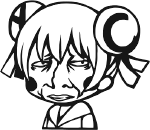
Java就业班-第93天  链接：[https://www.acfun.cn/a/ac10343240](https://www.acfun.cn/a/ac10343240" \t "_blank)

IT培训机构实地观察日记 Java就业班-第94天

前言声明：

       此系列文章绝非网络推广文、软文、黑或洗白文等或出于任何商业目的而投稿的文章，仅出于UP主为记录并分享自身真实经历的目的而发表，不为文中出现的任何事物作出保证，全文均属于完全个人主观内容，无任何刻意诋毁或吹捧意图。文本、照片中会尽量避免出现敏感信息，如有不合适内容举报还请及时联系我立即删除。

       如果有同一个地点，同一个时间，甚至与我在同一个班的兄弟、朋友、Acer们，还请继续当作没有看见，请尽量配合避免暴露不必要的真实信息，以免造成一些不必要的麻烦，十分感谢。

或者换句话说，有什么问题请直接冲我来，我们厕所解决，谢谢！

更新规律：

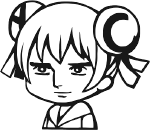
       基本上有课程的当天晚上(18：00---24：00)之间会进行更新投稿。休息日一般为星期天与节假日，无特殊理由将不会更新。如有任何建议、问题还请大家随意在评论区里留言，或者直接私信我，我看到后会尽快给予回复，感谢大家支持！

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

友情提示：

本日记中包含大量低质量代码，

对代码内容不感兴趣的朋友可以放心跳过至最后部分，丝毫不影响阅读。

(不过那也就没什么内容了。但是可以省时间。)

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

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(12天)               该阶段已结束

WEBBASIC(11天)        该阶段已结束

JQUERY(2天)                    该阶段已结束

DATABASE(5天)            该阶段已结束

JDBC(2天)                         该阶段已结束

SERVLETJSP(14天)        该阶段已结束

SPRINGMYBATIS01(8天) ←当前为该阶段第6天

AJAX(2天)

SPRINGMYBATIS02(12天) （该阶段已被取消）

LINUX(2天)                        （该阶段已被取消）

NGINX(2天)                      （该阶段已被取消）

REDIS(2天)                       （该阶段已被取消）

T-STORE(5天)                             （该阶段已被取消）

预计合计：101天

预计剩余：9天

注1：上述信息随时可能发生更改，仅供参考。

注2：被取消表示若按照正常流程就业则直接上班，不用再继续到来机构上课。

反之，则需要继续到机构上完全部课程。

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

**2019年6月22日**

**Java就业班-第94天**

**7:20实训室2-早自习**

今天是招聘会，到了9点左右各个公司会到机构就位开始进行笔试、面试。

 我非常清楚自己的水平有多烂，所以今天早早地就赶到机构来进行临时抱佛脚式的复习。主要也就是把昨天晚上看过的面试题目又粗略的过了一遍加深印象，至于实际上会有多大的效果那就拿不准了。

我的目标应聘公司是9点开始笔试，10点以后会开始发布安排面试学员的名单。

现在时间还早，先把昨天的课程笔记下载下来补上吧。

昨日课程笔记：

### 1. SpringMVC中的拦截器(Interceptor)

#### 1.1. 基本概念

在SpringMVC中的拦截器可以是运行在控制器(Controller)之前的组件，可以设置拦截器应用于哪些请求路径，当发生这些请求时，拦截器会自动执行，在执行过程中，可以对请求相关数据进行判断，选择阻止继续向后执行，或选择放行。

\*\*注意：拦截器是一个若干种请求都会经历的执行过程，但是，并不一定需要阻止继续运行，只要是若干种请求都需要做相同的事情，也许每种请求的处理过程都是选择放行，也可以使用拦截器。\*\*

#### 1.2. 开发拦截器

首先，所有的拦截器类都必须实现`HandlerInterceptor`接口，可以自定义`LoginInterceptor`：

      public class LoginInterceptor implements HandlerInterceptor {

             public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)

                           throws Exception {

                    System.out.println("LoginInterceptor.preHandle()");

                    return false;

             }

             public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,

                           ModelAndView modelAndView) throws Exception {

                    System.out.println("LoginInterceptor.postHandle()");

             }

             public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)

                           throws Exception {

                    System.out.println("LoginInterceptor.afterCompletion()");

             }

      }

在拦截器的3个方法中，只有`preHandle()`方法是运行在控制器(Controller)之前的，另2个方法是运行在控制器之后的，所以，只有`preHandle()`具有真正意义的“拦截”功能，该方法的返回值是`boolean`类型的，当返回`true`时表示放行，返回`false`时将阻止继续向后执行，即控制器并不会被执行：

      public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)

                    throws Exception {

             System.out.println("LoginInterceptor.preHandle()");

             // 获取Session

             HttpSession session = request.getSession();

             // 检查Session中是否有登录信息

             if (session.getAttribute("uid") == null) {

                    // 没有登录信息，重定向到登录页

                    response.sendRedirect("../user/login.do");

                    // 执行拦截

                    return false;

             }

             // 放行

             return true;

      }

\*\*注意：即使已经决定了重定向，还是需要return false;否则处理流程会继续向执行，控制器中的方法还是会被调用，达不到阻止运行的效果！\*\*

所有的拦截器都需要在Spring的配置文件中进行配置，在SpringMVC框架中，允许使用若干个拦截器，形成拦截器链，即某个请求可能需要经过多个拦截器，仅当每个拦截器都放行时，才会执行控制器中的方法！在配置文件中，配置的先后顺序决定了多个拦截器的执行顺序：

      <!-- 配置拦截器链 -->

      <mvc:interceptors>

             <!-- 配置第1个拦截器 -->

             <mvc:interceptor>

                    <!-- 拦截路径 -->

                    <mvc:mapping path="/main/index.do"/>

                    <!-- 拦截器类 -->

                    <bean class="cn.tedu.spring.LoginInterceptor"></bean>

             </mvc:interceptor>

             <!-- 配置第2个拦截器 -->

      </mvc:interceptors>

在配置每个拦截器时，允许使用若干个`<mvc:mapping>`节点以配置若干个拦截路径，例如：

      <mvc:interceptor>

             <!-- 拦截路径 -->

             <mvc:mapping path="/main/index.do"/>

             <mvc:mapping path="/user/password.do"/>

             <mvc:mapping path="/user/info.do"/>

             <mvc:mapping path="/user/handle\_password.do"/>

             <mvc:mapping path="/user/handle\_info.do"/>

             <!-- 拦截器类 -->

             <bean class="cn.tedu.spring.LoginInterceptor"></bean>

      </mvc:interceptor>

在配置路径，还可以使用`\*`作为通配符，例如：

      <mvc:interceptor>

             <!-- 拦截路径 -->

             <mvc:mapping path="/main/\*"/>

             <!-- 拦截器类 -->

             <bean class="cn.tedu.spring.LoginInterceptor"></bean>

      </mvc:interceptor>

在使用`\*`作为通配符时，需要注意，1个星号只能匹配1层路径，例如`/main/\*`可以匹配上`/main/index.do`，也可以匹配`/main/hello.do`，但是，不可以匹配上`/main/a/index.do`！

如果一定要匹配若干层路径，必须使用2个星号，例如配置为`/main/\*\*`，可以匹配上`/main/index.do`，也可以匹配`/main/a/hello.do`，甚至可以匹配`/main/a/b/c/d/hello.do`，即无视路径中后续的层级。

所以，可以配置为：

      <mvc:interceptor>

             <!-- 拦截路径 -->

             <mvc:mapping path="/main/\*\*"/>

             <mvc:mapping path="/user/\*\*"/>

             <!-- 拦截器类 -->

             <bean class="cn.tedu.spring.LoginInterceptor"></bean>

      </mvc:interceptor>

如果通配符匹配的路径过多，需要从中去除某些请求路径，还可以添加例外：

      <mvc:interceptor>

             <!-- 拦截路径：黑名单 -->

             <mvc:mapping path="/main/\*\*"/>

             <mvc:mapping path="/user/\*\*"/>

             <!-- 例外路径：白名单 -->

             <mvc:exclude-mapping path="/user/reg.do" />

             <mvc:exclude-mapping path="/user/handle\_reg.do" />

             <mvc:exclude-mapping path="/user/login.do" />

             <mvc:exclude-mapping path="/user/handle\_login.do" />

             <!-- 拦截器类 -->

             <bean class="cn.tedu.spring.LoginInterceptor"></bean>

      </mvc:interceptor>

也就是说，凡`/user/`下的路径都会经过该拦截器，但是，`/user/login.do`是不被处理的！添加到“例外”中的路径，在请求时，并不是拦截器直接放行，而是拦截器根本就不执行！

### 2. 请求参数的乱码解决方案

      <filter>

             <filter-name>CharacterEncodingFilter</filter-name>

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

             <init-param>

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

                    <param-value>utf-8</param-value>

             </init-param>

      </filter>

      <filter-mapping>

             <filter-name>CharacterEncodingFilter</filter-name>

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

      </filter-mapping>

### 3. 在SpringMVC中统一处理异常

在Java中，异常的体系结构：

      Throwable

             Error

                    OutOfMemoryError

             Exception

                    SQLException

                    IOException

                           FileNotFoundException

                    RuntimeException

                           NullPointerException

                           ClassCastException

                           NumberFormatException

                           IndexOutOfBoundsException

                                  ArrayIndexOutOfBoundsException

                                  StringIndexOutOfBoundsException

在`Exception`中，除了`RuntimeException`及其子孙类异常，其它的异常都是必须通过语法进行处理的！处理方式可以是`try...catch`进行捕获并处理，也可以是在方法的签名中添加`throws`声明抛出。

其实，处理异常的本质应该是：给予用户某些提示，告之操作失败的原因，避免用户再次提交错误的数据，另外，也可能在处理过程中，对已经发生的错误尽可能的补救，例如关闭某些已经打开的流对象。

某一种异常在项目的多个请求处理过程中，都可能出现，在SpringMVC中，提供了统一处理异常的做法：将`@ExceptionHandler`添加在自定义的用于处理异常的方法之前，该自定义方法应该：

1. 使用`public`权限；

2. 返回值的设计原则与处理请求的方法的相同；

3. 方法名称可以自由定义；

4. 方法中必须添加异常类型的参数，另外还可以添加例如`HttpServletRequest`等参数，但不可以添加其它参数。

例如：

      @ExceptionHandler({IndexOutOfBoundsException.class, NullPointerException.class})

      public String handleException(Throwable ex, HttpServletRequest request) {

             String errorMessage = null;

             if (ex instanceof NullPointerException) {

                    errorMessage = "错误：请提交用户名！";

             } else if (ex instanceof IndexOutOfBoundsException) {

                    errorMessage = "错误：使用的索引超出了界限！";

             }

             request.setAttribute("msg", errorMessage);

             return "error";

      }

在`@ExceptionHandler`注解中，可以配置需要处理的异常的种类，当配置后，仅当指定的异常出现时，才会调用匹配的方法进行处理，而其它异常是不予处理的！如果没有配置需要处理哪些异常，则任何异常出现都会进行处理！

在处理时，`@ExceptionHandler`只能作用于当前控制器类！

### 4. SpringMVC小结

1. 记住SpringMVC的执行流程图；

2. 掌握`@RequestMapping`的使用；

3. 掌握`@RequestParam`的使用；

4. 掌握获取请求参数的方式；

5. 掌握转发数据与重定向的做法；

6. 掌握拦截器的使用；

7. 掌握处理异常的做法。

### 5. 作业

1. 创建名为`tedu\_ums`的数据库；

2. 在该数据库中创建名为`t\_user`的数据表，数据表中至少包括：id、用户名、密码、年龄、手机号码、邮箱这些字段；

3. 向该表中插入不少于10条数据；

4. 删除指定id的数据，例如删除id=3的用户的数据；

5. 将所有用户的密码全部修改为`8888`；

6. 将指定用户的密码修改为`123456`，例如将id=6的用户的密码改为`123456`；

7. 统计当前有多少条用户数据；

8. 查询指定id的用户数据，例如查询id=5的用户数据；

9. 查询用户名和密码是指定值的用户数据，例如查询用户名为`root`且密码是`1234`的用户数据；

10. 删除多个指定id的数据，例如一次性删除id=4,id=2,id=8的这3条数据；

11. 查询年龄第2大的用户数据。

**补充知识内容：**

### 9. 关于Tomcat启动时报告错误

如果启动时报告错误，且弹出的提示框中的信息里有端口号，则大多是因为端口号被占用，需要先停止已经在运行的Tomcat，然后再次尝试启动。如果此前运行的Tomcat是非法关闭的，可能在Eclipse或相关软件中已经无法停止(停止按钮是灰色的)，但Tomcat仍处于运行状态，则需要切换到Tomcat目录下的bin目录中，找到\*\*shutdown\*\*指令并执行。

如果弹出的提示框中的信息并不包含端口号，可以先检查控制台的错误日志，主要观察每段错误日志的第1个异常，如果出现了`ZipException`，则是最近添加的依赖的jar包可能是损坏的，则应该更新所使用的依赖，或删除本地仓库重新下载相关jar包；如果没有`ZipException`，都是`LifecycleException`，则可能是Tomcat缓存数据有误，可以尝试：

1. 尝试再次启动；

2. 如果以上方式无效，则在\*\*Servers\*\*面板中找到Tomcat，点击右键，选择\*\*Clean\*\*，并尝试再次启动；

3. 如果以上方式无效，则点击Eclipse的\*\*Project\*\*菜单，选择\*\*Clean\*\*，并尝试再次启动；

4. 如果以上方式无效，则在\*\*Servers\*\*面板中删除现有的Tomcat，并在Eclipse中删除名为\*\*Servers\*\*的项目，然后重新添加Tomcat，并尝试再次启动；

5. 如果以上方式无效，则在\*\*Servers\*\*面板中删除现有的Tomcat，并在Eclipse中删除名为\*\*Servers\*\*的项目，另行下载新Tomcat或其它版本的Tomcat，然后重新添加Tomcat，并尝试再次启动；

### 10. POST与GET的区别

使用GET方式提交时，请求参数是体现在URL中的，所以，存在不安全的隐患，并且请求的数据受到长度限制，通常是2K(并不绝对，取决于浏览器等客户端，也取决于服务器)，但是，适合分享，并且，通信过程更快。

使用POST方式提交时，请求参数是封装在请求体中的，所以，相对安全，不受到请求参数的数据长度限制，但是，刷新时会提交表单，也许与用户操作本意不相符，并且，不适合分享，通信过程相对慢一点。

在实际使用时，虽然可能不太严谨，但是，如果以获取数据为主要目标的请求，通常设计为GET请求，如果需要向服务器提交数据，或者不提交数据，是以服务器执行特定的数据处理时，应该优先使用POST请求。

### 11. 转发与重定向的区别

转发是客户端发出1次请求，当请求被服务器端的控制器接收到后，也许控制器并不能直接完成请求任务，则会转到到服务器端内部的JSP等相关组件，然后再响应给客户端，整个转发过程是在服务器端内部完成的，对于客户端来说是不可见的，所以，在客户端的浏览器中，只会显示最初请求的URL。

重定向是服务器端的控制器接收到请求后，并不足以完成后续的任务，则直接响应给客户端，使客户端向第2个位置再次发出请求，然后才完成请求任务，由于客户端明确的知道第2次请求的目标，所以，在客户端的浏览器中，地址栏的URL会发生变化。

其实，转发的本质在于“控制器能够完成数据的访问，却不便于呈现数据，就需要转发到JSP”，所以，在处理请求时，重点在于显示某些数据，则应该使用转发！而重定向却不便于执行此类任务，因为重定向是2次请求，HTTP协议又是无状态协议，即使第1次请求得到了所需要显示的数据，如果使用转发，是无法将这些数据给到处理第2次请求的组件中的！

另外，如果需要客户端的浏览器中URL发生变化，不会重复提交此前的请求，就必须使用重定向，例如“用户登录成功后显示主页”，就必须使用重定向，即用户登录成功后重定向到主页，如果使用转发，可能客户端的URL还是`handle\_login.do`，在这种情况下，如果用户刷新页面，就会再次提交登录请求，而在显示主页时，用户如果要刷新页面，其操作目标应该只是单纯的刷新而已，并不是要再次登录！

### 12. 哪些数据可以存储在Session中

通常，会使用Session进行存储的数据有：

1. 用户身份的唯一标识，例如用户的id等；

2. 高频率使用数据，例如用户名、用户的头像等；

3. 不便于使用其它技术传输或存储的数据。

### 13. 拦截器与过滤器的区别

拦截器(Interceptor)是SpringMVC中的组件，过滤器(Filter)是JavaEE中的组件。

拦截器是运行在`DispatcherServlet`之后且在所有控制器(Controller)之前的组件，所以，仅当被`DispatcherServlet`接收的请求才可能被拦截器处理，例如`DispatcherServlet`配置的是`\*.do`，那拦截器就只能处理某些`\*.do`的请求，而例如`\*.html`或`\*.jpg`都不在拦截器可处理的范围之内！过滤器是运行在所有的Servlet之前的组件，甚至可以处理所有的请求，是根据它在\*\*web.xml\*\*中配置的`<url-pattern>`节点的值来决定的。

拦截器的配置更加灵活，可以有若干个拦截路径(黑名单)，也可以有若干个例外路径(白名单)，而过滤器只能配置1个过滤路径，如果使用了通配符，例如配置为`\*.do`或`/\*`等，只能在过滤器类中编写代码添加例外。

### 14. 乱码问题

计算机能够直接识别并处理的是二进制数据，每个二进数中的0或1占据存储空间中的1个二进制位(bit)，每个二进制位不足以表达任何有效的信息含义，计算机中最基本的存储单位是字节(Byte)，每个字节由8个二进制位组成。

在实际使用时，除去最高位的符号位，还有7个二进制位可以表示内容，则每个字节可以表示128种不同的可能，早期的ASCII码就约定了字符与二进制数之间的对应关系，例如字母`a`对应的就是`110 0001`，但是，由于ASCII码都只占1个字节，只能128种不同的可能，所以，无法表示中文。

由于中文的字符种类偏多，所以，至少需要2个字节才可以表示，当使用2个字节时，有效位数最多可高达15个！

能够支持中文的编码格式有：UTF-8、GBK、GB2312……等等，不同的编码格式的编码规则是不相同的！即同一个汉字，使用不同的编码时，对应的二进制数的序列是不相同的！

所以，乱码问题的根本原因在于使用的编码不统一！即存和取的时候使用了不同的编码！要解决这个问题的方案就是：始终使用同一种编码即可！

需要设置编码的位置有：项目的源代码，前端页面，控制器，数据库连接，数据库。

### 15. VARCHAR和CHAR的区别

`CHAR`是定长的，即固定长度，假设为`CHAR(20)`，存储`ABC`这3个字符，实际存储时，会补充17个空格，实现固定长度；而`VARCHAR`是变化的，即存储的内容长度可以发生变化，假设为`VARCHAR(20)`，存储`ABC`这3个字符，实际只存储这3个字符，但是，实际占据存储空间是21，在使用`VARCHAR`类型，MySQL会另使用空间存储实际存储的字节数，也就是存储`ABC`时，MySQL会另使用1个字节记录`3`这个值，如果使用`VARCHAR`时的设置值超过255时，MySQL会另使用2个字节记录实际使用的字节数。

另：需要注意的是，在使用`CHAR`或`VARCHAR`这2种类型时，设置的数值大小表示占用的字节数！至于能够存储多少字符，取决于使用的字符编码和存储的内容，以`VHARCHAR(20)`为例，最多可以存储20个ASCII码范围内的字符，如果存储的是中文，则还需要考虑使用的字符编码，如果使用UTF-8，则最多可以存储6个汉字！在MySQL中，默认使用的UTF-8是utf8mb3，即任何字符最多占3字节，另外还有utf8mb4，用于存储非常规字符。

      a     0110 0001

      ?     110 xxxxx     10 xxxxxx

      ?     1110 xxxx    10 xxxxxx     10 xxxxxx

      ?     11110 xxx    10 xxxxxx     10 xxxxxx     10 xxxxxx

**10:41实训室2-正课**

目标公司的笔试结束，题目比较简单，全部都是java相关的题目，一共17道题，其中12道选择，5道简答。但即便是如此我还是错了很多题，直接证明了我的基础确实是很烂。

剩下的就是等待群里发通知安排面试名单，我感觉很悬。

**13:10实训室2-午休时间**

机构的就业负责人发来消息安排了我到另一家公司去面试，那家公司没有安排笔试，直接去面试就可以了。但是那家公司的工作地点是在上海，4K包食宿。不是我的目标，但是闲着也是闲着，不如去体验一下吧，总会有些收获的。

…………

**15:40自家-休息**

回家了。心情复杂。不知道该从哪里说起比较好。让我整理一下过程……

今天算是走了一个过场，体验了一把笔试和面试的感觉。毕竟课都没上完，也从来没有复习过，被机构推上去面试，我都替那些面试官感到难过。简直和检查流水线产品没什么区别，换做是我坐那一天搞面试我也烦。

也就是说面试官的心情是肯定很绝望的，毕竟“次品”占大多数。

按照这种效率，一家公司面试一天撑死50人，然而参加面试的有200以上之多，最终公司也会只招3-8个人。虽然是我意料之中的事情吧，但现场体验过后还是蛮震撼的。那个面试官一整天就坐在一个小单间里，外面人挤人的过道排了一整个楼层的学员，一直排到安全通道，这场面甚至有点超出我的想象。不由得感叹机构真的是什么样的人(包括我这样的)都敢往外塞，而且学员是真的多。

负责面试的都是各个公司搞技术的，似乎没有人事hr，所以学员准备的简历基本上都不带看一眼的。

首先我参加笔试的那家目标公司公布的面试名单里没有我，这也很正常吧，毕竟比我优秀的人大把大把的。由于笔试的过程中有2名监考，所以也没有办法拍下笔试题目的照片。其实也没有拍下来的太大必要，毕竟题目是真的很简单，错了的话就表明基础是真的差。

然后下午1点参加面试的那一家公司……面试其实也就是知识问答，但分成2次面试，第1次面试的内容是JAVA基础，第2次面试则由另一个面试官在另一件房间进行提问，提问的内容则是项目经验相关（写在简历里的项目经验）。

而我直接在第1次面试中当场怒跪……所以也就没有第2次面试了。

最后我向面试官请教了一下我在这次面试中存在的问题，可能是由于态度诚恳吧，面试官摇了摇头又叹了口气并向我解释了原因：

1.    回答问题时，口述用词不严谨（毫无逻辑瞎鸡儿乱说），回答问题不全面，漏洞百出。不进行提示的话就无法完整的回答上来提出的问题。

2.    基础太烂。

以上。

…………

回到教室后顺带打听了一下我和班上其他学员在这家公司的面试中被问到的问题：

（我被问到的问题）

1.    说出你知道的几种线程创建方法。

2.    Integer的数值比较（f1=100; f2=100; f3=130; f4=130;  f1==f2输出？f3==f4输出？以及为什么。）

3.    final finally finalize※的区别

4.    有几种遍历集合的方法？

（其他学员被问到的问题）

1.  视图和表格的区别

2.  什么是值传递

3.  static和final等关键字用法及其含义。

4.  集合相关由浅至深的全部问题

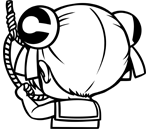
5.  事物的四大特性

6.  JDBC的使用方法与执行流程

……（全部的JAVA基础内容）

说来惭愧，这些问题我也并不是不懂不会，而是面试时不知道该如何正确的组织语言进行表达，同时也无法想起那些曾经学过的知识内容与注意事项。终归是练少了、题刷少了，答不上来也不怨任何人，不过这次面试倒是积累到了应得的经验。

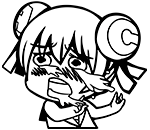
总之，今天的招聘会我是肯定是没戏啦，确实需要回炉重造。反倒是班上有几个本科生貌似是有戏，班主任也在群里说这次我们班的学员通过率很高……这下好了，我成了最稀烂的那一个，当然这也是纯属自作自受吧。昨天评论里有位老哥说的很对，真亏了我还每天坚持写日记，搞得自己学习很认真似的，然而现实是认真没有卵用，拿不出结果来，废物终究是废物。

当然，自暴自弃不是我的风格，既然有过这一次的耻辱，终将有一天我要双倍奉还。并且那一天不会太晚，最多不超过下个星期（下周四还有一场招聘会）。否则？那我就去日本乞讨要饭吧……

**16:33自家-休息**

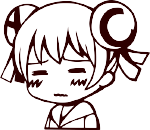
直到现在机构的就业群里还在发布各个公司的面试、复试名单。不得不说在这情况下确实可以很直观的感觉到，什么是竞争压力。我甚至觉得工资哪怕是有3K，需求摆在这里照样有人会为了工作岗位而挤到爆。

据我的观察，今天参加应聘的学员总人数保守估计超过500人。

P·S：昨天评论区里的一位老哥说的是对的，我真特么不该穿那么正式的，把自己整的干干净净衬衫领带样样齐全，结果整个机构的人就跟看SB似的看着我。在日企上班久了把自己的脑回路都整秀逗了，马德法克！

今日感想总结：虽然下周的课程依然正常进行，但我已经无心上课了。剩下的时间应该全部用在写代码和刷面试题上才对了。

今日评分：20/10分



今天就到这里，明天照例休息一天，下周一继续投稿。

感谢支持！

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

**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）：**

Address：github.com/wt62635