# Java高手班-手写系列

**内参保密不得外传**

**达内Java培优方向**

**陈子枢 主讲**

****

**今日头条极速版**

**2020**

# 前言

## 框架的发展史

随着计算机硬件的高速发展386、486、586，奔腾系列PII、PIII、酷睿系列i3、i5、i7。内存从128k、256k，1m，1g，4g、8g、16g。硬盘从4g、80g、160g、500g、1t、2t、ssd高速固态硬盘，秒启win系统。

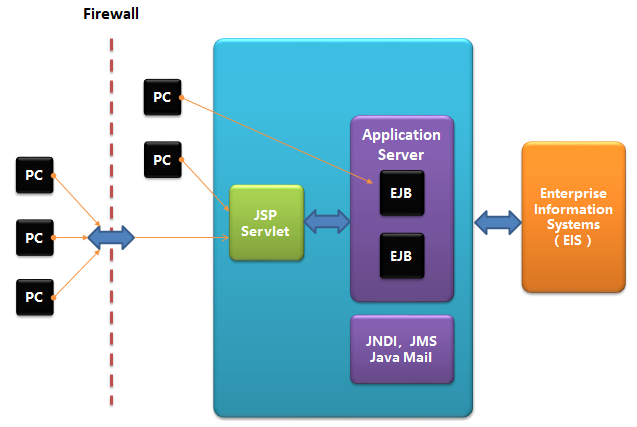


性能不断翻倍提升，个头不断变小，价格却一再低廉。1998年工资月薪几百，电脑上万，到2020年工资月薪几万，电脑几千。

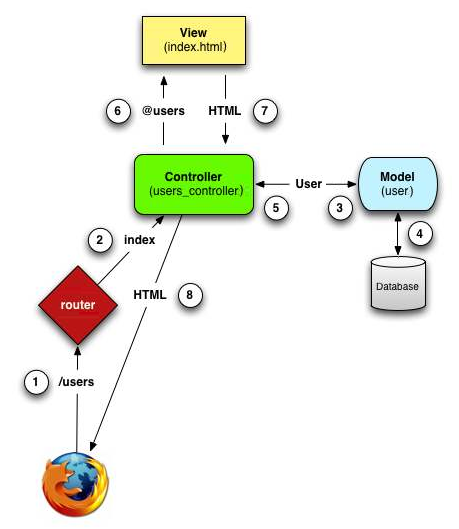


伴随硬件的高速发展，全球软件市场也爆发式增长，项目规模越来越大，业务越来越复杂，小系统已经不能满足日益发展的需求。开发速度慢，软件质量低下，用户量暴增，维护繁琐低效，C/S桌面方式萎缩，B/S浏览器互联网方式大行其道。

90年代软件在国内开始兴起，那个时代开发者都傻傻的埋头编程，恨不得一个类把所有事情做完（个体劳动）。随着OOP面向对象概念的普及，java2提出的J2EE分层体系：EJB（Enterprise JavaBean）框架的横空出世，分层体系慢慢渗透，逐渐改变程序的开发结构和组织架构（协作分工）。

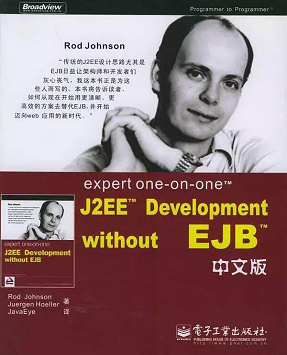


后又衍生出MVC（Model View Controller）模型并深入人心，进而又形成分层巨头，逐渐垄断各层，形成割据。



struts1、struts2负责表现层跟页面打交道，完成用户界面展现和交互，hibernate负责持久层跟数据库打交道，完成数据的存取。

随着spring作者的一本书《J2EE Development without EJB》的上市，EJB2.x刚落地就夭折了，后续推出的EJB3.0几乎无人问津。



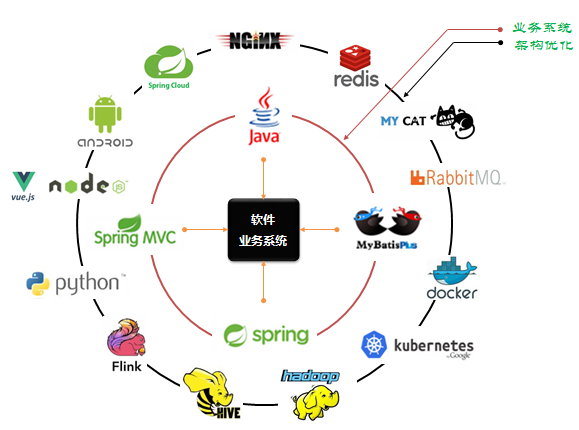
EJB开启了新时代，开启了框架framework的序幕，其分层体系被开发者广泛认可和接受。设计思想被接受，EJB代码却终结，取而代之的Spring业务层框架开始走上历史舞台，并逐渐统治全球。随着Spring的兴起，三分天下的格局逐渐形成，业界先形成了旧三大框架格局：SSH（struts2+spring+hibernate）。

时间推移，软件变革进一步加剧，SSH旧三大框架的缺点也浮出水面。Struts2配置繁琐，性能慢，安全漏洞一个又一个。hibernate完美的应用了面向对象的方式，但新潮没有解决性能问题，随着热潮退去，人们开始吐槽其逻辑的复杂和性能的差强人意，以及致命的bug漏洞。

在此风雨飘摇的关键时刻，注解编程方式兴起。这个另类开始并不被大家看好，而Spring独具慧眼，立刻全面铺开注解方式。而struts2、hibernate都没有改变。依赖此时机，Spring不仅自身变得强大，还出品了SpringMVC，它新的方式消除了struts2的繁琐xml配置，号称xml零配置，同时性能远超struts2一个数量级。

这样风光10年的struts2，在吐槽声中被SpringMVC替代。而Hibernate也没逃过厄运，天生的性能瓶颈越来越被鄙视，尽管hibernate一直在改进，可天生的毛病，怎么改？开发者不只是为OOP面向对象的实现，热潮过后回归理性，人们在思考软件开发本质是什么？简单和快捷，尤其在意性能。而hibernate做不到，渐渐的mybatis逐渐取而代之。新三大框架应运而生，鼎足而立：SSM（SpringMVC+Spring+Mybatis）。

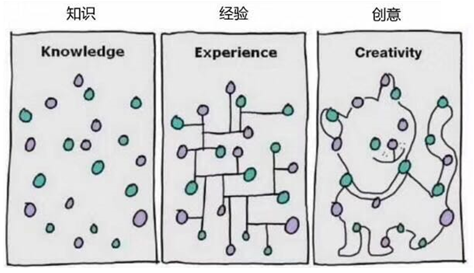
一晃新旧三大框架屹立20年，已经成为目前软件开发的核心，而且大有学习开发知识的重点已经从语言转变为框架的趋势。目前市场主流技术：大前端、移动端、微服务、互联网架构、开发运维一体化、大数据、甚至人工智能。技术浩如烟海，远比20年前门槛高的多。但不论软件技术如何发展，它都有一个天生的使命，首要任务就是实现用户的需求，完成业务的功能，而这个核心中，外圈的大前端、移动端、微服务、互联网架构、大数据、甚至人工智能都要依托内圈的技术，只有java基础和三大框架先实现了系统的业务功能，外圈的技术才能进行延伸和优化。内圈是根本，外圈是延伸，是优化，是锦上添花，一定要搞清它们之间的关系。



了解的软件框架的发展史，从中可以看到三大框架的地位之高，能力之强，乃至对现在主流技术的影响。是不心生敬佩，心生向往？学懂学透三大框架是成为一个计算机高手、高级程序员、架构师、技术总监必由之路，必会之技，你想要高薪，你想要稳定的工作，你想要有发展的工作，就必须从它开始。那还等什么，立刻开始我们的手写三大框架之旅吧！

## 论高手的修炼

现在的it技术已经成为一个庞杂的知识体系，面对这么庞杂的知识从来就不是一蹴而就，一学就会，特别经典深奥的三大框架，其中又以Spring为博大精深。把大名鼎鼎的J2EE的EJB框架干掉，把横行数十载的Struts2干掉，推出了微服务框架SpringCloud全家桶干掉了不可一世的阿里的Dubbo微服务，更是横扫全球企业级开发架构。甚至有人说现在学java核心已经不是学java，而是学习spring框架，足见其有多霸气。



那要把这么霸气的知识体系变成经验，变成创意是需要漫长的技术沉淀与持续积累。有人常问我怎么快速的学成？不要问我这样愚蠢的问题，你只要沉下心来，进行系统的学习，不要抬头东张西望，不要去钻小的问题，甚至问都不要问，想都不要想，把问题默默记录下来，记录到你的小本本上。先把老师教的学会，注意不是学懂！

当你把知识点都能朗朗上口，什么是反射？什么是注解？什么是动态代理？什么是IoC控制反转，什么是DI依赖注入？随便拿一个就能运用。这时你会发现你会了，原来只知道傻傻的练功，为什么而练懵懵的，练完有什么作用？不知道。可长期积累，滴水串石，代码写上30遍以上，也许在某个寂静的夜晚，一瞬之间你的任督二脉就打通了，啊，恍然大悟原来如此，这就是编程带给我们的愉悦，这就是佛陀所说的顿悟吧。当你经过漫长的黑暗，终于看到阳光那一刻，开心，只有开心。那一刻所有的阴霾顷刻间散去，正所谓厚积薄发，正所谓水到渠成，而在这之前你问的都是傻问题！

## 手写核心



三大框架是经典中的经典，简直就是it中的四大名著，这种东西必须拜读。但它可不是小说，精彩引人入胜。而是枯燥，如林黑夜。各不会有大篇幅先介绍来龙去脉，而是突然一个技术就蹦出来，根本不知道它为何物。同时，各种逻辑深藏其中，去实现每个细节，太繁杂晦涩难懂，底层技术穿插其中，初学者这些技术还没搞懂，更别说去搞懂框架的源码了。

所以这次我特意推出手写教程，首先一上来不是先讲三大框架，而是把框架底层这些技术一一罗列出来，通过案例讲清楚，这样在读框架源码时这些技术就不在是绊脚石了。同时，不是完全实现三大框架，而是把最核心的要点实现，起到抛砖引玉的作用，这就达到我们这次旅程的目的。

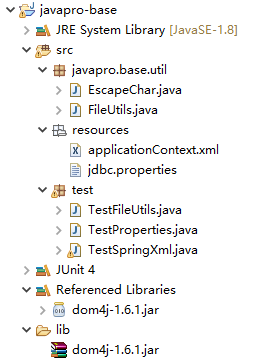
## 手写目标

1. 了解底层实现的相关技术
2. 体验真实框架实现，模拟核心关键点实现
3. 通过30遍以上的练习，达到不会说来也会吟，通过反复练习去体会其中的深意

# 基础：目录递归+路径转换+读取XML

## 基础

### 项目结构



## 转义字符

### 概念

有些字符我们能直接看到如a~z，0~9，但有些字符是看不到的，例如你敲入回车，那计算机怎么知道是个回车呢？

为了让开发者能看到这个回车，ASCII码规范中就定义了转义字符，例如：\n就代表换行。\就代表转义了。



也就是java中定义了一些字母前加"\"来表示常见的那些不能显示的ASCII字符，如\0,\t,\n等，就称为转义字符，因为后面的字符，都不是它本来的ASCII字符意思了。

### EscapeChar.java

package javapro.base.util;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:39:49

\* @description 描述：转义字符

\*/

public class EscapeChar {

public static void main(String[] args) {

System.out.println("-----");

System.out.println("我是字母：n");

// \\n，第一个斜杠是转义，第二个斜杠是斜杠字符，两个就表示输出一个斜杠

// \n，一个斜杠就代表转义，后面跟着的n就不是字母，而是换行符了

System.out.println("我不是字母，我是转义字符，我是换行符\\n：\n");

System.out.println("-----");

}

}

## 目录递归

### 概念

递归（recursion）是一种特殊的算法，有人夸它，有人贬它。不论大家如何评价，它依然神一样的存在，无可替代。

递归做为一种算法在程序设计语言中必有应用。一个过程或函数在其定义或说明中有直接或间接调用自身的一种方法，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。递归的能力在于用有限的语句来定义对象的无限集合。一般来说，递归需要有边界条件、递归前进段和递归返回段。当边界条件不满足时，递归前进；当边界条件满足时，递归结束。

### 三要素

* 自身调用自身
* 不同参数值，但处理过程一样
* 必须有结束条件，否则死循环

### 评论

高度抽象，不易理解，难以读程，后期难维护。可以从设计着手，避免使用递归。实际项目中要少用，慎用，尽量不用。它的思想比它实际的用途更让人赞叹！

### FileUtils.java

package javapro.base.util;

import java.io.File;

import java.util.List;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:49:20

\* @description 描述：利用递归遍历目录，返回里面所有文件名带路径

\*/

public class FileUtils {

/\*

\* 目录遍历开发步骤：

\* 1、用到File相关api

\* 2、先根据指定目录来创建File文件对象

\* 3、判断如果是目录，递归调用（调用自身）

\* 4、判断如果是文件，就记录到List集合中

\* 5、遍历完成，返回集合

\*/

static public List<String> getFileName(String dir, List<String> fileList){

File file = new File(dir);

//如果目录存在则处理

if( file.exists() ) {

//获取当前目录下的内容

File[] files = file.listFiles();

for(File f : files ) {

if( f.isDirectory() ) { //是目录

getFileName( f.getAbsolutePath(), fileList ); //参数值必须改变

}else { //是文件

fileList.add( f.getAbsolutePath() );

}

}

return fileList;

}

return null;

}

}

**File的关键API方法小结：**

File file = new File(path); #创建文件对象，指向一个目录

file.exists() #判断目录或者文件是否存在

File[] files = file.listFiles(); #获取当前目录下所有的目录和文件

#只获取当前目录下的内容，不含子目录

f.isDirectory() #判断是目录还是文件

f.getAbsolutePath() #获取文件的绝对路径

### TestFileUtils.java

package test;

import java.util.ArrayList;

import java.util.List;

import javapro.base.util.FileUtils;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:01:47

\* @description 描述：测试目的递归遍历

\*/

public class TestFileUtils {

public static void main(String[] args) {

//转义\\，eclipse自动帮你变成2个斜杠，傻傻的

String dir = "C:\\bak\\ACT-2020-05-NEW\\Day07-数据结构与算法基础";

List<String> fileList = new ArrayList<String>();

FileUtils.getFileName(dir, fileList);

for(String filename : fileList) {

System.out.println(filename);

}

}

}

## 路径转换

### 首字母大写

获取第一个字母变成大写，其它内容不变：

private static String toUpperFirst(String name) {

name = (""+name.charAt(0)).toUpperCase() + name.substring(1);

return name;

}

### 分离出类名

分割字符串，根据字符串中的分隔符，把结果存入一个数组，可以通过下标访问某个值。

String classFullName = "javase.base.HelloWorld";

//转义字符，正则表达式中.有特殊含义，加\转义即可

//就是告诉计算机.这里的点是分隔符的意思

//参数为正则表达式regex，先忽略后期讲，现在死记写法

String[] arr = classFullName.split("\\.");

//对象方式的for循环，实际开发非常频繁的使用，死记

for(String s: arr) {

System.out.println(s);

}

//获取最后一个部分

System.out.println("类名：" + arr[arr.length-1]);

执行结果：

javase

base

HelloWord

类名：HelloWord

### 路径转类全路径

**业务需求：**

已知文件的路径，提取转换出java的类全路径

**业务分析：**

那先要明白什么是“类文件路径”，什么是“类全路径”

类文件路径：

D:\java\ws\power7\javase\src\javase\base\HelloWorld.java

可以看出文件存储的路径和名称

类全路径：

javase.base.HelloWorld

可以看出就是package名称+类名嘛

**解决方案：**

1. 基准路径剥离D:\java\ws\power7\javase\src\
2. 去掉.java文件后缀
3. 将\替换为. 目录是\作为分隔符，包路径是.作为分隔符

代码实现：

//第一步：去掉基准路径

//项目文件的根目录

String basePath = "D:\\java\\ws\\power7\\javase\\src\\";

//类文件目录

String fileName = "D:\\java\\ws\\power7\\javase\\src\\javase\\base\\HelloWorld.java";

fileName = fileName.substring(basePath.length());

System.out.println(fileName);

//第二步：去掉后缀

if(fileName.endsWith(".java")) {

fileName = fileName.replaceAll("\\\\", "\\.");

}

System.out.println(fileName);

### 目录路径的处理

//转义字符，有些字符无法表达例如tab键，习惯就用\t表示，

//同样目录的分隔符也是\，那我们就前面在加一个\表示转义

//不转义就报错

String filename = "D:\\java\\env\\jdk1.8\\bin\\java.exe";

//参数为正则表达式regex，死记写法

String[] arr = filename.split("\\\\");

//变量对象的for循环，很好用，非常常用哦

for(String s :arr) {

System.out.println(s);

}

//最后一个部分就是文件名

System.out.println("文件名为：" + arr[arr.length-1]);

执行结果：

D:

java

env

jdk1.8

bin

java.exe

文件名为：java.exe

注意：东西有点多，转义字符、正则表达式、对象的for循环

评论：【死记】非常常用，必须牢记! 其它概念慢慢熟悉。

### 驼峰规则转换类

package javase.mybatis.util;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:42:37

\* @description 描述：驼峰规则转换

\*/

public class CoverUtil {

public static final char UNDERLINE = '\_';

//驼峰规则转成数据库下划线方式 userName user\_name

public static String camelToUnderline(String s) {

if(s ==null || "".equals(s.trim())) {

return "";

}

int len = s.length();

StringBuilder sb = new StringBuilder();

for(int i=0; i< len; i++) {

char ch = s.charAt(i);

if( Character.isUpperCase(ch)) {

sb.append(UNDERLINE);

sb.append(Character.toLowerCase(ch)); //把大写字符转换小写字符

}else {

sb.append(ch);

}

}

return sb.toString();

}

//数据库下划线方式转换驼峰规则 user\_name userName

public static String underlineToCamel(String s) {

if(s==null || "".equals(s.trim())) {

return "";

}

s = s.toLowerCase(); //不论大小写，都先强制转换小写

int len = s.length();

StringBuilder sb = new StringBuilder();

for( int i=0; i<len; i++) {

char ch = s.charAt(i);

if(ch == UNDERLINE) {

if(++i < len) { //获取下标后一个值，判断是否已经越界

sb.append(Character.toUpperCase(s.charAt(i)));

}

}else {

sb.append(ch);

}

}

return sb.toString();

}

}

## 属性文件

### 概念

Properties属性文件是比较常用的一种配置文件，纯文本，简单易读，形式简单KV结构，在JDK中有很好的支持。

### jdbc.properties

设置文件字符集为UTF-8，否则中文乱码。

#数据库链接配置

#\u6570\u636E\u5E93\u94FE\u63A5\u914D\u7F6E

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://127.0.0.1:3306/yhmisdb?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true

jdbc.username=root

jdbc.password=root

### 格式要求

* 属性文件是纯文本类型，本质就是txt文件
* #后面代表单行注释，尽量不要在属性文件中出现中文
* 每行都是kv结构，中间用=号分割
* Key不能重复，文件不报错，读取后的集合中只有最后一个的值

### TestProperties.java

读取属性文件所有内容和某一个值

package test;

import java.io.FileInputStream;

import java.util.Properties;

import org.junit.Test;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午2:26:00

\* @description 描述：读取属性配置文件

\*/

public class TestProperties {

@Test //读取文件

public void load() throws Exception {

/\*

\* 开发步骤：

\* 1、获取当前类所在的目录

\* 2、创建属性对象Properties

\* 3、读取当前类根目录下的资源目录下的属性文件

\* 4、读取属性文件内容

\* 5、关闭流

\* 6、打印全部内容

\* 7、获取单个属性

\*/

//获取当前类所在目录

String dir = TestProperties.class.getResource("/").getPath();

Properties prop = new Properties(); //创建属性文件对象

//创建输入文件流对象

FileInputStream is = new FileInputStream( dir + "/resources/jdbc.properties");

prop.load(is); //调用load方法进行文件内容的读取

is.close(); //流必须关闭

System.out.println("文件所有内容："+prop);

//读取某个值

String s = prop.getProperty("jdbc.username");

System.out.println("username的值："+s);

}

}

读取结果：

文件所有内容：{jdbc.url=jdbc:mysql://127.0.0.1:3306/yhmisdb?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true, jdbc.username=root, jdbc.driver=com.mysql.jdbc.Driver, jdbc.password=root}

username的值：root

## XML文件

### 概念

XML（eXtensible Markup Language）可拓展标记语言。

它可以用来标记数据、定义数据类型，是一种允许用户对自由定义的源语言。 它非常适合网络传输，提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。是Internet网络环境中跨平台的、依赖于内容的技术，也是当今处理分布式结构信息的有效工具。早在1998年，W3C就发布了XML1.0规范，使用它来简化Internet网络中的文档信息传输。

简而言之：就是一个纯文本语言，和txt不同的是，它引入很多标记，这样就可以从中挑选自己需要的数据，而不是全部读入。

<select id="list" resultType="Order">

SELECT \* FROM tb\_order

</select>

上面通过<select></select>标签，它有属性id和resultType，还有值“SELECT \* FROM tb\_order”，而txt文件一般就是全部读入，无法详细标识每个部分。

### 评价

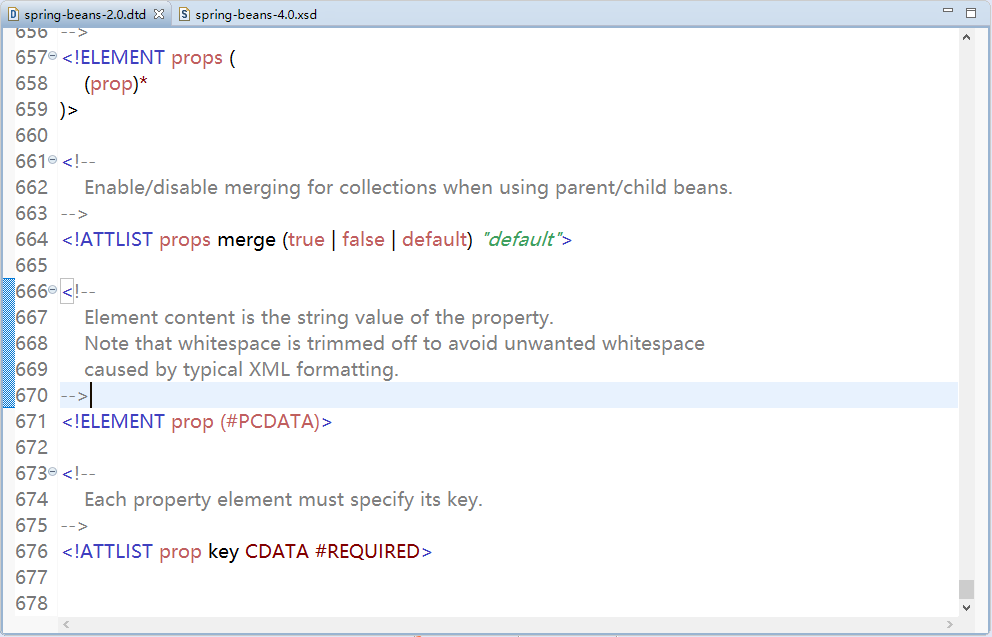
xml被后期出现的json技术所取代，所以了解会用即可，不必深究。

### DTD

DTD（Document Type Definition）文档类型定义，对标签的规范进行描述。

早期XML文件校验格式为DTD，后期改进为XSD，都是淘汰技术，了解即可。

http://www.springframework.org/dtd/spring-beans-2.0.dtd



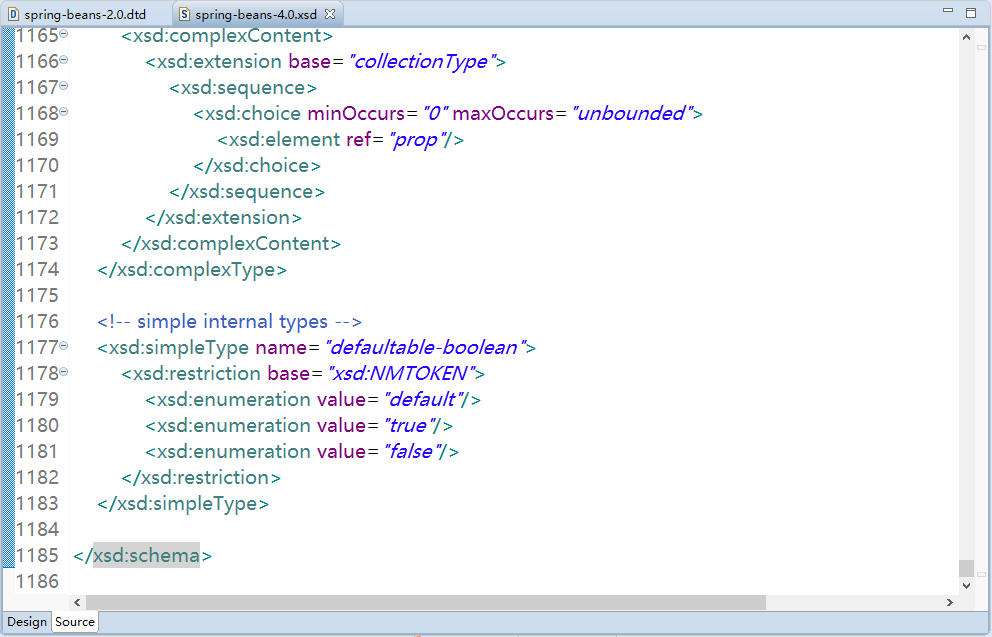
### XSD

早期XML文件校验格式为DTD，后期改进为XSD，都是淘汰技术，了解即可。

XSD（xml Schemas Definition），描述了xml文档结构。xsd本身是一个XML文档，它符合XML语法结构。可以用通用的XML解析器解析它。

Spring框架3.0版本就已经使用XSD替代掉DTD。

http://www.springframework.org/schema/beans/spring-beans-4.0.xsd



### DTD和XSD区别

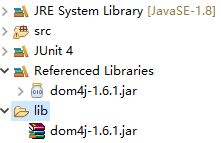
* dtd 是使用非 XML 语法编写的
* dtd不可扩展，不支持命名空间，只提供非常有限的数据类型
* xsd支持数据类型，支持命名空间

## 读取XML文件

### dom4j

dom4j是一个Java的XML API，是jdom的升级品，用来读写XML文件的。dom4j是一个十分优秀的JavaXML API，具有性能优异、功能强大和极其易使用的特点，它的性能超过sun公司官方的dom技术。完全支持DOM，SAX和JAXP解析方式，DOM是早期方式，当XML巨大时，性能低，JAVXP（Java API for [XML](https://baike.baidu.com/item/XML)Processing，XML处理的Java API）是java自身提供的xml解析api，性能差强人意，sun公司自己都不用，目前SAX方式使用居多。

导包：dom4j-1.6.1.jar



### applicationContext.xml

习惯放在resources目录下

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

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

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

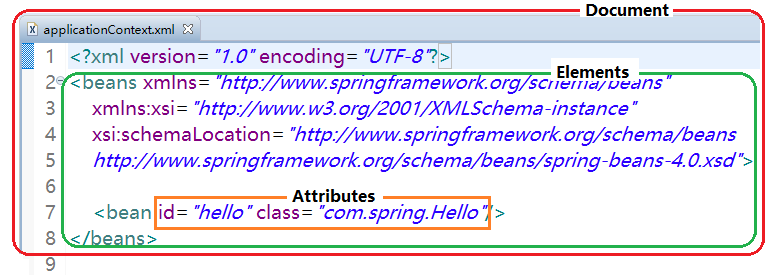
xsi:schemaLocation="http://www.springframework.org/schema/beans

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

<bean id="hello" class="com.spring.Hello"/>

</beans>

### dom的概念



### TestSpringXml.xml

package test;

import java.io.InputStream;

import java.util.Iterator;

import org.dom4j.Attribute;

import org.dom4j.Document;

import org.dom4j.Element;

import org.dom4j.io.SAXReader;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午12:34:37

\* @description 描述：dom4j读取xml文件获取<bean>标签的属性信息

\*/

public class TestSpringXml1 {

public static void main(String[] args) throws Exception {

/\*

\* 读取xml文件步骤：

\* 1、创建输入流

\* 2、创建SAXReader对象

\* 3、获取Document对象

\* 4、获取根元素

\* 5、遍历跟节点中的bean标签

\* 6、遍历每个bean标签的属性和属性值

\*/

//读取spring的核心配置文件applicationContext.xml文件，创建输入流

InputStream is = TestSpringXml.class.getResourceAsStream("/resources/applicationContext.xml");

//创建SAXReader对象，使用dom4j提供的SAX解析方式

SAXReader reader = new SAXReader();

//获取doc对象

Document doc = reader.read(is);

//获取根元素

Element root = doc.getRootElement();

//遍历跟元素，只有一个<bean>节点，可以有多个

for(Iterator<Element> els = root.elementIterator("bean"); els.hasNext(); ) {

Element ele = els.next();

//遍历当前节点的属性

for(Iterator<Attribute> attrs = ele.attributeIterator(); attrs.hasNext(); ) {

Attribute attr = attrs.next();

//获取属性名称和属性值

System.out.println(attr.getName()+"="+attr.getValue());

}

}

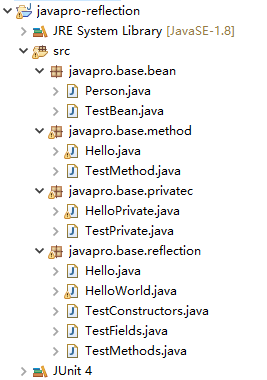
}

}

# 基础：反射+注解+动态代理+动态编程

## 反射

### 项目结构



### 概念

Reflection(反射) 是 Java 程序开发语言的特征之一，它允许运行中的 Java 程序对自身进行检查，或者说“自审”，也有称作“自省”。反射非常强大，它甚至能直接操作程序的私有属性。我们学习java基础都有一个概念，private修饰的属性只能类内部访问，外部是不行的，但这个规定被反射赤裸裸的打破了。

反射就像一面镜子，它可以在运行时动态获取一个类的所有信息，可以获取到任何定义的信息（包括成员变量，成员方法，构造方法等），并且可以操纵这些内容，访问属性，调用方法。

### 为什么需要反射？

同学们肯定会有疑问，好好的我们直接创建对象new User(); 不是很好，为什么要去通过反射的方式创建对象呢？

那得先说说静态编译和动态编译，从名字大家也能猜出来，所谓静态编译，就是我们日常的开发方式，代码提前写好，类型规划好，怎么运行，执行什么结果都在预料当中。那动态编译呢？在程序运行中我们根据动态的情况，可以动态的进行调整，这样相比静态什么都提前定死，显然程序会更加灵活。

看到这里你觉得静态编译好呢，还是动态编译好呢？

那java中如何支持这种更灵活的方式呢？java就提供了反射这样的机制，让我们可以在运行中去动态的执行代码。

一句话：java提供了一套反射机制，反射的api工具类，让我们可以动态编程，程序更加灵活。

### HelloWorld.java

package javapro.base.reflection;

import org.junit.Test;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:34:58

\* @description 描述：传统方式和反射方式比较

\*/

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello World.");

}

@Test

public void reflect() throws Exception {

HelloWorld hw1 = new HelloWorld();

System.out.print("传统方式执行：");

hw1.main(null);

//反射方式

Class<?> clazz = Class.forName("javapro.base.reflection.HelloWorld");

HelloWorld hw2 = (HelloWorld)clazz.newInstance();

System.out.print("反射方式执行：");

hw2.main(null);

}

}

从上面描述可以看出，要玩转反射，就先要对反射的api熟练，了解有哪些api，能做哪些事，具体怎么调用。

## 获取类信息

### Hello.java

一个类由三部分组成：构造方法、属性、业务方法，它们反射都可以获取并执行

package javapro.base.reflection;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午2:56:13

\* @description 描述：

\*/

public class Hello {

public String name = "tony";

public Hello() {

System.out.println("默认构造");

}

public Hello(String name) {

this.name = name;

System.out.println("带参构造："+name);

}

public void run() {

System.out.println("run");

}

public String show(String what) {

return what;

}

}

### TestConstructors.java

package javapro.base.reflection;

import java.lang.reflect.Constructor;

import org.junit.Test;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午2:57:07

\* @description 描述：

\*/

public class TestConstructors {

@Test

public void defaultContructor() throws Exception{

Class<?> clazz = Class.forName("javapro.base.reflection.Hello");

Constructor<?> c = clazz.getConstructor();

c.newInstance();

}

@Test

public void paramContructor() throws Exception {

Class<?> clazz = Class.forName("javapro.base.reflection.Hello");

Constructor<?> c = clazz.getConstructor(String.class);

c.newInstance("张居正");

}

@Test

public void all() throws Exception {

Class<?> clazz = Class.forName("javapro.base.reflection.Hello");

Constructor<?>[] cs = clazz.getConstructors();

for(Constructor<?> c : cs) {

System.out.println(c);

}

}

}

### TestFields.java

package javapro.base.reflection;

import java.lang.reflect.Field;

import org.junit.Test;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午2:59:20

\* @description 描述：

\*/

public class TestFields {

@Test

public void publicName() throws Exception {

Class<?> clazz = Class.forName("javapro.base.reflection.Hello");

Object obj = clazz.newInstance();

Field f = clazz.getField("name");

String name = (String) f.get(obj);

System.out.println(name);

}

@Test

public void all() throws Exception{

Class<?> clazz = Class.forName("javapro.base.reflection.Hello");

Object obj = clazz.newInstance();

Field[] fields = clazz.getDeclaredFields();

for(Field f : fields) {

f.setAccessible(true);

System.out.println(f);

System.out.println(f.get(obj));

}

}

}

### TestMethods.java

package javapro.base.reflection;

import java.lang.reflect.Method;

import org.junit.Test;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:00:39

\* @description 描述：

\*/

public class TestMethods {

@Test

public void method() throws Exception {

Class<?> clazz = Class.forName("javapro.base.reflection.Hello");

Object obj = clazz.newInstance();

Method m = clazz.getMethod("run");

m.invoke(obj);

Method show = clazz.getMethod("show", String.class);

String s = (String)show.invoke(obj, "哥有钱，哥自信");

System.out.println(s);

}

@Test

public void all() throws Exception {

Class<?> clazz = Class.forName("javapro.base.reflection.Hello");

Method[] ms = clazz.getMethods();

for(Method m : ms) {

System.out.println(m);

}

}

}

## 剥夺类的创建权

控制反转IoC（ Inversion of Control ），不是主动new出来，而是被动运行时创建的。

### Person.java

package javapro.base.bean;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午4:43:54

\* @description 描述：pojo对象

\*/

public class Person {

private Integer id;

private String name;

private Integer age;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

@Override

public String toString() {

return "Person [id=" + id + ", name=" + name + ", age=" + age + "]";

}

}

### TestSpringBean.java

package javapro.base.bean;

import java.lang.reflect.Method;

import org.junit.Test;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午4:45:16

\* @description 描述：

\*/

public class TestSpringBean {

@Test //传统方式

public void old() {

Person p = new Person();

p.setId(100);

p.setName("张居正");

p.setAge(55);

System.out.println("编号：" + p.getId());

System.out.println("姓名：" + p.getName());

System.out.println("年龄：" + p.getAge());

}

@Test //Spring方式，动态创建bean，通过反射设置和访问

public void spring() throws Exception {

Class<?> clazz = Class.forName("javapro.base.bean.Person");

Person p = (Person)clazz.newInstance();

Method setNameMethod = clazz.getMethod("setName", String.class);

setNameMethod.invoke(p, "高供");

Method getNameMethod = clazz.getMethod("getName");

String name = (String)getNameMethod.invoke(p);

System.out.println("姓名：" + name);

}

}

## 访问私有

### HelloPrivate.java

私有成员变量，私有方法访问

package javapro.base.privatec;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:08:15

\* @description 描述：私有属性和私有方法，反射认可访问

\*/

public class HelloPrivate {

private String name = "私有的你看不见，哈";

private void show() {

System.out.println("我是私有的方法，不让你访问");

}

}

### TestPrivate.java

package javapro.base.privatec;

import java.lang.reflect.Field;

import java.lang.reflect.Method;

import org.junit.Test;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:10:11

\* @description 描述：

\*/

public class TestPrivate {

@Test

public void privateField() throws Exception {

Class<?> clazz = Class.forName("javapro.base.privatec.HelloPrivate");

Object obj = clazz.newInstance();

Field f = clazz.getDeclaredField("name");

f.setAccessible(true);

String name = (String)f.get(obj);

System.out.println(name);

f.set(obj, "改变了吧，小样");

System.out.println(f.get(obj));

}

@Test

public void privateMethod() throws Exception{

Class<?> clazz = Class.forName("javapro.base.privatec.HelloPrivate");

Object obj = clazz.newInstance();

Method m = clazz.getDeclaredMethod("show");

m.setAccessible(true);

m.invoke(obj);

}

}

## getDeclaredMethods和getMethods的区别

### Hello.java

package javapro.base.method;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:12:28

\* @description 描述：

\*/

public class Hello {

public void hello() {

System.out.println("hello()");

}

public void hi() {

System.out.println("hi()");

}

private void welcome() {

System.out.println("welcome()");

}

}

### TestMethod.java

package javapro.base.method;

import java.lang.reflect.Method;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:12:55

\* @description 描述：

\*/

public class TestMethod {

public static void main(String[] args) throws Exception{

Class<?> clazz = Class.forName("javapro.base.method.Hello");

Method[] ms = clazz.getMethods();

for(Method m : ms) {

String methodName = m.getName();

System.out.println(methodName);

}

}

}

getMethods方法执行结果： 自己公有的，父类的

hello

hi

wait

wait

wait

equals

toString

hashCode

getClass

notify

notifyAll

getDeclaredMethods方法执行结果： 自己公有的，私有的

hello

hi

welcome

## 小结

### 创建对象三种方式

User user = new User();

User user = (User)clazz.newInstance();

Constructor<?> c = clazz.getConstructor();

User u = (User)c.newInstance();

### Declare有无的区别

* getDeclaredMethod：获取当前类的所有声明的方法，包括public、protected和private修饰的方法。需要注意的是，这些方法一定是在当前类中声明的，从父类中继承的不算，实现接口的方法由于有声明所以包括在内。
* getMethod：获取当前类和父类的所有public的方法。这里的父类，指的是继承层次中的所有父类。比如说，A继承B，B继承C，那么B和C都属于A的父类。

Field和Constructor雷同。

### 破坏面向对象的封装性

可以看到反射太强大了，我们封装为私有本意就不让其它类访问，反射却可以。但这样也破坏了面向对象提倡的封装性。好事坏事？难于言表。是不有生生相克之感，真是一物降一物。

## 注解

### 概念

注解是jdk 1.5引入的重要特性，它太优秀了，随着时间推移已经成为主流开发方式。它可以增强我们的java代码，注解配合反射技术就可以轻松扩展程序实现更多强大功能，如自动创建Bean、自动装配、配置事务、配置缓存等等。三大框架这么流行，其中最重要的功劳就是注解的助力，注解被广泛应用于三大框架底层。说的这么强大，还要配合反射这样的高难度技术，那开发时是不是很难啊！这里大家一定要记住这些增强功能是框架去实现，我们只需要加几个注解标识一下，而无需自己去写具体的实现代码。

传统我们通过xml文本文件声明方式，而现在最主流的开发都是基于注解方式，代码量少，框架可以根据注解去自动生成很多代码，从而减少代码量，程序更易读。例如最火爆的SpringCloud微服务体系就完全基于注解技术实现。

注解设计非常精巧，初学时觉得它很另类，甚至多余，甚至垃圾。有了java代码干嘛还要有@注解呢？但熟练之后你会赞叹，它竟然可以超越java代码的功能，让java代码瞬间变得强大，大家慢慢体会吧。

### 分类

注解有三类：

* JDK自带注解
* 元注解
* 自定义注解

### JDK注解

JDK提供的注解，就5个：

* @Override 重写，非常常见吧，如pojo对象时重写toString方法
* @Deprecated标记方法已经过时了，后期版本可能去除。如果就要用，风险自负
* @SuppressWarnings告诉编译器忽略指定的警告，在编译完成后不出现警告信息
* @SafeVarargs jdk1.7出现，堆污染，不常用
* @FunctionallInterface jdk1.8出现，配合函数式编程拉姆达表达式，不常用

### 元注解

描述注解的注解，就5个：

* @Target 注解用在哪里：类上、方法上、属性上，最常用
* @Retention 注解的声明周期：源文件中、class文件中、运行中，最常用
* @Inherited 允许子注解继承
* @Documented 生成javadoc时会包含注解，不常用
* @Repeatable注解为可重复类型注解，可以在同一个地方多次使用，不常用

## 元注解

### @Target

描述注解的使用范围：

* ElementType.ANNOTATION\_TYPE 应用于注释类型
* ElementType.CONSTRUCTOR 应用于构造方法
* ElementType.FIELD 应用于属性
* ElementType.LOCAL\_VARIABLE 应用于局部变量
* ElementType.METHOD 应用于方法
* ElementType.PACKAGE 应用于包声明
* ElementType.PARAMETER 应用于方法的参数
* ElementType.TYPE 应用于类

### @Retention

定义了该注解被保留的时间长短，某些注解仅出现在源代码中，而被编译器丢弃；而另一些却被编译在class文件中；编译在class文件中的注解可能会被虚拟机忽略，而另一些在class被装载时将被读取。

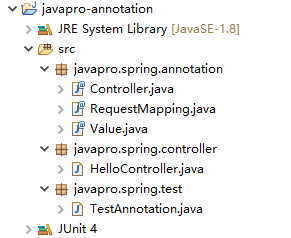
* SOURCE 在源文件中有效（即源文件保留）
* CLASS 在class文件中有效（即class保留）
* RUNTIME 在运行时有效（即运行时保留）

**为何要分有没有呢？**

本意注解的目标是在编译时增强，所以设计之初过河拆桥，在编译时用完就被干掉，这样对运行的java代码没有任何侵入和负担。但随着注解的使用，发现在动态编程时注解也能简化编程，于是将注解保留在class中，动态编程时就可以使用，如反射也可以拿到注解进行处理。

## 获取注解

### 项目结构



### @Controller.java

package javapro.spring.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:14:12

\* @description 描述：类上的注解

\*/

@Target({ElementType.TYPE}) //类上标识

@Retention(RetentionPolicy.RUNTIME) //运行时

public @interface Controller { //标识控制层类

String value() default "";

}

注意：注解的语法写法和常规java的语法写法不同

### @Value.java

package javapro.spring.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:14:12

\* @description 描述：属性上的注解

\*/

@Target(ElementType.FIELD)

@Retention(RetentionPolicy.RUNTIME)

public @interface Value {

String value() default "";

}

### @RequestMapping.java

package javapro.spring.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:14:12

\* @description 描述：方法上的注解

\*/

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface RequestMapping {

String value() default "";

}

### HelloController.java

package javapro.spring.controller;

import javapro.spring.annotation.Controller;

import javapro.spring.annotation.RequestMapping;

import javapro.spring.annotation.Value;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:15:40

\* @description 描述：模拟业务Controller类，调用自定义注解

\*/

@Controller("helloController")

public class HelloController {

@Value("tony")

private String name;

@RequestMapping("/hi")

public String hi() {

return "hi "+name;

}

}

### TestAnnotation.java

判断注解是否存在

package javapro.spring.test;

import java.lang.reflect.Field;

import java.lang.reflect.Method;

import org.junit.Test;

import javapro.spring.annotation.Controller;

import javapro.spring.annotation.RequestMapping;

import javapro.spring.annotation.Value;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:16:16

\* @description 描述：通过反射获取自定义注解

\*/

public class TestAnnotation {

@Test //获取类上的注解

public void classAnnotation() throws ClassNotFoundException {

Class<?> clazz = Class.forName("javapro.spring.controller.HelloController");

Controller c = clazz.getAnnotation(Controller.class);

if( c != null ) {

System.out.println(c);

System.out.println(c.value());

}

}

@Test //获取属性上的注解

public void fieldAnnotation() throws Exception {

Class<?> clazz = Class.forName("javapro.spring.controller.HelloController");

Field f = clazz.getDeclaredField("name");

//f.setAccessible(true);

Value v = f.getAnnotation(Value.class);

if( v != null) {

System.out.println(v);

System.out.println(v.value());

}

}

@Test //获取方法上的注解

public void methodAnnotation() throws Exception {

Class<?> clazz = Class.forName("javapro.spring.controller.HelloController");

Method m = clazz.getMethod("hi");

RequestMapping rm = m.getAnnotation(RequestMapping.class);

if( rm != null ) {

System.out.println(rm);

System.out.println(rm.value());

}

}

}

## 代码复用

Java中代码复用有三种方式：继承、组合、代理。

### 继承

继承（Extends、Implements）我们用的很多了，最好理解：

Teacher extends People; 老师是人不是神

Student extends People;

Cat implements Animal; 猫是动物，遵守动物的特性

Tiger implements Animal;

public class DispatcherServlet extends FrameworkServlet{} 框架

### 组合

组合（Component）其实我们已经用的很多了，如Controller中使用Service：

public class UserController{

@Autowired

private UserService userService;

@Autowired

private DeptService deptService;

}

### 代理

代理（roxy）

package javapro.proxy;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:07:23

\* @description 描述：血馒头

\*/

public class Trader {

public static void main(String[] args) {

Proxy proxy = new Proxy();

proxy.buy();

System.out.println("销售价：一个馒头二块");

}

}

class Proxy {

private Factory factory = new Factory();

public void buy() {

System.out.println("进货价：一个馒头五毛");

factory.sale();

}

}

class Factory {

public void sale() {

System.out.println("出厂价：一个馒头一毛");

}

}

### 继承、组合、代理的区别？

**继承和组合的区别？**

那什么时候使用继承，什么时候使用组合呢？

* 如果新类和已有类需要具有一些相似的方法和属性时，就采用继承的形式
* 如果新类只是为了借用已有类的一些方法和属性时，且两者没有很多相似之处时就采用组合的形式

**代理和组合的区别？**

代理的模式和组合有点类似，但又有差别。代理隔开了新类和已有类的直接关系，使得已有类的方法不直接暴露在新类面前，组合的方式会将已有类的非private的方法和属性直接暴露在新类中。与此同时，代理拿到了足够的好处。这也是框架推崇的代码的松耦合方式。厂家和消费者被代理隔开，消费者不直接和厂家发生关系。

## 代理模式

### 概述

按照代理的创建时期，代理类可以分为两种：

* 静态：由程序员创建代理类或特定工具自动生成源代码再对其编译，在程序运行前代理类的.class文件就已经存在了
* 动态：在程序运行时运用反射机制动态创建代理类，产生的类是在内存中的，类加载器动态加载的

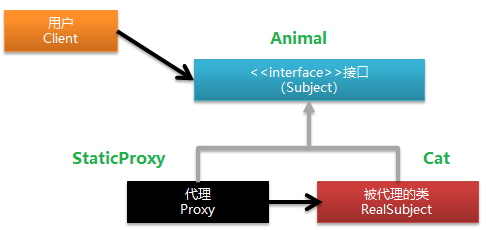
## 静态代理

### 实现过程

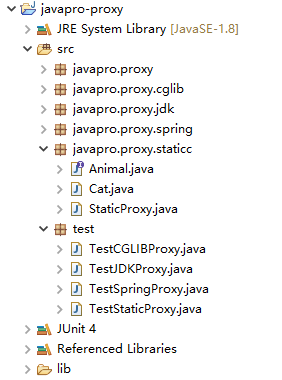
静态代理实现：

* 被代理类Cat和代理类StaticProxy实现了共同的接口Animal
* 在StaticProxy代理类中实现对被代理类的依赖（以成员变量方式体现）
* 在StaticProxy代理类的带参构造方法把被代理类作为参数传入
* 在StaticProxy代理类实现接口方法中，包裹被代理的方法，从而实现在方法前方法后加入增强的代码

最终实现在不修改被代理类的基础上，增加其他额外的功能，实现类的扩展。实现用户和被代理类的松耦合，用户依赖的是代理类，而不是具体的实现类。



### 项目结构



### Animal.java

package javapro.proxy.staticc;

public interface Animal {

public void eat();

}

### Cat.java

package javapro.proxy.staticc;

public class Cat implements Animal{

@Override

public void eat() {

System.out.println("我是猫，我喜欢吃鱼");

}

}

### StaticProxy.java

package javapro.proxy.staticc;

public class StaticProxy implements Animal{

private Cat cat;

public StaticProxy(Cat cat) {

super();

this.cat = cat;

}

@Override

public void eat() {

System.out.println("执行前");

cat.eat();

System.out.println("执行后");

}

}

### TestStaticProxy.java

package test;

import javapro.proxy.staticc.Cat;

import javapro.proxy.staticc.StaticProxy;

public class TestStaticProxy {

public static void main(String[] args) {

Cat cat = new Cat();

StaticProxy proxy = new StaticProxy(cat);

proxy.eat();

}

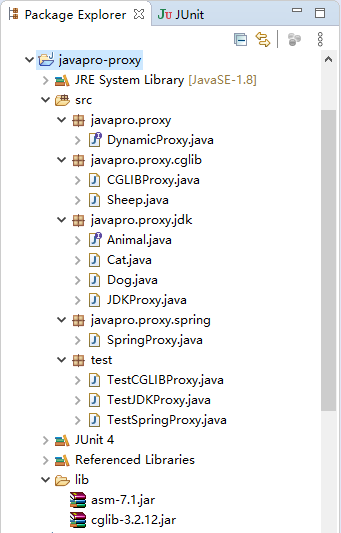
}

### 评价

静态代理实际开发中较少使用，它有个致命的缺点：如果接口添加一个新的方法，所有的实现类和代理类都要做更改，去实现这个新方法的代码，这就增加了代码的复杂度和代码的维护量，而动态代理就可以解决这个问题。所以在实际开发中很少使用静态代理，更多的是使用动态代理，特别在三大框架中动态代理被广泛应用。

## 动态代理

### 项目结构



### 概念

动态代理和静态代理，区别和显然，静态要预先写好，而动态代理是动态生成代理对象。动态代理有两大特点：一个是实现代理对象，一个是要动态实现。

传统编程中，我们都是自己的事情自己做，而代理技术出现后，别人可以帮你去做。如：生活中我们快递，原本我们要自己把东西送到客户手中，但有了快递员，他可以帮我们送过去。可见有了快递员，我们就非常方便了。

不仅如此，细细想想，他能帮你送，能不能帮别人送呢？也行是吧。动态代理也是这个好处，一处代码实现，很多地方都可以使用。而且你需要就用，你不需要随时去掉。如我就要自己送，不用快递员，难道不行吗？

这样的好处显而易见，一些公用的代码就可以通过动态代理技术实现，而这些对我们的代码没有侵入性，我们不是去改动自己的代码来满足新的要求，而且不需要时，也不用去删除这部分的代码，直接弃用代理类即可。这样的结构让我们的程序非常灵活，被很多核心底层代码广泛使用，如我们的spring业务层框架、mybatis持久层框架。

### 常见实现方式

* JDK动态代理，由java自身提供的
* CGLIB（Code Generation Library）动态代理，基于ASM小型字节码框架实现

## JDK动态代理

### 核心方法

自Java 1.3以后，JDK就提供了动态代理技术，允许开发者在运行期创建接口的代理实例。JDK动态代理主要涉及java.lang.reflect包下边的两个类：InvocationHandler接口和Proxy类。

InvoicationHandler.java

package java.lang.reflect;

/\*\*

\* @author Peter Jones

\* @see Proxy

\* @since 1.3

\*/

public interface InvocationHandler {

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable;

}

**JDK动态代理规定：**

* 实现类必须实现InvocationHandler接口
* 去实现这个接口中的唯一的方法invoke回调方法。

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable;

/\*

\* invoke的参数含义：

\* proxy: - 指所代理的那个真实对象

\* method:　 - 指所要调用真实对象的某个方法

\* args:　　 - 指调用真实对象某个方法时的参数

\*/

Proxy类的作用就是用来动态创建一个代理类对象的类，它提供了许多的方法，但是我们用的最多的就是newProxyInstance这个方法。

public static Object newProxyInstance(ClassLoader loader,

Class<?>[] interfaces,

InvocationHandler h)

throws IllegalArgumentException

/\*

\* loader: - 定义了由哪个ClassLoader对象来对生成的代理对象进行加载，保证类只加载一次

\* interfaces: - 表示被代理的对象接口

\* h: - 表示被代理对象在调用方法的时候，会关联到哪一个InvocationHandler对象上

\*/

### JDKProxy.java

下面的过程有点绕哦，多练习多思考！

* 实现InvocationHandler接口
* 调用Proxy.newProxyInstance创建代理对象
* 实现接口的invoke方法，在方法中添加需要实现的内容
* 在调用业务方法时，会自动的回调上面的invoke方法

实现的核心代码：

package javapro.proxy;

import java.lang.reflect.InvocationHandler;

import java.lang.reflect.Method;

import java.lang.reflect.Proxy;

public class JDKProxy implements InvocationHandler, DynamicProxy{

private Object target;

public Object getInstance(Object target) {

this.target = target;

//jdk底层提供的获取代理对象的方法

return Proxy.newProxyInstance(target.getClass().getClassLoader(), target.getClass().getInterfaces(), this);

}

@Override

public Object invoke(Object o, Method method, Object[] args) throws Throwable {

System.out.println("调用前");

Object proxy = method.invoke(target, args); //调用业务方法

System.out.println("调用后");

return proxy;

}

}

## CGLIB动态代理

### 核心方法

CGLIB是第三方库，它依赖ASM字节码处理框架，需要导包支持。

asm-7.1.jar

cglib-3.2.12.jar

**CGLIB规定：**

* 实现MethodInterceptor接口
* 使用Enhancer动态创建代理类，把被代理类作为父类，使用继承进行扩展

Enhancer eh = new Enhancer();

eh.setSuperclass(this.target.getClass());

eh.setCallback(this); //回调方法

return eh.create();

### CGLIBProxy.java

实现的核心代码：

package javapro.proxy;

import java.lang.reflect.Method;

import net.sf.cglib.proxy.Enhancer;

import net.sf.cglib.proxy.MethodInterceptor;

import net.sf.cglib.proxy.MethodProxy;

public class CGLIBProxy implements MethodInterceptor, DynamicProxy{

private Object target;

public Object getInstance(Object target) {

this.target = target;

Enhancer eh = new Enhancer();

eh.setSuperclass(this.target.getClass()); //设置被代理的对象，作为父类

eh.setCallback(this); //回调方法

return eh.create();

}

@Override

public Object intercept(Object o, Method method, Object[] objects, MethodProxy methodProxy) throws Throwable {

System.out.println("调用前");

Object proxy = methodProxy.invokeSuper(o, objects); //回调业务方法

System.out.println("调用后");

return proxy;

}

}

## 环境准备

### Animal.java

package javapro.proxy.jdk;

public interface Animal {

public void eat();

}

### Cat.java

package javapro.proxy.jdk;

public class Cat implements Animal{

@Override

public void eat() {

System.out.println("我是猫，我喜欢吃鱼");

}

}

### Dog.java

package javapro.proxy.jdk;

public class Dog implements Animal{

@Override

public void eat() {

System.out.println("我是狗，我喜欢啃骨头");

}

}

### DynamicProxy.java

统一接口，实现JDK动态代理和CGLIB动态代理形式相近，方便以后使用。通过getInstance()方法的参数传入被代理对象，返回代理类：

package javapro.proxy;

public interface DynamicProxy {

public Object getInstance(Object target) ;

}

## JDK动态代理案例

### JDKProxy.java

package javapro.proxy.jdk;

import java.lang.reflect.InvocationHandler;

import java.lang.reflect.Method;

import java.lang.reflect.Proxy;

import javapro.proxy.DynamicProxy;

public class JDKProxy implements InvocationHandler, DynamicProxy{

private Object target;

public Object getInstance(Object target) {

this.target = target;

//jdk底层提供的获取代理对象的方法

return Proxy.newProxyInstance(target.getClass().getClassLoader(), target.getClass().getInterfaces(), this);

}

@Override

public Object invoke(Object o, Method method, Object[] args) throws Throwable {

System.out.println("调用前");

Object proxy = method.invoke(target, args); //调用业务方法

System.out.println("调用后");

return proxy;

}

}

注意：这里Object o是指代理对象，如果需要获取代理对象的一些信息，可以通过o来调用。这是初学者容易搞不清的地方。

### TestJDKProxy.java

package test;

import org.junit.Test;

import javapro.proxy.DynamicProxy;

import javapro.proxy.jdk.Animal;

import javapro.proxy.jdk.Cat;

import javapro.proxy.jdk.Dog;

import javapro.proxy.jdk.JDKProxy;

public class TestJDKProxy {

@Test

public void cat() {

DynamicProxy proxy = new JDKProxy();

Animal cat = (Animal) proxy.getInstance( new Cat());

cat.eat();

}

@Test

public void dog() {

DynamicProxy proxy = new JDKProxy();

Animal dog = (Animal) proxy.getInstance( new Dog());

dog.eat();

}

}

## CGLIB动态代理案例

### Sheep.java

增加一个实现类，没有去实现接口的类。

package javapro.proxy.cglib;

public class Sheep {

public void eat() {

System.out.println("我是喜洋洋，我吃青草");

}

}

### CGLIBProxy.java

package javapro.proxy.cglib;

import java.lang.reflect.Method;

import javapro.proxy.DynamicProxy;

import net.sf.cglib.proxy.Enhancer;

import net.sf.cglib.proxy.MethodInterceptor;

import net.sf.cglib.proxy.MethodProxy;

public class CGLIBProxy implements MethodInterceptor, DynamicProxy{

private Object target;

public Object getInstance(Object target) {

this.target = target;

Enhancer eh = new Enhancer();

eh.setSuperclass(this.target.getClass()); //设置被代理的对象，作为父类

eh.setCallback(this); //回调方法

return eh.create();

}

/\*

\* Object o：代理对象，一般不用

\* method：业务方法

\* Object[] args：方法的参数

\* MethodProxy：方法的代理

\*/

@Override

public Object intercept(Object o, Method method, Object[] args, MethodProxy methodProxy) throws Throwable {

System.out.println("调用前");

Object proxy = methodProxy.invokeSuper(o, args); //回调业务方法

System.out.println("调用后");

return proxy;

}

}

### TestCGLIBProxy.java

package test;

import javapro.proxy.DynamicProxy;

import javapro.proxy.cglib.CGLIBProxy;

import javapro.proxy.cglib.Sheep;

public class TestCGLIBProxy {

public static void main(String[] args) {

DynamicProxy proxy = new CGLIBProxy();

Sheep sheep = (Sheep) proxy.getInstance( new Sheep());

sheep.eat();

}

}

## Spring应用动态代理

### 概述

Spring框架的AOP切面实现的底层就使用了动态代理设计模式，其中就使用了JDK动态代理和CGLIB动态代理两种方式。

如果被代理的对象有接口就使用JDK动态代理方式，如果被代理对象没有接口只有实现类则使用CGLIB动态代理方式。

### SpringProxy.java

package javapro.proxy;

import javapro.proxy.cglib.CGLIBProxy;

import javapro.proxy.jdk.JDKProxy;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:16:37

\* @description 描述：模拟实现spring框架底层使用动态代理方式

\*/

public class SpringProxy {

//工具类方法，获取动态代理对象，JDK？CGLIB？是否有接口进行判断

static public DynamicProxy getProxy(Object target) {

//被代理对象是否有接口？

Class<?> clazz = target.getClass();

Class<?>[] interfaces = clazz.getInterfaces(); //获取接口

for(Class<?> cls : interfaces) {

System.out.println("==接口========");

System.out.println(cls);

}

System.out.println();

if(interfaces.length==0) { //没有接口

return new CGLIBProxy();

}else { //有接口

return new JDKProxy();

}

}

}

### TestSpringProxy.java

下面我们就模拟Spring的底层实现：

package test;

import org.junit.Test;

import javapro.proxy.DynamicProxy;

import javapro.proxy.SpringProxy;

import javapro.proxy.cglib.Sheep;

import javapro.proxy.jdk.Animal;

import javapro.proxy.jdk.Cat;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:21:04

\* @description 描述：

\*/

public class TestSpringProxy {

@Test //有接口的情况

public void haveInterface() {

Cat target = new Cat();

DynamicProxy proxy = SpringProxy.getProxy(target);

Animal proxyClass = (Animal)proxy.getInstance(target);

proxyClass.eat();

}

@Test //无接口的情况

public void noInterface() {

Sheep target = new Sheep();

DynamicProxy proxy = SpringProxy.getProxy(target);

Sheep proxyClass = (Sheep)proxy.getInstance(target);

proxyClass.eat();

}

}

## 面试题

### 静态代理和动态代理的区别？

* 代理模式分为静态代理和动态代理两种
* 静态代理，代理类需要自己编写代码写成
* 动态代理，代理类通过 Proxy.newInstance() 方法生成
* 不管是静态代理还是动态代理，代理与被代理者都要实现同一个接口，它们的实质是面向接口编程，形成统一规范
* 静态代理和动态代理的区别是在于要不要开发者自己定义 Proxy 类
* 动态代理通过 Proxy 动态生成 proxy class，但是它也指定了一个 InvocationHandler 的实现类
* 代理模式本质上的目的是为了增强现有代码的功能，而无需改动被代理类

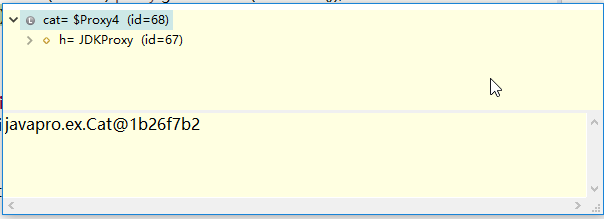
### 动态代理解决了什么问题？

首先它是一个代理机制，如果熟悉设计模式中的代理模式，我们会知道，代理可以看作是对调用目标的一个包装，这样我们对目标代码的调用不是直接发生的，而是通过代理完成，通过代理可以让调用者与实现者之间解耦。比如进行 RPC 调用，通过代理，可以提供更加友善的界面；还可以通过代理，可以实现一个全局的拦截器。

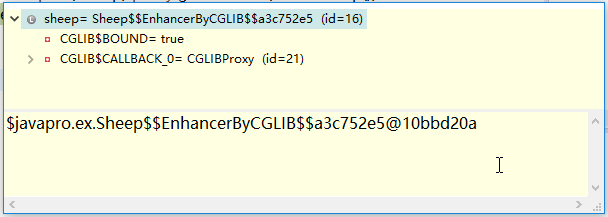
### 怎么知道是JDK代理类还是CGLIB代理类？

JDK动态代理产生的类：

通过 Proxy.newProxyInstance创建的代理对象是在jvm运行时动态生成的一个对象，命名方式都是这个形式（以$开头，Proxy在中，最后一个数字表示对象的标号）



CGLIB动态代理产生的类：



### JDK动态代理和CGLIB动态代理的区别？

JDK动态代理必须基于接口实现，无需依赖任何第三方jar包，平滑jdk升级。

CGLIB不需要接口实现，可以直接代理普通类，但需要添加第三方asm.jar的依赖，升级会有兼容性问题。本质其是作为被代理类的子类，因此代理类不能是最终类（final），否则就会报错 java.lang.IllegalArgumentException: Cannot subclass final class xxx。

### JDK动态代理为何必须要有接口？

JDK动态代理是面向接口的动态代理，它的底层实现必须传入实现类的接口，没有接口，创建代理实例方法将无法调用。

public static Object newProxyInstance(ClassLoader loader,

Class<?>[] interfaces,

InvocationHandler h)

throws IllegalArgumentException

### 性能谁高？

在早期的JDK版本中，反射性能是较低的，这时的情况下，CGLIB性能是远远高于JDK动态代理的，有人做过测试，高10倍。但CGLIB要动态创建对象，首次加载时性能就慢很多。所以在这样的情况下，对于singleton单例的代理对象或者具有实例池的代理，因为无需频繁的创建代理对象，所以比较适合采用CGLIB动态代理，反之，则比较适用JDK动态代理。

但在JDK1.8的时候，这一切变了，JDK动态代理的速度已经比CGLIB动态代理的速度快很多了。

在spring框架中，默认情况，如果有接口就采用JDK方式，如果没接口就采用CGLIB方式。

### 动态代理和反射的关系是什么？

Jdk动态代理底层是利用反射技术实现；CGLIB不是利用反射实现，而是直接剖析字节码文件，利用ASM工具包实现。

## 动态编程

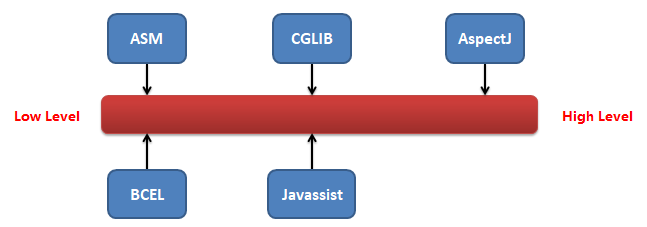
### 动态生成代码

操作java字节码的工具有两个比较流行，一个是ASM，一个是Javassit。

ASM ：直接操作字节码指令，执行效率高，要求使用者掌握Java类字节码文件格式及指令，对使用者的要求比较高。

Javassit 提供了更高级的API，执行效率相对较差，但无需掌握字节码指令的知识，对使用者要求较低。

ASM就类似低级语言汇编，性能高，没几个人会；而Javassit就类似java，高级语言，易学，易用，满大街的都是java程序员。



### Javassist

下面是mybatis框架的依赖包，可以看到mybatis底层也使用了动态代理jdk、cglib的，并且使用了更厉害的Javassist。如在延迟加载中就使用了javassist方式。



Javassist是一个开源的分析、编辑和创建Java字节码的类库。是由东京工业大学的数学和计算机科学系的 Shigeru Chiba （千叶 滋）所创建的。它已加入了开放源代码JBoss应用服务器项目，通过使用Javassist对字节码操作为JBoss实现动态AOP框架。javassist是jboss的一个子项目，其主要的优点在于简单，而且快速。直接使用java编码的形式，而不需要了解虚拟机指令，就能动态改变类的结构，或者动态生成类。

### 核心方法说明

官网手册：

http://www.javassist.org/tutorial/tutorial.html

CtClass 表示 class 文件。

ClassPool是 CtClass 对象的容器，它按需读取类文件来构造 CtClass 对象，并且保存 CtClass 对象以便以后使用。

注意的是 ClassPool 会在内存中维护所有被它创建过的 CtClass，当 CtClass 数量过多时，会占用大量的内存，解决方案是如不再使用，调用CtClass的detach()方法释放内存。

**ClassPool的方法：**

* getDefault : 返回默认的ClassPool 是单例模式的，一般通过该方法创建我们的ClassPool；
* appendClassPath, insertClassPath : 将一个ClassPath加到类搜索路径的末尾位置 或 插入到起始位置。通常通过该方法写入额外的类搜索路径，以解决多个类加载器环境中找不到类的尴尬；
* toClass : 将修改后的CtClass加载至当前线程的上下文类加载器中，CtClass的toClass方法是通过调用本方法实现。需要注意的是一旦调用该方法，则无法继续修改已经被加载的class，类会被冻结；
* get , getCtClass : 根据类路径名获取该类的CtClass对象，用于后续的编辑。

**CtClass的方法：**

* freeze : 冻结一个类，使其不可修改
* isFrozen : 判断一个类是否已被冻结
* prune : 删除类不必要的属性，以减少内存占用。调用该方法后，许多方法将无法正常使用，慎用
* defrost : 解冻一个类，使其可以被修改。如果事先知道一个类会被defrost， 则禁止调用 prune 方法
* detach : 将该class从ClassPool中删除
* writeFile : 根据CtClass生成 .class 文件
* toClass : 通过类加载器加载该CtClass

上面我们创建一个新的方法使用了CtMethod类。CtMthod代表类中的某个方法，可以通过CtClass提供的API获取或者CtNewMethod新建，通过CtMethod对象可以实现对方法的修改。

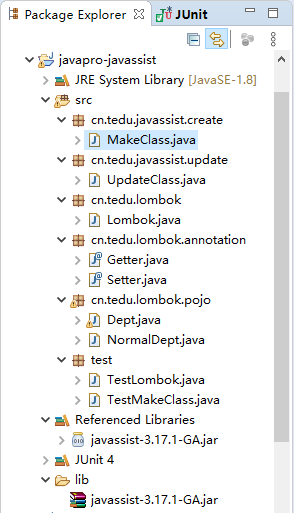
**CtMethod中的一些重要方法：**

* insertBefore : 在方法的起始位置插入代码
* insterAfter : 在方法的所有 return 语句前插入代码以确保语句能够被执行，除非遇到exception
* insertAt : 在指定的位置插入代码
* setBody : 将方法的内容设置为要写入的代码，当方法被 abstract修饰时，该修饰符被移除
* make : 创建一个新的方法

## Javassist入门

### 项目结构

导包：javassist.jar



### MakeClass.java

package cn.tedu.javassist.create;

import javassist.ClassPool;

import javassist.CtClass;

import javassist.CtConstructor;

import javassist.CtField;

import javassist.CtMethod;

import javassist.CtNewConstructor;

import javassist.CtNewMethod;

import javassist.Modifier;

import javassist.bytecode.AccessFlag;

//利用javassist技术动态创建新类

public class MakeClass {

public static void main(String[] args) throws Exception {

ClassPool pool = ClassPool.getDefault();

CtClass ct = pool.makeClass("cn.tedu.hello.HelloWorld"); //创建类

System.out.println(ct.getURL().getPath());

System.out.println(ct.getName()); // 获取类名

System.out.println("\tpackage " + ct.getPackageName()); // 获取包名

System.out.print("\t" + Modifier.toString(ct.getModifiers()) + " class " + ct.getSimpleName()); // 获取限定符和简要类名

System.out.print(" extends " + ct.getSuperclass().getName()); // 获取超类

//添加属性

CtField fId = new CtField(CtClass.intType, "id", ct); //创建int类型字段id

fId.setModifiers(AccessFlag.PUBLIC);

ct.addField(fId); //绑定字段到类上

CtField fName = new CtField(pool.get(String.class.getName()), "name", ct); //创建int类型字段id

fName.setModifiers(AccessFlag.PRIVATE);

ct.addField(fName); //绑定字段到类上

//添加构造方法

CtConstructor constructor = CtNewConstructor.make("public HelloWorld(){ System.out.println(\"我是默认构造方法\"); }", ct);

ct.addConstructor(constructor);

//添加有参的构造函数

CtConstructor conparam = new CtConstructor(new CtClass[]{pool.get("java.lang.String")}, ct);

// $0=this / $1,$2,$3... 代表方法参数

conparam.setBody("{$0.name = $1;}");

ct.addConstructor(conparam);

//添加方法

CtMethod hiMethod = CtNewMethod.make("public void hi(String name){ System.out.println(name); }", ct);

ct.addMethod(hiMethod);

ct.writeFile("bin/"); //生成的.class存盘，放在当前项目的bin目录下

}

}

### TestMakeClass.java

package test;

import java.lang.reflect.Constructor;

import java.lang.reflect.Field;

import java.lang.reflect.Method;

//创建类

public class TestMakeClass {

public static void main(String[] args) throws Exception {

Class<?> clazz = Class.forName("cn.tedu.hello.HelloWorld");

System.out.println("==构造方法==");

Constructor<?>[] cs = clazz.getConstructors();

for(Constructor<?> c :cs) {

System.out.println(c);

}

System.out.println("\n==属性==");

Field[] fields = clazz.getDeclaredFields(); //才能展示私有的属性

for(Field c : fields) {

System.out.println(c);

}

System.out.println("\n==方法==");

Method[] ms = clazz.getDeclaredMethods();

for(Method m :ms) {

System.out.println(m);

}

System.out.println("\n==方法调用==");

Method mHi = clazz.getMethod("hi", String.class);

System.out.println(mHi);

Object instance = clazz.newInstance();

mHi.invoke(instance, "tony");

}

}

### UpdateClass.java

package cn.tedu.javassist.update;

import javassist.ClassPool;

import javassist.CtClass;

import javassist.CtMethod;

import javassist.CtNewMethod;

import javassist.Loader;

public class UpdateClass {

public static void main(String[] args) throws Exception {

ClassPool pool = ClassPool.getDefault();

//因为下面的类修改后要重新加载，必须告知其类加载器，否则报错：attempted duplicate class definition（类重复加载了）

ClassLoader loader = new Loader(pool);

Class<?> clazz = loader.loadClass("cn.tedu.hello.HelloWorld");

CtClass ct = pool.get(clazz.getName());

CtClass[] paramTypes = { pool.get(String.class.getName()) };

CtMethod method = ct.getDeclaredMethod("hi", paramTypes);

CtMethod newMethod = CtNewMethod.copy(method, ct, null);

ct.defrost(); //冻结一个类

//修改原来的方法

String newMethodName = method.getName() + "$Impl";

//注意：下面将新方法好旧方法关联，不能写错

method.setName(newMethodName);

//所有实参。例如, m($$) 等价于 m($1,$2,...)

StringBuilder sbBody = new StringBuilder();

sbBody.append("{");

sbBody.append("System.out.println(\"执行前\");");

sbBody.append(newMethodName);

sbBody.append("($$);");

sbBody.append("System.out.println(\"执行后\");");

sbBody.append("}");

newMethod.setBody(sbBody.toString());

ct.addMethod(newMethod);

clazz = ct.toClass(); //重新编译

clazz.getMethod("hi", String.class).invoke(clazz.newInstance(), "tony");

ct.freeze();

}

}

## 手写Lombok核心

### 概述

项目开发中pojo已经成为不可或缺的类，可作为MVC各层数据传递媒介，可作为数据库数据封装，可作为网络传输数据封装，几乎出现在项目各处。但其内容傻傻的，一堆属性还情有可原，存放数据，可一堆属性的get/set方法，写在类里，使类臃肿，混淆视听。如果没有它们该多么的清爽啊。

一方面我们希望类简洁清爽，一方面又需要get/set方法，怎么鱼和熊掌兼得呢？Lombok插件出现后，这个问题得到完美的解决。



Lombok项目是一个Java库，它会自动插入编辑器和构建工具中，Lombok提供了一组有用的注释，用来消除Java类中的大量样板代码。仅五个字符(@Data)就可以替换数百行代码从而产生干净，简洁且易于维护的Java类。

下面我们就模拟其核心思路：在类上通过注解标识，属性是否产生get和set方法。通过javassist动态产生其属性的get和set方法。直接改变原来的字节码文件，将新的方法写入其中。

这样用户编辑时，只有属性；而用户使用时却已经有了get和set方法。

注意：只可加入一次，再次加入就会报错，我们采用再次执行时忽略错误。

### NormalDept.java

package cn.tedu.lombok.pojo;

public class NormalDept {

private Dept parentDept; //父部门，自关联

private Integer id; //部门编号

private String name; //部门名称

private Integer sort; //排序号

private String note; //备注信息

public Dept getParentDept() {

return parentDept;

}

public void setParentDept(Dept parentDept) {

this.parentDept = parentDept;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Integer getSort() {

return sort;

}

public void setSort(Integer sort) {

this.sort = sort;

}

public String getNote() {

return note;

}

public void setNote(String note) {

this.note = note;

}

}

### Dept.java

package cn.tedu.lombok.pojo;

import cn.tedu.lombok.annotation.Getter;

import cn.tedu.lombok.annotation.Setter;

@Getter

@Setter

public class Dept{

private Dept parentDept; //父部门，自关联

private Integer id; //部门编号

private String name; //部门名称

private Integer sort; //排序号

private String note; //备注信息

}

### @Getter.java

package cn.tedu.lombok.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target({ElementType.FIELD, ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

public @interface Getter {

}

### @Setter.java

package cn.tedu.lombok.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target({ElementType.FIELD, ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

public @interface Setter {

}

### Lombok.java

package cn.tedu.lombok;

import cn.tedu.lombok.annotation.Getter;

import cn.tedu.lombok.annotation.Setter;

import javassist.ClassPool;

import javassist.CtClass;

import javassist.CtField;

import javassist.CtMethod;

import javassist.Modifier;

//利用发射获取类上的注解，根据注解生成对应的方法

public class Lombok {

//生成

public static void generate(String className) throws Exception {

Class<?> cls = Class.forName(className);

boolean isGetter = false;

if(cls.isAnnotationPresent(Getter.class)) {

isGetter = true;

}

boolean isSetter = false;

if(cls.isAnnotationPresent(Setter.class)) {

isSetter = true;

}

ClassPool pool = ClassPool.getDefault();

CtClass ct = pool.get(cls.getName());

ct.defrost();

CtField[] fields = ct.getDeclaredFields(); //展示私有的

for(CtField field : fields){

String fieldname = toUpperFirst(field.getName());

if( isGetter ) {

CtMethod getter = new CtMethod(pool.get(field.getType().getName()),

"get" + fieldname, null, ct);

getter.setModifiers(Modifier.PUBLIC);

getter.setBody("{ return this."+field.getName()+"; }");

try {

ct.addMethod(getter);

}catch(Exception e) {

}

}

if( isSetter ) {

CtMethod setter = new CtMethod(CtClass.voidType,

"set" + fieldname,

new CtClass[]{ pool.get(field.getType().getName())}, ct);

setter.setModifiers(Modifier.PUBLIC);

setter.setBody("{ this."+field.getName()+"=$1; }");

try {

ct.addMethod(setter);

}catch(Exception e) {

}

}

}

ct.writeFile("bin/");

ct.freeze();

}

//首字母大写

private static String toUpperFirst(String name) {

name = (""+name.charAt(0)).toUpperCase() + name.substring(1);

return name;

}

}

### TestLombok.java

package test;

import java.lang.reflect.Method;

import org.junit.Test;

import cn.tedu.lombok.Lombok;

public class TestLombok {

@Test

public void updateClass() throws Exception {

String className = "cn.tedu.lombok.pojo.Dept";

Lombok.generate(className);

}

@Test

public void test() throws Exception {

String className = "cn.tedu.lombok.pojo.Dept";

Class<?> clazz = Class.forName(className);

Method[] ms = clazz.getDeclaredMethods();

for(Method m : ms) {

System.out.println(m);

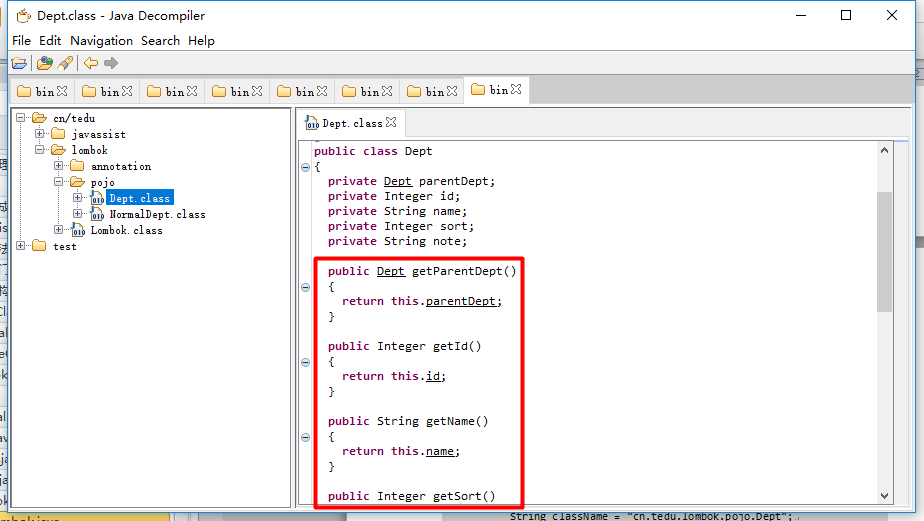
}

}

}

### 验证

使用反编译工具，可以清晰看到，生成的class已经拥有了get和set方法：



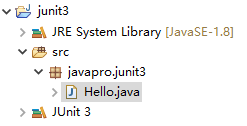
# 小试：手写Junit+依赖注入+自动装配

## 演示Junit3.x

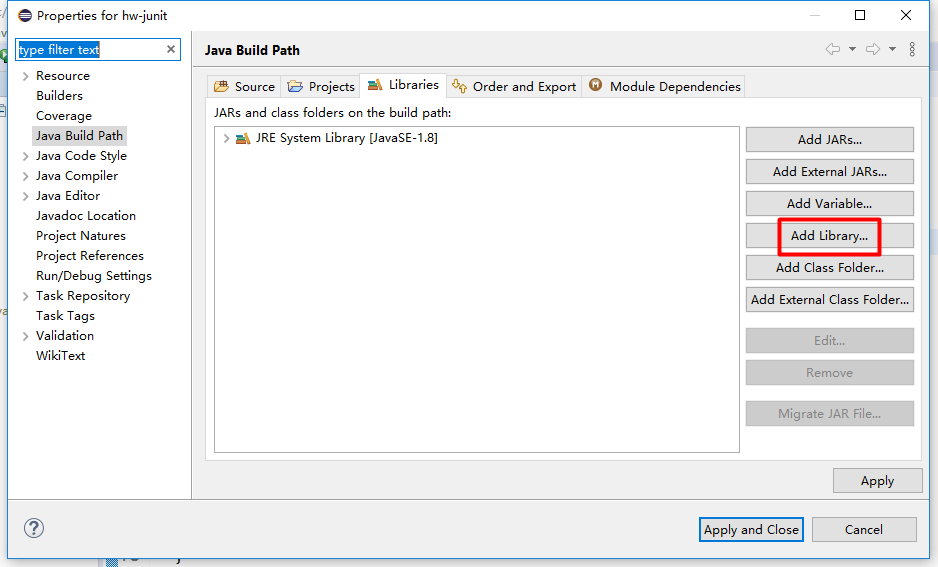
### 规则

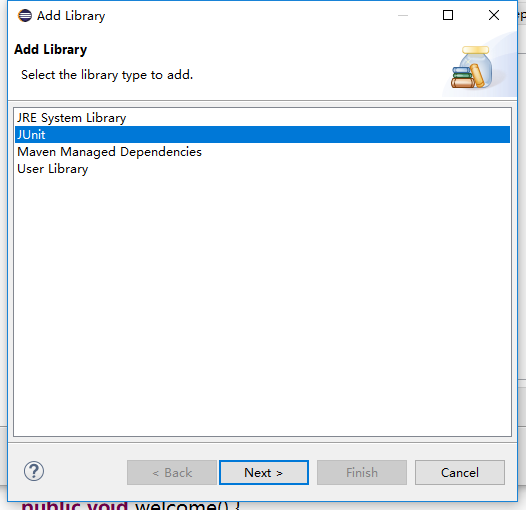
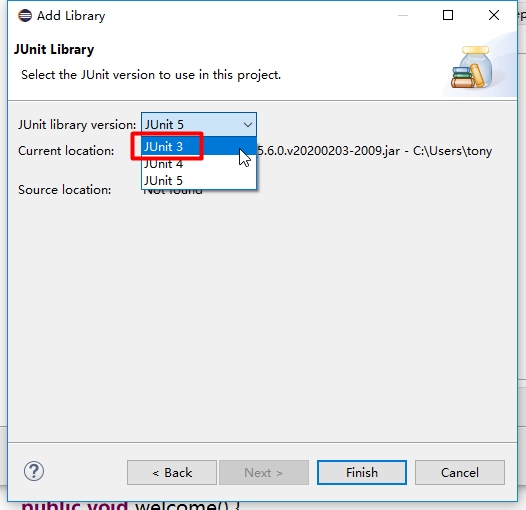
* public的
* void的
* 无方法参数
* 方法名称必须以test开头
* 测试类必须继承TestCase类

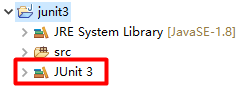
### 项目结构



### 导包





### Hello.java

Junit3的单元测试类必须继承TestCase类：

package cn.tedu.junit3;

import junit.framework.TestCase;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午7:28:06

\* @description 描述：

\*/

public class Hello extends TestCase{

public void testHi() {

System.out.println("hi()");

}

public void testSay() {

System.out.println("say()");

}

public void welcome() {

System.out.println("welcome()");

}

}

## 手写Junit3.x

### TestJunit3.java

执行类中的所有test开头的方法

package javapro.junit3;

import java.lang.reflect.Method;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午7:47:58

\* @description 描述：

\*/

public class TestJunit3 {

public static void main(String[] args) throws Exception {

Class<?> clazz = Class.forName("javapro.junit3.Hello");

//创建类的实例

Object obj = clazz.newInstance();

if("junit.framework.TestCase".equals(clazz.getSuperclass().getName())){

//获取类的所有方法

Method[] ms = clazz.getMethods();

for(Method m : ms) {

//获取类的方法名

String methodName = m.getName();

//System.out.println(methodName);

//判断方法名是否为test前缀

if(methodName.startsWith("test")) {

//回调方法

m.invoke(obj);

}

}

}

}

}

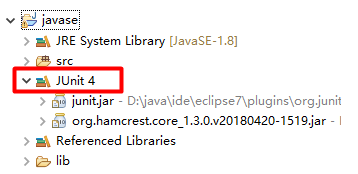
注意：从反射代码结构可知，其是没有方法的执行先后顺序的，方法自身也没有顺序的特性，先执行谁都一样。

## 演示Junit4.x

### 规则

junit4.x全面引入了注解Annotation开发方式，更加简洁方便。

在项目中引入Junit单元测试，怎么代码中增加一个注解@Test就可以运行了呢？



其实很简单，它利用了我们java中两个最核心的技术，一个叫注解，一个叫反射。注解我们已经见过了、使用了，非常简单@Test，但光它显然不够，它背后的黑手是谁呢？接着我们就来大解密。别小看这个秘密，如果你把这个基本原理搞懂了，你就打开了java世界通向底层的一道大门。

### Hello.java

package javapro.reflect.junit4;

import org.junit.Test;

public class Hello {

@Test

public void hello() {

System.out.println("testHello");

}

@Test

public void hi() {

System.out.println("testHi");

}

public void hi(String name) {

System.out.println("hi：" +name);

}

}

## 手写Junit4.x

### @Test.java

自定义注解

package javapro.junit;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface Test {

}

### Hello.java

修改@Test注解为上面自定义的注解，而不再是Junit。

package javapro.reflect.junit4;

//import org.junit.Test; //junit工具包

import javapro.junit.Test; //自定义注解

public class Hello {

@Test

public void hello() {

System.out.println("testHello");

}

@Test

public void hi() {

System.out.println("testHi");

}

public void hi(String name) {

System.out.println("hi：" +name);

}

}

### TestJunit4.java

执行类中的所有标识了@Test注解的方法

package javapro.reflect.junit4;

import java.lang.reflect.Method;

import javapro.junit.Test;

public class TestJunit4 {

// 执行所有@Test注解的方法，没有的就不执行

public static void main(String[] args) throws Exception {

Class<?> clazz = Class.forName("javapro.reflect.junit4.Hello");

Object obj = clazz.newInstance();

// 获取类上面的所有注解 @Test

Method[] ms = clazz.getMethods();

for (Method m : ms) {

// 判断方法上有@Test注解才可以执行

Test t = m.getAnnotation(Test.class);

if (t != null) { // 只处理@Test方法

// System.out.println(m);

m.invoke(obj);

}

}

}

}

## 依赖注入概念

### 概念

在框架出现之前，开发人员写代码硬编码居多。如设置属性或者对象关联关系都是硬编码。如下面的例子：

Emp emp = new Emp();

emp.setName("tony"); //硬编码，运行时进行设置

Dept dept = new Dept();

dept.setName("软件部");

emp.setDept(dept); //对象关系绑定，硬编码

但随着框架的出现，提倡代码的松耦合结构。把上面紧耦合的代码关系，硬编码的写死代码，变成外部配置的松耦合关系，形成动态配置。如spring把上面这些关系配置在applicationContext.xml文件中。松耦合结构使代码复杂度变高，逻辑复杂，显得更啰嗦，但代码的灵活度变高，扩展性变高，代码配置动态化，对大型项目而言更加有益，小项目就有点得不偿失了。

Spring框架也经过漫长的迭代，由开始的xml臃肿的配置过渡到现今主流的注解开发方式，大大简化了开发，越来越展示了其松耦合这套机制的完美，已经成为事实的开发规范。谁不会Spring都不好意思说自己是合格程序员！

有这么强大啊，大家赶快随我看看，它怎么实现的。

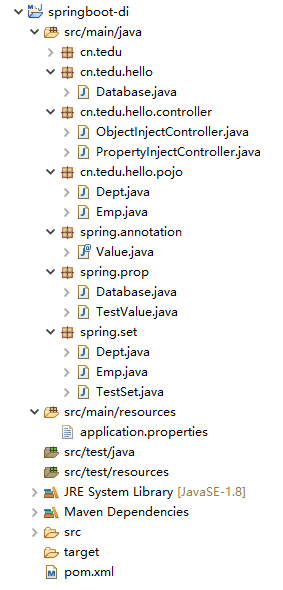
### 前提

那Spring框架到底如何实现的呢？接着我们就开始解密，解密前我们先进行环境准备。这里我直接使用SpringBoot来构建项目。不会使用SpringBoot的同学也不必担心，跟着我一步步做就可以。

但有个前提哦，你的Maven环境必须安装配置好。Maven已经成为开发者必备的基础技能，我们是高手班，我就不在这里多说了。

## 环境准备创建SpringBoot项目

### 项目结构



### pom.xml

官网生成pom.mxl文件：https://start.spring.io/

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

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.3.0.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>cn.tedu.hello</groupId>

<artifactId>springboot-di</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springboot-di</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

### application.properties

这里我们配置全球最快的Hikari连接池，比阿里的Druid快速稳定。

注意：

* 高版本的mysql的driver更改了：com.mysql.cj.jdbc.Driver
* 而且url链接需要加上时区：&serverTimezone=Asia/Shanghai

server.port=8090

spring.datasource.type=com.zaxxer.hikari.HikariDataSource

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/jtdb?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai

spring.datasource.username=root

spring.datasource.password=root

### RunApp.java

package cn.tedu;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:39:07

\* @description 描述：

\*/

@SpringBootApplication

public class RunApp {

public static void main(String[] args) {

SpringApplication.run(RunApp.class, args);

}

}

## 演示属性注入

### 实现过程

演示数据库链接的四个参数从属性文件中自动获取，并注入到私有属性中。Spring框架已经为我们提供了注解@Value。

具体的实现过程：

1. 创建的SpringBoot项目，自动集成了Spring。
2. Spring底层通过包扫描机制，扫描注解@Component。从而创建该对象，将其放在Spring容器中。
3. 扫描@Value注解，读取默认的配置文件application.properties。
4. 从中获取对应的属性。如果有则读取其值，把其值赋值给对应的属性；如果没有，则忽略。

### Database.java

package cn.tedu.hello;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:31:26

\* @description 描述：封装数据库的4个参数

\*/

@Component

public class Database {

@Value("${spring.datasource.driver-class-name}")

private String driver;

@Value("${spring.datasource.url}")

private String url;

@Value("${spring.datasource.username}")

private String username;

@Value("${spring.datasource.password}")

private String password;

public String getDriver() {

return driver;

}

public void setDriver(String driver) {

this.driver = driver;

}

public String getUrl() {

return url;

}

public void setUrl(String url) {

this.url = url;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

### PropertyInjectController.java

package cn.tedu.hello.controller;

import org.springframework.beans.factory.annotation.Autowired;

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

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

import cn.tedu.hello.Database;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:34:54

\* @description 描述：属性注入

\*/

@RestController

public class PropertyInjectController {

@Autowired

private Database db;

@RequestMapping("/dbInfo")

public String dbInfo() {

StringBuilder sb = new StringBuilder();

sb.append("驱动：").append(db.getDriver()).append("<br/>");

sb.append("链接：").append(db.getUrl()).append("<br/>");

sb.append("用户：").append(db.getUsername()).append("<br/>");

sb.append("密码：").append(db.getPassword()).append("<br/>");

return sb.toString();

}

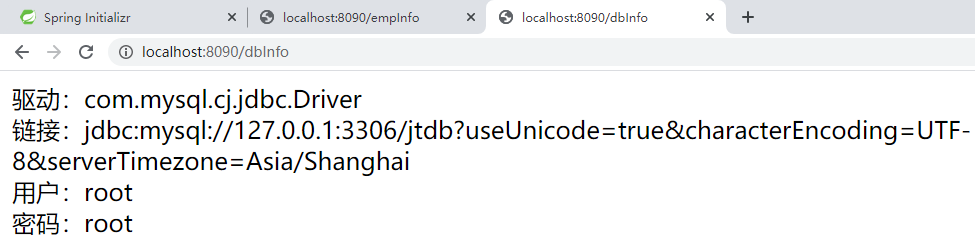
}

### 测试

运行RunApp.java，访问链接：

http://localhost:8090/dbInfo

页面展示：



## 演示对象注入

### Dept.java

package cn.tedu.hello.pojo;

import org.springframework.stereotype.Component;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:32:02

\* @description 描述：

\*/

@Component

public class Dept {

private String name = "软件部";

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Dept [name=" + name + "]";

}

}

### Emp.java

package cn.tedu.hello.pojo;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:33:12

\* @description 描述：

\*/

@Component

public class Emp {

@Autowired //对象关联，所属部门

private Dept dept;

private String name = "陈子枢";

public Dept getDept() {

return dept;

}

public void setDept(Dept dept) {

this.dept = dept;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Emp [dept=" + dept + ", name=" + name + "]";

}

}

### ObjectInjectController.java

package cn.tedu.hello.controller;

import org.springframework.beans.factory.annotation.Autowired;

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

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

import cn.tedu.hello.pojo.Emp;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:38:17

\* @description 描述：

\*/

@RestController

public class ObjectInjectController {

@Autowired

private Emp emp;

@RequestMapping("/empInfo")

public Emp empInfo() {

return emp;

}

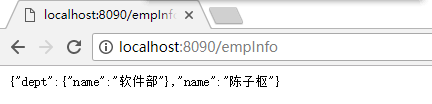
}

### 测试

运行RunApp.java，访问链接：

http://localhost:8090/empInfo

页面展示：



## 手写@Value~属性注入

### 目标

模拟 @Value注解实现从属性文件中读取数据，动态设置在私有属性上。

### application.properties

server.port=8090

spring.datasource.type=com.zaxxer.hikari.HikariDataSource

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/jtdb?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai

spring.datasource.username=root

spring.datasource.password=root

### @Value.java

package spring.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:44:47

\* @description 描述：模拟实现Spring框架中的@Value，来实现属性注入

\*/

@Target(ElementType.FIELD)

@Retention(RetentionPolicy.RUNTIME)

public @interface Value {

String value() default "";

}

### Database.java

导入我们自定义的注解@Value。

package spring.prop;

import spring.annotation.Value;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:46:44

\* @description 描述：

\*/

public class Database {

@Value("${spring.datasource.driver-class-name}")

private String driver;

@Value("${spring.datasource.url}")

private String url;

@Value("${spring.datasource.username}")

private String username;

@Value("${spring.datasource.password}")

private String password;

@Override

public String toString() {

return "Database [driver=" + driver + ", url=" + url + ", username=" + username + ", password=" + password

+ "]";

}

}

### TestValue.java

package spring.prop;

import java.io.FileInputStream;

import java.io.InputStream;

import java.lang.reflect.Field;

import java.util.Enumeration;

import java.util.HashMap;

import java.util.Map;

import java.util.Properties;

import spring.annotation.Value;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:44:28

\* @description 描述：模拟实现Spring框架属性注入过程

\*/

public class TestValue {

public static void main(String[] args) throws Exception {

//1. 读取属性文件，内容放入map集合中

Map<String,String> map = getProperties(TestValue.class, "application.properties");

System.out.println(map);

//2. 通过反射获取类，获取属性，获取属性注解

Class<?> clazz = Class.forName("spring.prop.Database");

Object obj = clazz.newInstance();

//3. 从map取出，设置到私有属性上

Field[] fs = clazz.getDeclaredFields();

for(Field f : fs) {

Value v = f.getAnnotation(Value.class);

if(null != v) {

//？处理el表达式

String key = v.value().substring(2, v.value().length()-1);

String value = map.get(key);

//写value到属性上

f.setAccessible(true);

f.set(obj, value);

}

}

//4. 打印对象

System.out.println(obj);

}

//读取配置文件，转换存储到map中

static public Map<String,String> getProperties(Class<?> clazz, String filename) throws Exception{

String dir = clazz.getResource("/").getPath(); //注意路径位置

Properties props = new Properties();

InputStream is = new FileInputStream(dir + filename);

props.load(is); //加载内容到Properties对象中

is.close();

//最高效遍历资源对象方式

Enumeration<?> e = props.propertyNames();

Map<String,String> map = new HashMap<String,String>();

while(e.hasMoreElements()) {

String key = (String) e.nextElement();

String value = props.getProperty(key);

map.put(key, value);

}

return map;

}

}

## 手写set对象注入

由原来的手动set方法设置替代为自动实现内容的设置，这个过程称为自动装配，也称为set注入。

下面分别实现Emp员工属性的注入和关联部门对象的注入。

### Dept.java

package spring.set;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:01:31

\* @description 描述：

\*/

public class Dept {

private String name = "软件部";

@Override

public String toString() {

return "Dept [name=" + name + "]";

}

}

### Emp.java

package spring.set;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:02:24

\* @description 描述：

\*/

public class Emp {

private Dept dept; //所属部门，对象注入

private String name;

@Override

public String toString() {

return "Emp [dept=" + dept + ", name=" + name + "]";

}

}

### TestSet.java

利用反射调用set方法，所以这种方式称为set注入，后续会进一步升级为@Autowired注解方式。

package spring.set;

import java.lang.reflect.Field;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:03:26

\* @description 描述：set注入方式，注解自动装配

\*/

public class TestSet {

public static void main(String[] args) throws Exception {

//1. 利用反射创建所需要对象实例

Class<?> clazzDept = Class.forName("spring.set.Dept");

Dept dept = (Dept)clazzDept.newInstance();

Class<?> clazzEmp = Class.forName("spring.set.Emp");

Emp emp = (Emp)clazzEmp.newInstance();

//2. 获取类的私有属性，通过反射set方法设置具体值

Field fName = clazzEmp.getDeclaredField("name");

fName.setAccessible(true);

fName.set(emp, "陈子枢");

//自动装配

Field fDept = clazzEmp.getDeclaredField("dept");

fDept.setAccessible(true);

fDept.set(emp, dept); //set注入

System.out.println(dept);

System.out.println(emp);

}

}

# 框架：手写Spring框架IoC+DI+AOP

## 概念

### 为什么会有IoC控制反转？

Spring诞生的年代可不一般，那时Struts和Hibernate如日中天，特别Hibernate把不可一世的EJB干趴下。在这样群雄辈出的时代，Spring如何异军突起，最终一统江湖呢？Spring强大在哪里呢？

它强大在，它剥夺了对象的创建权，从而控制了对象的生命周期。也就是对象是Spring创建，销毁。其它框架再牛，它也得现有对象啊，这就是Spring的厉害，釜底抽薪。最终其他英雄都成了它的幕僚。

怀念老的方式：

Girl girlfriend = new Girl();

Java中创建对象一去不复返了，以前没对象的时候，心血来潮想有几个创建几个。可现在取而代之是Spring的方式，甚至万物皆对象也都崩塌了，改成了万物皆Bean。

那Spring要达到这“阴险”的目的，就那么轻松吗？这可不然，Spring也是煞费苦心，一套盘古开天，神仙降生的理论。什么IoC控制反转，什么DI依赖注入，什么AOP面向切面编程，这些理论几十年经久不衰，可见多牛。

那Spring具体是怎么实现的呢？它突破性的创造了IoC控制反转的理论思想，不得不佩服作者这么烧脑的东西也能想的出来，现在很多开发者都说不清什么是IoC，面试官常拿来刁难面试者，其实他自己也说不清。IoC是一套理论，那怎么具体代码实现呢？Spring又衍生出四大发明：包扫描、容器、getBean获取对象，实际开发中不光要创建对象，还有对象关联，于是又加上一项发明，DI依赖注入。

## 四大发明

### 包扫描componentScan

替代繁琐臃肿的xml配置，几乎零配置。实现自动化处理，对程序员透明，程序员不用写一句代码。

通过注解标识哪些类需要进行Spring来管理。加载、创建对象实例，配置属性和对象关联。

@Controller、@Service、@Repository、@Component、@Mapper

### 容器context

没有了new对象，那用到对象怎么办？spring进行预处理，在加载项目时就给你创建好了，好了？那放在哪里呢？这就放在“容器”中。

### 获取对象getBean

我们不在new，而是要从容器中获取。那容器中那么多对象，不对改叫bean，怎么获得呢？Spring提供了getBean()方法，从容器中获取需要的对象。

### 依赖注入

以前我们手动绑定对象关系，现在Spring可以直接注入，就像我们之前使用的@Value和@Autowired注解实现。

## 演示xml配置方式IoC

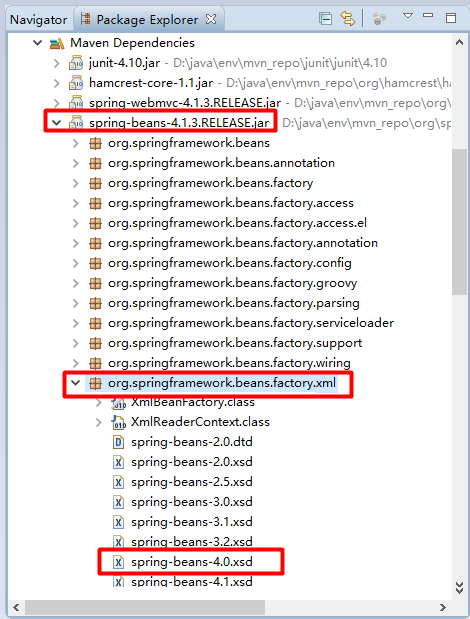
Inversion of Control 控制反转方式来实现对象的创建。

### 导入约束自动提示

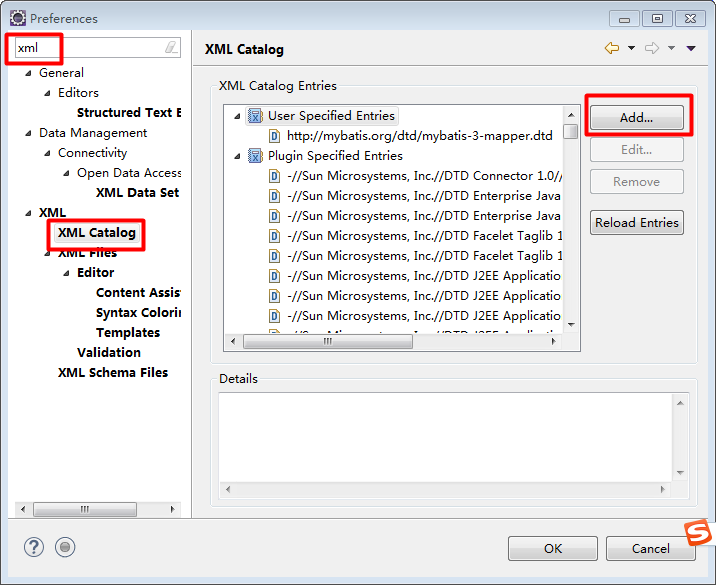
如果有网络可以无需配置就有标签提示，如果离线就需要进行下面配置才会有提示。熟练后复制进行修改有无下面配置都没有关系，初学者又可能没有网络就配置下。

配置说明可参考：http://schema.tedu.cn/proxy/

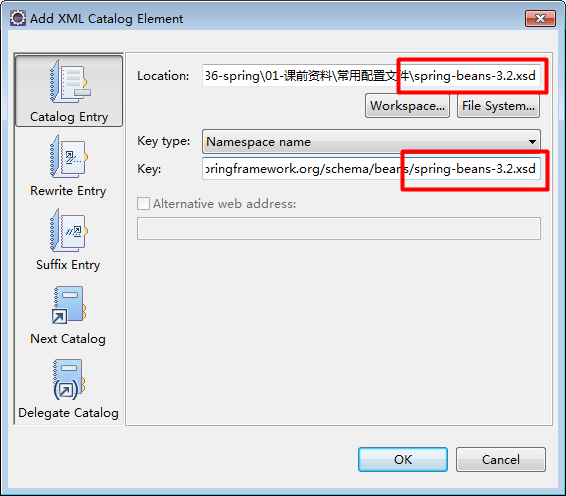
找到jar包中的xsd文件：



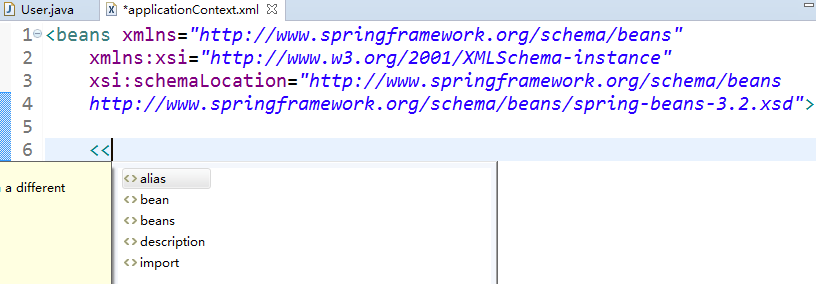
自定义xsd提示



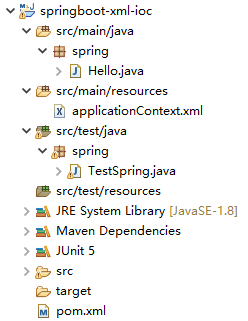
浏览xsd文件，复制拼接到下面的链接中



关闭xml文件，再次打开，提示出现



### 项目结构



### pom.xml

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

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.3.0.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>spring</groupId>

<artifactId>springboot-xml-ioc</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springboot-xml-ioc</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

### Hello.java

package spring;

public class Hello {

public void say() {

System.out.println("Spring Hello World!");

}

}

### applicationContext.xml

配置bean：

<?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-4.0.xsd">

<!-- Spring容器中id不能重复，唯一，class指向Hello.class -->

<bean id="hello" class="spring.Hello"></bean>

<bean id="h1" class="spring.Hello"></bean>

</beans>

### TestSpringIoC.java

测试类：

package spring;

import org.junit.Test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestSpringIoC {

@Test //原始方式创建java对象

public void oldStyle() {

Hello hello = new Hello();

hello.say();

}

@Test //spring来创建对象

public void newStyle() {

//1.加载核心配置文件，创建spring容器，会创建所有bean

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

//2.从spring容器中获取指的bean

Hello hello = (Hello)ac.getBean("hello"); //对应配置文件中的bean标签的id名称

hello.say();

Hello h1 = (Hello)ac.getBean("h1");

h1.say();

}

}

### 小结

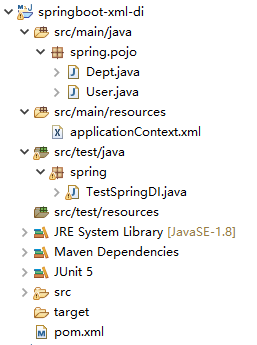
这就是spring框架的IoC，控制反转。之前我们自己new出新类。new User();变成由一个初始化的xml配置文件来创建，也就是由spring容器来创建。遍历xml配置文件，读取到<bean>，获取到class属性的类的全路径，利用反射创建这个类。

可见有了IoC，Spring如有神助，重要程度一日强过一日。在java范畴中万物皆Object，而在Spring中万物皆Bean。Bean是Spring的核心、基础、根源。

## 演示xml配置方式DI

Dependency Injection 依赖注入，设置对象的属性和其它对象关联。

### 项目结构



### pom.xml

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

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.3.0.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>spring</groupId>

<artifactId>springboot-xml-ioc</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springboot-xml-ioc</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

### Dept.java

package spring.pojo;

public class Dept {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Dept [name=" + name + "]";

}

}

### User.java

package spring.pojo;

public class User {

//创建一个对象关联：当前这个用户属于软件部

private Dept dept;

private String name;

public Dept getDept() {

return dept;

}

public void setDept(Dept dept) {

this.dept = dept;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "User [dept=" + dept + ", name=" + name + "]";

}

}

### applicationContext.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"

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

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

<bean id="dept" class="spring.pojo.Dept">

<!-- 配置属性初始化值 -->

<property name="name" value="软件研发部"/>

</bean>

<bean id="user" class="spring.pojo.User">

<property name="name" value="tony"/>

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

</bean>

</beans>

设置属性：property的value

关联对象：property的ref

### TestSpringDI.java

package spring;

import org.junit.Test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import spring.pojo.Dept;

import spring.pojo.User;

public class TestSpringDI {

@Test

public void di() {

//1. 获取容器

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

//2. 从容器中获取bean

Dept dept = (Dept)ac.getBean("dept");

System.out.println(dept);

User user = (User)ac.getBean("user");

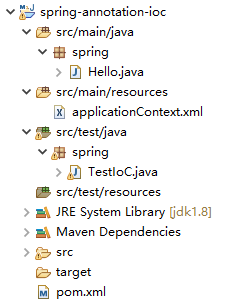
System.out.println(user);

}

}

## 演示注解方式IoC

### 项目结构



### pom.xml

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>cn.tedu</groupId>

<artifactId>spring</artifactId>

<version>0.0.1-SNAPSHOT</version>

<!-- 集中定义依赖版本号 -->

<properties>

<junit.version>4.10</junit.version>

<spring.version>4.1.3.RELEASE</spring.version>

</properties>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

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

<scope>test</scope>

</dependency>

<!-- Spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

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

</dependency>

</dependencies>

</project>

### Hello.java

package spring;

import org.springframework.stereotype.Component;

@Component

public class Hello {

public void hi() {

System.out.println("Hello Spring.");

}

}

### applicationContext.xml

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

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

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

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-4.0.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd

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

<!-- 扫描包 -->

<context:component-scan base-package="spring" />

</beans>

设置属性：property的value

关联对象：property的ref

### TestSpringIoC.java

测试类：

package spring;

import org.junit.Test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestSpringIoC {

@Test

public void bean() {

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

Hello hello = (Hello)ac.getBean("hello");

System.out.println(hello);

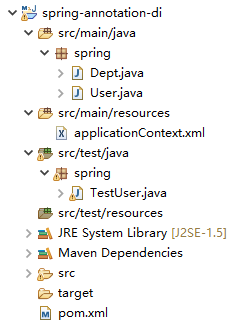
hello.hi();

}

}

## 演示注解方式DI

### 项目结构



### pom.xml

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>cn.tedu</groupId>

<artifactId>spring</artifactId>

<version>0.0.1-SNAPSHOT</version>

<!-- 集中定义依赖版本号 -->

<properties>

<junit.version>4.10</junit.version>

<spring.version>4.1.3.RELEASE</spring.version>

</properties>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

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

<scope>test</scope>

</dependency>

<!-- Spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

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

</dependency>

</dependencies>

</project>

### Dept.java

package spring;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

@Component

public class Dept {

@Value(value = "软件部")

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Dept [name=" + name + "]";

}

}

### User.java

package spring;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

@Component

public class User {

@Autowired

private Dept dept;

@Value(value = "tony")

private String name;

public Dept getDept() {

return dept;

}

public void setDept(Dept dept) {

this.dept = dept;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "User [dept=" + dept + ", name=" + name + "]";

}

}

### applicationContext.xml

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

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

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

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-4.0.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd

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

<!-- 扫描包 -->

<context:component-scan base-package="spring" />

</beans>

### TestSpringDI.java

package spring;

import org.junit.Test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestSpringDI {

@Test

public void show() {

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

User u = (User)ac.getBean("user");

System.out.println(u);

}

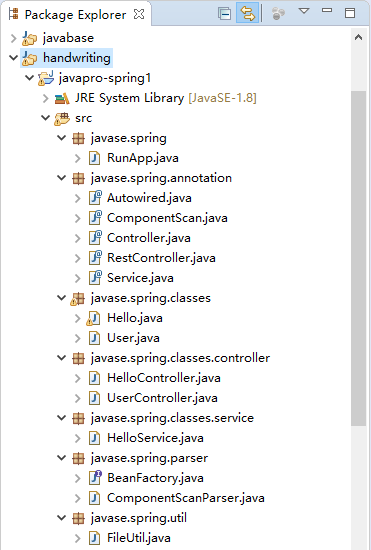
}

## 手写Spring框架

### 开发步骤

1. 注解：
   1. @ComponentScan 设置包扫描的起始目录路径
   2. @Controller 处理控制层
   3. @Service 处理业务层
2. 加载：
   1. 包扫描，扫描：@Controller、@Service
   2. 容器，创建对象，放入容器
3. 依赖注入：
   1. 注解：@Autowired
   2. 实现注入
4. 运行：
   1. 启动，加载，业务访问

### 创建java工程



### FileUtil.java

package javase.spring.util;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

//文件工具类

public class FileUtil {

// 首字母小写

public static String toLowerCaseFirstOne(String s) {

if (Character.isLowerCase(s.charAt(0))) {

return s;

} else {

return (new StringBuilder()).append(Character.toLowerCase(s.charAt(0))).append(s.substring(1)).toString();

}

}

// 递归，只读取某个文件夹下的所有文件，含子目录

static public List<String> readClassFile(String dir, List<String> fileList){

File file = new File(dir);

//如果目录存在则处理

if( file.exists() ) {

//获取当前目录下的内容

File[] files = file.listFiles();

for(File f : files ) {

if( f.isDirectory() ) { //是目录

readClassFile( f.getAbsolutePath(), fileList ); //参数值必须改变

}else { //是文件

fileList.add(coverClassName(f.getAbsolutePath())); // 去掉文件后缀，特殊字符使用转义字符

}

}

return fileList;

}

return null;

}

//转换文件路径为类的全路径形式

static public String coverClassName(String filename) {

filename = filename.substring(filename.indexOf("bin") + 4, filename.length() - 6); // 去掉根目录，默认编译后文件存放在当前项目的bin目录下

return filename.replaceAll("\\\\", "."); // 去掉文件后缀，特殊字符使用转义字符

}

}

### @Service.java

package javase.spring.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target({ ElementType.TYPE }) // 类上标识

@Retention(RetentionPolicy.RUNTIME) // 运行时

public @interface Service { // 标识业务层类

String value() default "";

}

### @Controller.java

package javase.spring.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target({ElementType.TYPE}) //类上标识

@Retention(RetentionPolicy.RUNTIME) //运行时

public @interface Controller { //标识控制层类

String value() default "";

}

### @Autowired.java

package javase.spring.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target({ ElementType.FIELD }) // 属性上标识

@Retention(RetentionPolicy.RUNTIME) // 运行时

public @interface Autowired { //实现属性的设置，也称为注入

String value() default "";

}

### @ComponentScan.java

package javase.spring.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target({ElementType.TYPE}) //类上标识

@Retention(RetentionPolicy.RUNTIME) //运行时

public @interface ComponentScan {

String value() default "";

}

### ApplicationContext.java

package javase.spring.parser;

//接口：确定有哪些行为，规范要做什么，但不实现，交给具体的实现类实现

public interface ApplicationContext {

//初始化配置所有类，进行池化，使用时直接调用，性能高

public void config(String packagePath) throws Exception;

//从容器中获取bean

public <T> T getBean(String beanName); //泛型方法，不确定得到的bean是谁，进行通配

}

### ComponentScanParser.java

包扫描核心类，同时创建容器，提供getBean方法

package javase.spring.parser;

import java.lang.annotation.Annotation;

import java.lang.reflect.Field;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import javase.spring.annotation.Autowired;

import javase.spring.annotation.Controller;

import javase.spring.annotation.Service;

import javase.spring.util.FileUtil;

//包扫描

public class ComponentScanParser implements ApplicationContext {

// 容器（池）

private Map<String, Object> beans = new HashMap<String, Object>();

//执行初始化过程

public void config(String packagePath) throws Exception {

System.out.println("初始化过程：第一步：遍历容器....................................... ");

scan(packagePath);

// 遍历容器

for (String beanName : beans.keySet()) {

System.out.println(beanName + " : " + beans.get(beanName));

}

System.out.println("\n初始化过程：第二步：注入对象....................................... ");

inject();

}

// 执行包扫描，初始化容器，装配对象

private void scan(String packageName) throws Exception {

String dir = this.getClass().getResource("/").getPath(); //获取当前类所在路径

dir = dir+packageName.replaceAll("\\.", "/"); //替换为路径的写法

List<String> fileList = new ArrayList<String>();

for (String className : FileUtil.readClassFile(dir, fileList)) {

Class<?> clazz1 = Class.forName(className); // 获取类

// 获取指定的注解

Controller ac = clazz1.getAnnotation(Controller.class);

Service as = clazz1.getAnnotation(Service.class);

// 判断类中是否存在注解

if (ac != null || as != null) { // 含有Controller或Server注解才放容器中

Object obj = clazz1.newInstance(); // 创建对象实例

// 放入实例到容器中

String beanName = className.substring(className.lastIndexOf(".") + 1); // 去掉后缀

beans.put(FileUtil.toLowerCaseFirstOne(beanName), obj);

}

}

}

//处理注入过程

private void inject() throws Exception {

// 自动注入对象

for (String beanName : beans.keySet()) {

Object instance = beans.get(beanName); // 获取对象实例

Field[] fields = instance.getClass().getDeclaredFields(); // 利用发射，可以访问私有的属性

for (Field f : fields) {

f.setAccessible(true); // 即使是private甚至final关键字标示过得属性也可以有访问权限！

Annotation aa = f.getAnnotation(Autowired.class);

if (aa != null) { // 含有Autowired注解才进行注入

// 获取属性的实例

String t = f.getType().getSimpleName();

Object obj = beans.get(FileUtil.toLowerCaseFirstOne(t));

f.set(instance, obj); // 利用set方法注入到bean对象上

}

}

}

}

// 获取对应bean，泛型类，这样无需强制转换

@SuppressWarnings("unchecked") // 下面用了泛型，编译时就会造成不确定性，编译器就会提示，这个注解强制无需提示

public <T> T getBean(String beanName) {

return (T) beans.get(beanName);

}

}

### HelloService.java

package javase.spring.classes.service;

import javase.spring.annotation.Service;

@Service //业务逻辑层

public class HelloService {

public String hello() {

return "tony"; //模拟处理业务逻辑

}

}

### HelloController.java

package javase.spring.classes.controller;

import javase.spring.annotation.Autowired;

import javase.spring.annotation.Controller;

import javase.spring.classes.service.HelloService;

@Controller //业务的控制层

public class HelloController {

@Autowired

private HelloService helloService; //由容器自动注入

public String hello() {

return "hello "+helloService.hello(); //调用逻辑层业务逻辑

}

}

### Hello.java

package javase.spring.classes;

//未加注解的类，容器忽略不做处理

public class Hello {

private String name = "pony";

}

### RunApp.java

package javase.spring;

import javase.spring.annotation.ComponentScan;

import javase.spring.classes.controller.HelloController;

import javase.spring.parser.ApplicationContext;

import javase.spring.parser.ComponentScanParser;

/\*

\* 功能要求：

\* 1. 形成容器，形成池化，不是现用现new，而是提前把对象创建存储起来，

\* 这样使用时无需创建，性能提升很多

\* 2. 对象关联，HelloController中要调用HelloService.hello()业务方法，

\* 那如何让Controller对象拥有Service对象呢？利用反射set设置，也称为注入

\* 3. 利用注解标识对象的关系，利用反射实现类的扫描创建对象，创建对象的关联

\*

\* 学习的意义：

\* 这里综合使用了：注解、反射、泛型，模拟后期学习的最重要框架的底层实现，

\* 加深对注解、反射、泛型难点知识的理解，也为后期理解Spring底层实现铺路

\*

\* 实现步骤：

\* 1. 指定目录，获取目录中所有文件，根据类注解确定是否创建对象

\* 2. 根据名称创建对象，存入容器中Map<K,V> key=beanName，value=object实例

\* 3. 设置对象关系，根据注解，获取实例，利用反射set对象关系

\* 4. 获取对象，调用业务

\*

\* 实现难点：

\* 1. 从包路径获取文件的目录路径

\* 2. 扫描目录，根据注解Controler或Service判断是否创建对象实例

\* 3. 将对象实例放入到map集合中

\* 4. 发生获取类的注解Autowired，有注解则利用set方法设置对象属性关系

\*/

@ComponentScan("javase.spring.classes")

public class RunApp {

public static void main(String[] args) throws Exception {

RunApp run = new RunApp();

ComponentScan cs = run.getClass().getAnnotation(ComponentScan.class); //获取类上的注解

String packagePath = cs.value(); //得到注解的value属性值

System.out.println("扫描的包路径："+packagePath+"\n");

//1.初始化环境

ApplicationContext context = new ComponentScanParser();

context.config(packagePath);

//2.获取bean来调用

System.out.println("\n业务调用过程：....................................... ");

HelloController h = context.getBean("helloController");

//3.执行业务

String rtn = h.hello();

System.out.println(rtn);

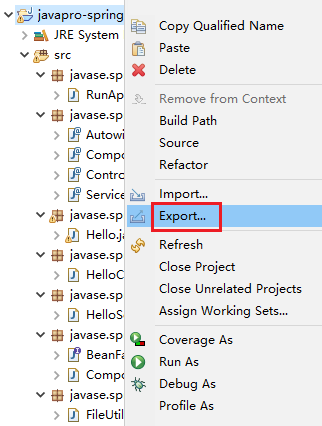
}

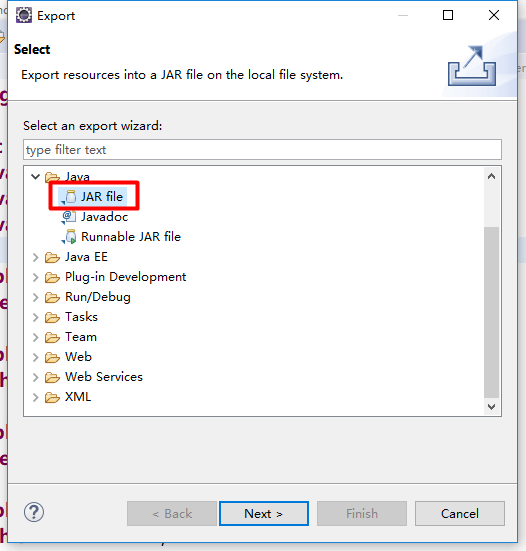
}

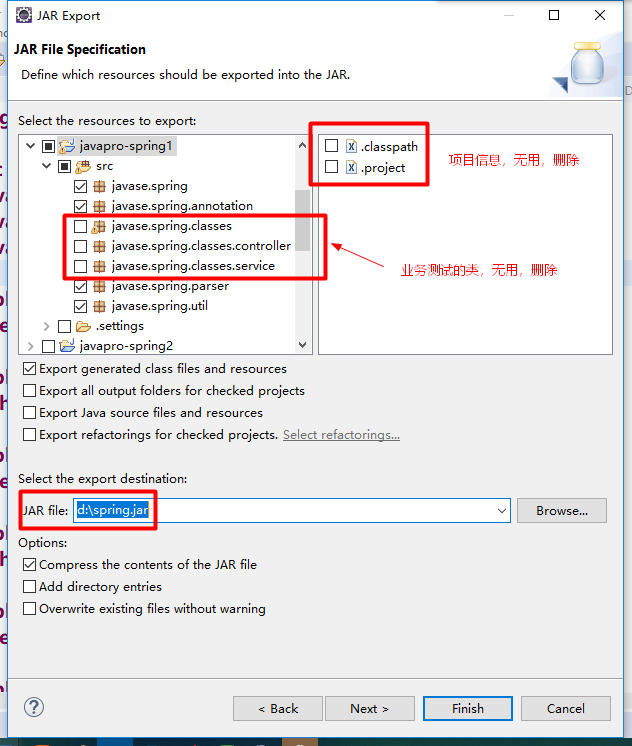
## 使用spring框架

### 导出jar包

可以把手写的spring框架打包，这样其它项目就可以直接导入，就像我们日常导入spring框架一样。

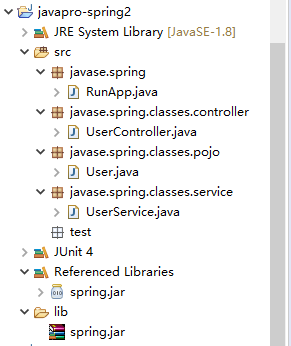






## 测试自己手写的spring框架

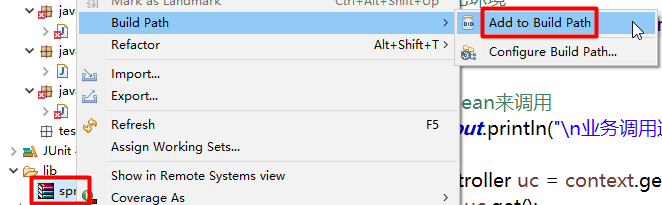
### 创建java工程



### 导入jar包

创建lib目录，把上面导出的spring.jar文件放入其中

Jar包上鼠标右键，Build Path。



### User.java

package javase.spring.classes.pojo;

public class User {

private Integer id;

private String name;

private String company;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCompany() {

return company;

}

public void setCompany(String company) {

this.company = company;

}

@Override

public String toString() {

return "User [id=" + id + ", name=" + name + ", company=" + company + "]";

}

}

### UserService.java

package javase.spring.classes.service;

import javase.spring.annotation.Service;

import javase.spring.classes.pojo.User;

@Service

public class UserService {

//获取用户信息，最终在从数据库获取即可

public User get(Integer id) {

User u = new User();

u.setId(1);

u.setName("tony");

u.setCompany("达内科技教育集团");

return u;

}

}

### UserController.java

package javase.spring.classes.controller;

import javase.spring.annotation.Autowired;

import javase.spring.annotation.Controller;

import javase.spring.classes.pojo.User;

import javase.spring.classes.service.UserService;

@Controller

public class UserController {

@Autowired

private UserService userService;

//获取用户信息

public User get() {

User u = userService.get(1);

return u;

}

}

### RunApp.java

package javase.spring;

import javase.spring.annotation.ComponentScan;

import javase.spring.classes.controller.UserController;

import javase.spring.classes.pojo.User;

import javase.spring.parser.ApplicationContext;

import javase.spring.parser.ComponentScanParser;

@ComponentScan("javase.spring.classes")

public class RunApp {

public static void main(String[] args) throws Exception {

RunApp run = new RunApp();

ComponentScan cs = run.getClass().getAnnotation(ComponentScan.class); //获取类上的注解

String packagePath = cs.value(); //得到注解的value属性值

System.out.println("扫描的包路径："+packagePath+"\n");

//1.初始化环境

ApplicationContext context = new ComponentScanParser();

context.config(packagePath);

//2.获取bean来调用

System.out.println("\n业务调用过程：....................................... ");

UserController uc = context.getBean("userController");

User u = uc.get();

System.out.println(u.toString());

}

}

### 评论

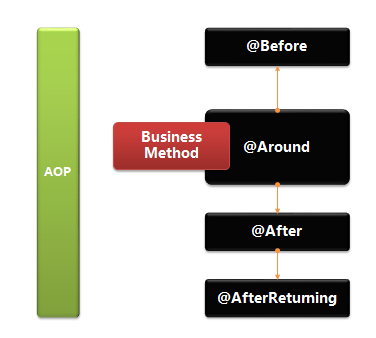
太酷了，可以看到，我们可以使用自己的spring框架了。只要按约定，指定包扫描的路径，要扫描的类上写明处理的注解，“底层”就自动包扫描，创建对象，存放在容器中，并且自动依赖注入。而我们不关心“底层”过程，直接写业务，获取对象来使用。

## 演示SpringAOP

### 通知的执行顺序

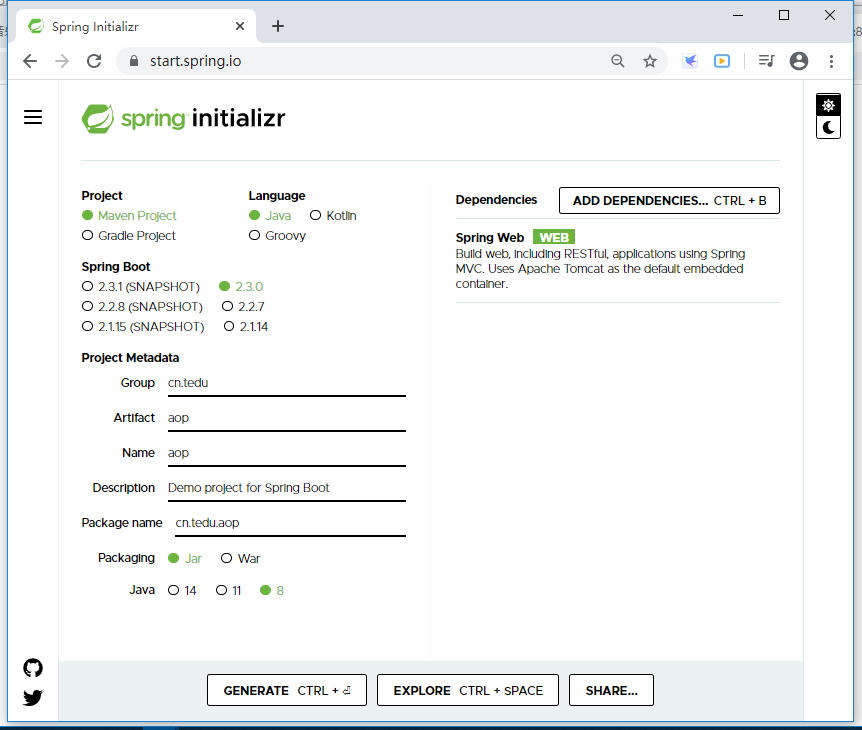
Spring框架实现了AOP面向切面，其引入了第三方AspectJ框架来具体实现。AspectJ提供了五种切入方式，术语称为通知advice。具体五种为：前置通知before、后置通知after、环绕通知around、返回后通知afterReturning、异常通知afterThrowing。异常通知特殊，这里暂不讨论。

可以看到，分别在业务方法（Business Method）的执行前后进行拦截，执行指定的代码。



### 创建springboot项目

官网：https://start.spring.io/



### pom.xml

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

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.3.0.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>cn.tedu</groupId>

<artifactId>aop</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>aop</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

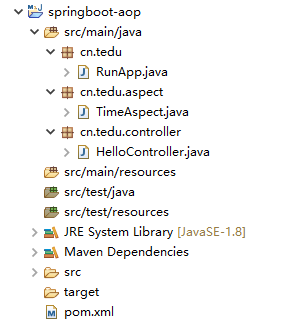
</plugin>

</plugins>

</build>

</project>

### 项目结构



### 添加aop依赖

注意需要自己手动加上，整合aspectJ

<dependency>

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

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

</dependency>

### HelloController.java

package cn.tedu.controller;

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

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

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:59:43

\* @description 描述：

\*/

@RestController

public class HelloController {

@RequestMapping("/hello")

public String hello() {

try {

Thread.sleep(10);

} catch (InterruptedException e) {

e.printStackTrace();

}

return "hi aop";

}

@RequestMapping("/welcome")

public String welcome() {

try {

Thread.sleep(800);

} catch (InterruptedException e) {

e.printStackTrace();

}

return "welcome aop";

}

}

### TimeAspect.java

package cn.tedu.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.AfterReturning;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.Pointcut;

import org.springframework.stereotype.Component;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:56:46

\* @description 描述：Aspect切面

\*/

@Component //组件，交给spring容器

@Aspect //切面

public class TimeAspect {

String methodName; //方法名

long startTime; //开始时间

//切点表达式，处理所有controller下包括其子目录中的\*.\*所有类的所有方法

//匹配所有目标类的public方法，第一个\*为返回类型，第二个\*为方法名

@Pointcut("execution( public \* cn.tedu.controller..\*.\*(..))")

public void aopPointCut() {}

@Before("aopPointCut()") //前置通知，业务执行前

public void doBefore(JoinPoint joinPoint) {

methodName = joinPoint.getSignature().getDeclaringTypeName() + "." + joinPoint.getSignature().getName();

startTime = System.currentTimeMillis();

}

@After("aopPointCut()") //后置通知，业务执行后

public void doAfter() {

long endTime = System.currentTimeMillis() - startTime;

System.out.println("执行 " + methodName + " 耗时为：" + endTime + "ms");

}

//正常返回通知，返回后执行

@AfterReturning(returning = "object", pointcut = "aopPointCut()")

public void doAfterReturning(Object object) {

System.out.println("response={"+object.toString()+"}");

}

@Around("aopPointCut()") //环绕通知

public Object doAround(ProceedingJoinPoint joinPoint) throws Throwable{

long start = System.currentTimeMillis();

try {

Object result = joinPoint.proceed(); //回调业务方法

long end = System.currentTimeMillis();

System.out.println("==around " + joinPoint + "\t耗时 : " + (end - start) + " ms!");

return result;

} catch (Throwable e) {

long end = System.currentTimeMillis();

System.out.println("==around " + joinPoint + "\t耗时 : " + (end - start) + " ms with exception : " + e.getMessage());

throw e;

}

}

}

### RunApp.java

package cn.tedu;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午4:05:37

\* @description 描述：

\*/

@SpringBootApplication

public class RunApp {

public static void main(String[] args) {

SpringApplication.run(RunApp.class, args);

}

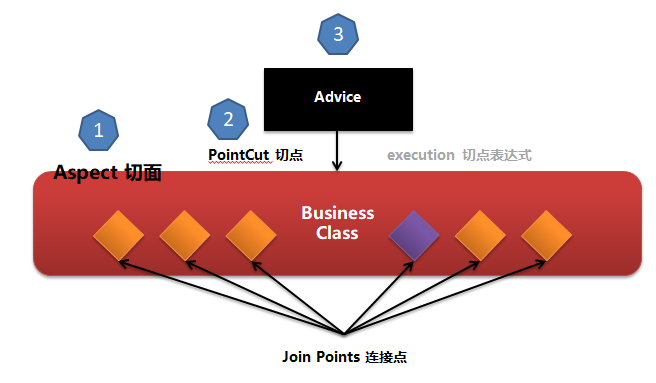
}

## 手写Spring框架之AOP

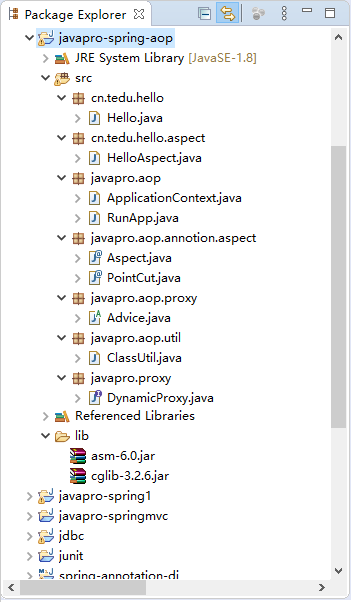
### 概念

Spring核心特征中除了IoC控制反转、DI依赖注入，还有一个核心就是强大的面向切面编程AOP（Aspect Oriented Programming）的实现。

Sring AOP有三要素：Aspect定义切面；具体通过切点（PointCut）配置切点表达式（expression）来指定在哪些类的哪些方法上织入横切逻辑；通过通知（Advice）来指定在切点上具体做什么事情。如方法前做什么，方法后做什么，抛出异常做什么。从而实现对象功能的增强。



### 项目结构



### ClassUtil.java

package javapro.aop.util;

import java.io.File;

import java.io.FileFilter;

import java.io.IOException;

import java.net.URL;

import java.net.URLDecoder;

import java.util.Enumeration;

import java.util.HashSet;

import java.util.Set;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:38:31

\* @description 描述：模拟aop底层实现工具类

\*/

public class ClassUtil {

//获取类加载器

public static ClassLoader getClassLoader() {

return Thread.currentThread().getContextClassLoader();

}

//加载指定包下的所有类（包扫描）

public static Set<Class<?>> getClassSet(String packageName) throws IOException{

Set<Class<?>> classSet = new HashSet<>();

//高效遍历资源文件对象，jdk新版本特性

//根路径+包路径

Enumeration<URL> urls = getClassLoader().getResources( packageName.replace(".", "/"));

while(urls.hasMoreElements()) {

URL url = urls.nextElement(); //把磁盘文件单做网络地址形式访问：http://abc，file://abc

if(url != null) {

String protocal = url.getProtocol(); //获取协议头

if( protocal.equals("file")) {

//可能有中文乱码，转码

String packagePath = URLDecoder.decode(url.getFile(), "UTF-8");

addClass(classSet, packagePath, packageName);

}

}

}

return classSet;

}

//目录下有很多文件，进行过滤.class的文件才处理，放入set集合中

private static void addClass(Set<Class<?>> classSet, String packagePath, String packageName) {

//过滤文件

File[] files = new File(packagePath).listFiles(new FileFilter() {

@Override

public boolean accept(File file) {

return (file.isFile() && file.getName().endsWith(".class") || file.isDirectory());

}

});

for(File file : files) {

String fileName = file.getName();

if ( file.isFile() ) { //文件

String className = fileName.substring(0, fileName.lastIndexOf("."));

if( null != packageName) {

className = packageName + "." + className; //全局限定名

doAddClass(classSet, className);

}

}else { //子目录

String subPackagePath = fileName;

if(null != packageName) {

subPackagePath = packagePath + "/" + subPackagePath;

}

String subPackageName = fileName;

if(null != packageName) {

subPackageName = packageName +"." + subPackageName;

}

//递归，遍历所有子目录

addClass(classSet, subPackagePath, subPackageName);

}

}

}

//获取className可以发生获取类对象

private static void doAddClass(Set<Class<?>> classSet, String className) {

//反射获取类

try {

Class<?> clazz = Class.forName(className);

classSet.add(clazz);

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

//首字母小写

public static String toLowerCaseFirstOne(String s) {

String first = "" + s.charAt(0);

return first.toLowerCase() + s.substring(1);

}

}

### DynamicProxy.java

package javapro.proxy;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:03:31

\* @description 描述：

\*/

public interface DynamicProxy {

public Object getInstance(Object target);

}

### Advice.java

package javapro.aop.proxy;

import java.lang.reflect.Method;

import javapro.proxy.DynamicProxy;

import net.sf.cglib.proxy.Enhancer;

import net.sf.cglib.proxy.MethodInterceptor;

import net.sf.cglib.proxy.MethodProxy;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:04:30

\* @description 描述：使用cglib动态代理方式增强被代理对象

\*/

public abstract class Advice implements MethodInterceptor, DynamicProxy {

private Object target; //被代理目标对象

private String proxyMethodName; //被代码对象具体方法名

@Override //使用cglib动态代理工具类创建代理对象

public Object getInstance(Object target) {

this.target = target;

Enhancer enhancer = new Enhancer();

enhancer.setSuperclass(this.target.getClass());

enhancer.setCallback(this);

return enhancer.create();

}

@Override

public Object intercept(Object proxy, Method method, Object[] args, MethodProxy methodProxy) throws Throwable {

Object result = null;

//判断和切点表达式一致方法，才进行加强，否则不进行加强的处理

if(proxyMethodName.equals( method.getName())) {

doBefore();

result = methodProxy.invokeSuper(proxy, args); //业务方法回调

doAfter();

}else {

result = methodProxy.invokeSuper(proxy, args); //业务方法回调

}

return result;

}

//抽象方法，延迟子类去实现

public abstract void doBefore(); //模拟前置通知

public abstract void doAfter(); //模拟后置通知

public String getProxyMethodName() {

return proxyMethodName;

}

public void setProxyMethodName(String proxyMethodName) {

this.proxyMethodName = proxyMethodName;

}

}

### @Aspect.java

package javapro.aop.annotation.aspect;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:18:31

\* @description 描述：模拟AOP切面

\*/

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

public @interface Aspect {

}

### @PointCut.java

package javapro.aop.annotation.aspect;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:19:21

\* @description 描述：模拟AOP切点

\*/

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface PointCut {

String value(); //设置切点表达式

}

### ApplicationContext.java

package javapro.aop;

import java.lang.reflect.Method;

import java.util.Set;

import java.util.concurrent.ConcurrentHashMap;

import javapro.aop.annotation.aspect.Aspect;

import javapro.aop.annotation.aspect.PointCut;

import javapro.aop.proxy.Advice;

import javapro.aop.util.ClassUtil;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:24:52

\* @description 描述：

\*/

public class ApplicationContext {

//容器（存放代理对象）

public static ConcurrentHashMap<String, Object> beans = new ConcurrentHashMap<String, Object>();

public ApplicationContext( String componentScan) {

initAopBeanMap(componentScan);

}

//初始化AOP容器，根据切点表达式规则进行解析，IoC

private void initAopBeanMap(String basePackage) {

try {

//遍历包路径下所有类

Set<Class<?>> classSet = ClassUtil.getClassSet(basePackage);

for(Class<?> clazz : classSet) {

//判断类上面是否有Aspect注解

if(clazz.isAnnotationPresent(Aspect.class)) {

Method[] methods = clazz.getDeclaredMethods();

for( Method method : methods) {

//判断是否满足切点表达式

PointCut pointCut = method.getAnnotation(PointCut.class);

if(null != pointCut) {

//形式：package.className\_methodName

String execution = pointCut.value(); //获取切点表达式

String[] pointCutArr = execution.split("\_");

String className = pointCutArr[0];

String methodName = pointCutArr[1];

//被代理对象

Object target = Class.forName(className).newInstance();

//创建切面类的代理对象

Advice proxy = (Advice)clazz.newInstance();

proxy.setProxyMethodName(methodName);

Object object = proxy.getInstance(target); //利用Advice通知target加强

if(object != null) {

String beanName = ClassUtil.toLowerCaseFirstOne(target.getClass().getSimpleName());

beans.put(beanName, object);

}

}

}

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

//从容器中获取对象实例

@SuppressWarnings("unchecked")

public <T> T getBean(String beanName) {

return (T)beans.get(beanName);

}

}

### Hello.java

package cn.tedu.hello;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午5:43:51

\* @description 描述：业务类

\*/

public class Hello {

public void hi() {

System.out.println("hi");

}

public void welcome() {

System.out.println("welcome");

}

}

### HelloAspect.java

package cn.tedu.hello.aspect;

import javapro.aop.annotation.aspect.Aspect;

import javapro.aop.annotation.aspect.PointCut;

import javapro.aop.proxy.Advice;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:46:38

\* @description 描述：

\*/

@Aspect

public class HelloAspect extends Advice{

//切点表达式：形式：全局限定名\_方法名

@PointCut("cn.tedu.hello.Hello\_hi")

public void testAspect() {}

@Override

public void doBefore() {

System.out.println("do before()");

}

@Override

public void doAfter() {

System.out.println("do after()");

}

}

### RunApp.java

package cn.tedu;

import cn.tedu.hello.Hello;

import javapro.aop.ApplicationContext;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午11:50:04

\* @description 描述：

\*/

public class RunApp {

public static void main(String[] args) {

//定义包扫描路径

String componentScan = "cn.tedu.hello";

ApplicationContext context = new ApplicationContext(componentScan);

Hello hello = context.getBean("hello");

hello.hi();

System.out.println();

hello.welcome();

}

}

## 小结

### 面试：IoC和DI

在平时的java应用开发中，我们要实现某一个功能或者说是完成某个业务逻辑时可能需要多个对象来协作完成，在没有使用Spring的时候，每个对象在需要使用他的合作对象时，自己均要使用像new object() 这样的语法来将合作对象创建出来，这个合作对象是由自己主动创建出来的，创建合作对象的主动权在自己手上，自己需要哪个合作对象，就主动去创建，创建合作对象的主动权和创建时机是由自己把控的，而这样就会使得对象间的耦合度高了，A对象需要使用合作对象B来共同完成一件事，A要使用B，那么A就对B产生了依赖，也就是A和B之间存在一种耦合关系，并且是紧密耦合在一起，而使用了Spring之后就不一样了，创建合作对象B的工作是由Spring来做的，Spring创建好B对象，然后存储到一个容器里面，当A对象需要使用B对象时，Spring就从存放对象的那个容器里面取出A要使用的那个B对象，然后交给A对象使用，至于Spring是如何创建那个对象，以及什么时候创建好对象的，A对象不需要关心这些细节问题(你是什么时候生的，怎么生出来的我可不关心，能帮我干活就行)，A得到Spring给我们的对象之后，两个人一起协作完成要完成的工作即可。

所以控制反转IoC(Inversion of Control)是说创建对象的控制权进行转移，以前创建对象的主动权和创建时机是由自己把控的，而现在这种权力转移到第三方，比如转移交给了IoC容器，它就是一个专门用来创建对象的工厂，你要什么对象，它就给你什么对象，有了 IoC容器，依赖关系就变了，原先的依赖关系就没了，它们都依赖IoC容器了，通过IoC容器来建立它们之间的关系。

总结一句话：IoC是设计思想，IoC有三个核心：BeanFactory、反射、DI。BeanFactory利用反射实现对象的创建，容器来存储对象，DI实现对象关系的绑定。

### 自动装配

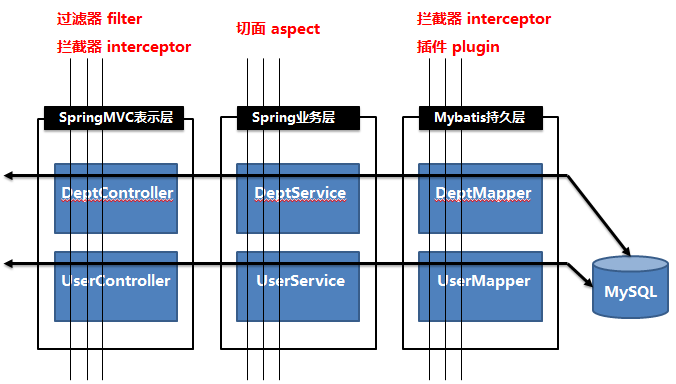
利用注解方式，我们只需要写@Autowired注解，底层就会去容器中找对应的对象，如果有获取到，反射调用其对应的set方法设置。而这个调用过程都是自动，我们没有手工去写set方法，所以这个过程也称为自动装配。

### AOP面向切面和OOP面向对象的关系？

OOP面向对象全面实现编程，AOP面向切面只是其中一个特例，AOP是OOP的有益补充。实际项目中两者结合使用。主要功能：日志记录、性能统计、安全控制、事务、缓存、异常处理等。

AOP实现原理是java动态代理，但是JDK的动态代理必须实现接口，所以Spring的AOP是用CGLIB方式实现的，CGLIB使用直接操纵字节码的ASM框架，所以可以做到不使用接口的情况下实现动态代理。

### 过滤器、拦截器、AOP面向切面



Filter过滤器、Interceptor拦截器、切面Aspect有相似之处，都是AOP思想的体现，都能实现效率统计、权限检查、日志记录等。

不同的是：

过滤器Filter是针对页面，页面的请求和响应，依赖于servlet容器，在实现上基于方法的回调。对所有请求进行过滤，比如：在过滤器中修改字符编码；在过滤器中修改HttpServletRequest的一些参数，包括：过滤低俗文字、危险字符等。

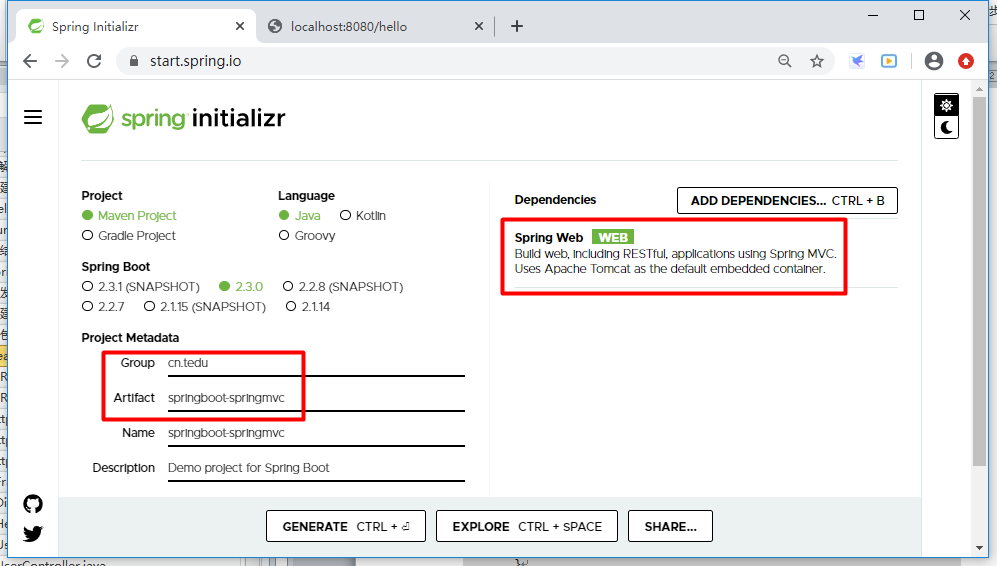
拦截器Interceptor依赖于web框架，在SpringMVC Interceptor中就是依赖于SpringMVC框架。在实现上基于Java的反射机制。由于拦截器是基于web框架的调用，因此可以使用Spring的依赖注入（DI）进行一些业务操作，同时一个拦截器实例在一个controller生命周期之内可以多次调用。但是缺点是只能对controller请求进行拦截，对其他的一些比如直接访问静态资源的请求则没办法进行拦截处理。

Aspect切面依赖于Spring框架，依赖于第三方AOP技术AspectJ。在实现上基于Java CGLIB动态代理技术，动态织入，本质也属于回调机制实现。

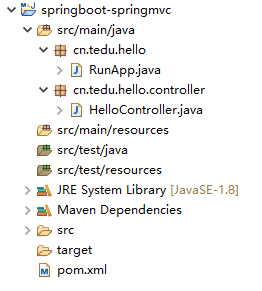
# 框架：手写SpringMVC框架

## 演示注解配置方式

### 产生pom文件



### 创建springboot项目



### HelloController.java

package cn.tedu.hello.controller;

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

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

@RestController

public class HelloController {

@RequestMapping("/hello")

public String hello() {

return "hello springmvc!";

}

}

### RunApp.java

package cn.tedu.hello;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class RunApp {

public static void main(String[] args) {

SpringApplication.run(RunApp.class, args);

}

}

### 小结

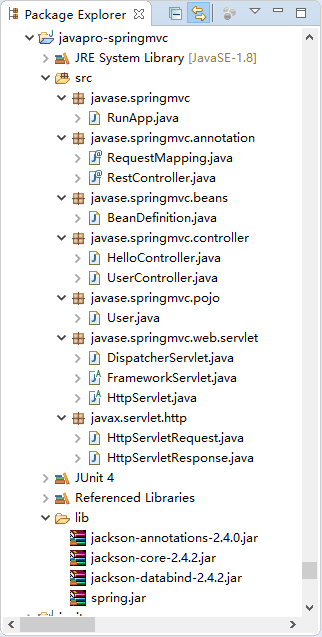
大家想想，怎么我们在浏览器中输入链接，最终就找到HelloController的hello方法，并执行呢？看似简单，其实这个是非常复杂的。下面我们就模拟这个过程。

## 手写SpringMVC框架

### 开发步骤

1. 启动加载
   1. 包扫描，创建对象Controller，加载Controller到容器中
   2. 包扫描，HandlerMapping负责，创建url和Controller方法的映射
2. DispatcherServlet
   1. 请求，解决url的解析，<http://localhost:8090/hello，得到/hello>
   2. 根据url找到对应的controller方法，反射调用方法
   3. 得到返回值，将结果对象转换为json字符串
   4. 响应，将json字符串返回

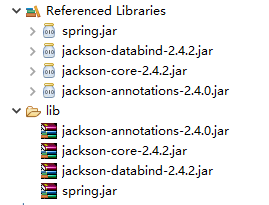
### 创建java项目



### 导包

我们手写的spring框架包，直接用它完成包扫描，创建对象，存入容器。

同时导入第三方json工具类jackson，3个jar包都会用到。



### BeanDefinition.java

package javase.springmvc.beans;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午8:38:19

\* @description 描述：

\*/

public class BeanDefinition {

private String uri; //浏览器uri

private String beanName;

private String methodName;

private boolean json; //是否进行json转换，RestController

public String getUri() {

return uri;

}

public void setUri(String uri) {

this.uri = uri;

}

public String getBeanName() {

return beanName;

}

public void setBeanName(String beanName) {

this.beanName = beanName;

}

public String getMethodName() {

return methodName;

}

public void setMethodName(String methodName) {

this.methodName = methodName;

}

public boolean isJson() {

return json;

}

public void setJson(boolean json) {

this.json = json;

}

}

### @RestController.java

package javase.springmvc.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午8:40:02

\* @description 描述：

\*/

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

public @interface RestController {

String value() default "";

}

### @RequestMapping.java

package javase.springmvc.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午8:41:00

\* @description 描述：

\*/

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface RequestMapping {

String value() default "";

}

### HttpServletRequest.java

模拟tomcat等中间件的请求对象：

package javax.servlet.http;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午8:42:24

\* @description 描述：模拟Sevlet容器，请求对象

\*/

public class HttpServletRequest {

protected String url;

public void setUrl(String url) {

this.url = url;

}

//请求：http://localhost:8080/hello uri=/hello

public String getRequestURI() {

String[] eles = url.split("/");

int pos = (eles[0]+1 + eles[1]+1 + eles[2]).length();

return url.substring(pos);

}

//获取方法名

public String getMethodName(String uri) {

return uri.substring(1); //去掉最前面斜杠

}

}

### HttpServletResponse.java

模拟tomcat等中间件的响应对象：

package javax.servlet.http;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午8:48:22

\* @description 描述：模拟servlet容器，响应（链式编程）

\*/

public class HttpServletResponse {

//链式编程的写法

public HttpServletResponse getWriter() {

return this;

}

public HttpServletResponse print( Object obj ) {

System.out.println(obj);

return this;

}

}

### HttpServlet.java

模拟Servlet容器

package javase.springmvc.web.servlet;

import java.lang.reflect.Method;

import java.util.HashMap;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.fasterxml.jackson.databind.ObjectMapper;

import javase.springmvc.beans.BeanDefinition;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:29:07

\* @description 描述：模拟servlet容器

\*/

public abstract class HttpServlet {

//引入jackson工具类转换java对象成json字符串

private static final ObjectMapper MAPPER = new ObjectMapper();

//容器，springmvc子容器

protected Map<String, Object> beans = new HashMap<String, Object>();

//requestMapping 和 controller映射关系

protected Map<String, BeanDefinition> uriMapping = new HashMap<String, BeanDefinition>();

//1. 初始化 init，交给框架初始化

public abstract void init(String packagePath) throws Exception;

//2. doService处理请求doGet/doPost

protected void doService(HttpServletRequest request, HttpServletResponse response) throws Exception {

BeanDefinition bd = uriMapping.get( request.getRequestURI() );

String beanName = bd.getBeanName();

Object instance = beans.get(beanName);

Class<?> clazz = instance.getClass();

Method m = clazz.getMethod( bd.getMethodName());

Object rtn = m.invoke(instance); //对controller的方法回调

if(bd.isJson()) {

rtn = MAPPER.writeValueAsString(rtn);

}

response.getWriter().print(rtn);

}

}

### FrameworkServlet.java

模拟springmvc包扫描，创建对象，放入容器

package javase.springmvc.web.servlet;

import java.lang.reflect.Method;

import java.util.ArrayList;

import java.util.List;

import javase.spring.annotation.Controller;

import javase.spring.util.FileUtil;

import javase.springmvc.annotation.RequestMapping;

import javase.springmvc.annotation.RestController;

import javase.springmvc.beans.BeanDefinition;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:39:37

\* @description 描述：模拟springmvc框架处理核心

\*/

public abstract class FrameworkServlet extends HttpServlet{

public void init(String packagePath) throws Exception{

//1、包扫描，处理controller

scanController(packagePath);

//2、RequestMapping注解扫描形成BeanDefinition

scanRequestMappings();

}

//只扫描controller，不实现di

private void scanController(String packagePath) throws Exception {

String dir = this.getClass().getResource("/").getPath();

dir += packagePath.replaceAll("\\.", "/");

List<String> fileList = new ArrayList<String>();

FileUtil.readClassFile(dir, fileList);

for(String className : fileList) {

Class<?> clazz = Class.forName(className);

//获取类上注解

Controller ac = clazz.getAnnotation(Controller.class);

RestController rc = clazz.getAnnotation(RestController.class);

//判断是否为controller或者restController

if( ac != null || rc != null) {

Object instance = clazz.newInstance();

String beanName = className.substring(className.lastIndexOf(".") +1);

beanName = FileUtil.toLowerCaseFirstOne(beanName);

beans.put(beanName, instance); //包扫描，创建controller，放入容器

}

}

}

//获取所有RequstMapping注解，获取value，解析

private void scanRequestMappings() {

//遍历所有的controller

for(String beanName : beans.keySet()) {

Object instance = beans.get(beanName);

//获取类中的所有方法

Method[] methods = instance.getClass().getDeclaredMethods();

for(Method m : methods) {

RequestMapping rm = m.getAnnotation(RequestMapping.class);

if(rm !=null) {

BeanDefinition bd = new BeanDefinition();

bd.setUri(rm.value());

bd.setBeanName(beanName);

bd.setMethodName(m.getName());

//是否转换json

RestController rc = instance.getClass().getAnnotation(RestController.class);

if( rc != null) {

bd.setJson(true);

}

uriMapping.put(bd.getUri(), bd);

}

}

}

}

@SuppressWarnings("unchecked")

public <T> T getBean(String beanName) {

return (T) beans.get(beanName);

}

}

### DispatcherServlet.java

模拟springmvc核心分发器

package javase.springmvc.web.servlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javase.spring.parser.ApplicationContext;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:56:12

\* @description 描述：模拟springmvc核心分发器

\*/

public class DispatcherServlet extends FrameworkServlet{

public void initStrategies(ApplicationContext context, String packagePath) throws Exception{

init(packagePath);

}

public void doDispatch(HttpServletRequest request, HttpServletResponse response) throws Exception {

doService(request, response);

}

}

### HelloController.java

package javase.springmvc.controller;

import javase.spring.annotation.Controller;

import javase.springmvc.annotation.RequestMapping;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午2:16:13

\* @description 描述：

\*/

@Controller

public class HelloController {

@RequestMapping("/hello")

public String hello() {

return "hello springmvc";

}

@RequestMapping("/welcome")

public String welcome() {

return "welcome springmvc";

}

}

### User.java

package javase.springmvc.pojo;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午2:40:11

\* @description 描述：

\*/

public class User {

private Integer id;

private String name;

private String company;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCompany() {

return company;

}

public void setCompany(String company) {

this.company = company;

}

}

### UserController.java

package javase.springmvc.controller;

import javase.springmvc.annotation.RequestMapping;

import javase.springmvc.annotation.RestController;

import javase.springmvc.pojo.User;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午2:40:49

\* @description 描述：

\*/

@RestController

public class UserController {

@RequestMapping("/user/get")

public User get() {

User u = new User();

u.setId(7);

u.setName("tony");

u.setCompany("达内科技");

return u;

}

}

### RunApp.java

package javase.springmvc;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javase.spring.annotation.ComponentScan;

import javase.spring.parser.ApplicationContext;

import javase.spring.parser.ComponentScanParser;

import javase.springmvc.web.servlet.DispatcherServlet;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午2:17:55

\* @description 描述：

\*/

/\*

\* 两条主线：

\* 1、DispatcherServlet来实现包扫描controller，创建对象，放入容器

\* 2、用户访问url、经过解析，知道对于controller来进行处理，返回json

\* http://localhost:8080/hello

\* HelloController.hello()

\*/

@ComponentScan("javase.springmvc.controller")

public class RunApp {

public static void main(String[] args) throws Exception {

RunApp run = new RunApp();

ComponentScan cs = run.getClass().getAnnotation(ComponentScan.class);

String packagePath = cs.value();

System.out.println("扫描的包路径：" + packagePath + "\n");

//1、初始化环境

ApplicationContext context = new ComponentScanParser();

//2、获取bean来调用

System.out.println("业务调用过程......................");

//解析url获取到controller，并传入参数

DispatcherServlet dispatcher = new DispatcherServlet();

dispatcher.initStrategies(context, packagePath);

//模拟浏览器发起请求后，请求内容被封装到request对象中

HttpServletResponse response = new HttpServletResponse();

HttpServletRequest request = new HttpServletRequest();

//3、模拟用户浏览器请求

String url = "http://localhost:8080/hello";

System.out.println("访问url：" + url);

request.setUrl(url);

dispatcher.doDispatch(request, response);

//4、RestController支持，返回值json字符串

url = "http://localhost:8080/user/get";

System.out.println("访问url：" + url);

request.setUrl(url);

dispatcher.doDispatch(request, response);

}

}

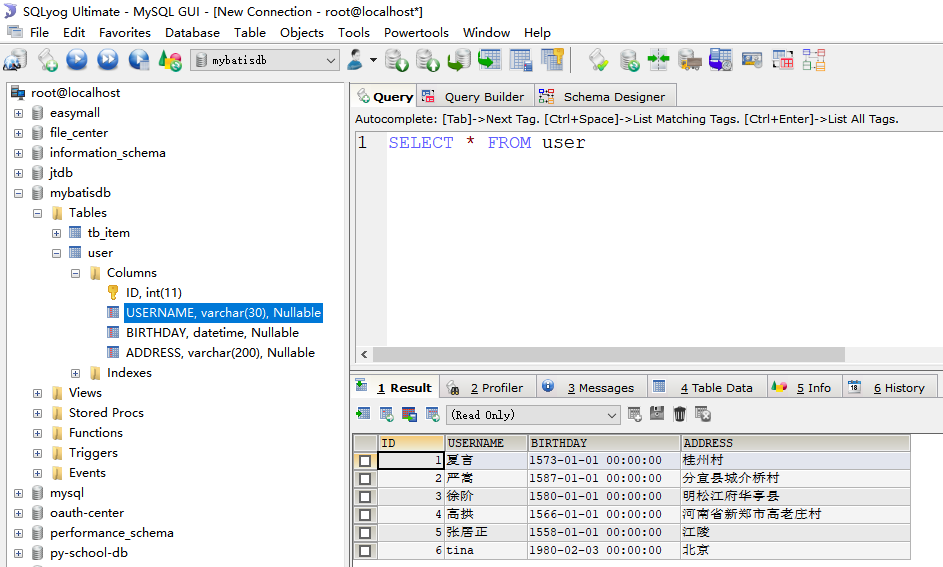
# 框架：手写Mybatis框架

## 演示xml配置方式

### 运行过程

1. 利用mybatis提供的工具类，读取sqlMapConfig.xml核心配置文件，获取配置信息，如：数据库链接信息，封装在组件Configuration
2. 获取sqlMapConfig.xml中的<mappers>，组件mapperStatement，解析sql
3. 创建SqlSessionFactory对象
4. fatory.openSession();创建SqlSession对象
5. 根据statement参数去mapperStatement集合中获取对应的SQL语句
6. Jdbc执行sql访问数据库
7. 将返回的数据封装到java对象中
8. 打印展示

### 数据库脚本



DROP TABLE IF EXISTS `user`;

CREATE TABLE `user` (

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

`USER\_NAME` varchar(30) DEFAULT NULL,

`BIRTHDAY` datetime DEFAULT NULL,

`ADDRESS` varchar(200) DEFAULT NULL,

PRIMARY KEY (`ID`)

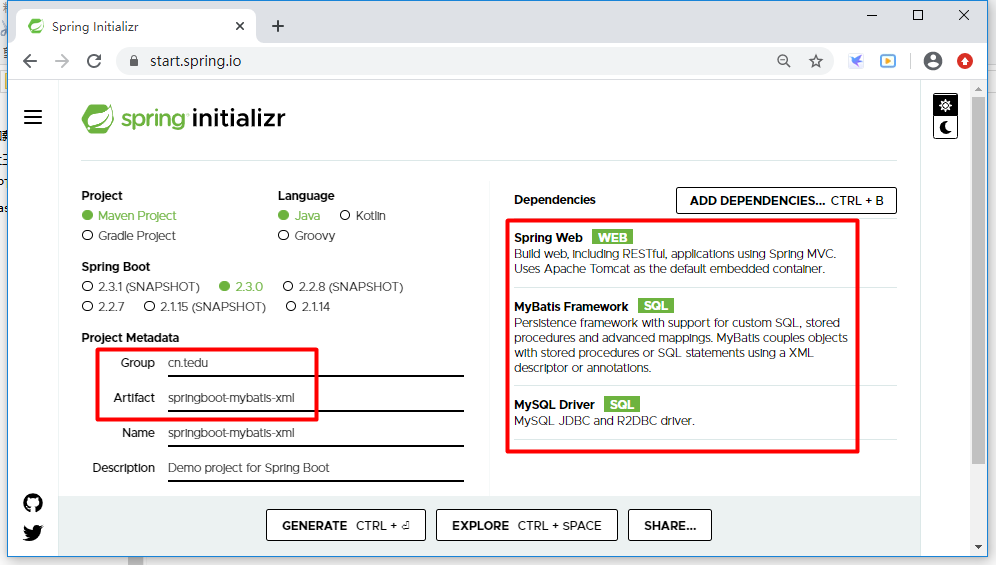
) ENGINE=InnoDB AUTO\_INCREMENT=8 DEFAULT CHARSET=utf8;

/\*Data for the table `user` \*/

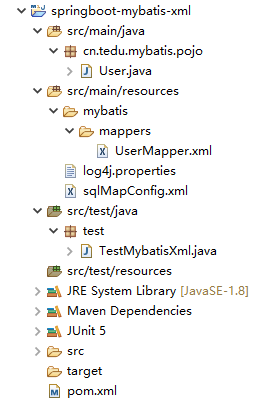
insert into `user`(`ID`,`USER\_NAME`,`BIRTHDAY`,`ADDRESS`) values (1,'夏言','1573-01-01 00:00:00','桂州村'),(2,'严嵩','1587-01-01 00:00:00','分宜县城介桥村'),(3,'徐阶','1580-01-01 00:00:00','明松江府华亭县'),(4,'高拱','1566-01-01 00:00:00','河南省新郑市高老庄村'),(5,'张居正','1558-01-01 00:00:00','江陵'),(6,'tina','1980-02-03 00:00:00','北京');

### 创建springboot工程

到官网https://start.spring.io/上生成pom.xml文件



### 项目结构



### pom.xml

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

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.3.0.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>cn.tedu</groupId>

<artifactId>springboot-mybatis-xml</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springboot-mybatis-xml</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

<version>2.1.2</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

### log4j.properties

Mybatis内置log4j日志系统，配置log4j可自动打印执行的SQL语句，方便我们开发时对SQL的调试，大多错误都是SQL语句的拼接和参数错误。

注意：注意下面的配置路径，路径错误，不会SQL打印

log4j.rootLogger=INFO,A1

log4j.logger.cn.tedu.mybatis.pojo=DEBUG

log4j.appender.A1=org.apache.log4j.ConsoleAppender

log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n

### sqlMapConfig.xml

注意mysql数据库驱动升级后，需要增加链接参数：

jdbc:mysql://localhost:3307/mybatisdb

?serverTimezone=GMT%2B8&amp;useUnicode=true&amp;characterEncoding=utf8&amp;useSSL=false

GMT@2B代表+号，&amp;代表&符合。

<?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>

<environments default="test">

<environment id="test">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

<property name="driver" value="com.mysql.cj.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3307/mybatisdb?serverTimezone=GMT%2B8&amp;useUnicode=true&amp;characterEncoding=utf8&amp;useSSL=false"/>

<property name="username" value="root"/>

<property name="password" value="root"/>

</dataSource>

</environment>

</environments>

<mappers>

<mapper resource="mybatis/mappers/UserMapper.xml"/>

</mappers>

</configuration>

### User.java

package cn.tedu.mybatis.pojo;

import java.util.Date;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午3:29:14

\* @description 描述：

\*/

public class User {

private Integer id;

private String username;

private Date birthday;

private String address;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public Date getBirthday() {

return birthday;

}

public void setBirthday(Date birthday) {

this.birthday = birthday;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

@Override

public String toString() {

return "User [id=" + id + ", username=" + username + ", birthday=" + birthday + ", address=" + address + "]";

}

}

### UserMapper.xml

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

<!DOCTYPE mapper

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

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

<!-- namespace命名空间 -->

<mapper namespace="cn.tedu.mybatis.mapper.UserMapper">

<!-- 查询所有，resultType返回值，标识返回类型List<> -->

<select id="findAll" resultType="cn.tedu.mybatis.pojo.User">

select \* from user

</select>

<!-- 获取一个用户的信息（一条记录）#{id}参数，解析获取id值 -->

<select id="get" parameterType="integer" resultType="cn.tedu.mybatis.pojo.User">

SELECT \* FROM USER WHERE id = #{id}

</select>

</mapper>

### TestMybatisXml.java

package test;

import java.io.IOException;

import java.io.InputStream;

import java.util.List;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import org.junit.Test;

import cn.tedu.mybatis.pojo.User;

public class TestMybatisXml {

//成员变量，线程安全问题：

private SqlSessionFactory factory; //线程安全

@Test //查询所有记录

public void find() throws IOException{

//读取核心配置文件，路径要求：

InputStream is = Resources.getResourceAsStream("sqlMapConfig.xml");

//获取到工厂对象

factory = new SqlSessionFactoryBuilder().build(is);

//创建SqlSession对象

SqlSession session = factory.openSession();

//查询所有的记录，namespace+"."+id

String statement = "cn.tedu.mybatis.mapper.UserMapper.findAll";

//selectList查询，找到映射文件，找到其中一个方法

List<User> userList = session.selectList(statement);

//遍历对象

for(User u:userList){

System.out.println(u);

}

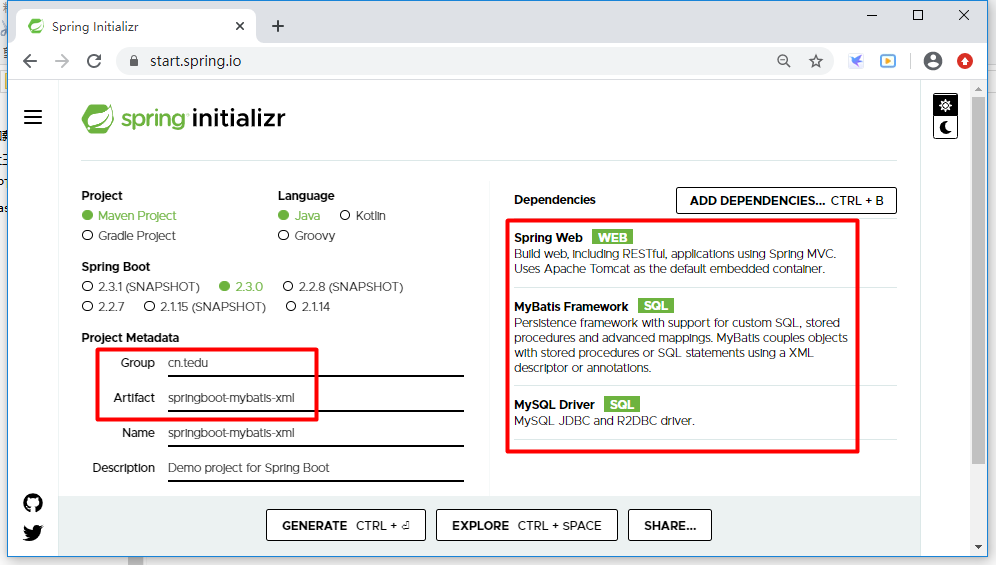
}

}

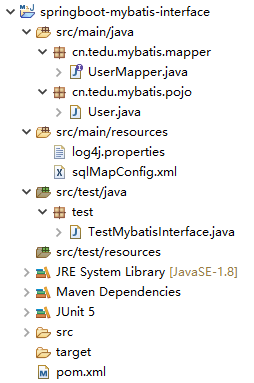
## 演示接口配置方式

### 创建springboot工程

到官网https://start.spring.io/上生成pom.xml文件



### 项目结构



### pom.xml

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

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.2.5.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>cn.tedu</groupId>

<artifactId>springboot-mybatis</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springboot-mybatis</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

<version>2.1.2</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

### log4j.properties

Mybatis内置log4j日志系统，配置log4j可自动打印执行的SQL语句，方便我们开发时对SQL的调试，大多错误都是SQL语句的拼接和参数错误。

注意：注意下面的配置路径，路径错误，不会SQL打印

log4j.rootLogger=INFO,A1

log4j.logger.cn.tedu.mybatis.mapper=DEBUG

log4j.appender.A1=org.apache.log4j.ConsoleAppender

log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n

### sqlMapConfig.xml

注意mysql数据库驱动升级后，需要增加链接参数：

jdbc:mysql://localhost:3307/mybatisdb

?serverTimezone=GMT%2B8&amp;useUnicode=true&amp;characterEncoding=utf8&amp;useSSL=false

GMT@2B代表+号，&amp;代表&符合。

<?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>

<environments default="test">

<environment id="test">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

<property name="driver" value="com.mysql.cj.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3307/mybatisdb?serverTimezone=GMT%2B8&amp;useUnicode=true&amp;characterEncoding=utf8&amp;useSSL=false"/>

<property name="username" value="root"/>

<property name="password" value="root"/>

</dataSource>

</environment>

</environments>

<mappers>

<mapper class="cn.tedu.mybatis.mapper.UserMapper"/>

</mappers>

</configuration>

### User.java

package cn.tedu.mybatis.pojo;

import java.util.Date;

public class User {

private Integer id;

private String name;

private Date birthday;

private String address;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public java.util.Date getBirthday() {

return birthday;

}

public void setBirthday(java.util.Date birthday) {

this.birthday = birthday;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

@Override

public String toString() {

return "User [id=" + id + ", name=" + name + ", birthday=" + birthday + ", address=" + address + "]";

}

}

### UserMapper.java

package cn.tedu.mybatis.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Select;

import cn.tedu.mybatis.pojo.User;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午5:23:44

\* @description 描述：mybatis映射接口文件

\*/

public interface UserMapper {

@Select("select \* from user")

public List<User> findAll();

@Select("SELECT \* FROM USER WHERE id = #{id}")

public User get(Integer id);

}

### TestMybatisInterface.java

package test;

import java.io.IOException;

import java.io.InputStream;

import java.util.List;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import org.junit.jupiter.api.Test;

import cn.tedu.mybatis.mapper.UserMapper;

import cn.tedu.mybatis.pojo.User;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午5:31:38

\* @description 描述：

\*/

public class TestMybatisInterface {

//成员变量，线程安全问题：

private SqlSessionFactory factory; //线程安全

@Test //查询所有记录

public void find() throws IOException{

//读取核心配置文件，路径要求：

InputStream is = Resources.getResourceAsStream("sqlMapConfig.xml");

factory = new SqlSessionFactoryBuilder().build(is);

SqlSession session = factory.openSession();

UserMapper mapper = session.getMapper(UserMapper.class);

List<User> userList = mapper.findAll();

for(User u :userList) {

System.out.println(u);

}

}

}

## xml和接口方式差异

学习一个技术，其中有一点非常重要技巧，就是比较，比较前后两者的差异，搞清哪些代码是死的，哪些代码是活的，使用时需要注意什么。了解这些，洞悉差异，能更快的掌握一门新的技术。

### log4j日志配置

xml:

log4j.logger.cn.tedu.mybatis.pojo=DEBUG

接口：

log4j.logger.cn.tedu.mybatis.mapper=DEBUG

注意：配置到pojo也可以

### sqlMapConfig.xml

xml:

<mappers>

<mapper resource="mybatis/mappers/UserMapper.xml"/>

</mappers>

接口：

<mappers>

<mapper class="cn.tedu.mybatis.mapper.UserMapper"/>

</mappers>

### 映射配置

xml：

UserMapper.xml

接口：

UserMapper.java

### 访问api

xml：

List<User> userList = session.selectList(statement);

接口：

UserMapper mapper = session.getMapper(UserMapper.class);

List<User> userList = mapper.findAll();

## 解析mybatis映射文件

### UserMapper.xml

简单点，好做练习

注意路径：resources/mybatis/mappers/UserMapper.xml，这是项目管理路径，真正文件保存路径是target/classes/mybatis/mappers/UserMapper.xml。

解析要求：

根据传入的namespace.id找到对应的标签，然后解析出其SQL语句

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

<!DOCTYPE mapper

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

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

<mapper namespace="cn.tedu.mybatis.mapper.UserMapper">

<!-- 查询所有，resultType返回值默认List<User> -->

<select id="findAll" resultType="cn.tedu.mybatis.pojo.User">

select \* from user

</select>

<!-- 查询一条 -->

<select id="get" parameterType="integer" resultType="cn.tedu.mybatis.pojo.User">

select \* from user where id = #{id}

</select>

</mapper>

### MapperXmlParser.java

package test;

import java.io.FileInputStream;

import java.io.InputStream;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import org.dom4j.Attribute;

import org.dom4j.Document;

import org.dom4j.Element;

import org.dom4j.io.SAXReader;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午4:35:36

\* @description 描述：

\*/

public class MapperXmlParser {

//集合，statement=namespace.id = sql

private static Map<String, String> mapperStatements = new HashMap<String, String>();

public void configuration(String namespace) throws Exception {

/\*

\* 开发步骤：

\* 1、创建SAX解析对象SAXReader

\* 2、获取当前映射文件所在路径

\* 3、创建输入流，读取路径下的UserMapper.xml

\* 4、读取文件流，转换成Document对象

\* 5、获取xml的根节点，遍历

\* 6、拿到里面元素节点，判断类型，获取id属性

\* 7、获取属性值id等和内容sql语句

\* 8、保存map中（key=namespace.id，value=sql）

\*/

SAXReader reader = new SAXReader();

String dir = this.getClass().getResource("/").getPath();

dir += "../classes/mybatis/mappers/"; //真正存放文件路径

InputStream is = new FileInputStream(dir + "UserMapper.xml");

Document doc = reader.read(is); //读取到内存中

is.close();

Element root = doc.getRootElement(); //获取根元素

for(Iterator<Element> it0 = root.elementIterator(); it0.hasNext(); ) {

Element ele = it0.next();

//System.out.println(ele.asXML());

//获取属性

for(Iterator<Attribute> it1 = ele.attributeIterator(); it1.hasNext(); ) {

Attribute attr = it1.next();

if("id".equals(attr.getName())) { //属性的名字

String key = namespace + "." + attr.getValue(); //属性的值

String sql = ele.getStringValue().trim(); //元素内容

mapperStatements.put(key, sql);

}

}

}

}

public static void main(String[] args) throws Exception {

MapperXmlParser parser = new MapperXmlParser();

String namespace = "cn.tedu.mybatis.mapper.UserMapper";

parser.configuration(namespace);

for(String id : mapperStatements.keySet()) {

System.out.println(id+"="+mapperStatements.get(id));

}

}

}

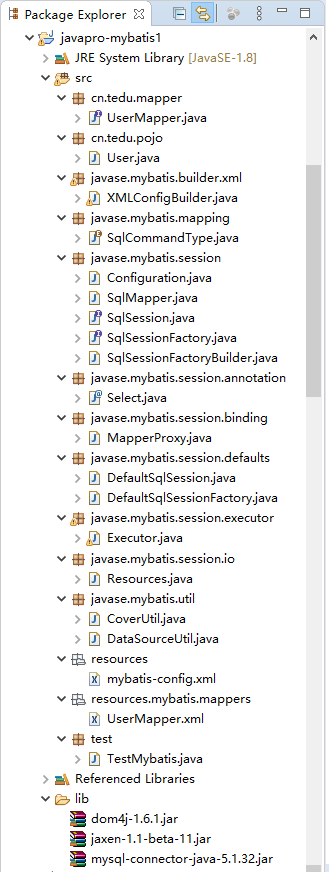
执行结果：

cn.tedu.mybatis.mapper.UserMapper.get=select \* from user where id = #{id}

cn.tedu.mybatis.mapper.UserMapper.findAll=select \* from user

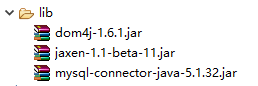
## 手写Mybatis框架

### 创建java工程

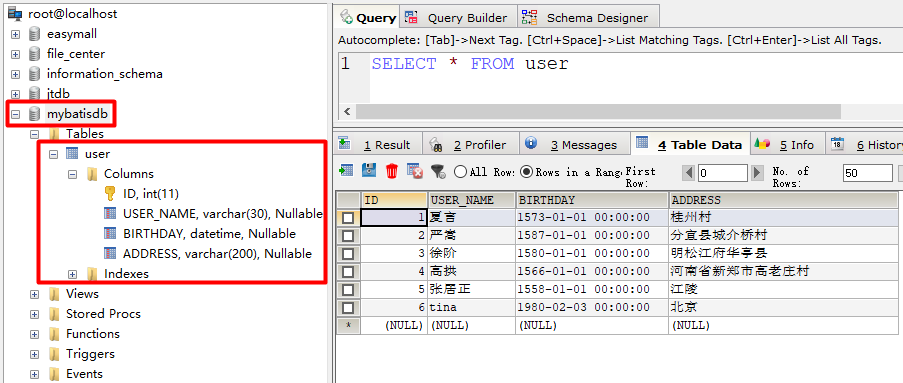


### 导包

解析XML文件我使用了dom4j和xpath方式，需要前两个包支持。同时，我们进行数据库访问，所以需要mysql驱动包。



### 数据库脚本



脚本：

CREATE DATABASE /\*!32312 IF NOT EXISTS\*/`mybatisdb` /\*!40100 DEFAULT CHARACTER SET utf8 \*/;

USE `mybatisdb`;

/\*Table structure for table `tb\_item` \*/

DROP TABLE IF EXISTS `tb\_item`;

CREATE TABLE `tb\_item` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '商品id，同时也是商品编号',

`title` varchar(100) NOT NULL COMMENT '商品标题',

`sell\_point` varchar(500) DEFAULT NULL COMMENT '商品卖点',

`price` bigint(20) NOT NULL COMMENT '商品价格，单位为：分',

`num` int(10) NOT NULL COMMENT '库存数量',

`barcode` varchar(30) DEFAULT NULL COMMENT '商品条形码',

`image` varchar(500) DEFAULT NULL COMMENT '商品图片',

`cid` bigint(10) NOT NULL COMMENT '所属类目，叶子类目',

`status` tinyint(4) NOT NULL DEFAULT 1 COMMENT '商品状态，1-正常，2-下架，3-删除',

`created` datetime NOT NULL COMMENT '创建时间',

`updated` datetime NOT NULL COMMENT '更新时间',

PRIMARY KEY (`id`),

KEY `cid` (`cid`),

KEY `status` (`status`),

KEY `updated` (`updated`),

KEY `title` (`title`)

) ENGINE=InnoDB AUTO\_INCREMENT=1474391957 DEFAULT CHARSET=utf8 COMMENT='商品表';

/\*Data for the table `tb\_item` \*/

insert into `tb\_item`(`id`,`title`,`sell\_point`,`price`,`num`,`barcode`,`image`,`cid`,`status`,`created`,`updated`) values (1473724868,'三星（SAMSUNG）HG48AC460KJ 48英寸LED液晶电视超薄窄边 正品特价','',2999000,99999,NULL,'http://image.jt.com/jd/1c949a7322304ed4991fb232ae094f11.jpg',76,1,'2015-03-08 21:29:33','2015-03-08 21:29:33'),(1473743901,'里奥(NEO) KP-MX4 冒险者 移动4G手机 双卡双待 白色','5英寸OGS全贴合屏四核64位1.5GHz1300万像素双卡双待移动4G手机 情人节特惠',999000,99999,NULL,'http://image.jt.com/jd/20c515bbd1e24636889c714667c5a90e.jpg',560,1,'2015-03-08 21:33:47','2015-03-08 21:33:47'),(1474318759,'奥克斯(AUX) M288 移动联通2G老人手机 双卡双待 土豪金','老年机大字体大按键大音量 超长待机 触屏手写 手电筒外放收音机语音王QQ电子书',139000,99999,NULL,'http://image.jt.com/jd/c97a5b84a5d740da9bee3626f1732bcb.jpg',560,1,'2015-03-08 21:33:47','2015-03-08 21:33:47'),(1474391927,'neeea 小米（MI）小米米键 颜色随机 官方正品','智键手机快捷键防尘塞',19000,99999,'','http://image.jt.com/images/2015/12/29/2015122903370299506833.png,http://image.jt.com/images/2015/12/29/2015122903370372008423.png,http://image.jt.com/images/2015/12/29/2015122903370381301388.jpg,http://image.jt.com/images/2015/12/29/201512290337038890690.jpg',560,1,'2015-03-08 21:33:48','2015-06-04 15:47:05'),(1474391945,'星星 Galaxy S7 edge（G9350）32G版 铂光金移动联通电信4G手机 双卡双待 骁龙820手机','4月23日-5月8日可参加白条12期免息活动！骁龙820手机！好评过万！',568800,100,'12345657','http://image.jt.com/images/2016/04/25/2016042502095921402458.jpg,http://image.jt.com/images/2016/04/25/2016042502100039909929.jpg,http://image.jt.com/images/2016/04/25/2016042502100056201599.jpg,http://image.jt.com/images/2016/04/25/2016042502100072208750.jpg,http://image.jt.com/images/2016/04/25/2016042502100084509017.jpg',4,1,'2016-04-25 14:12:10','2016-10-08 17:14:37'),(1474391947,'魅蓝note113 全网通公开版 16GB 银色 移动联通电信4G手机 双卡双待','【已售罄！12月12日0点开抢！】2GB+16GB！5.5英寸！500万+1300万像素！4100mAh大电池！推荐购买套装版\r\n选择移动/联通/电信优惠购，流量话费送不停，惊喜不断，优惠多多！！！',79900,4,'5123123','http://image.jt.com/images/2016/12/09/1481270026450296.jpg,http://image.jt.com/images/2016/12/09/1481270026702644.jpg,http://image.jt.com/images/2016/12/09/1481270026779116.jpg,http://image.jt.com/images/2016/12/09/1481270026932421.jpg,http://image.jt.com/images/2016/12/09/1481270027064761.jpg',560,1,'2016-12-09 15:54:17','2017-01-12 16:45:36'),(1474391948,'华为(HUAWEI) Mate9 4G手机 双卡双待 香槟金 全网通(4GB RAM+64GB ROM)','【电子发票】有原装耳机+后壳+膜(包装内赠)部分地区就近发货 买曲面mate9Pro',394800,100,'1234567','http://image.jt.com/images/2017/02/09/1486606889663134.jpg,http://image.jt.com/images/2017/02/09/1486606889945429.jpg,http://image.jt.com/images/2017/02/09/1486606890081361.jpg,http://image.jt.com/images/2017/02/09/1486606890232569.jpg,http://image.jt.com/images/2017/02/09/1486606890379738.jpg',560,1,'2017-02-09 10:27:30','2017-02-21 15:59:11'),(1474391953,'华为 G9 青春版 黑色 移动联通电信4G手机 双卡双待','下单立减200！骁龙芯片！3GB运行内存！索尼1300万摄像头！更多优惠请点击！\r\n选择下方购买方式的【移动】【电信】【联通】优惠购，套餐有优惠，还有话费返还！',159900,100,'1234657','http://image.jt.com/images/2017/03/14/1489473034025569.jpg,http://image.jt.com/images/2017/03/14/1489473034224700.jpg,http://image.jt.com/images/2017/03/14/1489473034350227.jpg,http://image.jt.com/images/2017/03/14/1489473034598105.jpg,http://image.jt.com/images/2017/03/14/1489473034783649.jpg',560,1,'2017-03-14 14:33:08','2017-03-14 14:33:08'),(1474391954,'东芝（TOSHIBA）55U66EBC 55英寸4K智能火箭炮音响系统 液晶电视（银灰黑）','爆品！仅限3.21号3499元抢购！东芝曲面4K新品首发上市！',349900,100,'1234567','http://image.jt.com/images/2017/03/21/1490080380892150.jpg,http://image.jt.com/images/2017/03/21/1490080381080439.jpg',76,1,'2017-03-21 15:14:25','2017-03-21 15:14:25');

/\*Table structure for table `user` \*/

DROP TABLE IF EXISTS `user`;

CREATE TABLE `user` (

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

`USER\_NAME` varchar(30) DEFAULT NULL,

`BIRTHDAY` datetime DEFAULT NULL,

`ADDRESS` varchar(200) DEFAULT NULL,

PRIMARY KEY (`ID`)

) ENGINE=InnoDB AUTO\_INCREMENT=8 DEFAULT CHARSET=utf8;

/\*Data for the table `user` \*/

insert into `user`(`ID`,`USER\_NAME`,`BIRTHDAY`,`ADDRESS`) values (1,'夏言','1573-01-01 00:00:00','桂州村'),(2,'严嵩','1587-01-01 00:00:00','分宜县城介桥村'),(3,'徐阶','1580-01-01 00:00:00','明松江府华亭县'),(4,'高拱','1566-01-01 00:00:00','河南省新郑市高老庄村'),(5,'张居正','1558-01-01 00:00:00','江陵'),(6,'tina','1980-02-03 00:00:00','北京');

### CoverUtil.java

package javase.mybatis.util;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:42:37

\* @description 描述：驼峰规则转换

\*/

public class CoverUtil {

public static final char UNDERLINE = '\_';

//驼峰规则转成数据库下划线方式 userName user\_name

public static String camelToUnderline(String s) {

if(s ==null || "".equals(s.trim())) {

return "";

}

int len = s.length();

StringBuilder sb = new StringBuilder();

for(int i=0; i< len; i++) {

char ch = s.charAt(i);

if( Character.isUpperCase(ch)) {

sb.append(UNDERLINE);

sb.append(Character.toLowerCase(ch)); //把大写字符转换小写字符

}else {

sb.append(ch);

}

}

return sb.toString();

}

//数据库下划线方式转换驼峰规则 user\_name userName

public static String underlineToCamel(String s) {

if(s==null || "".equals(s.trim())) {

return "";

}

s = s.toLowerCase(); //不论大小写，都先强制转换小写

int len = s.length();

StringBuilder sb = new StringBuilder();

for( int i=0; i<len; i++) {

char ch = s.charAt(i);

if(ch == UNDERLINE) {

if(++i < len) { //获取下标后一个值，判断是否已经越界

sb.append(Character.toUpperCase(s.charAt(i)));

}

}else {

sb.append(ch);

}

}

return sb.toString();

}

}

### @Select.java

package javase.mybatis.session.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:14:38

\* @description 描述：模拟mybatis中接口方法上注解 select查询

\*/

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface Select {

String value(); //存放sql语句

}

### SqlCommandType.java

package javase.mybatis.mapping;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午8:45:33

\* @description 描述：完全模拟mybatis SQL类型

\*/

public enum SqlCommandType {

UNKNOWN, INSERT, UPDATE, DELETE, SELECT, FLUSH;

}

### SqlMapper.java

package javase.mybatis.session;

import javase.mybatis.mapping.SqlCommandType;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:16:31

\* @description 描述：类似pojo对象，存储sql和返回值类型字符串

\*/

public class SqlMapper {

private String id; //xml标签id，接口方法名

private SqlCommandType sqlCommandType;

private String queryString; //查询sql语句

private String resultType; //返回值类型字符串

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public SqlCommandType getSqlCommandType() {

return sqlCommandType;

}

public void setSqlCommandType(SqlCommandType sqlCommandType) {

this.sqlCommandType = sqlCommandType;

}

public String getQueryString() {

return queryString;

}

public void setQueryString(String queryString) {

this.queryString = queryString;

}

public String getResultType() {

return resultType;

}

public void setResultType(String resultType) {

this.resultType = resultType;

}

}

### Configuration.java

package javase.mybatis.session;

import java.util.HashMap;

import java.util.Map;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:18:14

\* @description 描述：读取数据库信息和mapper映射信息

\*/

public class Configuration {

private String driver;

private String url;

private String username;

private String password;

//key=namespace.id（.findAll），value=（sql,User)

private Map<String, SqlMapper> sqlMappers = new HashMap<>();

public String getDriver() {

return driver;

}

public void setDriver(String driver) {

this.driver = driver;

}

public String getUrl() {

return url;

}

public void setUrl(String url) {

this.url = url;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public Map<String, SqlMapper> getSqlMappers() {

return sqlMappers;

}

public void setSqlMappers(Map<String, SqlMapper> sqlMappers) {

this.sqlMappers = sqlMappers;

}

}

### DataSourceUtil.java

package javase.mybatis.util;

import java.sql.Connection;

import java.sql.DriverManager;

import javase.mybatis.session.Configuration;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:21:35

\* @description 描述：获取数据库链接

\*/

public class DataSourceUtil {

public static Connection getConnection(Configuration cfg) {

try {

Class.forName(cfg.getDriver());

return DriverManager.getConnection(cfg.getUrl(), cfg.getUsername(), cfg.getPassword());

} catch (Exception e) {

//如果数据库链接失败，抛出一个运行时异常，中断程序

throw new RuntimeException(e);

}

}

}

### mybatis-config.xml

resources目录下：

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

<configuration>

<environments default="test">

<environment id="test">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

<property name="driver" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3307/mybatisdb"/>

<property name="username" value="root"/>

<property name="password" value="root"/>

</dataSource>

</environment>

</environments>

<!--xml配置方式：-->

<mappers>

<mapper resource="resources/mybatis/mappers/UserMapper.xml"/>

</mappers>

</configuration>

注意：xml配置或者接口配置，配置一种即可。

### Resources.java

package javase.mybatis.session.io;

import java.io.InputStream;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:53:47

\* @description 描述：模拟mybatis提供工具类，读取核心配置文件

\*/

public class Resources {

public static InputStream getResourceAsStream(String filePath) {

String prefix = "resources/"; //资源目录

if(!filePath.startsWith(prefix)) {

filePath = prefix + filePath;

}

return Resources.class.getClassLoader().getResourceAsStream(filePath);

}

}

### XMLConfigBuilder.java

package javase.mybatis.builder.xml;

import java.io.IOException;

import java.io.InputStream;

import java.lang.reflect.Method;

import java.lang.reflect.ParameterizedType;

import java.lang.reflect.Type;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.dom4j.Attribute;

import org.dom4j.Document;

import org.dom4j.Element;

import org.dom4j.io.SAXReader;

import javase.mybatis.mapping.SqlCommandType;

import javase.mybatis.session.Configuration;

import javase.mybatis.session.SqlMapper;

import javase.mybatis.session.annotation.Select;

import javase.mybatis.session.io.Resources;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:57:18

\* @description 描述：模拟mybatis解析核心配置文件内容，存放内容到配置对象中

\*/

public class XMLConfigBuilder {

//定义封装数据库链接配置对象

protected static final Configuration cfg = new Configuration();

public static Configuration parse(InputStream in) {

try {

//dom4j解析xml核心配置文件

SAXReader reader = new SAXReader();

Document doc = reader.read(in);

Element root = doc.getRootElement(); //获取根元素

//1、处理数据库相关参数配置对象

database( root );

//2、映射接口信息、xml信息

mappers( root );

return cfg;

} catch (Exception e) {

throw new RuntimeException(e);

} finally {

try {

in.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

//解析映射xml文件或者映射接口

private static void mappers(Element root) throws Exception {

List<Element> mapperEls = root.selectNodes("//mappers/mapper");

for(Element e: mapperEls) {

Attribute attrRes = e.attribute("resource");

if( attrRes != null) {

System.out.println("使用XML配置方式");

String mapperPath = attrRes.getValue();

//把映射配置文件的内容获取处理，封装成map

Map<String, SqlMapper> sqlMappers = mapperConfiguration(mapperPath);

cfg.setSqlMappers(sqlMappers);

}

//上面处理完xml方式，接着处理接口方式，因为key相同，就会覆盖上面xml配置

Attribute attrClass = e.attribute("class");

if( attrClass != null){

System.out.println("使用接口配置方式");

String mapperClassPath = attrClass.getValue();

Map<String, SqlMapper> sqlMappers = mapperAnnotation(mapperClassPath);

cfg.setSqlMappers(sqlMappers);

}

}

}

/\*

\* 根据传入的参数，得到Mapper接口中方法上的注解select

\* 根据方法名称和类名，以及方法上注解vlaue属性的值（sql），组成sqlMapper对象

\*/

private static Map<String, SqlMapper> mapperAnnotation(String mapperClassPath) throws Exception {

Map<String, SqlMapper> sqlMappers = new HashMap<>();

//获得接口类

Class<?> daoClass = Class.forName(mapperClassPath);

Method[] ms = daoClass.getMethods();

for(Method m : ms) {

if( m.isAnnotationPresent(Select.class)) {

SqlMapper sqlMapper = new SqlMapper();

Select selectAnno = m.getAnnotation(Select.class);

sqlMapper.setId(m.getName());

sqlMapper.setSqlCommandType(SqlCommandType.SELECT);

String queryString = selectAnno.value(); //sql语句

sqlMapper.setQueryString(queryString);

//获取方法的返回值，下面利用反射解析出参数类型

Type type = m.getGenericReturnType(); //List<User>

if( type instanceof ParameterizedType) {

ParameterizedType ptype = (ParameterizedType) type;

//得到参数化类型中的实际类型参数

Type[] types = ptype.getActualTypeArguments();

//取出第一个参数，User

Class<?> domainClass = (Class<?>)types[0];

String resultType = domainClass.getName();

sqlMapper.setResultType(resultType);

}

String methodName = m.getName();

String className = m.getDeclaringClass().getName();

String key = className + "." + methodName;

sqlMappers.put(key, sqlMapper);

}

}

return sqlMappers;

}

/\*

\* 根据传入的参数，解析XML，并且封装到map中

\* 约定：

\* key是Mapper接口的全局限定名.方法名组成：myabtis中statement=namespace.id

\* value是Mapper对象，里面存放执行sql和返回值的全局限定名

\*/

private static Map<String, SqlMapper> mapperConfiguration(String mapperPath) throws IOException {

InputStream in = null;

try {

Map<String, SqlMapper> sqlMappers = new HashMap<>();

in = Resources.getResourceAsStream(mapperPath);

SAXReader reader = new SAXReader();

Document doc = reader.read(in);

Element root = doc.getRootElement();

String namespace = root.attributeValue("namespace");

//处理<select>标签

List<Element> selectElements = root.selectNodes("//select");

for(Element e : selectElements) {

String id = e.attributeValue("id");

String resultType = e.attributeValue("resultType");

String queryString = e.getText(); //获取标签内容，就是sql语句

String key = namespace + "." + id;

SqlMapper sqlMapper = new SqlMapper();

sqlMapper.setId(id);

sqlMapper.setSqlCommandType(SqlCommandType.SELECT);

sqlMapper.setQueryString(queryString);

sqlMapper.setResultType(resultType);

sqlMappers.put(key, sqlMapper);

}

return sqlMappers;

} catch (Exception e) {

throw new RuntimeException(e);

} finally {

in.close();

}

}

private static void database(Element root) {

//双斜杠代表：从任意位置的节点上选择名称为item节点

List<Element> propertyEls = root.selectNodes("//property");

for(Element e : propertyEls) {

String name = e.attributeValue("name");

switch (name) {

case "driver":

cfg.setDriver(e.attributeValue("value"));

break;

case "url":

cfg.setUrl(e.attributeValue("value"));

break;

case "username":

cfg.setUsername(e.attributeValue("value"));

break;

case "password":

cfg.setPassword(e.attributeValue("value"));

break;

}

}

}

}

### Executor.java

package javase.mybatis.session.executor;

import java.beans.PropertyDescriptor;

import java.lang.reflect.Method;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.ResultSetMetaData;

import java.util.ArrayList;

import java.util.List;

import javase.mybatis.session.SqlMapper;

import javase.mybatis.util.CoverUtil;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:09:13

\* @description 描述：模拟mybatis的执行器，底层就直接执行sql

\*/

public class Executor {

//执行查询sql语句

public <E> List<E> selectList(SqlMapper sqlMapper, Connection conn){

PreparedStatement pstm = null;

ResultSet rs = null;

try {

String queryString = sqlMapper.getQueryString();

String resultType = sqlMapper.getResultType();

Class<?> domainClass = Class.forName(resultType);

//执行sql

pstm = conn.prepareStatement(queryString);

rs = pstm.executeQuery();

//解析结果集rs，把其中数据存放到pojo对象中

List<E> list = new ArrayList<>();

while(rs.next()) {

E obj = (E) domainClass.newInstance();

ResultSetMetaData rsmd = rs.getMetaData(); //表的元数据，列的信息

int columnCount = rsmd.getColumnCount(); //字段数

for( int i=1; i<= columnCount; i++) { //注意字段起始值1

//user\_name userName

String columnName = rsmd.getColumnName(i);

String propertyName = CoverUtil.underlineToCamel(columnName);

Object columnValue = rs.getObject(columnName);

//使用bean工具类api，它可以按名称反射到对应写入方法 name setName()进行回调

PropertyDescriptor pd = new PropertyDescriptor(propertyName, domainClass);

Method writeMethod = pd.getWriteMethod();

writeMethod.invoke(obj, columnValue);

}

list.add(obj);

}

return list;

} catch(Exception e) {

throw new RuntimeException();

} finally {

release(pstm, rs);

}

}

private void release(PreparedStatement pstm, ResultSet rs) {

if( rs != null) {

try {

rs.close();

} catch (Exception e) {

e.printStackTrace();

}

}

if(pstm != null) {

try {

pstm.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

### MapperProxy.java

package javase.mybatis.session.binding;

import java.lang.reflect.InvocationHandler;

import java.lang.reflect.Method;

import java.sql.Connection;

import java.util.Map;

import javase.mybatis.mapping.SqlCommandType;

import javase.mybatis.session.SqlMapper;

import javase.mybatis.session.executor.Executor;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午9:26:15

\* @description 描述：接口代理，底层使用jdk动态代理

\*/

public class MapperProxy implements InvocationHandler{

private Map<String, SqlMapper> sqlMappers;

private Connection connection;

//通过带参构造方法传入参数

public MapperProxy(Map<String, SqlMapper> sqlMappers, Connection connection) {

this.sqlMappers = sqlMappers;

this.connection = connection;

}

@Override //调用代理对象时，会自动调用下面方法

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

String methodName = method.getName();

String className = method.getDeclaringClass().getName();

String key = className + "." + methodName;

SqlMapper sqlMapper = sqlMappers.get(key);

Object rtn = null; //返回对象

if(sqlMapper.getSqlCommandType() == SqlCommandType.SELECT) {

rtn = new Executor().selectList(sqlMapper, connection);

}

return rtn;

}

}

### SqlSession.java

package javase.mybatis.session;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:12:51

\* @description 描述：

\*/

public interface SqlSession {

<T> T getMapper(Class<T> mapperClass);

void close();

}

### DefaultSqlSession.java

package javase.mybatis.session.defaults;

import java.lang.reflect.Proxy;

import java.sql.Connection;

import java.sql.SQLException;

import java.util.Map;

import javase.mybatis.session.SqlMapper;

import javase.mybatis.session.SqlSession;

import javase.mybatis.session.binding.MapperProxy;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:14:21

\* @description 描述：

\*/

public class DefaultSqlSession implements SqlSession{

private Map<String, SqlMapper> sqlMappers;

private Connection connection;

//以带参构造方式传参

public DefaultSqlSession(Map<String, SqlMapper> sqlMappers, Connection connection) {

this.sqlMappers = sqlMappers;

this.connection = connection;

}

@SuppressWarnings("unchecked")

@Override //利用jdk动态代理，为什么mybatis不写接口实现类，依然可以调用

public <T> T getMapper(Class<T> mapperClass) {

return (T) Proxy.newProxyInstance(mapperClass.getClassLoader(),

new Class[] { mapperClass },

new MapperProxy(sqlMappers, connection));

}

@Override

public void close() {

if(connection != null) {

try {

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

### SqlSessionFactory.java

package javase.mybatis.session;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:20:17

\* @description 描述：

\*/

public interface SqlSessionFactory {

SqlSession openSession();

}

### DefaultSqlSessionFactory.java

package javase.mybatis.session.defaults;

import javase.mybatis.session.Configuration;

import javase.mybatis.session.SqlSession;

import javase.mybatis.session.SqlSessionFactory;

import javase.mybatis.util.DataSourceUtil;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:21:12

\* @description 描述：

\*/

public class DefaultSqlSessionFactory implements SqlSessionFactory{

private Configuration configuration;

public DefaultSqlSessionFactory(Configuration configuration) {

this.configuration = configuration;

}

@Override

public SqlSession openSession() {

return new DefaultSqlSession(configuration.getSqlMappers(),

DataSourceUtil.getConnection(configuration)

);

}

}

### SqlSessionFacotryBuilder.java

package javase.mybatis.session;

import java.io.InputStream;

import javase.mybatis.builder.xml.XMLConfigBuilder;

import javase.mybatis.session.defaults.DefaultSqlSessionFactory;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:24:32

\* @description 描述：

\*/

public class SqlSessionFactoryBuilder {

public SqlSessionFactory build(InputStream inputStream) {

return new DefaultSqlSessionFactory( XMLConfigBuilder.parse(inputStream) );

}

}

### User.java

package cn.tedu.pojo;

import java.util.Date;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：上午10:36:42

\* @description 描述：

\*/

public class User {

private Integer id;

private String userName;

private Date birthday;

private String address;

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;

}

public Date getBirthday() {

return birthday;

}

public void setBirthday(Date birthday) {

this.birthday = birthday;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

@Override

public String toString() {

return "User [id=" + id + ", userName=" + userName + ", birthday=" + birthday + ", address=" + address + "]";

}

}

### UserMapper.xml

resources/mybatis/mappers目录下：

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

<!DOCTYPE mapper

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

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

<mapper namespace="cn.tedu.mapper.UserMapper">

<select id="findAll" resultType="cn.tedu.pojo.User">

select \* from user

</select>

</mapper>

### UserMapper.java

package javase.mybatis.mapper;

import java.util.List;

import javase.mybatis.pojo.User;

import javase.mybatis.session.annotations.Select;

public interface UserMapper {

@Select("select \* from user")

List<User> findAll();

}

### TestMybatis.java

package test;

import java.io.IOException;

import java.io.InputStream;

import java.util.List;

import javase.mybatis.mapper.UserMapper;

import javase.mybatis.pojo.User;

import javase.mybatis.session.SqlSession;

import javase.mybatis.session.SqlSessionFactory;

import javase.mybatis.session.SqlSessionFactoryBuilder;

import javase.mybatis.session.io.Resources;

public class TestMybatis {

public static void main(String[] args) throws IOException {

//1、读取配置文件,将配置文件转换为输入流

InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

//2、创建SqlSessionFactory工厂

SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

//返回DefaultSqlSessionFactory对象，利用XMLConfigBuilder类从输入流中读取配置信息

//封装成Configuration对象传入DefaultSqlSessionFactory的构造方法

SqlSessionFactory factory = builder.build(in);

//3、使用工厂(DefaultSqlSessionFactory)生产SqlSession对象,

SqlSession session = factory.openSession();

//4、使用SqlSession创建Dao接口的代理对象

UserMapper mapper = session.getMapper(UserMapper.class);

//5、使用代理对象执行方法

List<User> users = mapper.findAll();

for (User user : users) {

System.out.println(user);

}

//6.释放资源

session.close();

try {

in.close();

} catch (IOException e) {

e.printStackTrace();

}

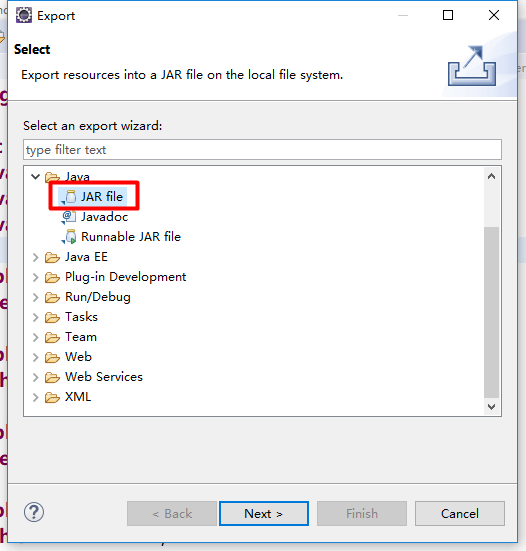
}

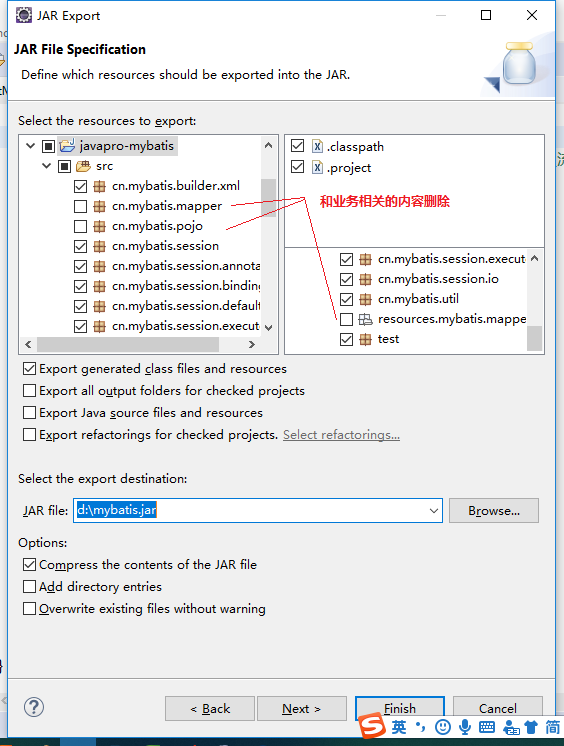
}

## 使用mybatis框架

### 导出jar包

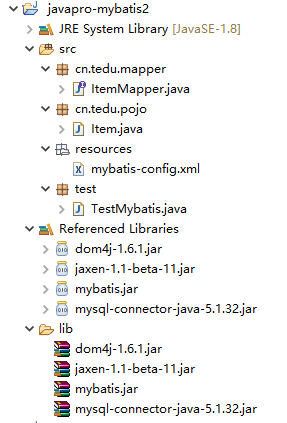
可以把手写的mybatis框架打包，这样其它项目就可以直接导入，就像我们日常导入mybatis框架一样。





## 测试自己手写的mybatis框架

### 创建java工程



### Item.java

package cn.tedu.pojo;

import java.util.Date;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午5:00:20

\* @description 描述：京淘电商商品表

\*/

public class Item {

private Long id; //注意类型映射，不对就会报错，bigint=Long

private String title;

private String sellPoint;

private Long price;

private Integer num;

private String barcode;

private String image;

private Long cid;

private Integer status;

private Date created;

private Date updated;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getSellPoint() {

return sellPoint;

}

public void setSellPoint(String sellPoint) {

this.sellPoint = sellPoint;

}

public Long getPrice() {

return price;

}

public void setPrice(Long price) {

this.price = price;

}

public Integer getNum() {

return num;

}

public void setNum(Integer num) {

this.num = num;

}

public String getBarcode() {

return barcode;

}

public void setBarcode(String barcode) {

this.barcode = barcode;

}

public String getImage() {

return image;

}

public void setImage(String image) {

this.image = image;

}

public Long getCid() {

return cid;

}

public void setCid(Long cid) {

this.cid = cid;

}

public Integer getStatus() {

return status;

}

public void setStatus(Integer status) {

this.status = status;

}

public Date getCreated() {

return created;

}

public void setCreated(Date created) {

this.created = created;

}

public Date getUpdated() {

return updated;

}

public void setUpdated(Date updated) {

this.updated = updated;

}

@Override

public String toString() {

return "Item [id=" + id + ", title=" + title + ", sellPoint=" + sellPoint + ", price=" + price + ", num=" + num

+ ", barcode=" + barcode + ", image=" + image + ", cid=" + cid + ", status=" + status + ", created="

+ created + ", updated=" + updated + "]";

}

}

### ItemMapper.java

package cn.tedu.mapper;

import java.util.List;

import cn.tedu.pojo.Item;

import javase.mybatis.session.annotation.Select;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午5:03:45

\* @description 描述：商品表的映射

\*/

public interface ItemMapper {

@Select("select \* from tb\_item")

public List<Item> findAll();

}

### mybatis-config.xml

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

<configuration>

<environments default="test">

<environment id="test">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

<property name="driver" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3307/mybatisdb"/>

<property name="username" value="root"/>

<property name="password" value="root"/>

</dataSource>

</environment>

</environments>

<!--xml接口方式：-->

<mappers>

<mapper class="cn.tedu.mapper.ItemMapper"/>

</mappers>

</configuration>

### TestMybatis.java

package test;

import java.io.InputStream;

import java.util.List;

import cn.tedu.mapper.ItemMapper;

import cn.tedu.pojo.Item;

import javase.mybatis.session.SqlSession;

import javase.mybatis.session.SqlSessionFactory;

import javase.mybatis.session.SqlSessionFactoryBuilder;

import javase.mybatis.session.io.Resources;

/\*\*

\* @author 作者：tony chan

\* @email 邮箱：chenzs@tedu.cn

\* @version v.1.0 创建时间：下午5:04:51

\* @description 描述：

\*/

public class TestMybatis {

public static void main(String[] args) {

InputStream is = Resources.getResourceAsStream("mybatis-config.xml");

SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

SqlSessionFactory factory = builder.build(is);

SqlSession session = factory.openSession();

ItemMapper mapper = session.getMapper(ItemMapper.class);

List<Item> itemList = mapper.findAll();

for(Item i : itemList) {

System.out.println(i);

}

}

}