|  |
| --- |
| [公司名称] |
| 标题 |
| 副标题 |

|  |
| --- |
| 姚斯元  2018-10-30 |

目录

[一、 123 1](#_Toc3539499)

[1.1 1 1](#_Toc3539500)

[1.1.1 3 1](#_Toc3539501)

# 基本概念

## Spring

        Spring是一个开源框架，Spring是于2003 年兴起的一个轻量级的Java 开发框架，由Rod Johnson 在其著作Expert One-On-One J2EE Development and Design中阐述的部分理念和原型衍生而来。它是为了解决企业应用开发的复杂性而创建的。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。 简单来说，Spring是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。

## SpringMVC

        Spring MVC属于SpringFrameWork的后续产品，已经融合在Spring Web Flow里面。Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

## MyBatis

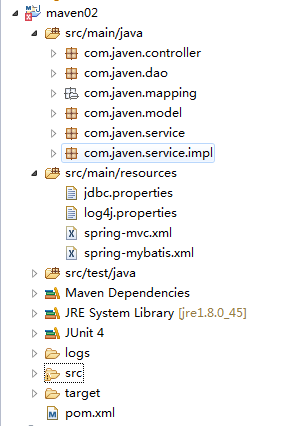
       MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。MyBatis是一个基于Java的持久层框架。iBATIS提供的持久层框架包括SQL Maps和Data Access Objects（DAO）MyBatis 消除了几乎所有的JDBC代码和参数的手工设置以及结果集的检索。MyBatis 使用简单的 XML或注解用于配置和原始映射，将接口和 Java 的POJOs（Plain Old Java Objects，普通的 Java对象）映射成数据库中的记录。

# 开发环境搭建以及创建Maven Web项目

**参看之前的博文：http://www.cnblogs.com/zyw-205520/p/4767633.html**

# SSM整合

 下面主要介绍三大框架的整合，至于环境的搭建以及项目的创建，参看上面的博文。这次整合我分了2个配置文件，分别是spring-mybatis.xml，包含spring和mybatis的配置文件，还有个是spring-mvc的配置文件，此外有2个资源文件：jdbc.propertis和log4j.properties。完整目录结构如下（最后附上源码下载地址）：



使用框架的版本：

       Spring 4.0.2 RELEASE

       Spring MVC 4.0.2 RELEASE

       MyBatis 3.2.6

## 3.1、Maven引入需要的JAR包

    在pom.xml中引入jar包

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javen.maven01</groupId>

<artifactId>maven01</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>maven01 Maven Webapp</name>

<url>http://maven.apache.org</url>

<properties>

<!-- spring版本号 -->

<spring.version>4.0.2.RELEASE</spring.version>

<!-- mybatis版本号 -->

<mybatis.version>3.2.6</mybatis.version>

<!-- log4j日志文件管理包版本 -->

<slf4j.version>1.7.7</slf4j.version>

<log4j.version>1.2.17</log4j.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<!-- 表示开发的时候引入，发布的时候不会加载此包 -->

<scope>test</scope>

</dependency>

<!-- <dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency> -->

<!-- spring核心包 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- mybatis核心包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>${mybatis.version}</version>

</dependency>

<!-- mybatis/spring包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.2</version>

</dependency>

<!-- 导入java ee jar 包 -->

<dependency>

<groupId>javax</groupId>

<artifactId>javaee-api</artifactId>

<version>7.0</version>

</dependency>

<!-- 导入Mysql数据库链接jar包 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.36</version>

</dependency>

<!-- 导入dbcp的jar包，用来在applicationContext.xml中配置数据库 -->

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.2.2</version>

</dependency>

<!-- JSTL标签类 -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- 日志文件管理包 -->

<!-- log start -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

<!-- 格式化对象，方便输出日志 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.1.41</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${slf4j.version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>${slf4j.version}</version>

</dependency>

<!-- log end -->

<!-- 映入JSON -->

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-mapper-asl</artifactId>

<version>1.9.13</version>

</dependency>

<!-- 上传组件包 -->

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.1</version>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.4</version>

</dependency>

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

<version>1.9</version>

</dependency>

</dependencies>

<build>

<finalName>maven01</finalName>

<plugins>

<plugin>

<groupId>org.eclipse.jetty</groupId>

<artifactId>jetty-maven-plugin</artifactId>

<version>9.2.8.v20150217</version>

<configuration>

<httpConnector>

<port>80</port>

</httpConnector>

<stopKey>shutdown</stopKey>

<stopPort>9966</stopPort>

</configuration>

</plugin>

</plugins>

</build>

</project>

复制代码

复制代码

[复制代码](javascript:void(0);)

## 3.2、整合SpringMVC

### 3.2.1、配置spring-mvc.xml

配置里面的注释也很详细，主要是自动扫描控制器，视图模式，注解的启动这三个。

<?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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

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

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.1.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

<!-- 自动扫描该包，使SpringMVC认为包下用了@controller注解的类是控制器 -->

<context:component-scan base-package="com.javen.controller" />

<!-- 扩充了注解驱动，可以将请求参数绑定到控制器参数 -->

<mvc:annotation-driven/>

<!-- 静态资源处理 css js imgs -->

<mvc:resources location="/resources/\*\*" mapping="/resources"/>

<!--避免IE执行AJAX时，返回JSON出现下载文件 -->

<bean id="mappingJacksonHttpMessageConverter"

class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter">

<property name="supportedMediaTypes">

<list>

<value>text/html;charset=UTF-8</value>

</list>

</property>

</bean>

<!-- 启动SpringMVC的注解功能，完成请求和注解POJO的映射 -->

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">

<property name="messageConverters">

<list>

<ref bean="mappingJacksonHttpMessageConverter" /> <!-- JSON转换器 -->

</list>

</property>

</bean>

<!-- 配置文件上传，如果没有使用文件上传可以不用配置，当然如果不配，那么配置文件中也不必引入上传组件包 -->

<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

<!-- 默认编码 -->

<property name="defaultEncoding" value="utf-8" />

<!-- 文件大小最大值 -->

<property name="maxUploadSize" value="10485760000" />

<!-- 内存中的最大值 -->

<property name="maxInMemorySize" value="40960" />

<!-- 启用是为了推迟文件解析，以便捕获文件大小异常 -->

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

</bean>

<!-- 配置ViewResolver 。可用多个ViewResolver 。使用order属性排序。 InternalResourceViewResolver 放在最后-->

<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">

<property name="order" value="1"></property>

<property name="mediaTypes">

<map>

<!-- 告诉视图解析器，返回的类型为json格式 -->

<entry key="json" value="application/json" />

<entry key="xml" value="application/xml" />

<entry key="htm" value="text/htm" />

</map>

</property>

<property name="defaultViews">

<list>

<!-- ModelAndView里的数据变成JSON -->

<bean class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />

</list>

</property>

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

</bean>

<!-- 定义跳转的文件的前后缀 ，视图模式配置-->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<!-- 这里的配置我的理解是自动给后面action的方法return的字符串加上前缀和后缀，变成一个 可用的url地址 -->

<property name="prefix" value="/WEB-INF/jsp/" />

<property name="suffix" value=".jsp" />

</bean>

</beans>

### 3.2.2、配置web.xml文件

 配置的spring-mvc的Servlet就是为了完成SpringMVC+MAVEN的整合。

web.xml

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

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

xmlns="http://java.sun.com/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

version="3.0">

<display-name>Archetype Created Web Application</display-name>

<!-- Spring和mybatis的配置文件 -->

<!-- <context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring-mybatis.xml</param-value>

</context-param> -->

<!-- 编码过滤器 -->

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<async-supported>true</async-supported>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- Spring监听器 -->

<!-- <listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener> -->

<!-- 防止Spring内存溢出监听器 -->

<!-- <listener>

<listener-class>org.springframework.web.util.IntrospectorCleanupListener</listener-class>

</listener> -->

<!-- Spring MVC servlet -->

<servlet>

<servlet-name>SpringMVC</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring-mvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

<async-supported>true</async-supported>

</servlet>

<servlet-mapping>

<servlet-name>SpringMVC</servlet-name>

<!-- 此处可以可以配置成\*.do，对应struts的后缀习惯 -->

<url-pattern>/</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>/index.jsp</welcome-file>

</welcome-file-list>

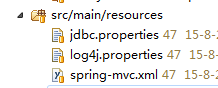
</web-app>

### 3.2.3、Log4j的配置

   为了方便调试，一般都会使用日志来输出信息，Log4j是Apache的一个开放源代码项目，通过使用Log4j，我们可以控制日志信息输送的目的地是控制台、文件、GUI组件，甚至是套接口服务器、NT的事件记录器、UNIX Syslog守护进程等；我们也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，我们能够更加细致地控制日志的生成过程。

      Log4j的配置很简单，而且也是通用的，下面给出一个基本的配置，换到其他项目中也无需做多大的调整，如果想做调整或者想了解Log4j的各种配置，参看我转载的一篇博文，很详细：<http://blog.csdn.net/zhshulin/article/details/37937365>

下面给出配置文件目录：



log4j.properties

log4j.rootLogger=INFO,Console,File

#定义日志输出目的地为控制台

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.Target=System.out

#可以灵活地指定日志输出格式，下面一行是指定具体的格式

log4j.appender.Console.layout = org.apache.log4j.PatternLayout

log4j.appender.Console.layout.ConversionPattern=[%c] - %m%n

#文件大小到达指定尺寸的时候产生一个新的文件

log4j.appender.File = org.apache.log4j.RollingFileAppender

#指定输出目录

log4j.appender.File.File = logs/ssm.log

#定义文件最大大小

log4j.appender.File.MaxFileSize = 10MB

# 输出所以日志，如果换成DEBUG表示输出DEBUG以上级别日志

log4j.appender.File.Threshold = ALL

log4j.appender.File.layout = org.apache.log4j.PatternLayout

log4j.appender.File.layout.ConversionPattern =[%p] [%d{yyyy-MM-dd HH\:mm\:ss}][%c]%m%n

### 3.2.4、使用Jetty测试

****

package com.javen.model;

public class User {

private Integer id;

private String userName;

private String password;

private Integer age;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName == null ? null : userName.trim();

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password == null ? null : password.trim();

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

@Override

public String toString() {

return "User [id=" + id + ", userName=" + userName + ", password="

+ password + ", age=" + age + "]";

}

}

package com.javen.controller;

import javax.servlet.http.HttpServletRequest;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import com.javen.model.User;

@Controller

@RequestMapping("/user")

// /user/\*\*

public class UserController {

private static Logger log=LoggerFactory.getLogger(UserController.class);

// /user/test?id=1

@RequestMapping(value="/test",method=RequestMethod.GET)

public String test(HttpServletRequest request,Model model){

int userId = Integer.parseInt(request.getParameter("id"));

System.out.println("userId:"+userId);

User user=null;

if (userId==1) {

user = new User();

user.setAge(11);

user.setId(1);

user.setPassword("123");

user.setUserName("javen");

}

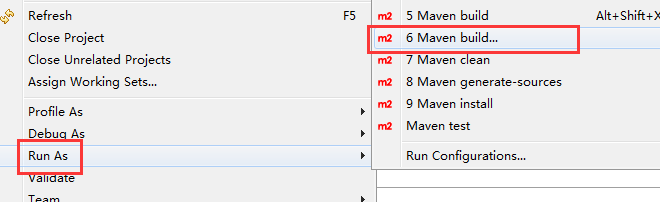
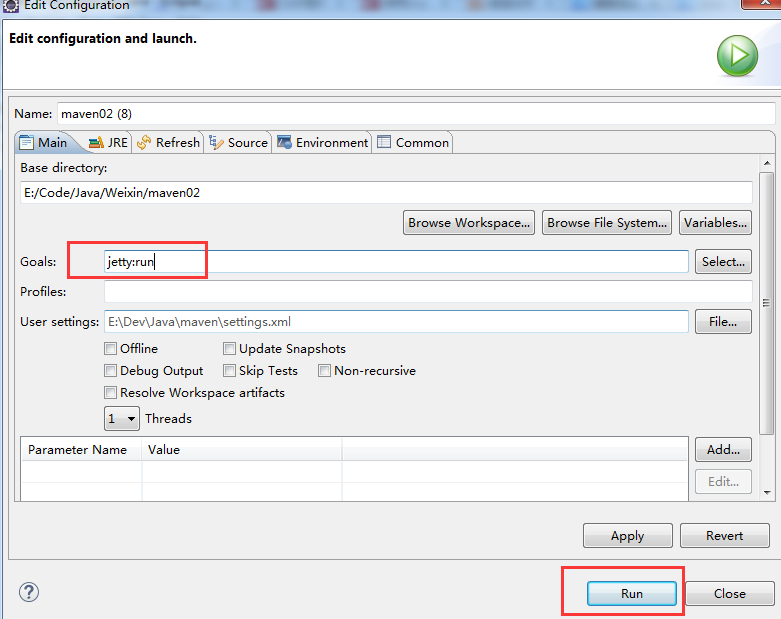
log.debug(user.toString());

model.addAttribute("user", user);

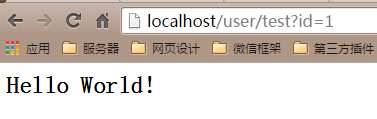
return "index";

}

}

** **

**在浏览器中输入：http://localhost/user/test?id=1**

****

**到此 SpringMVC+Maven 整合完毕**

### 3.3 Spring与MyBatis的整合

**取消3.2.2 web.xml中注释的代码**

3.3.1、建立JDBC属性文件

jdbc.properties（文件编码修改为utf-8）

driver=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/maven

username=root

password=root

#定义初始连接数

initialSize=0

#定义最大连接数

maxActive=20

#定义最大空闲

maxIdle=20

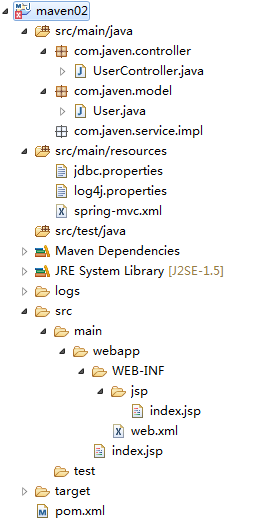
#定义最小空闲

minIdle=1

#定义最长等待时间

maxWait=60000

**此时的目录结构为**



### 3.3.2、建立spring-mybatis.xml配置文件

    这个文件就是用来完成spring和mybatis的整合的。这里面也没多少行配置，主要的就是自动扫描，自动注入，配置数据库。注释也很详细，大家看看就明白了。

spring-mybatis.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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

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

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.1.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

<!-- 自动扫描 -->

<context:component-scan base-package="com.javen" />

<!-- 引入配置文件 -->

<bean id="propertyConfigurer"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

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

</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"

destroy-method="close">

<property name="driverClassName" value="${driver}" />

<property name="url" value="${url}" />

<property name="username" value="${username}" />

<property name="password" value="${password}" />

<!-- 初始化连接大小 -->

<property name="initialSize" value="${initialSize}"></property>

<!-- 连接池最大数量 -->

<property name="maxActive" value="${maxActive}"></property>

<!-- 连接池最大空闲 -->

<property name="maxIdle" value="${maxIdle}"></property>

<!-- 连接池最小空闲 -->

<property name="minIdle" value="${minIdle}"></property>

<!-- 获取连接最大等待时间 -->

<property name="maxWait" value="${maxWait}"></property>

</bean>

<!-- spring和MyBatis完美整合，不需要mybatis的配置映射文件 -->

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

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

<!-- 自动扫描mapping.xml文件 -->

<property name="mapperLocations" value="classpath:com/javen/mapping/\*.xml"></property>

</bean>

<!-- DAO接口所在包名，Spring会自动查找其下的类 -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="basePackage" value="com.javen.dao" />

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"></property>

</bean>

<!-- (事务管理)transaction manager, use JtaTransactionManager for global tx -->

<bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

</beans>

### JUnit测试

  经过以上步骤，我们已经完成了Spring和mybatis的整合，这样我们就可以编写一段测试代码来试试是否成功了。

#### **创建测试用表**

既然我们需要测试，那么我们就需要建立在数据库中建立一个测试表，这个表建的很简单，SQL语句为：

-- ----------------------------

-- Table structure for `user\_t`

-- ----------------------------

DROP TABLE IF EXISTS `user\_t`;

CREATE TABLE `user\_t` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`user\_name` varchar(40) NOT NULL,

`password` varchar(255) NOT NULL,

`age` int(4) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of user\_t

-- ----------------------------

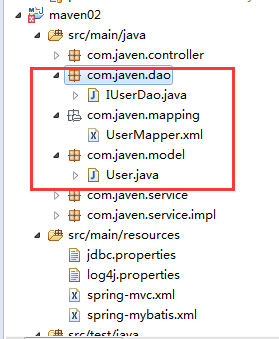
INSERT INTO `user\_t` VALUES ('1', '测试', '345', '24');

INSERT INTO `user\_t` VALUES ('2', 'javen', '123', '10');

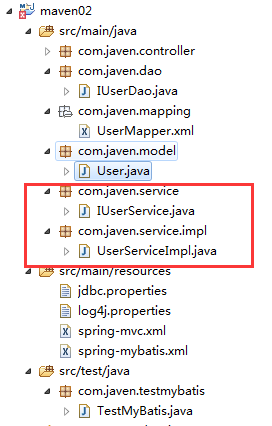
#### **利用MyBatis Generator自动创建代码**

参考博文：<http://blog.csdn.net/zhshulin/article/details/23912615>

 这个可根据表自动创建实体类、MyBatis映射文件以及DAO接口，当然，我习惯将生成的接口名改为IUserDao，而不是直接用它生成的UserMapper。如果不想麻烦就可以不改。完成后将文件复制到工程中。如图：



#### 建立Service接口和实现类

****

下面给出具体的内容：

IUserService.jave

package com.javen.service;

import com.javen.model.User;

public interface IUserService {

public User getUserById(int userId);

}

UserServiceImpl.java

package com.javen.service.impl;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;

import com.javen.dao.IUserDao;

import com.javen.model.User;

import com.javen.service.IUserService;

@Service("userService")

public class UserServiceImpl implements IUserService {

@Resource

private IUserDao userDao;

public User getUserById(int userId) {

// TODO Auto-generated method stub

return this.userDao.selectByPrimaryKey(userId);

}

}

#### **建立测试类**

 测试类在src/test/java中建立，下面测试类中注释掉的部分是不使用Spring时，一般情况下的一种测试方法；如果使用了Spring那么就可以使用注解的方式来引入配置文件和类，然后再将service接口对象注入，就可以进行测试了。

       如果测试成功，表示Spring和Mybatis已经整合成功了。输出信息使用的是Log4j打印到控制台。

package com.javen.testmybatis;

import javax.annotation.Resource;

import org.apache.log4j.Logger;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.alibaba.fastjson.JSON;

import com.javen.model.User;

import com.javen.service.IUserService;

@RunWith(SpringJUnit4ClassRunner.class) //表示继承了SpringJUnit4ClassRunner类

@ContextConfiguration(locations = {"classpath:spring-mybatis.xml"})

public class TestMyBatis {

private static Logger logger = Logger.getLogger(TestMyBatis.class);

// private ApplicationContext ac = null;

@Resource

private IUserService userService = null;

// @Before

// public void before() {

// ac = new ClassPathXmlApplicationContext("applicationContext.xml");

// userService = (IUserService) ac.getBean("userService");

// }

@Test

public void test1() {

User user = userService.getUserById(1);

// System.out.println(user.getUserName());

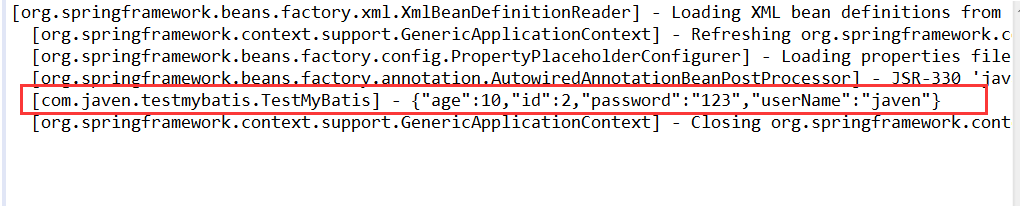
// logger.info("值："+user.getUserName());

logger.info(JSON.toJSONString(user));

}

}

测试结果



#### **3.4.5、建立UserController类**

UserController.java  控制器

package com.javen.controller;

import java.io.File;

import java.io.IOException;

import java.util.Map;

import javax.annotation.Resource;

import javax.servlet.http.HttpServletRequest;

import org.apache.commons.io.FileUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.multipart.MultipartFile;

import com.javen.model.User;

import com.javen.service.IUserService;

@Controller

@RequestMapping("/user")

// /user/\*\*

public class UserController {

private static Logger log=LoggerFactory.getLogger(UserController.class);

@Resource

private IUserService userService;

// /user/test?id=1

@RequestMapping(value="/test",method=RequestMethod.GET)

public String test(HttpServletRequest request,Model model){

int userId = Integer.parseInt(request.getParameter("id"));

System.out.println("userId:"+userId);

User user=null;

if (userId==1) {

user = new User();

user.setAge(11);

user.setId(1);

user.setPassword("123");

user.setUserName("javen");

}

log.debug(user.toString());

model.addAttribute("user", user);

return "index";

}

// /user/showUser?id=1

@RequestMapping(value="/showUser",method=RequestMethod.GET)

public String toIndex(HttpServletRequest request,Model model){

int userId = Integer.parseInt(request.getParameter("id"));

System.out.println("userId:"+userId);

User user = this.userService.getUserById(userId);

log.debug(user.toString());

model.addAttribute("user", user);

return "showUser";

}

// /user/showUser2?id=1

@RequestMapping(value="/showUser2",method=RequestMethod.GET)

public String toIndex2(@RequestParam("id") String id,Model model){

int userId = Integer.parseInt(id);

System.out.println("userId:"+userId);

User user = this.userService.getUserById(userId);

log.debug(user.toString());

model.addAttribute("user", user);

return "showUser";

}

// /user/showUser3/{id}

@RequestMapping(value="/showUser3/{id}",method=RequestMethod.GET)

public String toIndex3(@PathVariable("id")String id,Map<String, Object> model){

int userId = Integer.parseInt(id);

System.out.println("userId:"+userId);

User user = this.userService.getUserById(userId);

log.debug(user.toString());

model.put("user", user);

return "showUser";

}

// /user/{id}

@RequestMapping(value="/{id}",method=RequestMethod.GET)

public @ResponseBody User getUserInJson(@PathVariable String id,Map<String, Object> model){

int userId = Integer.parseInt(id);

System.out.println("userId:"+userId);

User user = this.userService.getUserById(userId);

log.info(user.toString());

return user;

}

// /user/{id}

@RequestMapping(value="/jsontype/{id}",method=RequestMethod.GET)

public ResponseEntity<User> getUserInJson2(@PathVariable String id,Map<String, Object> model){

int userId = Integer.parseInt(id);

System.out.println("userId:"+userId);

User user = this.userService.getUserById(userId);

log.info(user.toString());

return new ResponseEntity<User>(user,HttpStatus.OK);

}

//文件上传、

@RequestMapping(value="/upload")

public String showUploadPage(){

return "user\_admin/file";

}

@RequestMapping(value="/doUpload",method=RequestMethod.POST)

public String doUploadFile(@RequestParam("file")MultipartFile file) throws IOException{

if (!file.isEmpty()) {

log.info("Process file:{}",file.getOriginalFilename());

}

FileUtils.copyInputStreamToFile(file.getInputStream(), new File("E:\\",System.currentTimeMillis()+file.getOriginalFilename()));

return "succes";

}

}

#### **新建jsp页面**

file.jsp

<%@ page language="java" contentType="text/html; charset=utf-8"

pageEncoding="utf-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>Insert title here</title>

</head>

<body>

<h1>上传文件</h1>

<form method="post" action="/user/doUpload" enctype="multipart/form-data">

<input type="file" name="file"/>

<input type="submit" value="上传文件"/>

</form>

</body>

</html>

index.jsp

<html>

<body>

<h2>Hello World!</h2>

</body>

</html>

showUser.jsp

<%@ page language="java" import="java.util.\*" pageEncoding="utf-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>测试</title>

</head>

<body>

${user.userName}

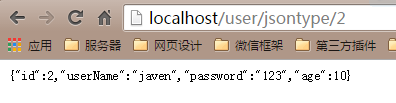
</body>

</html>

**至此，完成Spring+SpingMVC+mybatis这三大框架整合完成。**

#### 部署项目

**输入地址：http://localhost/user/jsontype/2**



  项目下载地址：<https://github.com/Javen205/SSM>

# [SSM框架下结合 log4j、slf4j打印日志](https://www.cnblogs.com/a8457013/p/7909780.html)

## 加入log4j和slf4j的jar包

<!-- 日志处理

<!-- slf4j日志包-->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.21</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.21</version>

</dependency>

<!-- 引入log4j日志包-->

<dependency>

<groupId>log4j</groupId> -->

<artifactId>log4j</artifactId>

<version>1.2.16</version>

</dependency>

Mybatis内置的日志工厂提供日志功能，具体的日志实现有以下几种工具：

SLF4J

Apache Commons Logging

Log4j 2

Log4j

JDK logging

1、具体选择哪个日志实现工具由MyBatis的内置日志工厂确定。它会使用最先找到的（按上文列举的顺序查找）。 如果一个都未找到，日志功能就会被禁用。

不少应用服务器的classpath中已经包含Commons Logging，如Tomcat和WebShpere， 所以MyBatis会把它作为具体的日志实现。记住这点非常重要。这将意味着，在诸如 WebSphere的环境中——WebSphere提供了Commons Logging的私有实现，你的Log4J配置将被忽略。 这种做法不免让人悲催，MyBatis怎么能忽略你的配置呢？事实上，因Commons Logging已经存 在了，按照优先级顺序，Log4J自然就被忽略了！不过，如果你的应用部署在一个包含Commons Logging的环境， 而你又想用其他的日志框架，你可以通过在MyBatis的配置文件mybatis-config.xml里面添加一项setting（配置）来选择一个不同的日志实现。

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

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 用于输出日志 log4j:LOG4J SLF4J -->

<settings>

<setting name="logImpl" value="SLF4J"/>

</settings>

<!-- 此文件可以为空文件, 但是必须要有这文件 -->

<!-- plugins在配置文件中的位置必须符合要求，否则会报错，顺序如下: properties?, settings?, typeAliases?,

typeHandlers?, objectFactory?,objectWrapperFactory?, plugins?, environments?,

databaseIdProvider?, mappers? -->

<plugins>

<!-- com.github.pagehelper为PageHelper类所在包名 -->

<plugin interceptor="com.github.pagehelper.PageHelper">

<!-- 4.0.0以后版本可以不设置该参数 -->

<property name="dialect" value="mysql" />

<!-- 该参数默认为false -->

<!-- 设置为true时，会将RowBounds第一个参数offset当成pageNum页码使用 -->

<!-- 和startPage中的pageNum效果一样 -->

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

<!-- 该参数默认为false -->

<!-- 设置为true时，使用RowBounds分页会进行count查询 -->

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

<!-- 设置为true时，如果pageSize=0或者RowBounds.limit = 0就会查询出全部的结果 -->

<!-- （相当于没有执行分页查询，但是返回结果仍然是Page类型） -->

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

<!-- 3.3.0版本可用 - 分页参数合理化，默认false禁用 -->

<!-- 启用合理化时，如果pageNum<1会查询第一页，如果pageNum>pages会查询最后一页 -->

<!-- 禁用合理化时，如果pageNum<1或pageNum>pages会返回空数据 -->

<!-- <property name="reasonable" value="false" /> -->

<!-- 3.5.0版本可用 - 为了支持startPage(Object params)方法 -->

<!-- 增加了一个`params`参数来配置参数映射，用于从Map或ServletRequest中取值 -->

<!-- 可以配置pageNum,pageSize,count,pageSizeZero,reasonable,orderBy,不配置映射的用默认值 -->

<!-- 不理解该含义的前提下，不要随便复制该配置 -->

<!-- <property name="params" value="pageNum=pageHelperStart;pageSize=pageHelperRows;" /> -->

<!-- 支持通过Mapper接口参数来传递分页参数 -->

<property name="supportMethodsArguments" value="false" />

<!-- always总是返回PageInfo类型,check检查返回类型是否为PageInfo,none返回Page -->

<property name="returnPageInfo" value="none" />

</plugin>

</plugins>

</configuration>

logImpl可选的值有：SLF4J、LOG4J、LOG4J2、JDK\_LOGGING、COMMONS\_LOGGING、STDOUT\_LOGGING、NO\_LOGGING 或者是实现了接口org.apache.ibatis.logging.Log的类的完全限定类名， 并且这个类的构造函数需要是以一个字符串（String类型）为参数的。 （可以参考org.apache.ibatis.logging.slf4j.Slf4jImpl.java的实现）

你根据需要调用如下的某一方法：

org.apache.ibatis.logging.LogFactory.useSlf4jLogging();

org.apache.ibatis.logging.LogFactory.useLog4JLogging();

org.apache.ibatis.logging.LogFactory.useJdkLogging();

org.apache.ibatis.logging.LogFactory.useCommonsLogging();

org.apache.ibatis.logging.LogFactory.useStdOutLogging();

如果的确需要调用以上的某个方法，请在其他所有MyBatis方法之前调用它。另外，只有在相应日志实现中存在 的前提下，调用对应的方法才是有意义的，否则MyBatis一概忽略。如你环境中并不存在Log4J，你却调用了 相应的方法，MyBatis就会忽略这一调用，代之默认的查找顺序查找日志实现。

## 配置web.xml

<!-- 加载log4j -->

<context-param>

<param-name>log4jConfigLocation</param-name>

<param-value>classpath:log4j.properties</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>

</listener>

## log4j.properties

只要在应用的classpath中创建一个名称为log4j.properties的文件， 文件的具体内容如下：

################################################################################

#①配置根Logger，其语法为：

#

#log4j.rootLogger=[level],appenderName,appenderName2,...

#level是日志记录的优先级，分为OFF,TRACE,DEBUG,INFO,WARN,ERROR,FATAL,ALL

##Log4j建议只使用四个级别，优先级从低到高分别是DEBUG,INFO,WARN,ERROR

#通过在这里定义的级别，您可以控制到应用程序中相应级别的日志信息的开关

#比如在这里定义了INFO级别，则应用程序中所有DEBUG级别的日志信息将不被打印出来

#appenderName就是指定日志信息输出到哪个地方。可同时指定多个输出目的

################################################################################

#OFF,systemOut,logFile,logDailyFile,logRollingFile,logMail,logDB,ALL

log4j.rootLogger=INFO,systemOut,DB

#输出到控制台

log4j.appender.systemOut=org.apache.log4j.ConsoleAppender

log4j.appender.systemOut.layout=org.apache.log4j.PatternLayout

log4j.appender.systemOut.layout.ConversionPattern=[%-5p][%-22d{yyyy/MM/ddHH:mm:ssS}][%l]%n%m%n

################################################################################

#指定特定包的输出特定的级别

#log4j.logger.org.springframework=DEBUG

################################################################################

#log4j.appender.systemOut.Threshold=DEBUG

log4j.appender.systemOut.ImmediateFlush=TRUE

log4j.appender.systemOut.Target=System.out

log4j.logger.org.mybatis=DEBUG

log4j.logger.com.yaosiyuan.dao=DEBUG

#mybatis显示SQL语句日志配置

#log4j.logger.org.mybatis=DEBUG

#log4j.logger.com.itzixi.mapper=DEBUG

#将日志登录到MySQL数据库

log4j.appender.DB=org.apache.log4j.jdbc.JDBCAppender

log4j.appender.DB.layout=org.apache.log4j.PatternLayout

log4j.appender.DB.Driver=com.mysql.jdbc.Driver

log4j.appender.DB.Threshold=DEBUG

# 定义字符集防止中文乱码

log4j.appender.DB.encoding=UTF-8

log4j.appender.DB.URL=jdbc:mysql://localhost:3306/ms

log4j.appender.DB.User=root

log4j.appender.DB.Password=root

#log4j.appender.DB.Sql=INSERT INTO t\_log4j(project\_name,create\_date,level,category,file\_name,thread\_name,line,all\_category,message)values('秒杀项目','%d{yyyy-MM-ddHH:mm:ss}','%p','%c','%F','%t','%L','%l','%m')

log4j.appender.DB.Sql=INSERT INTO t\_log4j(project\_name,create\_date,level,category,file\_name,thread\_name,line,all\_category,message)values("秒杀项目","%d{yyyy-MM-ddHH:mm:ss}","%p","%c","%F","%t","%L","%l","%m")

#Sql参数

#%p输出优先级，即DEBUG，INFO，WARN，ERROR，FATAL

#%r输出自应用启动到输出该log信息耗费的毫秒数

#%c输出所属的类目，通常就是所在类的全名

#%t输出产生该日志事件的线程名

#%n输出一个回车换行符，Windows平台为“rn”，Unix平台为“n”

#%d输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，比如：%d{yyyMMMddHH:mm:ss,SSS}，输出类似：2002年10月18日22：10：28，921

#%l输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数。举例：Testlog4.main(TestLog4.java:10)

#sql table

#DROP TABLE IF EXISTS `t\_log4j`;

#CREATE TABLE `t\_log4j` (

# `log\_id` int(11) NOT NULL AUTO\_INCREMENT,

# `project\_name` varchar(255) DEFAULT NULL COMMENT '目项名',

# `create\_date` varchar(255) DEFAULT NULL COMMENT '创建时间',

# `level` varchar(255) DEFAULT NULL COMMENT '优先级',

# `category` varchar(255) DEFAULT NULL COMMENT '所在类的全名',

# `file\_name` varchar(255) DEFAULT NULL COMMENT '输出日志消息产生时所在的文件名称 ',

# `thread\_name` varchar(255) DEFAULT NULL COMMENT '日志事件的线程名',

# `line` varchar(255) DEFAULT NULL COMMENT '号行',

# `all\_category` varchar(255) DEFAULT NULL COMMENT '日志事件的发生位置',

# `message` varchar(4000) DEFAULT NULL COMMENT '输出代码中指定的消息',

# PRIMARY KEY (`log\_id`)

#) ENGINE=InnoDB AUTO\_INCREMENT=170 DEFAULT CHARSET=utf8;

#用Email发送日志

log4j.appender.logMail=org.apache.log4j.net.SMTPAppender

log4j.appender.logMail.layout=org.apache.log4j.HTMLLayout

log4j.appender.logMail.layout.LocationInfo=TRUE

log4j.appender.logMail.layout.Title=MSMailLogFile

log4j.appender.logMail.Threshold=DEBUG

log4j.appender.logMail.SMTPDebug=FALSE

log4j.appender.logMail.SMTPHost=SMTP.163.com

log4j.appender.logMail.From=yaosiyuanmail@163.com

log4j.appender.logMail.To=yaosiyuanmail@gmail.com

#log4j.appender.logMail.Cc=xly3000@gmail.com

#log4j.appender.logMail.Bcc=xly3000@gmail.com

log4j.appender.logMail.SMTPUsername=yaosiyuan

log4j.appender.logMail.SMTPPassword=1234567

log4j.appender.logMail.Subject=Log4jLogMessages

#log4j.appender.logMail.BufferSize=1024

#log4j.appender.logMail.SMTPAuth=TRUE

#按DatePattern输出到文件

log4j.appender.logDailyFile=org.apache.log4j.DailyRollingFileAppender

log4j.appender.logDailyFile.layout=org.apache.log4j.PatternLayout

log4j.appender.logDailyFile.layout.ConversionPattern=[%-5p][%-22d{yyyy/MM/ddHH:mm:ssS}][%l]%n%m%n

log4j.appender.logDailyFile.Threshold=DEBUG

log4j.appender.logDailyFile.ImmediateFlush=TRUE

log4j.appender.logDailyFile.Append=TRUE

log4j.appender.logDailyFile.File=../Struts2/WebRoot/log/DailyFile/log4j\_Struts

log4j.appender.logDailyFile.DatePattern='.'yyyy-MM-dd-HH-mm'.log'

log4j.appender.logDailyFile.Encoding=UTF-8

#输出到文件

log4j.appender.logFile=org.apache.log4j.FileAppender

log4j.appender.logFile.layout=org.apache.log4j.PatternLayout

log4j.appender.logFile.layout.ConversionPattern=[%-5p][%-22d{yyyy/MM/ddHH:mm:ssS}][%l]%n%m%n

log4j.appender.logFile.Threshold=DEBUG

log4j.appender.logFile.ImmediateFlush=TRUE

log4j.appender.logFile.Append=TRUE

log4j.appender.logFile.File=../Struts2/WebRoot/log/File/log4j\_Struts.log

log4j.appender.logFile.Encoding=UTF-8

#设定文件大小输出到文件

log4j.appender.logRollingFile=org.apache.log4j.RollingFileAppender

log4j.appender.logRollingFile.layout=org.apache.log4j.PatternLayout

log4j.appender.logRollingFile.layout.ConversionPattern=[%-5p][%-22d{yyyy/MM/ddHH:mm:ssS}][%l]%n%m%n

log4j.appender.logRollingFile.Threshold=DEBUG

log4j.appender.logRollingFile.ImmediateFlush=TRUE

log4j.appender.logRollingFile.Append=TRUE

log4j.appender.logRollingFile.File=../Struts2/WebRoot/log/RollingFile/log4j\_Struts.log

log4j.appender.logRollingFile.MaxFileSize=1MB

log4j.appender.logRollingFile.MaxBackupIndex=10

log4j.appender.logRollingFile.Encoding=UTF-8

##显示SQL语句部分

#log4j.logger.com.ibatis=DEBUG

#log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG

#log4j.logger.com.ibatis.common.jdbc.ScriptRunner=DEBUG

#log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate=DEBUG

#log4j.logger.java.sql.Connection=DEBUG

#log4j.logger.java.sql.Statement=DEBUG

#log4j.logger.java.sql.PreparedStatement=DEBUG

# 定期滚动日志文件，每天都会生成日志

log4j.appender.dailyFile=org.apache.log4j.DailyRollingFileAppender

log4j.appender.dailyFile.encoding=UTF-8

log4j.appender.dailyFile.Threshold=INFO

# TODO 本地日志地址，正式环境请务必切换为阿里云地址

#log4j.appender.dailyFile.File=C:/logs/log.log4j

log4j.appender.dailyFile.File=/logs/log.log4j

log4j.appender.dailyFile.DatePattern='.'yyyy-MM-dd

log4j.appender.dailyFile.layout=org.apache.log4j.PatternLayout

log4j.appender.dailyFile.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%l] - [%p] %m%n

################################################################################

#②配置日志信息输出目的地Appender，其语法为：

#

#log4j.appender.appenderName=fully.qualified.name.of.appender.class

#log4j.appender.appenderName.optionN=valueN

#

#Log4j提供的appender有以下几种：

#1)org.apache.log4j.ConsoleAppender(输出到控制台)

#2)org.apache.log4j.FileAppender(输出到文件)

#3)org.apache.log4j.DailyRollingFileAppender(每天产生一个日志文件)

#4)org.apache.log4j.RollingFileAppender(文件大小到达指定尺寸的时候产生一个新的文件)

#5)org.apache.log4j.WriterAppender(将日志信息以流格式发送到任意指定的地方)

#

#1)ConsoleAppender选项属性

#-Threshold=DEBUG:指定日志消息的输出最低层次

#-ImmediateFlush=TRUE:默认值是true,所有的消息都会被立即输出

#-Target=System.err:默认值System.out,输出到控制台(err为红色,out为黑色)

#

#2)FileAppender选项属性

#-Threshold=INFO:指定日志消息的输出最低层次

#-ImmediateFlush=TRUE:默认值是true,所有的消息都会被立即输出

#-File=C:\log4j.log:指定消息输出到C:\log4j.log文件

#-Append=FALSE:默认值true,将消息追加到指定文件中，false指将消息覆盖指定的文件内容

#-Encoding=UTF-8:可以指定文件编码格式

#

#3)DailyRollingFileAppender选项属性

#-Threshold=WARN:指定日志消息的输出最低层次

#-ImmediateFlush=TRUE:默认值是true,所有的消息都会被立即输出

#-File=C:\log4j.log:指定消息输出到C:\log4j.log文件

#-Append=FALSE:默认值true,将消息追加到指定文件中，false指将消息覆盖指定的文件内容

#-DatePattern='.'yyyy-ww:每周滚动一次文件,即每周产生一个新的文件。还可以按用以下参数:

#'.'yyyy-MM:每月

#'.'yyyy-ww:每周

#'.'yyyy-MM-dd:每天

#'.'yyyy-MM-dd-a:每天两次

#'.'yyyy-MM-dd-HH:每小时

#'.'yyyy-MM-dd-HH-mm:每分钟

#-Encoding=UTF-8:可以指定文件编码格式

#

#4)RollingFileAppender选项属性

#-Threshold=ERROR:指定日志消息的输出最低层次

#-ImmediateFlush=TRUE:默认值是true,所有的消息都会被立即输出

#-File=C:/log4j.log:指定消息输出到C:/log4j.log文件

#-Append=FALSE:默认值true,将消息追加到指定文件中，false指将消息覆盖指定的文件内容

#-MaxFileSize=100KB:后缀可以是KB,MB,GB.在日志文件到达该大小时,将会自动滚动.如:log4j.log.1

#-MaxBackupIndex=2:指定可以产生的滚动文件的最大数

#-Encoding=UTF-8:可以指定文件编码格式

################################################################################

################################################################################

#③配置日志信息的格式(布局)，其语法为：

#

#log4j.appender.appenderName.layout=fully.qualified.name.of.layout.class

#log4j.appender.appenderName.layout.optionN=valueN

#

#Log4j提供的layout有以下几种：

#5)org.apache.log4j.HTMLLayout(以HTML表格形式布局)

#6)org.apache.log4j.PatternLayout(可以灵活地指定布局模式)

#7)org.apache.log4j.SimpleLayout(包含日志信息的级别和信息字符串)

#8)org.apache.log4j.TTCCLayout(包含日志产生的时间、线程、类别等等信息)

#9)org.apache.log4j.xml.XMLLayout(以XML形式布局)

#

#5)HTMLLayout选项属性

#-LocationInfo=TRUE:默认值false,输出java文件名称和行号

#-Title=StrutsLogMessage:默认值Log4JLogMessages

#

#6)PatternLayout选项属性

#-ConversionPattern=%m%n:格式化指定的消息(参数意思下面有)

#

#9)XMLLayout选项属性

#-LocationInfo=TRUE:默认值false,输出java文件名称和行号

#

#Log4J采用类似C语言中的printf函数的打印格式格式化日志信息，打印参数如下：

#%m输出代码中指定的消息

#%p输出优先级，即DEBUG,INFO,WARN,ERROR,FATAL

#%r输出自应用启动到输出该log信息耗费的毫秒数

#%c输出所属的类目,通常就是所在类的全名

#%t输出产生该日志事件的线程名

#%n输出一个回车换行符，Windows平台为“\r\n”，Unix平台为“\n”

#%d输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式

#如：%d{yyyy年MM月dd日HH:mm:ss,SSS}，输出类似：2012年01月05日22:10:28,921

#%l输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数

#如：Testlog.main(TestLog.java:10)

#%F输出日志消息产生时所在的文件名称

#%L输出代码中的行号

#%x输出和当前线程相关联的NDC(嵌套诊断环境),像javaservlets多客户多线程的应用中

#%%输出一个"%"字符

#

#可以在%与模式字符之间加上修饰符来控制其最小宽度、最大宽度、和文本的对齐方式。如：

#%5c:输出category名称，最小宽度是5，category<5，默认的情况下右对齐

#%-5c:输出category名称，最小宽度是5，category<5，"-"号指定左对齐,会有空格

#%.5c:输出category名称，最大宽度是5，category>5，就会将左边多出的字符截掉，<5不会有空格

#%20.30c:category名称<20补空格，并且右对齐，>30字符，就从左边交远销出的字符截掉

################################################################################

## test

import org.apache.log4j.Logger;

public class Test {

private static Logger logger = Logger.getLogger(Test.class);

public static void main(String[] args) {

logger.info("测试信息5");

}

}

## [LOG4J日志性能建议](https://somefuture.iteye.com/blog/2331383)

使用日志可能会让你的应用性能下降20% —— 很难相信吧，但是却是真的可能。

本文讨论一些尽可能提升日志性能的方法，  
  
**2** 关键设置

这里通过Junit，使用不同的Log4j配置来测试。下面会展示测试结果。实际上影响log4j性能的最主要因素有三：

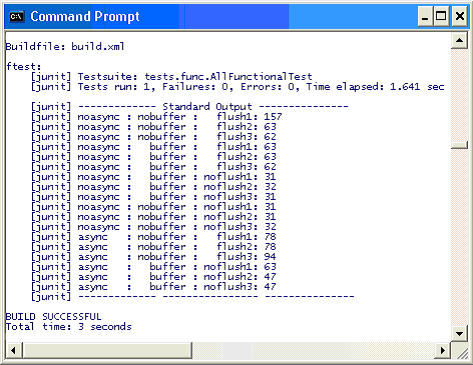
1. immediateFlush
2. bufferedIO
3. asycAppender

测试的结果可能和你想的不一样。比如，使用asyncAppender会降低本地日志文件的写入性能；bufferedIO不怎么样影响性能除非日志特别多；immediateFlush设为false几乎可以使写入时间减半。怎么回事呢？我们来看！

**3** **就是缓存**

上面三个都是使用缓存，他们分三级。第一级， immediateFlush=false 会开启 java.io.Writer/java.io.OutputStreamWriter 的缓存，消息到达后会缓存最长1024字节。第二级 bufferedIO=true 会把 java.io.Writer 包装成 java.io.BufferedWriter ，缓存默认大小是 1024\*8 字节。第三级是 **asyncAppender** ，会缓存Log4j事件，默认是128条。  
  
一般地，输出流在使用的时候其他写入会被阻塞。使用缓存后可以减少这种阻塞。但是如果本来就没有阻塞（比如在写入本地日志的时候），使用缓存反而会延迟写入。这样讲的话，使用缓存反而没提升性能。所以性能调优要看情况：你使用的是本地日志还是远程日志，什么会造成资源竞争。

**4** **测试结果**

这个测试很简单，使用log4j输出2560行，看看不同的配置使用的时间如何。我们考察上面的三个因素：  
  
   
  
每个相同的测试我们跑了三次。前三行结果是log4j的默认配置：同步+无缓存+立即刷入。第一条结果是157毫秒，我们忽略它，因为它和后两条（63毫秒，62毫秒）差距太大。  
  
接下来三行 (63, 63, 62 milliseconds) 开启了缓存但是还是立即刷入。你可能想到这没用，立即刷入让缓存无用武之地。所以结果和前面一致还是60多毫秒。  
然后我们看开启缓存后不立即刷入，也就是 immediateFlush=false，结果是(31, 32, 31 milliseconds)。差不多是前面的一半，好厉害！  
如果关闭缓存不立即刷入呢？这个和上面结果一样。  
接下来看异步日志。我们关闭缓存并且立即刷入，结果是 (78, 78, 94 milliseconds)。异步日志没有改善性能，反而降低了我们的性能。  
最后，我们使用异步日志结合缓存打开、不立即刷入，结果是 (63, 47, 47 milliseconds) ，依然没有同步的效果好。  
恩？咋回事呢？  
**5** **总结**

|  |
| --- |
| **Log4j性能建议** |
| **本地日志**:   1. 要速度快就关闭立即写入，缓存使用默认。缺点是系统崩溃的时候缓存里面没写入文件的消息都丢失了。 2. 别使用asycAppender，它实际上比同步用的时间长。因为没阻塞的时候异步逻辑比同步逻辑复杂时间更长。   **远程日志**:   1. 使用asyncAppender。只有异步队列满了才会开启和关闭一次网络连接。 2. 使用缓存bufferedIO ，它会把Writer 包装成BufferedWriter .   **注意**:   1. 如果日志带来的性能下降不严重，就不要调优。调优反而会带来其他问题。 2. 日志配置文件修改后一般不用重启就能生效。 3. 改用远程日志的时候一定要做性能测试。   **调试的时候**:   1. 使用log4j默认设置。 2. 开启缓存的时候使用tail -f 这样的命令读不到缓存中没写入的消息。 |

# 常见问题

## log4j:ERROR Failed to excute sql

异常信息

场景描述

log4j写入日志到数据库

异常原因

毫无疑问sql代码错了

解决方案

log4j.appender.DB.Sql=INSERT INTO t\_log4j(project\_name,create\_date,level,category,file\_name,thread\_name,line,all\_category,message)values('秒杀项目','%d{yyyy-MM-ddHH:mm:ss}','%p','%c','%F','%t','%L','%l','%m')

更改参数的引号 '为"

#log4j.appender.DB.Sql=INSERT INTO t\_log4j(project\_name,create\_date,level,category,file\_name,thread\_name,line,all\_category,message)values("秒杀项目","%d{yyyy-MM-ddHH:mm:ss}","%p","%c","%F","%t","%L","%l","%m")

参考资料