**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

# 什么是数据交换格式

客户端与服务器常用数据交换格式xml、json、html

# 数据交换格式用场景

移动端(安卓、IOS)通讯方式采用http协议+JSON格式 走restful风格。

很多互联网项目都采用Http协议+JSON

因为xml比较重WebService服务采用http+xml格式 银行项目使用比较多

同学们可以思考下？移动端和PC端服务器是接口是怎么设计的？

画图演示

# 数据交换格式

## JSON简单使用

### 什么是JSON

JSON([JavaScript](http://lib.csdn.net/base/javascript) Object Notation)是一种轻量级的数据交换格式，相比于xml这种数据交换格式来说，因为解析xml比较的复杂，而且需要编写大段的代码，所以客户端和服务器的数据交换格式往往通过JSON来进行交换。

样例:

|  |
| --- |
| {  "id": "10",  "name": "每特教育",  "items": [  {  "itemId": "20",  "itemName": "蚂蚁课堂"  },  {  "itemId": "21",  "itemName": "每特学院"  }  ]  } |

**JSON：JavaScript 对象表示法（JavaScript Object Notation）。**

**JSON的形式是用大括号“{}”包围起来的项目列表，每一个项目间用逗号（,）分隔，而项目就是用冒号（:）分隔的属性名和属性值。这是典型的字典表示形式，也再次表明javascript里的对象就是字典结构。不管多么复杂的对象，都可以用一句JSON代码来创建并赋值。在JSON中，名称 / 值对包括字段名称（在双引号中），后面写一个冒号，然后是值**

### JSON格式的分类

**JSON有两种结构**

**json简单说就是javascript中的对象和数组，所以这两种结构就是对象和数组两种结构，通过这两种结构可以表示各种复杂的结构**

**1、对象：对象在js中表示为“{}”括起来的内容，数据结构为 {key：value,key：value,...}的键值对的结构，在面向对象的语言中，key为对象的属性，value为对应的属性值，所以很容易理解，取值方法为 对象.key 获取属性值，这个属性值的类型可以是 数字、字符串、数组、对象几种。**

**2、数组：数组在js中是中括号“[]”括起来的内容，数据结构为 ["java","javascript","vb",...]，取值方式和所有语言中一样，使用索引获取，字段值的类型可以是 数字、字符串、数组、对象几种。  
经过对象、数组2种结构就可以组合成复杂的数据结构了。**

### 常用JSON解析框架

fastjson(阿里)、gson(谷歌)、jackson(SpringMVC自带)

### 使用fastjson解析json

添加jar fastjson-1.1.43 或引入maven依赖

|  |
| --- |
| <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.1.43</version>  </dependency> |

### 使用fastjson api

|  |
| --- |
| public static final Object parse(String text); // 把JSON文本parse为JSONObject或者JSONArray  public static final JSONObject parseObject(String text)； // 把JSON文本parse成JSONObject     public static final <T> T parseObject(String text, Class<T> clazz); // 把JSON文本parse为JavaBean  public static final JSONArray parseArray(String text); // 把JSON文本parse成JSONArray  public static final <T> List<T> parseArray(String text, Class<T> clazz); //把JSON文本parse成JavaBean集合  public static final String toJSONString(Object object); // 将JavaBean序列化为JSON文本  public static final String toJSONString(Object object, boolean prettyFormat); // 将JavaBean序列化为带格式的JSON文本  public static final Object toJSON(Object javaObject); 将JavaBean转换为JSONObject或者JSONArray。 |

### 解析json

|  |
| --- |
| **static String *jsonStr* = "{\"sites\":[{\"name\":\"蚂蚁课堂\",\"url\":\"www.itmayiedu.com\"},{\"name\":\"每特教育\",\"url\":\"http://meiteedu.com/\"}]}";**  **public static void main(String[] args) {**  **JSONObject jsonObject = new JSONObject();**  **// 将json字符串转为jsonbject**  **JSONObject jsonStrObject = jsonObject.*parseObject*(*jsonStr*);**  **JSONArray jsonArray = jsonStrObject.getJSONArray("sites");**  **for (Object object : jsonArray) {**  **JSONObject stObject = (JSONObject) object;**  **String name = stObject.getString("name");**  **String url = stObject.getString("url");**  **System.*out*.println(name + "---" + url);**  **}**  **}** |

### Json转对象

#### 定义类

|  |
| --- |
| class Item {  private String itemId;  private String itemName;  public String getItemId() {  return itemId;  }  public void setItemId(String itemId) {  this.itemId = itemId;  }  public String getItemName() {  return itemName;  }  public void setItemName(String itemName) {  this.itemName = itemName;  }  @Override  public String toString() {  return "Item [itemId=" + itemId + ", itemName=" + itemName + "]";  }  }  public class User {  private String id;  private String name;  private List<Item> items;  public String getId() {  return id;  }  public void setId(String id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public List<Item> getItems() {  return items;  }  public void setItems(List<Item> items) {  this.items = items;  }  @Override  public String toString() {  return "User [id=" + id + ", name=" + name + ", items=" + items + "]";  }  } |

#### Json转换对象api

|  |
| --- |
| **// 2.使用java反射机制 对应生成对象**  **User user = new JSONObject().*parseObject*(*json*, User.class);**  **System.*out*.println("user:" + user.toString());** |

### 组装json

#### Json api封装json

|  |
| --- |
| **// 1.json api 封装json**  **JSONObject root = new JSONObject();**  **root.put("id", "01");**  **root.put("name", "张三");**  **JSONArray arrayList = new JSONArray();**  **JSONObject object1 = new JSONObject();**  **object1.put("itemId", "20");**  **object1.put("itemName", "蚂蚁课堂");**  **JSONObject object2 = new JSONObject();**  **object2.put("itemId", "30");**  **object2.put("itemName", "每特学院");**  **arrayList.add(object1);**  **arrayList.add(object2);**  **root.put("items", arrayList);**  **System.*out*.println(root.toJSONString());** |

#### 将对象转换成json字符串

|  |
| --- |
| **User user = new User();**  **user.setId("01");**  **user.setName("张三");**  **List<Item> items =new ArrayList<Item>();**  **Item item1 = new Item();**  **item1.setItemId("20");**  **item1.setItemName("蚂蚁课堂");**  **Item item2 = new Item();**  **item2.setItemId("30");**  **item2.setItemName("每特学院");**  **items.add(item1);**  **items.add(item2);**  **user.setItems(items);**  **System.*out*.println(new JSONObject().*toJSONString*(user));** |

## XML简单使用

### 什么是XML？

它是可扩展标记语言（Extensible Markup Language，简称XML），是一种标记语言。

XML 全称为可扩展的标记语言。主要用于描述数据和用作配置文件。

XML 文档在逻辑上主要由一下 5 个部分组成：

XML 声明：指明所用 XML 的版本、文档的编码、文档的独立性信息

文档类型声明：指出 XML 文档所用的 DTD

元素：由开始标签、元素内容和结束标签构成

注释：以结束，用于对文档中的内容起一个说明作用

处理指令：通过处理指令来通知其他应用程序来处理非 XML 格式的数据，格式为

　　XML 文档的根元素被称为文档元素，它和在其外部出现的处理指令、注释等作为文档实体的子节点，根元素本身和其内部的子元素也是一棵树。

### XML样例？

|  |
| --- |
| **<?xml version="1.0" encoding="UTF-8"?>**  **<students>**  **<student1 id="001">**  **<微信公众号>@残缺的孤独</微信公众号>**  **<学号>20140101</学号>**  **<地址>北京海淀区</地址>**  **<座右铭>要么强大，要么听话</座右铭>**  **</student1>**  **<student2 id="002">**  **<新浪微博>@残缺的孤独</新浪微博>**  **<学号>20140102</学号>**  **<地址>北京朝阳区</地址>**  **<座右铭>在哭泣中学会坚强</座右铭>**  **</student2>**  **</students>** |

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

[xml文件](https://www.baidu.com/s?wd=xml%E6%96%87%E4%BB%B6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YdnjK9rjbvuWfLPAN9Ph7W0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHR4rjR1rHc4nWTYP10krj0Y)头部要写的话，说明了xml的版本和编码，utf-8一般是[网络传输](https://www.baidu.com/s?wd=%E7%BD%91%E7%BB%9C%E4%BC%A0%E8%BE%93&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YdnjK9rjbvuWfLPAN9Ph7W0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHR4rjR1rHc4nWTYP10krj0Y" \t "_blank)用的编码

### XML解析方式？

Dom4j、Sax、Pull

### Dom4j与Sax区别

 dom4j不适合大文件的解析，因为它是一下子将文件加载到内存中，所以有可能出现内存溢出，sax是基于事件来对xml进行解析的，所以他可以解析大文件的xml，也正是因为如此，所以dom4j可以对xml进行灵活的增删改查和导航，而sax没有这么强的灵活性，所以sax经常是用来解析大型xml文件，而要对xml文件进行一些灵活（crud）操作就用dom4j。

### 使用dom4j解析xml

解析XML过程是通过获取Document对象，然后继续获取各个节点以及属性等操作，因此获取Document对象是第一步，大体说来，有三种方式：

1.自己创建Document对象

|  |
| --- |
| **Document document = DocumentHelper.*createDocument*();**  **Element root = document.addElement("students");** |

其中students是根节点，可以继续添加其他节点等操作。

2.自己创建Document对象

|  |
| --- |
| **// 创建SAXReader对象**  **SAXReader reader = new SAXReader();**  **// 读取文件 转换成Document**  **Document document = reader.read(new File("XXXX.xml"));** |

3.读取XML文本内容获取Document对象

|  |
| --- |
| **String xmlStr = "<students>......</students>";**  **Document document = DocumentHelper.*parseText*(xmlStr);** |

### 解析xml代码

Xml配置:

|  |
| --- |
| **<?xml version="1.0" encoding="UTF-8"?>**  **<students>**  **<student1 id="001">**  **<微信公众号>每特学院</微信公众号>**  **<学号>20140101</学号>**  **<地址>北京海淀区</地址>**  **<座右铭>要么强大，要么听话</座右铭>**  **</student1>**  **<student2 id="002">**  **<新浪微博>蚂蚁课堂</新浪微博>**  **<学号>20140102</学号>**  **<地址>北京朝阳区</地址>**  **<座右铭>在哭泣中学会坚强</座右铭>**  **</student2>**  **</students>** |

Java代码

|  |
| --- |
| **public static void main(String[] args) throws DocumentException {**  **SAXReader saxReader = new SAXReader();**  **Document read = saxReader.read(new File("E://work//spring-ioc//src//main//resources//stu.xml"));**  **// 获取根节点**  **Element rootElement = read.getRootElement();**  ***getNodes*(rootElement);**  **}**  **static public void getNodes(Element rootElement) {**  **System.*out*.println("当前节点名称:" + rootElement.getName());**  **// 获取属性ID**  **List<Attribute> attributes = rootElement.attributes();**  **for (Attribute attribute : attributes) {**  **System.*out*.println("属性:" + attribute.getName() + "---" + attribute.getText());**  **}**  **if (!rootElement.getTextTrim().equals("")) {**  **System.*out*.println(rootElement.getName() + "--" + rootElement.getText());**  **}**  **// 使用迭代器遍历**  **Iterator<Element> elementIterator = rootElement.elementIterator();**  **while (elementIterator.hasNext()) {**  **Element next = elementIterator.next();**  ***getNodes*(next);**  **}**  **}** |

注意:

**this.getClass().getClassLoader().getResourceAsStream(xmlPath) 获取当前项目路径xmlfsfs**

## XML与JSON区别

Xml是重量级数据交换格式，占宽带比较大。

JSON是轻量级交换格式，xml占宽带小。

所有很多互联网公司都会使用json作为数据交换格式

很多银行项目，大多数还是在使用xml。

# 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>** |