## 前提。具有配置文件ApplicationContext.xml

## 1.接口类

|  |
| --- |
| **public** **interface** IOAXmlDealFactory {  **public** Result parseXML(OAMessageInfo requestEAMessageInfo, **byte**[] xml, String managercode);    **public** OAXMLHeadVO parseHead(**byte**[] xml, String managercode);  } |

## 2.实现类 注意service中的注释，其实可有可无，有的话，是注解内容起了作用，大写，没得话，就是类名起了作用

|  |
| --- |
| @Service("CENTERXmlDealFactory")  **public** **class** CenterXmlDealFactory **implements** IOAXmlDealFactory {    @Resource(type = OAXmlDealHandleOne.**class**)  OAXmlDealHandle oAXmlDealHandle;  @Override  **public** Result parseXML(OAMessageInfo requestEAMessageInfo, **byte**[] xml, String managercode) {  // **TODO** Auto-generated method stub  **return** oAXmlDealHandle.parseXML(requestEAMessageInfo, xml, managercode);  }  @Override  **public** OAXMLHeadVO parseHead(**byte**[] xml, String managercode){  // **TODO** Auto-generated method stub  **return** oAXmlDealHandle.parseHead(xml,managercode);  }} |

### 2.2 实现类2

|  |
| --- |
| @Service  **public** **class** GUANGDONGXmlDealFactory **implements** IOAXmlDealFactory{  @Resource(type=OAXmlDealHandleTwo.**class**)  OAXmlDealHandle oAXmlDealHandle;  @Override  **public** Result parseXML(OAMessageInfo requestEAMessageInfo, **byte**[] xml,  String managercode) {    **return** oAXmlDealHandle.parseXML(requestEAMessageInfo, xml, managercode);  }  @Override  **public** OAXMLHeadVO parseHead(**byte**[] xml, String managercode) {    **return** oAXmlDealHandle.parseHead(xml, managercode);  }  } |

## 3.利用配置接口类实现上述功能

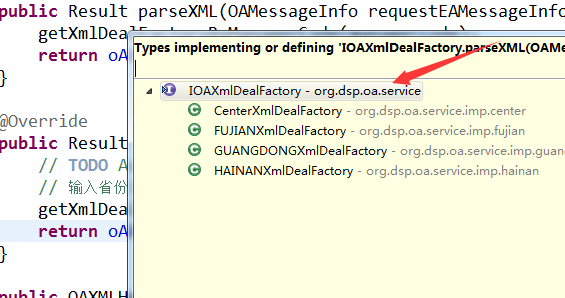
|  |
| --- |
| /\*\*  \* Copyright:http://www.chinalife.com.cn All rights reserved.  \* **@author**:huangshuren  \* **@date**:2012-8-28  \* **@Description**:  \* 保留applicationContext实例的引用  \* 实现 ApplicationContextAware，可设置context上下文用于应用中使用Spring  \* 实现BeanPostProcessor，可改变Bean的加载顺序，优先加载实现该接口的类  \*/  @Component  **public** **class** ApplicationContextHolder **implements** ApplicationContextAware,BeanPostProcessor {  /\*\*  \* Spring上下文  \*/  **private** **static** ApplicationContext *applicationContext*;    /\*\*  \* **@return**  \* **@Description**:  \* 获取 ApplicationContext  \*/  **public** **static** ApplicationContext getApplicationContext() {  **return** *applicationContext*;  }    /\*\*  \* **@param** <T>  \* **@param** name  \* **@return**  \* **@Description**:  \* 获取Bean  \*/  @SuppressWarnings("unchecked")  **public** **static** <T> T getBean(String name) {  **return** (T) *applicationContext*.getBean(name);  }    /\*\*  \* **@param** <T>  \* **@param** name  \* **@return**  \* **@Description**:  \* 获取Bean  \*/  **public** **static** <T> T getBean(Class<T> clazz) {  **return** (T) *applicationContext*.getBean(clazz);  }    /\*\*  \* **@param** <T>  \* **@param** name  \* **@return**  \* **@Description**:  \* 获取Bean  \*/  **public** **static** <T> T getBean(Class<T> clazz, String name) {  **return** (T) *applicationContext*.getBean(name, clazz);  }    @Override  **public** **void** setApplicationContext(ApplicationContext context) **throws** BeansException {  *applicationContext* = context;  }  @Override  **public** Object postProcessAfterInitialization(Object arg0, String arg1)  **throws** BeansException {  // **TODO** Auto-generated method stub  **return** arg0;  }  @Override  **public** Object postProcessBeforeInitialization(Object arg0, String arg1)  **throws** BeansException {  // **TODO** Auto-generated method stub  **return** arg0;  }  } |

## 4.使用上面的配置接口类，并给定接口所最终选择的实现类

|  |
| --- |
| @Service  **public** **class** OAInfoServiceImp **implements** IOAInfoService {    IOAXmlDealFactory oAXmlDealFactory;    /\*\*  \* 根据管理人编码获得  \* **@param** managercode  \*/  **private** **void** getXmlDealFactoryByManagerCode(String managercode){  **try**{  **if**(managercode!=**null** && !"".equals(managercode)){  //取得省份信息 广西和广东一样  String name=getProvinceName(managercode);  oAXmlDealFactory=ApplicationContextHolder.*getBean*(name+"XmlDealFactory");  }  }**catch**(Exception ex){  ex.printStackTrace();  **throw** **new** BusinessException("处理失败");  }    } |

## 5.使用某个实现类

|  |
| --- |
| **public** OAXMLHeadVO parseHead(**byte**[] xml, String managercode) **throws** BusinessException{  getXmlDealFactoryByManagerCode(managercode);  **return** oAXmlDealFactory.parseHead(xml, managercode);  } |



# 第二种抽象类（type），这里以举例OAXmlDealHandle 为一个抽象类

## 1、Resourse 直接写入注解 type，这样即使 到了

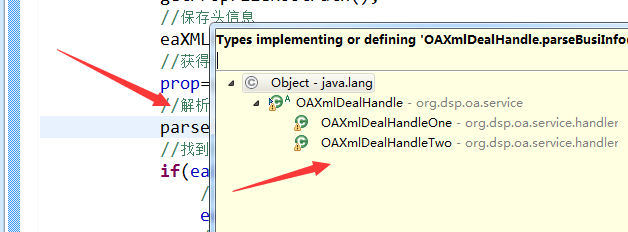
|  |
| --- |
| @Service  **public** **class** FUJIANXmlDealFactory **implements** IOAXmlDealFactory{  @Resource(type = OAXmlDealHandleOne.**class**)  OAXmlDealHandle oAXmlDealHandle;  @Override  **public** Result parseXML(OAMessageInfo requestEAMessageInfo, **byte**[] xml, String managercode) {  // **TODO** Auto-generated method stub  //子类中没有该方法，所以还是该抽象类中的自有方法  **return** oAXmlDealHandle.parseXML(requestEAMessageInfo, xml, managercode);  } |

## 2、抽象类中的带重写方法，注意抽象类头部不需要写注解

|  |
| --- |
| **public** **abstract** **class** OAXmlDealHandle {  /\*\*  \* 解析业务信息  \* **@param** eaMessage  \* **@throws** BusinessException  \*/  **protected** **abstract** **void** parseBusiInfo(OAMessageInfo eaMessage); |

## 3、抽象类内部自身调用子类中重写的方法。因为前面的类周玲使用了@Resource(type = OAXmlDealHandleOne.class)会自动识别子类

parseBusiInfo(eaMessage);



## 4、子类方法

|  |
| --- |
| @Service  **public** **class** OAXmlDealHandleOne **extends** OAXmlDealHandle {  @Override  **protected** **void** parseBusiInfo(OAMessageInfo eaMessage) {  Iterator<Element> iteEle=rootEle.elementIterator();  **while**(iteEle.hasNext()){  Element ele=iteEle.next();  String tagName=ele.getName();  **if**("pensioninfo".equals(tagName)){  dealBusiInfo(ele,eaMessage);  }  }    } |

# 第三种 抽象类 name

## 1、sevice 写入名字@Component("SendJIANGSU")

|  |
| --- |
| @Component("SendJIANGSU")  **public** **class** SendMessageToJIANGSU **extends** WsSender<client.transfer.SendMessage>{ |

## 2、使用@Resource(name="SendJIANGSU")

|  |
| --- |
| @Resource(name="SendJIANGSU")  **private** WsSender<client.transfer.SendMessage> sender; |

# 第四种（只有一个孩子），直接利用 ClassPathXmlApplicationContext

## 1、直接获取接口类（写上接口的名字，或者实现类名字就可以）

|  |
| --- |
| **public** **final** **static** ClassPathXmlApplicationContext *context* = **new** ClassPathXmlApplicationContext(  **new** String[] { "applicationContext-dsp.xml" });  **public** **final** **static** ISequenceDao *sequenceDao*=(ISequenceDao)*context*.getBean(ISequenceDao.**class**); |

## 2、接口如下

|  |
| --- |
| **public** **interface** ISequenceDao {  /\*\*  \* **@return**  \* **@Description**:  \* 获取批次号  \*/  **public** **abstract** String getBatchNo();  /\*\*  \* **@return**  \* **@Description**:  \* 获取交易序列号  \*/  **public** **abstract** String getAppseriono();    /\*\* |

## 3、子类如下

|  |
| --- |
| @Component  **public** **class** SequenceDaoImp **implements** ISequenceDao {    @Autowired  **private** JdbcTemplate jdbcTemplate;    @Override  **public** String getBatchNo() {  **return** get("batchno\_sequence", String.**class**);  } |

# 第五种，直接通过service标注的名字获取

## 1、建立service类 （使用spring自带的读取，一定要再service中添加名字，否则不能够成功，除非使用之前的Applicaionhoder才可以读取类名）

|  |
| --- |
| /\*\*  \* **@author** 作者 HealerJean  \* **@version** 创建时间：2017年9月8日 上午9:43:06  \* 类说明  \*/  @Service("AspectStyleService")  **public** **class** AspectStyleService {    **public** **void** save(String name) {  System.*out*.println("save方法");  }    } |

## 2、、取得配置文件获取bean，AspectStyleServiceName为上面的名字

|  |
| --- |
| /\*\*  \* **@author** 作者 HealerJean  \* **@version** 创建时间：2017年9月8日 上午9:33:38  \* 类说明  \*/  **public** **class** AspectStypeTest {  **public** **static** **void** main(String[] args) {    ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext-dsp.xml" );  AspectStyleService aspectStyleService = (AspectStyleService)context.getBean("AspectStyleServiceName");  aspectStyleService.save("HealerJean");  }  } |

# 第六种、@service中没有name参数也可以获取name，首字母小写即可

|  |
| --- |
| @Service  **public** **class** QueueSender {  @Autowired  **private** JmsTemplate jmsTemplate = **null**;  **public** **static** **void** main(String[] args)**throws** Exception {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext("spring-queue-applicationContext.xml");  QueueSender queueSender = (QueueSender)ctx.getBean("queueSender");  queueSender.jmsTemplate.send(**new** MessageCreator() {  **public** Message createMessage(Session session) **throws** JMSException {  TextMessage msg = session.createTextMessage("Spring msg（ spring 创建生产者成功）");  System.*out*.println("spring 创建生产者成功"); return msg; }  });  }  } |

# 第7中、@Qualifier 用来指定名字

|  |
| --- |
| @Autowired @Qualifier("co\_RelationService")  **private** Co\_RelationServiceImp co\_RelationService; |

|  |
| --- |
| @Service("co\_RelationService")  **public** **class** Co\_RelationServiceImp **implements** Co\_RelationService { |

# 第8种 @Primary优先使用

## 意思是在众多相同的bean中，优先使用用@Primary注解的bean.

@Primary  
@Bean(name = {"admoreRedisTemplate","redisTemplate"})  
public RedisTemplate<Object,Object> commonRedisTemplate(@Qualifier("admoreRedisConnectionFactory")RedisConnectionFactory redisConnectionFactory) throws UnknownHostException {  
 return redisTemplate(redisConnectionFactory);  
}

<https://www.cnblogs.com/1540340840qls/p/6970692.html>