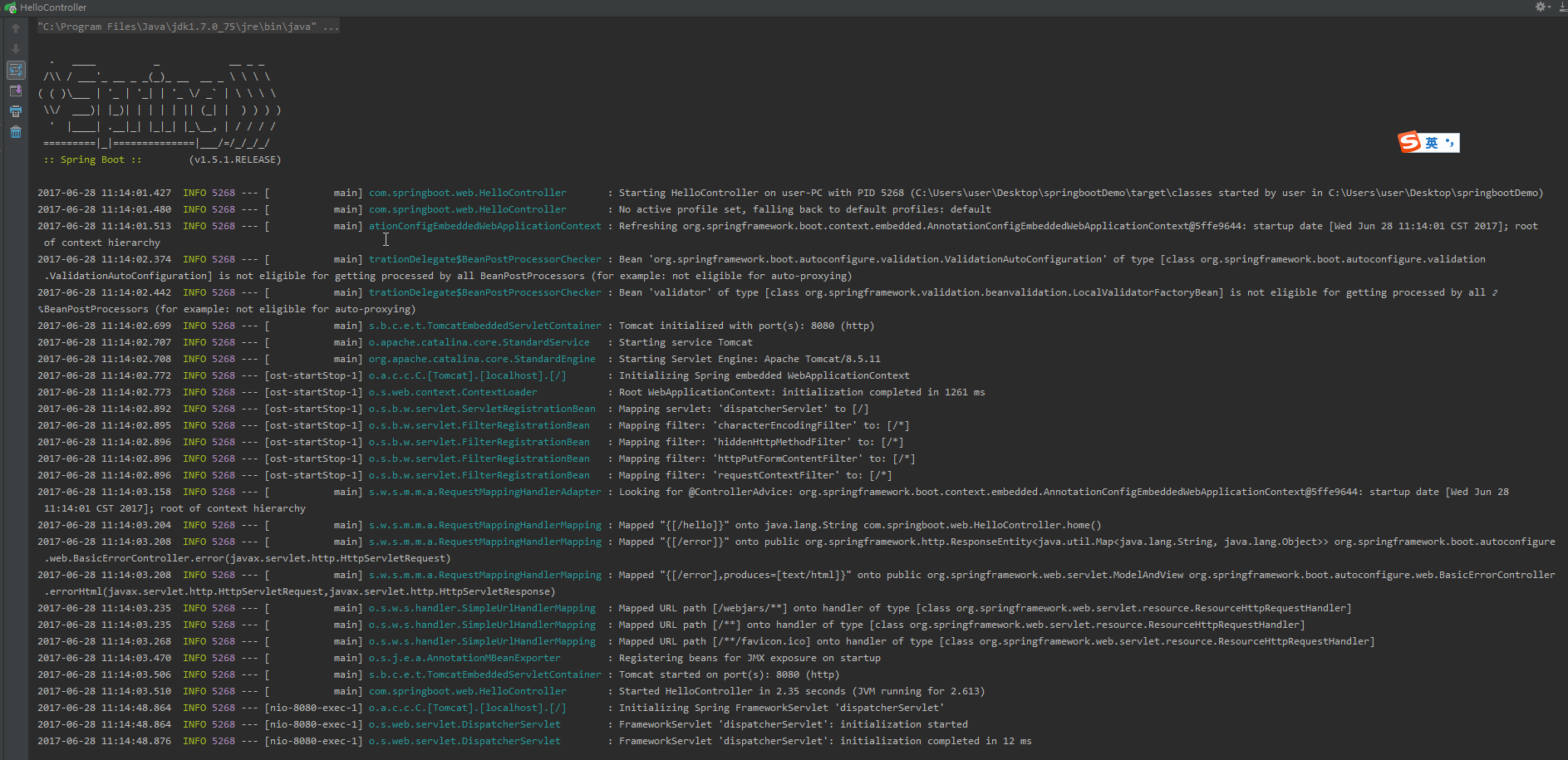
<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.1.RELEASE</version>  
</parent>

### 3.在src中书写controller类

package com.springboot.web;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
@Controller  
@EnableAutoConfiguration  
public class HelloController {  
  
 @RequestMapping("/hello")  
 @ResponseBody  
 String home() {  
 return "Hello ,spring boot!";  
 }  
  
 public static void main(String[] args) throws Exception {  
 SpringApplication.*run*(HelloController.class, args);  
 //运行之后在浏览器中访问：http://localhost:8080/hello  
 }  
}

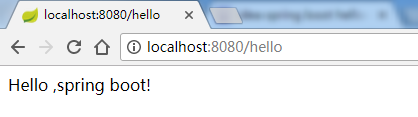
### 4.项目右键run HelloController

### 5.控制台日志输出:



### 6.浏览器输入<http://localhost:8080/hello>

获得页面:



刚写完helloworld,然后就找到了一个学习springboot系列的文章,可以跟着这篇文章一起学

## Spring boot 那些事-目录

Spring Boot，支持约定优于配置，让开发人员尽快启动并运行项目。针对 Spring Boot 的学习和总结准备写系列文章。

代码共享在【**[springboot-learning-example](https://github.com/JeffLi1993/springboot-learning-example)**】，spring boot 实践学习案例，是 spring boot 初学者及核心技术巩固的最佳实践。

任何问题，上「[Spring For All](http://spring4all.com/)」找我！

 文章如下：

**『 基础 - 入门篇 』**

《[Spring Boot 之 HelloWorld 详解](http://www.bysocket.com/?p=1124)》

《[Spring Boot 之配置文件详解](http://www.bysocket.com/?p=1786)》

**『 基础 - Web 业务开发篇 』**

《[Spring Boot 实现 Restful 服务，基于 HTTP / JSON 传输](http://www.bysocket.com/?p=1627)》

《[Spring Boot 之 RESRful API 权限控制](http://www.bysocket.com/?p=1080)》

《[Spring Boot 集成 FreeMarker](http://www.bysocket.com/?p=1666)》

《[Spring Boot HTTP over JSON 的错误码异常处理](http://www.bysocket.com/?p=1692)》

《Spring Boot 使用 Swagger2 构建 RESRful API 文档》

《Spring Boot 集成 JSP》

《Spring Boot 集成 Thymeleaf》

《Spring Boot 单元测试的使用》

《Spring Boot 热更新部署》

**『 基础 – 数据存储篇 』**

《[Spring Boot 整合 Mybatis 的完整 Web 案例](http://www.bysocket.com/?p=1610)》

《[Spring Boot 整合 Mybatis Annotation 注解案例](http://www.bysocket.com/?p=1811)》

《[Spring Boot 整合 Mybatis 实现 Druid 多数据源配置](http://www.bysocket.com/?p=1712)》

《Spring Boot 整合使用 JdbcTemplate》

《Spring Boot 整合 Spring-data-jpa》

《Spring Boot 声明式事务管理》

**『 基础 – 数据缓存篇 』**

《[Spring Boot 整合 Redis 实现缓存操作](http://www.bysocket.com/?p=1756)》

《Spring Boot 整合 Redis Annotation 实现缓存操作》

《Spring Boot 整合 MongoDB 实现缓存操作》

《Spring Boot 整合 EhCache 实现缓存操作》

**『 基础 – 日志管理篇 』**

《Spring Boot 默认日志 logback 配置解析》

《Spring Boot 使用 log4j 记录日志》

《Spring Boot 对 log4j 进行多环境不同日志级别的控制》

《Spring Boot 使用 log4j 记录日志到 MongoDB》

《Spring Boot 1.5.x 动态修改日志级别》

**『 基础 – 应用篇 』**

《Spring Boot Actuator 监控》

《Spring Boot Web 应用部署》

**『 提升 – 安全控制及权限篇 』**

《Spring Boot 整合 Spring Security》

《Spring Boot 整合 Shiro》

《Spring Boot 整合 Spring Session》

**『 提升 – 消息服务篇 』**

《Spring Boot 整合 RabbitMQ》

**『 提升 – 源码篇 』**

《Spring Boot 启动原理解析》

**『 其他篇 』**

《[Spring Boot 整合 Dubbo/ZooKeeper 详解 SOA 案例](http://www.bysocket.com/?p=1681)》

《[Spring Boot 中如何使用 Dubbo Activate 扩展点](http://www.bysocket.com/?p=1782)》

《[Spring Boot Dubbo applications.properties 配置清单](http://www.bysocket.com/?p=1805)》

《[Spring Boot 整合 Elasticsearch](http://www.bysocket.com/?p=1829)》

《Spring Boot 整合 Quartz》

## Spring boot 那些事-**『 基础 - 入门篇 』**

命令创建maven项目骨架:

mvn archetype:generate -DgroupId=springboot -DartifactId=springboot-helloworld -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

1.pom.xml中的依赖:

<!-- Spring Boot 启动父依赖 -->

    <parent>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-parent</artifactId>

        <version>1.3.3.RELEASE</version>

    </parent>

<!-- Spring Boot web依赖 -->

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-web</artifactId>

    </dependency>

</dependencies>

2.Controller层:

1. @RestController：提供实现了REST API，可以服务JSON,XML或者其他。这里是以String的形式渲染出结果。

2. @RequestMapping：提供路由信息，”/“路径的HTTP Request都会被映射到sayHello方法进行处理。

3.启动应用类:

1. @SpringBootApplication：Spring Boot 应用的标识

2. Application很简单，一个main函数作为主入口。SpringApplication引导应用，并将Application本身作为参数传递给run方法。具体run方法会启动嵌入式的Tomcat并初始化Spring环境及其各Spring组件。

# \*\*\*

要深入学习注解，我们就必须能定义自己的注解，并使用注解，在定义自己的注解之前，我们就必须要了解Java为我们提供的元注解和相关定义注解的语法。

# 元注解：

　　元注解的作用就是负责注解其他注解。Java5.0定义了4个标准的meta-annotation类型，它们被用来提供对其它 annotation类型作说明。Java5.0定义的元注解：  
　　　　1.@Target,  
　　　　2.@Retention,  
　　　　3.@Documented,  
　　　　4.@Inherited  
　　这些类型和它们所支持的类在java.lang.annotation包中可以找到。下面我们看一下每个元注解的作用和相应分参数的使用说明。

## ****@Target：****

　　　@Target说明了Annotation所修饰的对象范围：Annotation可被用于 packages、types（类、接口、枚举、Annotation类型）、类型成员（方法、构造方法、成员变量、枚举值）、方法参数和本地变量（如循环变量、catch参数）。在Annotation类型的声明中使用了target可更加明晰其修饰的目标。

　**作用：用于描述注解的使用范围（即：被描述的注解可以用在什么地方）**

**取值(ElementType)有：**

　　　　1.CONSTRUCTOR:用于描述构造器  
　　　　2.FIELD:用于描述域  
　　　　3.LOCAL\_VARIABLE:用于描述局部变量  
　　　　4.METHOD:用于描述方法  
　　　　5.PACKAGE:用于描述包  
　　　　6.PARAMETER:用于描述参数  
　　　　7.TYPE:用于描述类、接口(包括注解类型) 或enum声明

## ****@Retention：****

**@Retention**定义了该Annotation被保留的时间长短：某些Annotation仅出现在源代码中，而被编译器丢弃；而另一些却被编译在class文件中；编译在class文件中的Annotation可能会被虚拟机忽略，而另一些在class被装载时将被读取（请注意并不影响class的执行，因为Annotation与class在使用上是被分离的）。使用这个meta-Annotation可以对 Annotation的“生命周期”限制。

**作用：表示需要在什么级别保存该注释信息，用于描述注解的生命周期（即：被描述的注解在什么范围内有效）**

**取值（RetentionPoicy）有：**

　　　　1.SOURCE:在源文件中有效（即源文件保留）  
　　　　2.CLASS:在class文件中有效（即class保留）  
　　　　3.RUNTIME:在运行时有效（即运行时保留）

## ****@Documented:****

**@**Documented用于描述其它类型的annotation应该被作为被标注的程序成员的公共API，因此可以被例如javadoc此类的工具文档化。Documented是一个标记注解，没有成员。

## ****@Inherited：****

@Inherited 元注解是一个标记注解，@Inherited阐述了某个被标注的类型是被继承的。如果一个使用了@Inherited修饰的annotation类型被用于一个class，则这个annotation将被用于该class的子类。

　　注意：@Inherited annotation类型是被标注过的class的子类所继承。类并不从它所实现的接口继承annotation，方法并不从它所重载的方法继承annotation。

　　当@Inherited annotation类型标注的annotation的Retention是RetentionPolicy.RUNTIME，则反射API增强了这种继承性。如果我们使用java.lang.reflect去查询一个@Inherited annotation类型的annotation时，反射代码检查将展开工作：检查class和其父类，直到发现指定的annotation类型被发现，或者到达类继承结构的顶层。

# ****自定义注解：****

　　使用@interface自定义注解时，自动继承了java.lang.annotation.Annotation接口，由编译程序自动完成其他细节。在定义注解时，不能继承其他的注解或接口。@interface用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回值类型就是参数的类型（返回值类型只能是基本类型、Class、String、enum）。可以通过default来声明参数的默认值。

**定义注解格式：**  
　　public @interface 注解名 {定义体}

**注解参数的可支持数据类型：**

　　　　1.所有基本数据类型（int,float,boolean,byte,double,char,long,short)  
　　　　2.String类型  
　　　　3.Class类型  
　　　　4.enum类型  
　　　　5.Annotation类型  
　　　　6.以上所有类型的数组

　　Annotation类型里面的参数该怎么设定:   
　　第一,只能用public或默认(default)这两个访问权修饰.例如,String value();这里把方法设为defaul默认类型；　 　  
　　第二,参数成员只能用基本类型byte,short,char,int,long,float,double,boolean八种基本数据类型和 String,Enum,Class,annotations等数据类型,以及这一些类型的数组.例如,String value();这里的参数成员就为String;　　  
　　第三,如果只有一个参数成员,最好把参数名称设为"value",后加小括号.例:下面的例子FruitName注解就只有一个参数成员。

## 为什么用注解?

**优点**

配置文件

1，遵循OCP开发原则，修改配置文件即可进行功能扩展（OCP 开闭原则 Open Closed Principle）

2，集中管理对象和对象之间的组合关系，易于阅读

注解

1，开发速度快

2，编译期间容易发现错误的出处

**缺点**

配置文件

1，开发速度相对较慢；

2，编译时很难检查出错误，运行中的错误很难定位，调试难度较大。

注解

1，管理分散，基本每个类上都有；

2，扩展功能时，没有遵循OCP开发原则。

一个小特性

注解有类似继承这样的机制，A.java 实现了spring的@Component可以被注入到spring容器，但如果自定义的注解有spring的@Component注解的话，那么在具体使用这个自定义注解时候将不需要原本使用的spring注解

# 关于List与Map组合的用法

hashMap或者hashSet的高级用法

相信大家对简单的hashmap和hashset的使用都有些基础了，HashMap<k,v>中大都是**[Java](http://lib.csdn.net/base/java" \t "_blank" \o "Java 知识库)**基础类型的包装类，比如String，Integer，Byte，Character等；但是在实际应用中，我们可能需要更加复杂的**[数据结构](http://lib.csdn.net/base/datastructure" \t "_blank" \o "算法与数据结构知识库)**，比如key为一个Set、Map或者是一个类，而value同样也可以更为复杂;

## 1.直接返回List<>

**public List<Object[]> getJmtListByProductCodeList(List<String> productCodeList, Date startTime, Date endTime) {  
 if (productCodeList == null || productCodeList.size() == 0) {  
 return Lists.*newArrayList*();  
 }  
 String inProductCodes = "";  
 for (String productCode : productCodeList) {  
 inProductCodes = inProductCodes + "'" + productCode + "',";  
 }  
 inProductCodes = inProductCodes.substring(0, inProductCodes.length() - 1);  
 StringBuffer sb = new StringBuffer();  
 sb.append("SELECT ");  
 sb.append(" product\_code, ");  
 sb.append(" SUM(jmt\_price) jmt\_price, ");  
 sb.append(" SUM(jmt\_product\_count) jmt\_product\_count ");  
 sb.append("FROM ");  
 sb.append(" ord\_jmt ");  
 sb.append("WHERE ");  
 sb.append("product\_code in (" + inProductCodes + ")");  
 sb.append(" AND cancel\_item\_paid = 0");  
 if (startTime != null) {  
 sb.append(" AND jmt\_apply\_date > '" + ConvertUtil.*dateToStringE*(startTime) + "'");  
 }  
 if (endTime != null) {  
 sb.append(" AND jmt\_apply\_date < '" + ConvertUtil.*dateToStringE*(endTime) + "'");  
 }  
 sb.append(" GROUP BY product\_code");  
 List<Object[]> objList = entityManager.createNativeQuery(sb.toString()).getResultList();  
 return objList;  
}**

## 2.返回List<Map<String,Object>>组合

public List<Map<String, Object>> findJmtInfoByProductId(Long productId, String startDate, String endDate) {  
 List<Map<String, Object>> reL = new ArrayList<Map<String, Object>>();  
 StringBuffer sb = new StringBuffer();  
 sb.append("SELECT ");  
 sb.append(" b.jmt\_no, ");  
 sb.append(" b.jmt\_product\_count, ");  
 sb.append(" b.jmt\_price, ");  
 sb.append(" date\_format(b.jmt\_apply\_date, '%Y-%m-%d %H:%i:%s') jmt\_apply\_date ");  
 sb.append("FROM ord\_product\_items a, ord\_jmt b ");  
 sb.append("WHERE a.id = b.item\_id ");  
 sb.append(" AND a.item\_status = 9");  
 sb.append(" AND a.product\_id = " + productId);  
 sb.append(" AND b.jmt\_apply\_date >= date\_format('" + startDate + " 00:00:00', '%Y-%m-%d %H:%i:%s')");  
 sb.append(" AND b.jmt\_apply\_date <= date\_format('" + endDate + " 23:59:59', '%Y-%m-%d %H:%i:%s')");  
 List<Object[]> objList = entityManager.createNativeQuery(sb.toString()).getResultList();  
 for (Object[] objs : objList) {  
 Map<String, Object> childMap = Maps.*newHashMap*();  
 childMap.put("jmtNo", objs[0]);  
 childMap.put("jmtProductCount", objs[1]);  
 childMap.put("jmtPrice", objs[2]);  
 childMap.put("jmtApplyDate", objs[3]);  
 reL.add(childMap);  
 }  
 return reL;  
}

## 3.直接返回一个Map<String,String>

private Map<String, String> vitifyParam(HttpServletRequest request) {  
 // 获取支付宝POST过来反馈信息  
 Map<String, String> params = new HashMap<>();  
 @SuppressWarnings("rawtypes")  
 Map requestParams = request.getParameterMap();  
 for (@SuppressWarnings("rawtypes")  
 Iterator iter = requestParams.keySet().iterator(); iter.hasNext(); ) {  
 String name = (String) iter.next();  
 String[] values = (String[]) requestParams.get(name);  
 String valueStr = "";  
 for (int i = 0; i < values.length; i++) {  
 valueStr = (i == values.length - 1) ? valueStr + values[i]  
 : valueStr + values[i] + ",";  
 }  
 //乱码解决，这段代码在出现乱码时使用。如果mysign和sign不相等也可以使用这段代码转化  
 //valueStr = new String(valueStr.getBytes("ISO-8859-1"), "gbk");  
 params.put(name, valueStr);  
 }  
 return params;  
}

## 4.返回Map<long,HashMap<String,Double>>组合.

public Map<Long, HashMap<String, Double>> findProductJmtCntAndPrice(Long storeId, String startDate, String endDate) throws ServiceException {  
 Map<Long, HashMap<String, Double>> reM = new HashMap<Long, HashMap<String, Double>>();  
 StringBuffer sb = new StringBuffer();  
 sb.append("SELECT ");  
 sb.append(" a.product\_id, ");  
 sb.append(" sum(b.jmt\_product\_count) cnt, ");  
 sb.append(" sum(b.jmt\_price) price ");  
 sb.append("FROM ord\_product\_items a, ord\_jmt b ");  
 sb.append("WHERE a.id = b.item\_id ");  
 sb.append("AND a.whole\_sales\_type = 2 ");  
 sb.append("AND a.item\_status = 9 ");  
 sb.append("AND a.store\_id = " + storeId);  
 sb.append(" AND b.jmt\_apply\_date >= date\_format('" + startDate + " 00:00:00', '%Y-%m-%d %H:%i:%s') ");  
 sb.append("AND b.jmt\_apply\_date <= date\_format('" + endDate + " 23:59:59', '%Y-%m-%d %H:%i:%s') ");  
 sb.append("GROUP BY a.product\_id ");  
 List<Object[]> objList = entityManager.createNativeQuery(sb.toString()).getResultList();  
 for (Object[] objs : objList) {  
 HashMap<String, Double> childMap = Maps.*newHashMap*();  
 childMap.put("count", objs[1] == null ? 0D : Double.*parseDouble*(String.*valueOf*(objs[1])));  
 childMap.put("price", objs[2] == null ? 0D : Double.*parseDouble*(String.*valueOf*(objs[2])));  
 reM.put(Long.*parseLong*(String.*valueOf*(objs[0])), childMap);  
 }  
 return reM;  
}

## 5.HashSet<Map<Character,Integer> >组合

1. HashSet<Map<Character,Integer> > data = **new** HashSet<Map<Character,Integer>>();
2. **for**(**int** i=0;i<a.length;i++){
3. Map<Character,Integer> tem = **new** HashMap<Character, Integer>();
4. **for**(**int** j=0;j<a[i].length();j++)
5. **if**(tem.containsKey(a[i].charAt(j))){
6. **int** x=tem.get(a[i].charAt(j));
7. x++;
8. tem.put(a[i].charAt(j), x);
9. }**else**
10. tem.put(a[i].charAt(j), 1);
11. **if**(data.contains(tem)){
12. **continue**;
13. }**else**
14. data.add(tem);
15. }

Springboot

Springboot并非单单一个微框架的概念就可以概括,作者认为将springboot看做一种最佳实践会更为贴切:一种spring框架及其社区对”约定优先于配置”理念的最佳实践.

# Spring配置文件详解

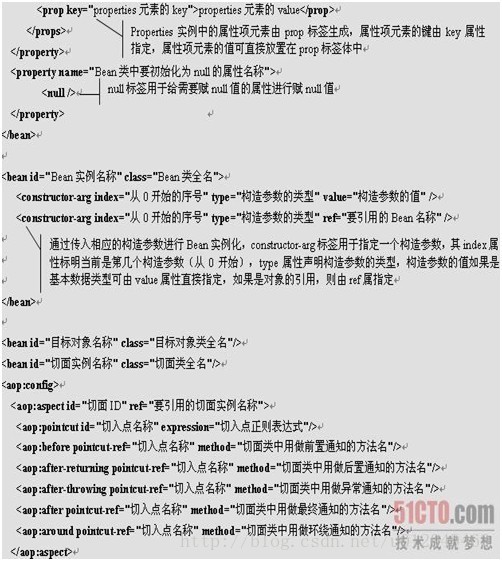
**[spring](http://lib.csdn.net/base/javaee" \t "_blank" \o "Java EE知识库)**配置文件是用于指导Spring工厂进行Bean生产、依赖关系注入（装配）及Bean实例分发的"图纸"。**[Java](http://lib.csdn.net/base/java" \t "_blank" \o "Java 知识库)**EE程序员必须学会并灵活应用这份"图纸"准确地表达自己的"生产意图"。Spring配置文件是一个或多个标准的XML文档，applicationContext.xml是Spring的默认配置文件，当容器启动时找不到指定的配置文档时，将会尝试加载这个默认的配置文件。

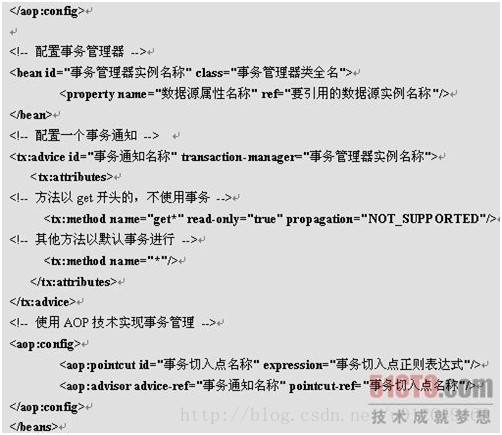
             下面列举的是一份比较完整的配置文件模板，文档中各XML标签节点的基本用途也给出了详细的解释，这些XML标签节点在后续的知识点中均会用到，熟练掌握了这些XML节点及属性的用途后，为我们动手编写配置文件打下坚实的基础。











下面，就上面的配置列举一个示例：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<!-- 定义使用C3P0连接池的数据源 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

<!-- 指定连接数据库的JDBC驱动 -->

<property name="driverClass">

<value>com.mysql.jdbc.Driver</value>

</property>

<!-- 连接数据库所用的URL -->

<property name="jdbcUrl">

<value>jdbc:mysql://localhost:3306/eportal?useUnicode=

true&characterEncoding=gbk</value>

</property>

<!-- 连接数据库的用户名 -->

<property name="user">

<value>root</value>

</property>

<!-- 连接数据库的密码 -->

<property name="password">

<value>root</value>

</property>

<!-- 设置数据库连接池的最大连接数 -->

<property name="maxPoolSize">

<value>20</value>

</property>

<!-- 设置数据库连接池的最小连接数 -->

<property name="minPoolSize">

<value>2</value>

</property>

<!-- 设置数据库连接池的初始化连接数 -->

<property name="initialPoolSize">

<value>2</value>

</property>

<!-- 设置数据库连接池的连接的最大空闲时间，单位为秒 -->

<property name="maxIdleTime">

<value>20</value>

</property>

</bean>

<!-- 定义Hibernate的SessionFactory -->

<bean id="sessionFactory"

class="org.springframework.orm.

hibernate3.LocalSessionFactoryBean">

<!-- 依赖注入上面定义的数据源dataSource -->

<property name="dataSource" ref="dataSource" />

<!-- 注册Hibernate的ORM映射文件 -->

<property name="mappingResources">

<list>

<value>com/eportal/ORM/News.hbm.xml</value>

<value>com/eportal/ORM/Category.hbm.xml</value>

<value>com/eportal/ORM/Memberlevel.hbm.xml</value>

<value>com/eportal/ORM/Cart.hbm.xml</value>

<value>com/eportal/ORM/Traffic.hbm.xml</value>

<value>com/eportal/ORM/Newsrule.hbm.xml</value>

<value>com/eportal/ORM/Merchandise.hbm.xml</value>

<value>com/eportal/ORM/Admin.hbm.xml</value>

<value>com/eportal/ORM/Orders.hbm.xml</value>

<value>com/eportal/ORM/Cartselectedmer.hbm.xml</value>

<value>com/eportal/ORM/Newscolumns.hbm.xml</value>

<value>com/eportal/ORM/Member.hbm.xml</value>

</list>

</property>

<!-- 设置Hibernate的相关属性 -->

<property name="hibernateProperties">

<props>

<!-- 设置Hibernate的数据库方言 -->

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<!-- 设置Hibernate是否在控制台输出SQL语句，开发调试阶段通常设为true -->

<prop key="show\_sql">true</prop>

<!-- 设置Hibernate一个提交批次中的最大SQL语句数 -->

<prop key="hibernate.jdbc.batch\_size">50</prop>

<prop key="show\_sql">50</prop>

</props>

</property>

</bean>

<!--定义Hibernate的事务管理器HibernateTransactionManager -->

<bean id="transactionManager"

class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<!-- 依赖注入上面定义的sessionFactory -->

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<!--定义Spring的事务拦截器TransactionInterceptor -->

<bean id="transactionInterceptor"

class="org.springframework.transaction.interceptor.TransactionInterceptor">

<!-- 依赖注入上面定义的事务管理器transactionManager -->

<property name="transactionManager" ref="transactionManager" />

<!-- 定义需要进行事务拦截的方法及所采用的事务控制类型 -->

<property name="transactionAttributes">

<props>

<!-- 以browse、list、load、get及is开头的所有方法采用只读型事务控制类型 -->

<prop key="browse\*">PROPAGATION\_REQUIRED,readOnly</prop>

<prop key="list\*">PROPAGATION\_REQUIRED,readOnly</prop>

<prop key="load\*">PROPAGATION\_REQUIRED,readOnly</prop>

<prop key="get\*">PROPAGATION\_REQUIRED,readOnly</prop>

<prop key="is\*">PROPAGATION\_REQUIRED,readOnly</prop>

<!-- 所有方法均进行事务控制，如果当前没有事务，则新建一个事务 -->

<prop key="\*">PROPAGATION\_REQUIRED</prop>

</props>

</property>

</bean>

<!-- 定义BeanNameAutoProxyCreatorf进行Spring的事务处理 -->

<bean

class="org.springframework.aop.framework.autoproxy.

BeanNameAutoProxyCreator">

<!-- 针对指定的bean自动生成业务代理 -->

<property name="beanNames">

<list>

<value>adminService</value>

<value>columnsService</value>

<value>newsService</value>

<value>crawlService</value>

<value>memberLevelService</value>

<value>memberService</value>

<value>categoryService</value>

<value>merService</value>

<value>cartService</value>

<value>ordersService</value>

<value>trafficService</value>

</list>

</property>

<!-- 这个属性为true时，表示被代理的是目标类本身而不是目标类的接口 -->

<property name="proxyTargetClass">

<value>true</value>

</property>

<!-- 依赖注入上面定义的事务拦截器transactionInterceptor -->

<property name="interceptorNames">

<list>

<value>transactionInterceptor</value>

</list>

</property>

</bean>

<!-- 装配通用数据库访问类BaseDAOImpl -->

<bean id="dao" class="com.eportal.DAO.BaseDAOImpl">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<!-- 部署系统用户管理业务逻辑组件AdminServiceImpl -->

<bean id="adminService" class="com.eportal.service.AdminServiceImpl">

<property name="dao" ref="dao" />

</bean>

<!-- 部署新闻栏目管理业务逻辑组件ColumnsServiceImpl -->

<bean id="columnsService" class="com.eportal.service.ColumnsServiceImpl">

<property name="dao" ref="dao" />

</bean>

<!-- 部署订单管理业务逻辑组件OrderServiceImpl -->

<bean id="ordersService" class="com.eportal.service.OrderServiceImpl">

<property name="dao" ref="dao" />

</bean>

<!-- 部署流量统计业务逻辑组件TrafficServiceImpl -->

<bean id="trafficService" class="com.eportal.service.TrafficServiceImpl">

<property name="dao" ref="dao" />

</bean>

<!-- 部署Struts 2负责系统用户管理的控制器AdminAction -->

<bean id="adminAction" class="com.eportal.struts.action.

AdminAction"

scope="prototype">

<property name="service" ref="adminService" />

</bean>

<!-- 部署Struts 2负责新闻栏目管理的控制器ColumnsAction -->

<bean id="columnsAction" class="com.eportal.struts.action.

ColumnsAction"

scope="prototype">

<property name="service" ref="columnsService" />

</bean>

<!-- 部署Struts 2负责新闻管理的控制器NewsAction -->

<bean id="newsAction" class="com.eportal.struts.action.

NewsAction"

scope="prototype">

<property name="service" ref="newsService" />

<property name="columnsService" ref="columnsService" />

</bean>

<!-- 部署Struts 2负责新闻采集规则管理的控制器CrawlAction -->

<bean id="crawlAction" class="com.eportal.struts.action.

CrawlAction"

scope="prototype">

<property name="service" ref="crawlService" />

<property name="columnsService" ref="columnsService" />

</bean>

</beans>

<where>

<if test=*"platformAccount !=null "*>

account = #{platformAccount, jdbcType=VARCHAR}

</if>

<if test=*"platform !=null "*>

and business = #{platform, jdbcType=VARCHAR}

</if>

<if test=*"currency !=null "*>

and currency = #{currency, jdbcType=VARCHAR}

</if>

<if test=*"month != null"*>

and MONTH(DATE) = #{month, jdbcType=VARCHAR}

</if>

</where>

GROUP BY EXTRACT(YEAR\_MONTH FROM DATE)