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

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2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：13天 （11天正课+2天自习）

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15日**

**Java就业班-第88天（自习）**

**--:--自家-休息**

**周六，上午到教室打卡签到，下午依旧是逃课回家。并没有学习，完全没有学习的欲望。**

**回到家后开始打DNF、搬砖，搬完砖后打暗黑地牢自虐了一晚上成功打造出全4级且无负面属性的完整小队。随即早上6点入睡，睡至周日下午2点起床，继续搬砖。**

**周日同上。**

**2019年6月17日**

**Java就业班-第89天**

**--:--自家-休息**

因为周末未能得到良好的休息，上午5点，准备在出门前躺在床上稍事休息，结果醒来时已经是下午4点。查看手机收到了几条消息：

1. 来自项目经理的未出勤询问。——说明了自己周末没有休息好，并承诺周二早上正常出勤上课。
2. 项目经理透露，本周末会有企业到机构进行招聘，提醒我抓紧时间复习。——回复收到。
3. 班主任在群里发布了机构内的模拟面试计划，针对我们班共安排了2天分时间段进行单人模拟面试。我的模拟面试时间定在：6/19周三18:00-19:00。要求：自行打印一份上周的自制简历用于模拟面试。
4. 班主任提到，自今日起，任何关于企业招聘的相关信息将会通过班级群进行发布，所以不再强制上课前收手机，务必随时关注。

以上。

既然今天已经旷课了，而我还是无心学习，那么只好继续搬砖。

不知道是不是因为压抑了太久的缘故，这种重复劳动性质的游戏方式让我欲罢不能。

仔细一想，这个周末我本来只是处于调整心态的目的而玩游戏的，然而怎么就变成了被游戏玩了？

关于我为什么会选择DNF，是因为那位照顾我的前辈就在玩，并且还多次拉我回归。本来我是拒绝的……结果就这样了。

一晚上又这么过去了……请叫我守夜冠军。

时间也差不多到6点了，准备出发。

**2019年6月18日**

**Java就业班-第90天**

**7:20实训室2-早自习**

**这应该是我第一次到的这么早，教室里空无一人。**

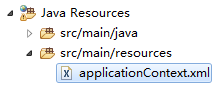
**昨天开始就是换新的课程和新的讲师了，很可惜我没能赶上。不过经过这两天的放纵和尽情的搬砖，我觉得我的学习热情有了明显的提升，心态也明显有所好转。剩下的时间应该不到2周了，最后加把劲吧。**

**首先抄一下昨天的课程笔记：**

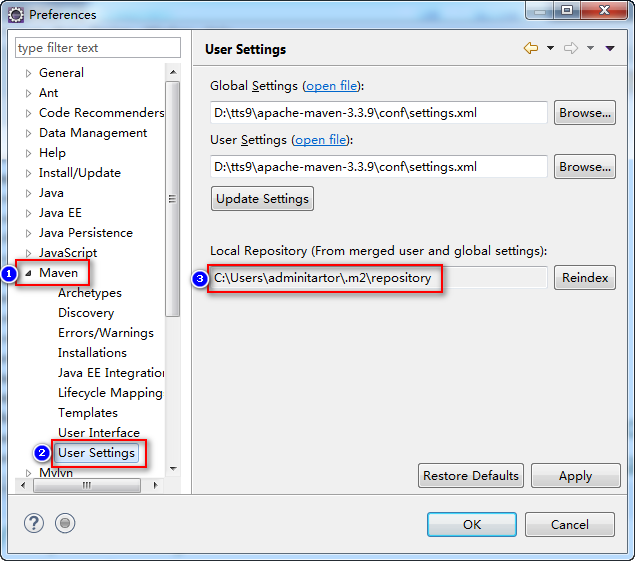
**### 1. Spring框架  
#### 1.1. 框架**  
框架：开发人员可以在项目开发过程中，引用某些框架，从而，在开发过程中，就可以不必关心某些功能的开发，而是由框架直接完成！

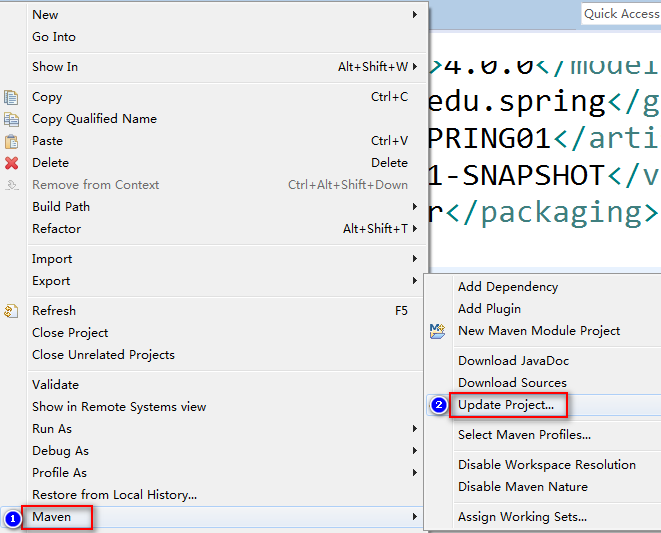
**#### 1.2. 解决的问题**  
Spring框架主要解决了创建对象和管理对象的问题！  
传统的创建对象的方法例如：  
    User user = new User();  
使用Spring框架之后，可以改为：  
    User user = 从框架中获取;  
这样的做法的好处主要在于解耦，即解除耦合度，表现为降低对象与对象之间的依赖关系。

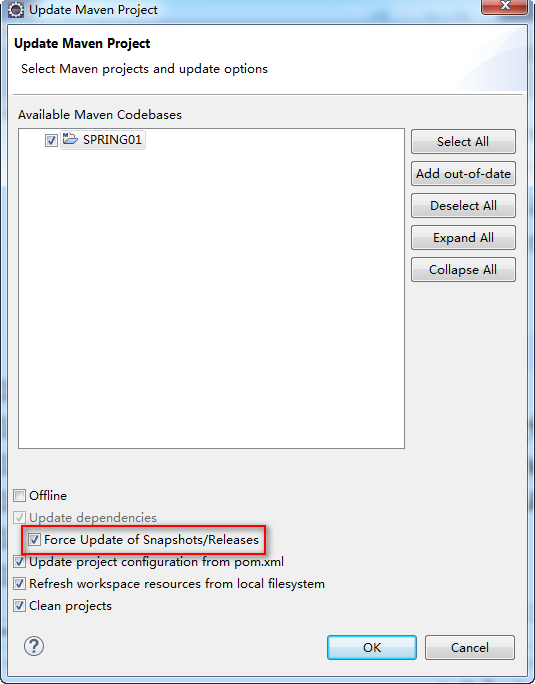
例如，在以下代码中，UserServlet就是依赖于UserDao的，因为功能的实现是通过UserDao中的reg()方法来实现的，如果没有UserDao类，则UserServlet无法完成用户注册功能：  
    public class UserDao {  
        public void reg() {  
        }  
    }  
  
    public class UserServlet {  
        public UserDao userDao = new UserDao();  
  
        public void doPost(...) {  
            userDao.reg();  
        }  
    }  
  
在以上UserServlet的代码中，明确的出现了UserDao的类名，也就是明确的表示了依赖于UserDao这个类，这种明确的依赖关系是不推荐的！万一随着项目的更新、维护等，UserDao这个类中的代码已经不足以满足后续的使用需求，希望通过另一个例如UserDao2这个类来替换它，则原有代码中：  
  
    public UserDao userDao = new UserDao();  
  
这样的语句都需要替换为：  
  
    public UserDao2 userDao = new UserDao2();  
  
这样的维护方式是非常不便利的！  
  
首先，可以通过接口解决其中一部分问题，例如：  
  
    public interface IUserDao {  
        void reg();  
    }  
  
然后：  
  
    public class UserDao implements IUserDao {}  
  
    public class UserDao2 implements IUserDao {}  
  
则在UserServlet中就可以使用接口去声明对象：  
  
    public IUserDao userDao = new UserDao();  
  
所以，无论使用哪个类，或者需要更换，以后都不用再调整以上变量的声明部分！  
  
如果希望对象的创建过程也不会体现出UserDao或UserDao2的名称，还可以使用设计模式中工厂模式来解决：  
  
    public class UserDaoFactory {  
        public static IUserDao newInstance() {  
            return new UserDao2();  
        }  
    }  
  
则之前的代码就可以调整为：  
  
    public IUserDao userDao = UserDaoFactory.newInstance();  
  
由于以上代码中完全没有出现UserDao或UserDao2的类名，即使后续需要替换这些类，也只需要将新的类也实现IUserDao接口，并修改工厂方法中的返回值，而项目中其它位置的代码是不需要调整的！  
  
    public class UserServlet {  
        public IUserDao userDao = UserDaoFactory.newInstance();  
  
        public void doPost(...) {  
            userDao.reg();  
        }  
    }  
  
对于代码中并不出现别的类的名称，却可以正确使用它，则称之为耦合度较低！  
  
在实际开发时，为每种定位的类都创建对应的工厂类是不现实的！所以，就有了Spring框架，可以把Spring框架理解为是一个庞大的工厂，可以创建各种各样的类的对象，在整个项目开发过程中，需要对象时，就找Spring框架去获取对象即可，从而实现解耦的目的，而Spring框架由于可以管理很多对象，所以也被称之为Spring容器。  
  
当然，关于Spring框架，还有其它的特性，例如：IoC、DI、AOP等，将在后续进行讲解。  
  
**#### 1.3. 准备Spring环境**  
  
创建`Maven Project`，在创建过程中，勾选**\*\*Create a simple project ...\*\***，**\*\*Artifact Id\*\***输入`xx.xxxx.spring`，`Group Id`输入`SPRING01`，**\*\*Packaging\*\***选择`war`。  
  
> 在输入项目信息时，Artifact Id应该是即将使用的包中，Group Id应该　能描述项目的特征，Packaging应该根据项目的定位来选择，由于后续使用Java语言开发项目基本上是Web项目，所以可以选择war。  
  
创建好项目后，需要先生成**\*\*web.xml\*\***文件。  
  
然后，在**\*\*pom.xml\*\***文件中添加**\*\*spring-webmvc\*\***依赖：  
  
    <dependency>  
        <groupId>org.springframework</groupId>  
        <artifactId>spring-webmvc</artifactId>  
        <version>4.3.8.RELEASE</version>  
    </dependency>  
  
> 其实此次只需要使用spring-context依赖，但是，后续学习到SpringMVC框架开始就必须使用spring-webmvc依赖，而且，spring-webmvc依赖中包含了spring-context依赖，所以，此次也可以直接使用spring-webmvc。  
  
另外，还需要下载Spring的配置文件压缩包，解压得到**\*\*applicationContext.xml\*\***文件，将文件复制到项目的**\*\*src/main/resources\*\***下：  
  
![](02.png)

  
  
**#### 1.4. 通过Spring创建并获取对象－通过无参数构造方法**  
  
首先创建`xx.xxxx.spring.User`类：  
  
    public class User {  
    }  
  
然后，在Spring的配置文件中添加配置：  
  
    <bean id="user" class="xx.xxxx.spring.User">  
    </bean>  
  
然后，添加单元测试的依赖：  
  
    <dependency>  
        <groupId>junit</groupId>  
        <artifactId>junit</artifactId>  
        <version>4.12</version>  
    </dependency>  
  
可以在**\*\*src/test/java\*\***下创建`xx.xxxx.spring.Tests`单元测试类，编写单元测试：  
  
    public class Tests {  
  
        @Test  
        public void test() {  
            // 加载Spring配置文件，获取Spring容器  
            // 从容器中获取所需的对象  
            // 测试  
            // 释放资源  
        }  
  
    }  
  
具体实现为：  
  
    public class Tests {  
      
        @Test  
        public void test() {  
            // 加载Spring配置文件，获取Spring容器  
            ClassPathXmlApplicationContext ac  
                = new ClassPathXmlApplicationContext(  
                    "applicationContext.xml");  
      
            // 从容器中获取所需的对象  
            User user = (User) ac.getBean("user");  
      
            // 测试  
            System.out.println(user);  
      
            // 释放资源  
            ac.close();  
        }  
      
    }  
  
**#### 1.5. 通过Spring创建并获取对象－通过静态工厂方法（不常用）**  
  
如果某个类中有静态的方法，并获取当前类的对象，例如`Calendar`类，通过`Calendar.getInstance()`即可获取`Calendar`类型的对象，则在配置时：  
  
    <!-- 通过静态工厂方法 -->  
    <bean id="cal"  
        class="java.util.Calendar"  
        factory-method="getInstance">  
    </bean>  
  
**#### 1.6. 通过Spring创建并获取对象－通过实例工厂方法（不常用）**  
  
表现为某个类无法直接创建对象，但是，却有另一个工厂类，可以先创建工厂类的对象，然后，再创建所需要类的对象，例如：  
  
    public class Phone {  
        public Phone(String name) {  
        }  
    }  
  
    public class PhoneFactory {  
          
        public Phone getInstance() {  
            return new Phone("XiaoMi");  
        }  
      
    }  
  
然后，在配置时：  
  
    <!-- 通过实例工厂方法 -->  
    <bean id="phoneFactory"   
        class="xx.xxxx.spring.PhoneFactory">  
    </bean>  
    <bean id="phone"   
        class="xx.xxxx.spring.Phone"  
        factory-bean="phoneFactory"  
        factory-method="getInstance">  
    </bean>  
  
可以看到，这种做法就必须先创建工厂类`PhoneFactory`的对象，然后调用工厂类的方法`getInstance()`，从而完成类的创建。  
  
这种做法还要求工厂类必须有无参数的构造方法！

**其他：**

**### 1. Maven添加依赖失败的解决方案**  
  
如果希望快速的解决jar包损坏且Eclipse不会重新下载的问题，可以更换jar包的版本！  
  
如果不确定哪些其它的版本是可用的，首先，应该先关闭Eclipse，然后找到Maven的本地仓库：  
  
![](A01.png)  


删除该文件夹中错误的jar所在的文件夹，甚至删除整个**\*\*.m2\*\***文件夹也是可以的！  
  
再次打开Eclipse，对项目点右键，依次选择**\*\*Maven\*\*** > **\*\*Update Project\*\***：  
  
![](A02.png)  


在弹出的对话框中，勾选**\*\*Force Update ...\*\***：  
  
![](A03.png)  


然后，等待更新结束即可！  
  
**### 2. 配置Spring时没有代码提示的问题**  
  
在Spring的配置中，使用了一系列的**\*\*xsd\*\***文件约定了文件的配置，仅当下载了这些文件时，才可以有代码提示！  
  
这些文件部署在`http://www.springframework.org/`网站中，是有可能连接不上的，进而无法下载对应的**\*\*xsd\*\***文件，则代码没有提示！  
  
  
**### 3. 关于排查异常的错误报告**  
  
通常，在大量的异常错误报告中，只需要关注最后一次出现的`Caused by`，如果没有，则关注第1行错误信息，即可找到错误的原因，例如：  
  
    org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'user' defined in class path resource [applicationContext.xml]: Instantiation of bean failed; nested exception is org.springframework.beans.BeanInstantiationException: Failed to instantiate [xx.xxxx.spring.User]: No default constructor found; nested exception is java.lang.NoSuchMethodException: xx.xxxx.spring.User.<init>()  
          
    Caused by: org.springframework.beans.BeanInstantiationException: Failed to instantiate [xx.xxxx.spring.User]: No default constructor found; nested exception is java.lang.NoSuchMethodException: xx.xxxx.spring.User.<init>()  
      
    Caused by: java.lang.NoSuchMethodException: xx.xxxx.spring.User.<init>()

**8:30实训室2-早自习**

**昨天的课看上去讲了很多的东西，但从笔记的内容来看没有特别难以理解的地方，希望今天正常讲课的时候不会掉队吧。**

**今日晨讲主题——数据库事务以及四大特性**

**四大特性：**

**原子性：事务的完整操作**

**一致性：当事务完成时数据必须处于一致的状态**

**隔离性：对数据进行修改彼此发生的间隔**

**持久性：当事务完成时，它对系统的影响是永久性的**

**…………**

**其他略。**

**我并没有仔细听晨讲的内容，而是把昨天我旷掉的课程笔记内容自习研究了一下，以免等会上课时跟不上进度。**

**从笔记上来看，昨天似乎是只在eclipse中创建了maven项目和一些关于spring框架作用的说明。**

**从同桌那打听到模拟面试其实从昨天晚上晚自习就已经开始了，担任模拟面试官的是项目经理，但是结果似乎并不理想，据说项目经理昨晚说道 “基础问题一个答不上来，就你们这样还怎么找工作？”。**

**其实这也是我一直在说的话……**

**9:02实训室2-正课**

**开始上课。这位讲师没有课程回顾的环节。**

**昨天的主要课程学习了如何通过框架创建对象。**

**今天的主要课程就是学习什么是Spring的IoC。**

**※Tips:什么是数据安全？实际上数据安全一般并不是指泄露、窃取、攻破等等黑客行为或安全漏洞。当设计好的程序得出意料之外的结果时，这个时候的数据就是不安全的。**

**11:10实训室2-课间休息**

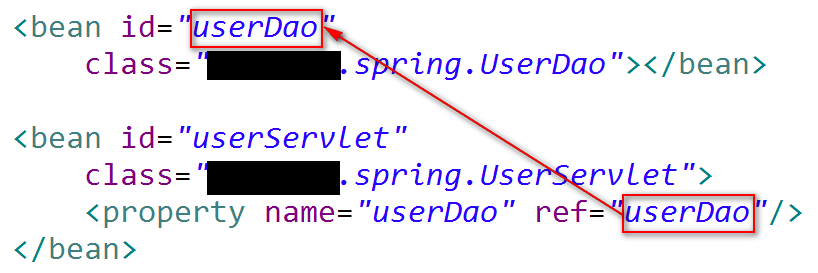
**守夜过后的debuff时间到了……实在是撑不住了，我得睡会。**

**16:02实训室2-正课**

**睡醒了，洗把脸。查看一下目前为止的课程笔记，希望没有掉队的太多……**

**17:43实训室2-下课时间**

**今天的课程就到这里了，今日课程笔记：**

**### 1. Spring管理的对象的作用域与生命周期（不常用）**  
  
由Spring管理的对象，默认都是单例的！并且，都是饿汉式的单例模式。  
  
在配置`<bean>`节点时，可以添加`scope`属性其是否单例，当取值为`singleton`时表示单例，该值也是默认值，当取值为`prototype`时表示非单例：  
  
    <bean id="user"   
        class="xx.xxxx.spring.User"  
        scope="prototype"></bean>  
  
在单例模式的基础之上，还可以通过`lazy-init`属性配置它是否为懒汉式的单例模式，默认值为`false`，即非懒汉式，也就是饿汉式的单例模式，当取值为`true`时，表示懒汉式的单例模式：  
  
    <bean id="user"   
        class="xx.xxxx.spring.User"  
        scope="singleton"  
        lazy-init="true"></bean>  
  
如果某个类被配置为单例模式，还可以配置它的生命周期方法：首先，在类中声明2个方法，这2个方法都应该是`public`方法，返回值都是`void`，方法名称可以自由定义，方法必须没有参数，例如：  
  
    public void init() {  
        System.out.println("User.init()");  
    }  
      
    public void destroy() {  
        System.out.println("User.destroy()");  
    }  
  
然后，在Spring的配置文件中，在`<bean>`节点中配置`init-method`和`destroy-method`属性，即可配置初始化方法和销毁方法，这2个属性的取值都是需要调用的方法的名称：  
  
    <bean id="user"   
        class="xx.xxxx.spring.User"  
        scope="singleton"  
        lazy-init="true"  
        init-method="init"  
        destroy-method="destroy"></bean>  
  
**### 2. Spring的IoC**  
  
**#### 2.1. 什么是IoC**  
  
IoC表示Inversion of control，即“控制反转”。传统模式下，对象的创建与管理都是由开发人员编写代码直接完成的，而使用Spring后，将创建与管理交给了框架，则称之为控制反转。  
  
其中，比较重要的环节是为对象的某些属性进行赋值，称之为DI，即Dependency Injection，表示“依赖注入”，通俗的来说，是为其属性赋值，也称之为“为其属性注入值”。  
  
Spring通过DI实现了IoC，即DI是实现手段，而IoC是需要实现的目标。  
  
**#### 2.2. 通过SET方式注入属性的值**  
  
假设`User`类中有名为`name`的属性，需要为该属性注入值，首先，需要为该属性添加SET/GET方法(其实只有SET方法是必须的)：  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
然后，在`<bean>`节点子级添加`<property>`节点进行配置：  
  
    <bean id="user"   
        class="xx.xxxx.spring.User">  
        <!-- 使用property节点为属性注入值 -->  
        <!-- name：属性名 -->  
        <!-- value：属性值 -->  
        <property name="name" value="Kitty"></property>  
    </bean>  
  
其实，框架在处理时，发现有`<property>`节点，就会尝试为属性赋值，它会基于该节点的`name`属性值得到SET方法的名称，规则就是`set`加上属性名且首字母改为大写，得到SET方法名称，如果属性名称是`name`，则框架将调用的方法就是`setName`，如果属性名称是`password`，则框架将调用的方法已经`setPassword`，然后，将`value`属性对应的值，作为将调用的方法的参数，以上示例代码中的配置，使得框架将调用`对象.setName("Kityy");`语句。  
  
所以，在`<property>`节点中，`name`属性配置的其实是SET方法的方法名右侧的部分，且首字母改为小写！  
  
但是，在实际使用时，SET方法都是由开发工具自动生成，生成规则与Spring框架处理时的规则是完全相同的，所以，也可以**\*\*简单的认为\*\***`<property>`节点中配置的`name`就是属性的名称！  
  
在某些情况下，需要注入的属性值并不是基本值(可以直接书写的值，例如数值、字符串等)，而是另一个类的对象时，可以先使得Spring也管理另一个类的对象，然后，注入值时，通过`ref`属性引用那个`<bean>`即可：  
  
![](01.png)  


**#### 2.3. 通过构造方法注入属性的值（不常用）**  
  
如果某个属性是通过构造方法设置值的，例如：  
  
    public class Person {  
      
        // 25  
        private Integer age;  
      
        public Person(Integer age) {  
            super();  
            this.age = age;  
        }  
      
        @Override  
        public String toString() {  
            return "Person [age=" + age + "]";  
        }  
      
    }  
  
则在配置时，应该使用`<constructor-arg>`节点进行配置：  
  
    <bean id="person" class="xx.xxxx.spring.Person">  
        <constructor-arg index="0" value="25" />  
    </bean>  
  
以上属性的配置中，`index`表示第几个参数，从0开始顺序编号，然后，根据值的类型选择使用`value`或`ref`属性进行配置即可！  
  
**#### 2.4. 注入集合类型的值**  
  
如果某个类中的属性是`List`集合类型的，并需要注入值：  
  
    public class SampleBean {  
          
        // Alex, Lucy, Kitty, Henry  
        public List<String> names;  
      
        public void setNames(List<String> names) {  
            this.names = names;  
        }  
      
    }  
  
然后，在Spring的配置文件中：  
  
    <bean id="sampleBean"  
        class="xx.xxxx.spring.SampleBean">  
        <property name="names">  
            <list>  
                <value>Alex</value>  
                <value>Lucy</value>  
                <value>Kitty</value>  
                <value>Henry</value>  
            </list>  
        </property>  
    </bean>  
  
如果需要注入`Set`类型的值，例如：  
  
    // Beijing, Shanghai, Guangzhou, Shenzhen  
    public Set<String> cities;  
  
在配置注入时，使用`<set>`节点即可：  
  
    <property name="cities">  
        <set>  
            <value>Beijing</value>  
            <value>Shanghai</value>  
            <value>Guangzhou</value>  
            <value>Shenzhen</value>  
        </set>  
    </property>  
  
另外，关于`Map`类型集合的配置例如：  
  
    <property name="session">  
        <map>  
            <entry key="username" value="Jack" />  
            <entry key="password" value="1234" />  
            <entry key="from" value="Nanjing" />  
        </map>  
    </property>  
  
关于数组类型集合的配置例如：  
  
    <property name="numbers">  
        <array>  
            <value>9</value>  
            <value>5</value>  
            <value>2</value>  
            <value>7</value>  
        </array>  
    </property>  
  
> 在配置数组时，也可以使用`<list>`节点，反之，在配置`List`集合时，也可以使用`<array>`节点，但是，推荐使用匹配的节点进行配置。  
  
关于`Properties`类型的配置：  
  
    <property name="config">  
        <props>  
            <prop key="driver">com.mysql.jdbc.Driver</prop>  
            <prop key="username">root</prop>  
            <prop key="password">root</prop>  
        </props>  
    </property>  
  
在配置以上集合类型的值时，也可以事先使用例如`<util:list>`这类节点先将值配置好：  
  
    <util:list id="names">  
        <value>Tom</value>  
        <value>Alex</value>  
        <value>Lucy</value>  
        <value>Kitty</value>  
        <value>Henry</value>  
    </util:list>  
  
然后再注入到属性中：  
  
    <property name="names" ref="names" />  
  
比较特殊的是读取`Properties`类型的数据，在Spring中，可以通过`<util:properties>`节点的`location`属性指定需要读取的文件：  
  
    <util:properties id="config"  
        location="classpath:db.properties" />  
  
然后，就可以注入到相应的属性中：  
  
    <property name="config" ref="config" />

其他知识补充笔记：

**### 4. List与Set集合的区别**  
  
`List`集合是序列的，其中的元素是允许重复的，常见的实现类是`ArrayList`和`LinkedList`，前者查询效率较高，后者修改效率较高；  
  
`Set`集合是散列的，其中的元素是不允许重复的，重复的标准是：2个对象的`equals()`对比结果为`true`，且2个对象的`hashCode()`值相同，常见的实现类是`HashSet`和`TreeSet`、`LinkedHashSet`。  
  
另外，所有的`Set`的本质都是只有key没有value的`Map`。  
  
**### 5. 什么情况下需要自定义构造方法**  
  
1. 创建对象时快速的为某些属性赋值，例如：  
  
    User user = new User("David", "1234", 25);  
  
2. 限制对象的创建过程，例如：单例模式。  
  
3. 强制要求必须提供某些参数才可以创建对象。  
  
**### 6. 内存溢出/泄漏**  
  
在JVM中有垃圾回收机制，可以回收垃圾数据，通常，丢失引用的对象就会被视为垃圾数据，会被回收！  
  
如果程序中出现某些异常错误，导致某个变量的作用域消失，但是对象的引用依然存在，不会被视为垃圾，就会导致无法回收！  
  
所有这类无法使用却不会被回收的数据，都可能导致内存溢出。  
  
其实，少量的内存溢出，并没有明显的危害，但是，每个程序员都应该尽量规避所有可能的内存溢出，例如及时释放资源，对于可能出现异常的代码片断中，使用`finally`释放资源。  
  
**### 7. 设计模式之单例模式**  
  
单例模式可以保证在同一时刻，某个类的实例(对象)只有1个！  
  
简单的单例模式可以是：  
  
    public class King {  
  
        private static King king = new King();  
  
        private King() {  
        }  
  
        public static King getInstance() {  
            return king;  
        }  
  
    }  
  
以上单例模式中，从一开始就已经创建好了`King`的对象，随时可供调用/获取，这种单例模式称之为**\*\*饿汉式\*\***单例模式，另外，还有一种**\*\*懒汉式\*\***的单例模式，它的特点是“不到逼不得已的时候，并不会创建对象”，代码例如：  
  
    public class King {  
  
        private static King king = null;  
        private static final Object lock = new Object();  
  
        private King() {  
        }  
  
        public static King getInstance() {  
            if (king == null) {  
                sychronized (lock) {  
                    if (king == null) {  
                        king = new King();  
                    }  
                }  
            }  
            return king;  
        }  
  
    }  
  
**18:53宿舍-休息**  
**回宿舍的路上打印好了明天晚自习模拟面试时需要用到的自制简历，根据今天班上已经进行了模拟面试的学员所述，所谓的模拟面试实际上就是教一些实际面试中的一些“话术”、“扬长避短”，简而言之就是如何“包装自己”，确保怎样才能提高面试被採用的几率。**

**最有趣的环节终于要开始表演了。**

**今天也是一如既往的没有上晚自习，回到宿舍自行研究学习因为睡觉而错过的今日课程笔记。现在上课可以明显的感觉到讲框架的这名讲师的功底确实很深厚，在讲解spring框架的作用原理的过程中还是不是穿插补充了一些之前课程中没有详细讲解的基础知识。课程进度不快不慢，还是非常适合我目前的学习状态的。**

**由于没能得到很好的休息的缘故，今天准备尽早休息。**

**对于拖更了2天这件事情，还请各位海涵……连我自己都不知道原来我已经积累了那么大的压力，好在这次放松的比较彻底（毕竟都旷了一天课），接下来的这2个星期，想必会趣事不断吧。**

**我会持续跟进并记录任何真实发生的趣闻，敬请期待。**

**今日感想总结：是时候再认真一次了。**

今日评分：8/10分

今天就到这里，明天继续投稿。

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