**数据交换格式与SpringIOC底层实现**

# 课程目标

XML和JSON

Java反射机制

手写SpringIOC

# Java反射机制

## 什么是Java反射

就是正在运行，动态获取这个类的所有信息。

## 反射机制的作用

  1，反编译：.class-->.java

   2．通过反射机制访问java对象的属性，方法，构造方法等；

## 反射机制的应用场景

Jdbc 加载驱动-----

Spring ioc

框架

## 反射机制获取类有三种方法

|  |
| --- |
| **//第一种方式：**  **Classc1 = Class.*forName*("Employee");**  **//第二种方式：**  **//java中每个类型都有class 属性.**  **Classc2 = Employee.class;**    **//第三种方式：**  **//java语言中任何一个java对象都有getClass 方法**  **Employeee = new Employee();**  **Classc3 = e.getClass(); //c3是运行时类 (e的运行时类是Employee)** |

## 反射创建对象的方式

|  |
| --- |
| **Class<?> forName = Class.*forName*("com.itmayiedu.entity.User");**  **// 创建此Class 对象所表示的类的一个新实例 调用了User的无参数构造方法.**  **Object newInstance = forName.newInstance();** |

实例化有参构造函数

|  |
| --- |
| **Class<?> forName = Class.*forName*("com.itmayiedu.entity.User");**  **Constructor<?> constructor = forName.getConstructor(String.class, String.class);**  **User newInstance = (User) constructor.newInstance("123", "123");** |

## 反射创建api

|  |  |
| --- | --- |
| 方法名称 | 作用 |
| getDeclaredMethods [] | 获取该类的所有方法 |
| getReturnType() | 获取该类的返回值 |
| getParameterTypes() | 获取传入参数 |
| getDeclaredFields() | 获取该类的所有字段 |
| setAccessible | 允许访问私有成员 |

## 使用反射为类私有属性赋值

|  |
| --- |
| **// 为user对象私有属性赋值**  **Class<?> classUser = Class.*forName*("com.itmayiedu.entity.User");**  **// 获取到当前的所有属性**  **Field[] fields = classUser.getDeclaredFields();**  **for (Field field : fields) {**  **System.*out*.println(field.getName());**  **}**  **// 获取当前所有的方法**  **Method[] declaredMethods = classUser.getDeclaredMethods();**  **for (Method method : declaredMethods) {**  **System.*out*.println(method.getName());**  **}**  **// 初始化对象**  **User user = (User) classUser.newInstance();**  **Field declaredField = classUser.getDeclaredField("id");**  **// 标记为true 允许反射赋值**  **declaredField.setAccessible(true);**  **declaredField.set(user, "20");**  **System.*out*.println("使用反射机制给id赋值为:"+user.getId());** |

## JDBC反射加载驱动

|  |
| --- |
| **public class DemoJDBC {**  **public static void main(String[] args) throws Exception {**  **// 加载驱动类**  **Class.*forName*("com.mysql.jdbc.Driver");**    **// 通过DriverManager获取数据库连接**  **String url = "jdbc:mysql://192.168.1.150/test";**  **String user = "teamtalk";**  **String password = "123456";**  **Connection connection = (Connection) DriverManager.*getConnection*(**  **url, user, password);**    **PreparedStatement statement = (PreparedStatement) connection.prepareStatement(**  **"insert persion (name, age) value (?, ?)");**  **statement.setString(1, "hdu");**  **statement.setInt(2, 21);**  **statement.executeUpdate();**    **ResultSet resultSet = statement.executeQuery("select \* from persion");**  **// 操作ResultSet结果集**  **while (resultSet.next()) {**  **// 第一种获取字段方式**  **System.*out*.println(resultSet.getString(1) + " " +**  **resultSet.getString(2) + " " + resultSet.getString(3));**  **}**  **// 关闭数据库连接**  **resultSet.close();**  **statement.close();**  **connection.close();**  **}**  **}** |

## 禁止使用反射机制初始化

将构造函数为私有化

# 手写SpringIOC框架

## 什么是SpringIOC

什么是SpringIOC，就是把每一个bean(实体类)与bean(实体了)之间的关系交给第三方容器进行管理。

Xml配置:

|  |
| --- |
| **<beans>**  **<bean id=*"user1"* class=*"com.itmayiedu.entity.UserEntity"*>**  **<property name=*"userId"* value=*"0001"*></property>**  **<property name=*"userName"* value=*"余胜军"*></property>**  **</bean>**  **<bean id=*"user2"* class=*"com.itmayiedu.entity.UserEntity"*>**  **<property name=*"userId"* value=*"0002"*></property>**  **<property name=*"userName"* value=*"张三"*></property>**  **</bean>**  **</beans>** |

Java代码:

|  |
| --- |
| //1.读取springxml配置  ClassPathXmlApplicationContext classPathXmlApplicationContext = **new** ClassPathXmlApplicationContext(  "applicationContext.xml");  //2.获取bean对象  TestService testService = (TestService) classPathXmlApplicationContext.getBean("testService");  System.***out***.println(testService.test()); |

## 什么是SpringIOC底层实现原理

1.读取bean的XML配置文件

2.使用beanId查找bean配置，并获取配置文件中class地址。

3.使用Java反射技术实例化对象

4.获取属性配置，使用反射技术进行赋值。

详细步骤

|  |
| --- |
| 1.利用传入的参数获取xml文件的流,并且利用dom4j解析成Document对象  2.对于Document对象获取根元素对象<beans>后对下面的<bean>标签进行遍历,判断是否有符合的id.  3.如果找到对应的id,相当于找到了一个Element元素,开始创建对象,先获取class属性,根据属性值利用反射建立对象.  4.遍历<bean>标签下的property标签,并对属性赋值.注意,需要单独处理int,float类型的属性.因为在xml配置中这些属性都是以字符串的形式来配置的,因此需要额外处理.  5.如果属性property标签有ref属性,说明某个属性的值是一个对象,那么根据id(ref属性的值)去获取ref对应的对象,再给属性赋值.  6.返回建立的对象,如果没有对应的id,或者<beans>下没有子标签都会返回null |

## 建立实体类

|  |
| --- |
| **public class User {**  **private String userId;**  **private String userName;**  **public String getUserId() {**  **return userId;**  **}**  **public void setUserId(String userId) {**  **this.userId = userId;**  **}**  **public String getUserName() {**  **return userName;**  **}**  **public void setUserName(String userName) {**  **this.userName = userName;**  **}**  **}** |

## 使用反射技术完成Java代码

|  |
| --- |
| **public class ClassPathXmlApplicationContext {**  **private String pathXml = null;**  **public ClassPathXmlApplicationContext(String pathXml) {**  **this.pathXml = pathXml;**  **}**  **public Object getBean(String beanId) throws Exception {**  **if (StringUtils.*isEmpty*(beanId)) {**  **throw new Exception("beanId is null");**  **}**  **SAXReader saxReader = new SAXReader();**  **Document read = saxReader.read(this.getClass().getClassLoader().getResource(pathXml));**  **// 获取到根节点**  **Element rootElement = read.getRootElement();**  **// 根节点下所有的子节点**  **List<Element> elements = rootElement.elements();**  **for (Element element : elements) {**  **// 获取到节点上的属性**  **String id = element.attributeValue("id");**  **if (StringUtils.*isEmpty*(id)) {**  **continue;**  **}**  **if (!id.equals(beanId)) {**  **continue;**  **}**  **// 使用java反射机制初始化对象**  **String beanClass = element.attributeValue("class");**  **Class<?> forName = Class.*forName*(beanClass);**  **Object newInstance = forName.newInstance();**  **List<Element> propertyElementList = element.elements();**  **for (Element el : propertyElementList) {**  **String name = el.attributeValue("name");**  **String value = el.attributeValue("value");**  **Field declaredField = forName.getDeclaredField(name);**  **declaredField.setAccessible(true);**  **declaredField.set(newInstance, value);**  **}**  **return newInstance;**    **}**  **return null;**  **}**  **public static void main(String[] args) throws Exception {**  **ClassPathXmlApplicationContext classPath = new ClassPathXmlApplicationContext("applicationContext.xml");**  **User user = (User) classPath.getBean("user2");**  **System.*out*.println(user.getUserId() + "---" + user.getUserName());**  **}**  **}** |

## Maven坐标

|  |
| --- |
| **<dependencies>**  **<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->**  **<dependency>**  **<groupId>org.apache.commons</groupId>**  **<artifactId>commons-lang3</artifactId>**  **<version>3.4</version>**  **</dependency>**  **<dependency>**  **<groupId>org.dom4j</groupId>**  **<artifactId>dom4j</artifactId>**  **<version>2.0.0</version>**  **</dependency>**  **</dependencies>** |