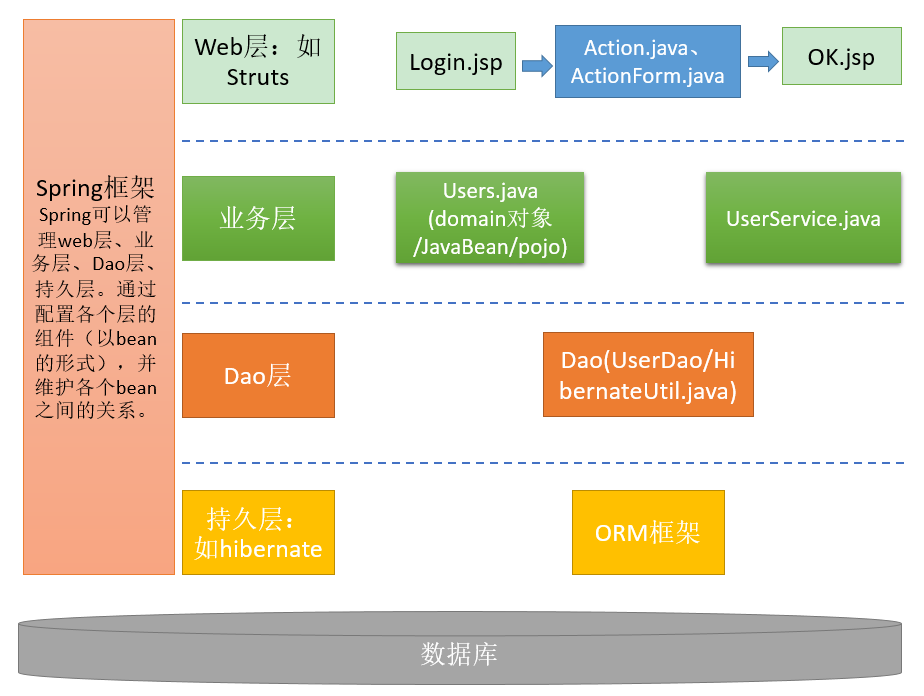
Spring

# Spring简介

Struts是web框架（jsp/action/actionfrom）。

Hibernate是ORM（object relation mapping）框架，处于持久层。

Spring属于容器框架，用于配置bean，维护bean之间的关系。



# Spring概念

Bean可以是java中的javabean/service/action/数据源/dao等等

IOC（控制反转，inverse of control）：将创建对象bean和维护对象的关系的权利从程序中转移动spring容器（applicationContext.xml），程序本身不再维护。

DI（dependency injection，依赖注入）：di和IOC实际相同，但开发者认为DI更能体现依赖关系，bean之间的关系用ref关系配置。

# 简单Spring项目

## 最小配置包

Spring包括aop、beans、core、jdbc、web等等.

写日志包commons-logging.jar.

## 创建spring核心文件applicationContext.xml

一般放置在src目录下，在maven中放到src/java/main下，此目录默认为classpath。Struts有核心struts-config.xml，hibernate有核心hibernate.cfg.xml。

applicationContext.xml路径设置方法：

1. Spring配置文件在WEB-INF根目录下，可以不进行配置，因为spring会默认去加载，如果一定要配置呢，可以这样

contextConfigLocation为WEB-INF/applicationContext.xml

2、Spring配置文件在WEB-INF下的某个文件夹下，比如config下，可以这样配置

contextConfigLocation为WEB-INF/config/applicationContext.xml

3、Spring配置文件在src根目录下面，可以这样配置

contextConfigLocation为WEB-INF/classes/applicationContext.xml

或者contextConfigLocation为classpath:applicationContext.xml

4、Spring配置文件在src下的某个包里，比如com.config，可以这样配置

contextConfigLocation为WEB-INF/classes/com/config/applicationContext.xml

或者contextConfigLocation为classpath:com/config/applicationContext.xml

## 配置bean

Bean中以id为唯一值，在java中根据id获取。可以理解成将java程序在xml加载过程中进行实例化。依赖注入使用ref依赖bean的id表示。

## Java反射

Spring的xml在加载的时候利用反射机制，和实际运行java代码相似，然后将实例化对象放入内存，但是必须要给目标的对象设置setter和getter方法，spring才能实现。

## 代码：

S1.SayByeService.java

package com.service;

public class SayByeService {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public void sayBye() {

System.out.println("Bye~ " + name);

}

}

S2.UserService.java 引用了SayByeService

package com.service;

public class UserService {

private String name;

private SayByeService sayByeService;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public SayByeService getSayByeService() {

return sayByeService;

}

public void setSayByeService(SayByeService sayByeService) {

this.sayByeService = sayByeService;

}

public void sayHello() {

System.out.println("Hello! " + name);

sayByeService.sayBye();

}

}

S3. applicationContext.xml

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

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

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

xmlns:cache="http://www.springframework.org/schema/cache" xmlns:aop="http://www.springframework.org/schema/aop"

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

xmlns:util="http://www.springframework.org/schema/util" xmlns:task="http://www.springframework.org/schema/task"

xmlns:oxm="http://www.springframework.org/schema/oxm" xmlns:tx="http://www.springframework.org/schema/tx"

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

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

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

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/task

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

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

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

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

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

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

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

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

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

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

http://www.springframework.org/schema/util/spring-util-3.1.xsd">

<!-- 配置spring bean -->

<!-- id 用于被获取，不可重复 -->

<beans:bean id="userService" class="com.service.UserService">

<!-- name 和 value体现出了注入 -->

<beans:property name="name">

<beans:value>使用spring输出的zyy</beans:value>

</beans:property>

<!-- ref 体现依赖注入 -->

<beans:property name="sayByeService" ref="sayByeServiceBean"></beans:property>

</beans:bean>

<beans:bean id="sayByeServiceBean" class="com.service.SayByeService">

<beans:property name="name" value="使用spring输出的小明"></beans:property>

</beans:bean>

</beans>

S4.test.java

package com.test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.service.SayByeService;

import com.service.UserService;

public class test {

public static void main(String[] args) {

// 传统方法调用

UserService userService = new UserService();

userService.setName("传统方法的zyy");

SayByeService sayByeService\_0 = new SayByeService();

sayByeService\_0.setName("传统方法的zyy\_小明");

userService.setSayByeService(sayByeService\_0);

userService.sayHello();

System.out.println("1 is over!");

// 传统方法调用

SayByeService sayByeService = new SayByeService();

sayByeService.setName("传统方法的小明");

//sayByeService;

sayByeService.sayBye();

System.out.println("2 is over!");

// spring的方法实例化

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

// 获取sayByeService

SayByeService sayByeService2 = (SayByeService) ac.getBean("sayByeServiceBean");

sayByeService2.sayBye();

System.out.println("3 is over!");

UserService userService2 = (UserService) ac.getBean("userService");

userService2.sayHello();

System.out.println("4 is over!");

}

}

# spring注意

spring读取的beans文件要保证唯一性，所以最好单独读取并设置成final。

# DI配合接口编程

Web层和业务层之间通过接口进行访问，实现解耦，将接口使用spring进行管理。

S1.TestMain.java

// 使用spring实现接口在beans.xml中进行转换

ApplicationContext ac = new ClassPathXmlApplicationContext("com\_2/beans.xml");

// 选择激活beans.xml中不同的changeLetter即可实现不同的接口实现，实现spring对业务控制

ChangeLetter changeLetter = (ChangeLetter) ac.getBean("changeLetter");

changeLetter.change();

System.out.println("3. spring的不同实现方法");

S2.beans.xml3

<!-- 两个 bean 的 id 不能相同，通过激活对接口 ChangeLetter 的不同的实现类，可以达到不同的业务逻辑 -->

<beans:bean id="changeLetter" class="com\_2.service.LowerLetter">

<beans:property name="string" value="Struts 3.0 "></beans:property>

</beans:bean>

<!--

<beans:bean id="changeLetter" class="com\_2.service.UpperLetter">

<beans:property name="string" value="Struts 3.0 "></beans:property>

</beans:bean>

-->

S3.ChangeLetter.java接口和两个实现类LowerLetter.java、UpperLetter.java。

# 装配bean

容纳bean的方法：bean工厂和应用上下文（ApplicationContext）。

1. Bean工厂

2. ApplicationContext

# 项目启动时启动线程

通过实现ApplicationListener接口进行启动监听。

@Component

public class StartupListener implements ApplicationListener {

@Resource

private FeedbackLuceneServiceImpl FeedbackLuceneServiceImpl;

// 工程发布的时候启用线程

@Override

public void onApplicationEvent(ApplicationEvent e) {

/\*\*

\* 只在项目启动的时候,发起一个分类判断线程

\*/

if (e instanceof ContextRefreshedEvent)

new Thread(new LuceneFeedback()).start();

}

private class LuceneFeedback implements Runnable {

@Override

public void run() {

try {

System.out.println("测试生成反馈索引");

FeedbackLuceneServiceImpl.ramFileIndexing();

FeedbackLuceneServiceImpl.query(CommonUtil.IK\_ANALYZER, "D:/tcm/usercenter/luceneFeedback","title","药物");

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

# 控制器

## 参数传递与页面返回

### ModelMap

ModelMap可以传参数，并且不需要return该map就可自动传递到前台，同时使用return返回界面。

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

public String home(ModelMap modelMap, HttpServletRequest request,

HttpServletResponse response) throws IOException {

JSONArray yearArray = adrAnalysisService.getYearStatistic();

JSONObject freqObject = adrAnalysisService.getKeyFreq();

modelMap.put("adrList", freqObject.get("adr"));

modelMap.put("drugList", freqObject.get("med"));

modelMap.put("yearAnalysisResult",

JSONFormatter.JSON2AdrYearAnalysisHighchart(yearArray));

return "adrhome";

}

### ModelAndView

ModelAndView中添加返回页面和参数，return modelAndView。

@RequestMapping(value = { "/", "/index", "/toIndex" })

public ModelAndView index(HttpServletRequest request, HttpServletResponse response) {

ModelAndView modelAndView = new ModelAndView();

List<MenuPo> rootList = MenuServiceImpl.queryRoot();

modelAndView.setViewName("index/index");

modelAndView.addObject("rootList", rootList);

return modelAndView;

}

# @Scheduled定时任务

参考资料：http://blog.csdn.net/sd4000784/article/details/7745947

## 定时任务Demo

### 开启扫描注解

spring.xml多加下面的内容

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

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

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

<!-- 启用task扫描注解 -->

<task:annotation-driven />

### 任务代码

@Component

public class Test {

@Resource

private ConstsServiceImpl constsServiceImpl;

@Scheduled(cron = "0/5 \* \* \* \* ?")//每隔5秒执行一次

public void doit(){

String logPath = constsServiceImpl.getStringByName("visitor\_log");

String logName = "visitor111.log";

try {

FileReader fr = new FileReader(logPath + logName);

BufferedReader br = new BufferedReader(fr);

String s = br.readLine();

while (s != null) {

System.err.println(s);

s = br.readLine();

}

br.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

# Spring MVC

## Service层

service一般分为interace与implement，代码结构一般是：

interface中：

baseinterface //放通用的方法，用public Object findByKey(Class class，Serializabled key)等通用类型较好

aaainterface extends baseinterface //关于aaa的方法，继承了baseinterface的方法

bbbinterface extends baseinterface

implement中：

baseimpl implements baseinterface //将接口方法实现为abstract，让aaaimpl，bbbimpl继承此类时实现了baseinterface的接口

aaaimpl extends baseimpl implements aaainterface //实现aaainterface并且继承了实现了baseinterface的baseimpl方法

bbbimpl extends baseimpl implements aaainterface

通过使用接口与继承，可以提升代码重用性，降低耦合：

interface中aaa、bbb继承了base的基础接口方法，不用写很多个接口中的方法；

implement中aaa、bbb需要实现base接口及父接口中的方法，只需继承baseimpl的实现就可以实现base接口中的方法。

调用service的时候使用接口，以支持动态加载：

aaainterface ain = new aImpl();

Aaa = ain.findByKey(Aaa.class,10);