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

计划课程安排：

FUNDAMENTAL(7天)         该阶段已结束

OOP(11天)           该阶段已结束

JAVASE01(10天) 该阶段已结束

JAVASE02(11天) ←当前为该阶段第10天

WEBBASIC(7天)

JQUERY(2天)

DATABASE(6天)

JDBC(3天)

SERVLETJSP(12天)

SPRINGMYBATIS01(8天)

AJAX(2天)

SPRINGMYBATIS02(12天)

LINUX(2天)

NGINX(2天)

REDIS(2天)

T-STORE(5天)

预计合计：101天

预计剩余：64天

注：上述信息随时可能发生更改，仅供参考。

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

**2019年4月23日**

**Java就业班-第47天**

**8:30实训室2-晨讲**

今日晨讲主题——RandomAccessFile的使用方法

这位大兄弟自身都没有理解，所以反而是受教育了。

**9:04实训室2-正课**

开始上课。

今天的课程内容已确定——反射。

首先写几个概念代码认识一下反射。

package reflect;

import java.lang.reflect.Method;

import java.util.Scanner;

/\*\*

\* 反射是java的一个动态机制。他将实例化对象，调用方法，操作属性等工作

\* 从编码期间确定转为在程序运行期间确定。这大大的提高了代码的灵活性。

\* 反射有更多的资源开销和较慢的运行效率，因此

\* 只在关键的地方使用反射，而不应大量应用。

\*

\* Class类

\* 类对象Class,他的每一个实例是用来表示JVM已经加载的一个类。每个被JVM加载的类都有且只有一个Class的实例与之对应。

\* 通过类对象我们可以得知其表示的类的一切信息：

\* 类名、有哪些方法、那些属性、那些构造器等。

\* 并且可以快速实例化其表示的类的实例。

\*

\* 反射的第一步就是获取要操作的类的类对象，而获取一个类的类对象有以下几种方式：

\* 1.每个类都有一个静态属性：class，可以直接得到这个类的类对象。

\* 但是由于这是调用静态属性，属于编码方式获取，所以不灵活。

\* 例如：Class cls = String.class;

\*

\* 2.通过Class的静态方法forName得到，例如：

\* Class cls = Class.forName("java.lang.String")

\*

\* 3.类加载器ClassLoader

\*

\* @author Administrator

\*

\*/

public class ReflectDemo01 {

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

//获取String的类对象

// Class cls = String.class;

//获取类名

// Class cls = Person.class;

Scanner sc = new Scanner(System.in);

System.out.println("请输入要加载的类的名字：");

String className = sc.nextLine();//reflect.Person

Class cls = Class.forName(className);

String name = cls.getName();

System.out.println("类名：" + name);

/\*

\* Method类

\* Method的每一个实例用于表示一个类中定义的一个方法。

\* 通过它可以得到这个方法的相关信息，并且也可以调用这个方法

\* 的相关信息（访问修饰符，返回值类型，方法名等），

\* 并且也可以调用这个方法。

\*/

// Method methods[] = cls.getMethods();

//获取本类自己定义的方法

Method methods[] = cls.getDeclaredMethods();

for(Method method :methods) {

System.out.println(method.getName());

}

}

}

创建一个测试用的Person和Student类。

人类：

package reflect;

public class Person {

private String name;

public Person(String name){

this.name = name;

}

public Person(){

}

private void dosome() {

System.out.println("这是一个私有方法！！");

}

public String toString() {

return "Person[" + name + "]";

}

public void sayHello(){

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

}

public void sayHi(){

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

}

public void say(String name){

System.out.println("大家好！我是:"+name);

}

public void say(String name,int age){

System.out.println(

"大家好！我是:"+name+",我今年"+age);

}

}

学生类：

package reflect;

public class Student {

public void say(){

System.out.println("hello everyone!I'm a student");

}

public void sayHi(){

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

}

}

**10:30实训室2-正课**

关于利用放射实例化对象。

package reflect;

import java.util.Scanner;

/\*\*

\* 利用反射实例化对象

\*

\* @author Administrator

\*

\*/

public class ReflectDemo02 {

public static void main(String[] args) throws ClassNotFoundException, InstantiationException, IllegalAccessException {

Person p = new Person();

System.out.println(p);

/\*

\* 实例化Person类实例

\*/

Scanner sc = new Scanner(System.in);

Class cls = Class.forName("reflect.Person");

String className = sc.next();

/\*

\* Class提供的实例化方法要求其表示的类必须具有无参构造器，否则实例化时会抛出异常

\*/

Object obj = cls.newInstance();

System.out.println(obj);

}

}

**10:50实训室2-正课**

关于利用反射调用方法。

package reflect;

import java.lang.reflect.Method;

import java.util.Scanner;

/\*\*

\* 利用反射调用方法

\* @author adminitartor

\*

\*/

public class ReflectDemo03 {

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

Person p = new Person();

p.sayHello();

/\*

\* 利用反射机制

\*/

Class cls = Class.forName("reflect.Person");

Object obj = cls.newInstance();

Method method = cls.getDeclaredMethod("sayHello");

method.invoke(obj);

}

}

**11:30实训室2-正课**

关于调用有参方法。

package reflect;

import java.lang.reflect.Method;

/\*\*

\* 调用有参方法

\* @author adminitartor

\*

\*/

public class ReflectDemo04 {

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

Class cls = Class.forName("reflect.Person");

Object obj = cls.newInstance();

//say(String,int)

Method method = cls.getDeclaredMethod(

"say",String.class,int.class);

method.invoke(obj,"Kamui",25);

}

}

**11:50实训室2-正课**

关于变长参数

package reflect;

/\*\*

\* 变长参数

\* @author adminitartor

\*

\*/

public class Test {

public static void main(String[] args) {

dosome(1,"one","two","three","four");

}

public static void dosome(int age,String... arg){

System.out.println(arg.length);

for(String str : arg){

System.out.println(str);

}

}

}

**14:21实训室2-正课**

package reflect;

import java.lang.reflect.Constructor;

import java.lang.reflect.Method;

/\*\*

\* 调用有参方法

\* @author adminitartor

\*

\*/

public class ReflectDemo05 {

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

Class cls = Class.forName("reflect.Person");

Constructor arr[] = cls.getDeclaredConstructors();

for (Constructor c : arr) {

System.out.println(c);

}

//Person = p = new Person("张三");

Constructor c = cls.getDeclaredConstructor(String.class);

Object o = c.newInstance("张三");

System.out.println(o);

}

}

**14:35实训室2-正课**

package reflect;

import java.lang.reflect.Method;

/\*\*

\* 利用反射机制访问一类的私有方法

\* @author Administrator

\*

\*/

public class ReflectDemo06 {

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

//正常情况无法访问私有方法

//Person p = new Person();

//p.dosome();

Class cls = Class.forName("reflect.Person");

Object o = cls.newInstance();

Method method = cls.getDeclaredMethod("dosome");

//强制访问该私有方法（此处仅作为了解，实际中因违背封装性，所以慎用！）

method.setAccessible(true);

method.invoke(o);

}

}

※只要是程序，灵活性和性能永远是成反比的。

**14:40实训室2-正课**

那么在我们了解了反射的用法之后，现在重新回到WebServer项目的代码重构。

新建一个项目WebServer\_v16，在core包中新建一个ServerContext类用于表示服务器相关配置信息

README.txt------------------------

本版本主要工作：利用反射机制完成请求与Servlet的对应

之前的版本中存在一个问题，每次新添加一个业务，我们除了要定义对应的页面和Servlet之外，

还要修改ClientHandler类，在处理请求的环节添加新的分支。

因此，我们应当利用反射机制根据请求找到对应的Servlet的名字并加以实例化，

使得ClientHandler不再关心具体的哪个请求对应哪个Servlet。

实现：

1.在com.webserver.core包中新建一个类：ServerContext

2.在ServerContext中定义静态属性：

Map servletMapping

并定义初始化方法，然后在静态块中调用。

初始化操作是通过解析conf/servlets.xml完成的

这里要利用反射来加载并实例化对应的Servlet.

3.定义静态方法：getServlet(String path)

使得ClientHandler可以根据请求获取到对应的业务处理类（某个Servlet为例）

4.修改ClientHandler的处理请求操作环节，将原有的利用若干分之判断请求与

对应Servlet的操作改为使用请求去ServerContext的getServlet方法找到

对应的Servlet然后调用其service方法处理业务，若没有找到则执行查找静态资源的后续操作。

经过本版本的改动后，将来在添加新的业务时，我们在准备完对应页面和Servlet后，只需要在conf/servletx.xml文件中

添加一个新的<servlet>标签，配置新得请求与对应Servlet的名字即可。ClientHandler就会根据该请求找到Servlet

并调用该业务了，而无须再添加新的分支操作。

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

功能实现：

package com.webserver.core;

import java.util.HashMap;

import java.util.Map;

import com.webserver.servlet.HttpServlet;

import com.webserver.servlet.LoginServlet;

import com.webserver.servlet.RegServlet;

import com.webserver.servlet.UpdateServlet;

/\*\*

\* 服务器相关配置信息

\* @author Administrator

\*

\*/

public class ServerContext {

/\*

\* 请求与对应Servler的关系

\* key:请求路径

\* value:处理该请求的Sevrlet实例

\*/

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

//静态代码块

static {

initSevrlet();

}

/\*\*

\* 初始化Servlet

\*/

private static void initSevrlet() {

servletMapping.put("/myweb/reg", new RegServlet());

servletMapping.put("/myweb/login", new LoginServlet());

servletMapping.put("/myweb/update", new UpdateServlet());

}

/\*\*

\* 根据请求路径获取对应的Servlet实例

\* @param path

\* @return

\*/

public static HttpServlet getServlet(String path) {

return servletMapping.get(path);

}

}

随后修改ClientHandler内相应代码。

**15:35实训室2-正课**

在项目conf的文件夹中新建一个servlets.xml

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

<servlets.xml>

<servlet path="/myweb/reg" className="com.webserver.servlet.RegServlet"/>

<servlet path="/myweb/login" className="com.webserver.servlet.LoginServlet"/>

<servlet path="/myweb/update" className="com.webserver.servlet.UpdateServlet"/>

</servlets.xml>

接下来将利用反射机制来完成请求处理的代码重构。

package com.webserver.core;

import java.io.File;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.dom4j.Document;

import org.dom4j.Element;

import org.dom4j.io.SAXReader;

import com.webserver.servlet.HttpServlet;

import com.webserver.servlet.LoginServlet;

import com.webserver.servlet.RegServlet;

import com.webserver.servlet.UpdateServlet;

/\*\*

\* 服务器相关配置信息

\* @author Administrator

\*

\*/

public class ServerContext {

/\*

\* 请求与对应Servler的关系

\* key:请求路径

\* value:处理该请求的Sevrlet实例

\*/

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

//静态代码块

static {

try {

initSevrlet();

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 初始化Servlet

\* @throws Exception

\* @throws InstantiationException

\*/

private static void initSevrlet(){

// servletMapping.put("/myweb/reg", new RegServlet());

// servletMapping.put("/myweb/login", new LoginServlet());

// servletMapping.put("/myweb/update", new UpdateServlet());

/\*

\* 解析conf/servletx.xml文件

\* 将根标签下所有的<servlet>标签解析出来

\* 并用其属性path的值作为key

\* className属性的值使用放射方式加载对应的Servlet类并进行实例化，

\* 然后将对应的实例作为value保存到servletMapping这个Map中。

\*/

try {

SAXReader reader = new SAXReader(); //创建SAXReader对象

//读取Document的结果付给doc对象

Document doc = reader.read(new File("./conf/servletx.xml"));

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

List<Element> list = root.elements("servlet");//获取所有子标签

for (Element element : list) {//将取得的值放入map集合

String path = element.attributeValue("path");

Class cls = Class.forName(element.attributeValue("className"));

HttpServlet servlet = (HttpServlet) cls.newInstance();

servletMapping.put(path, servlet);

}

}catch(Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 根据请求路径获取对应的Servlet实例

\* @param path

\* @return

\*/

public static HttpServlet getServlet(String path) {

return servletMapping.get(path);

}

}

**17:06实训室2-正课**

本来接下来要讲解线程池的内容，但是因为时间不够所以今天的课程就到这里了。

今天就先介绍一下什么是线程池：

讲解了一些关于线程重要性的比喻，如何延长线程的生命周期，如何循环利用线程等等……

**17:26实训室2-下课时间**

最后提示了一下，多掌握这些天写的代码，然后下课。

我中午是直接睡过去了，到现在为止这一天没吃任何东西……先去吃饭吧。

**18:30实训室2-晚自习**

班主任通知这周六休息（4月27日），周日上课（4月28日）。就是把原本的周六自习改为休息了，说是连续上9天课着实受不了。机构这确实是难得干了件对的事情。

**19:57实训室2-晚自习**

到目前为止还没有做任何的复习，主要是因为上一家公司的老总发邮件过来询问我之前在职时处理过的一些事情交接问题。由于我在那位老总的手下也干了比较长的一段时间吧，负责处理的事物也比较多，难免会有一些交接遗漏或无法预测到的交接之后的后续联络。尽管联系我的事情并不是什么很重要的事情，但从邮件的意思里我也能明显感觉到老总现在一个人处理不来所有的突发情况，需要有人协助他处理一些其他的工作。这也难怪，毕竟我离职的时候，我的职务没有人安排专人进行交接的，于是老总便把我所负责的工作下分给了其他部门处理。可想而知处理的效果并不怎么样了。

回到学习上来，项目经理貌似是要开始讲解一些练习题了，正好剩下这么点时间我也不知道干什么好，就听听看吧。

1. 下列属于传输层协议的是：
2. TCP
3. UDP
4. HTTP
5. SMTP

答案：AB

1. 下面关于ServerSocket描述正确的是：
2. ServerSocket类实现了服务器端的套接字
3. ServerSocket类的accept()方法侦听并接受到客户端套接字
4. 一个服务器端只能有一个客户端进行连接
5. 一个服务器端可以有多个客户端进行连接

答案：BD

1. 关于java.io.Serizalizable接口说法正确的是：
2. java.io.Serializable中有一个serialID属性，但是没有方法
3. 类通过实现java.io.Serializable接口以启用其对象的序列化功能。
4. java.io.Serializable中有一个run方法，但是没属性
5. java.io.Serializable接口没有方法或属性，仅用于标识可序列化的语义。

答案：BD

**20:40实训室2-晚自习**

项目经理最后通过复习讲解了一下HashMap中的HashSet集合，反复强调HashSet的原理是面试中经常会出现的问题，一定要重视。

项目经理强调说，很多原理性的东西确实都很难理解，没有多少可以实际看得到摸得着的东西可以去感受，但是在当前行业环境下也好，今后的水平提高也好，懂得各种方法、算法的原理才是作为一个程序员、写程序的、专业的码农必备的基础功。

…………

**20:55实训室2-晚自习**

真的好疲惫啊，主要是因为教室是真的热，室温35度，不开空调，几台电扇吹得都是热风，额头上汗如雨下，厉害了。

学习欲望急速降低，给机构反映了也没卵用，说什么空调不由他们控制，由物业的统一管理，51假期之后才给开。

NMD,WSM。

回宿舍洗洗睡了。

今日感想总结：班上越来越多的人开始和班主任预定4月班的座位了，为月考后的留级做准备。尽管现在的课程内容我还能跟上，但是身心上的压力也开始感觉越来越大了。

今日评分：6/10分

今天就到这里，明天继续投稿。

感想支持！

================================================================

**Acer程序员群：790482850**

**up主的java学习日记word文件及全部代码（随日记同步更新）：**

Adress：github.com/wt62635