**软件设计文档**

**研发技术规范手册**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 编 写 |  | | | | 时 间 | | 2016年08月25日 | |
| 审 核 |  | | | | 时 间 | |  | |
| 审 批 | （职务） | | | | 时 间 | | 年月日 | |
| 文档管理 | 页码 | 共页 | 修订次数 |  | | 版本 | |  |
| 编号 |  | | | | | | |

**版权所有**

文档中的全部内容属所有，

未经允许，不可全部或部分发表、复制、使用于任何目的。

**文档修订摘要**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **日期** | **修订号** | **描述** | **著者** | **审阅者** | **批准人/日期** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目 录

[AsiaInfo 软件设计文档 1](#_Toc460619715)

[产品研发技术规范手册 1](#_Toc460619716)

[目 录 1](#_Toc460619717)

[第1章 引言 1](#_Toc460619718)

[1.1 目的 1](#_Toc460619719)

[1.2 阅读对象 1](#_Toc460619720)

[1.3 参考资料 1](#_Toc460619721)

[1.4 名词解释 1](#_Toc460619722)

[第2章 编码规范及工程结构设计 1](#_Toc460619723)

[2.1 工程编码规范 1](#_Toc460619724)

[4.3.1 Web工程编码规范 1](#_Toc460619725)

[4.3.2 后场服务工程编码规范 11](#_Toc460619726)

[2.2 WEB工程结构 13](#_Toc460619727)

[2.3 后场服务工程结构 14](#_Toc460619728)

[第3章 基础设施SDK 16](#_Toc460619729)

[3.1 分布式配置服务CCS 17](#_Toc460619730)

[3.2 分布式缓存服务MCS 18](#_Toc460619731)

[3.3 分布式消息服务MDS 19](#_Toc460619732)

[3.4 分布式文档存储服务DSS 20](#_Toc460619733)

[第4章 SDK 21](#_Toc460619734)

[4.1 opt-sdk配置文件 21](#_Toc460619735)

[4.2 opt-sdk引用方法 21](#_Toc460619736)

[4.3 配置中心使用方法 21](#_Toc460619737)

[4.3.1 公共配置路径 22](#_Toc460619738)

[4.3.2 PaaS服务与密码映射 22](#_Toc460619739)

[4.3.3 数据源配置 23](#_Toc460619740)

[4.3.4 缓存MCS配置 25](#_Toc460619741)

[4.3.5 消息MDS配置 26](#_Toc460619742)

[4.3.6 文档存储DSS配置 27](#_Toc460619743)

[4.4 缓存中心MCS使用方法 28](#_Toc460619744)

[4.5 消息中心MDS使用方法 29](#_Toc460619745)

[4.6 文档存储DSS使用方法 31](#_Toc460619746)

[4.7 序列使用方法 31](#_Toc460619747)

[4.8 SDK工具包 33](#_Toc460619748)

[4.6.1 中心产品DUBBO服务启动 33](#_Toc460619749)

[4.6.2 DUBBO消费端接口获取 33](#_Toc460619750)

[4.6.3 常用工具类 34](#_Toc460619751)

[第5章 服务开发 34](#_Toc460619752)

[5.1 服务开发 35](#_Toc460619753)

[5.1.1 服务包结构 35](#_Toc460619754)

[5.1.2 服务接口签名要求 35](#_Toc460619755)

[5.1.3 服务参数要求 36](#_Toc460619756)

[5.1.4 服务实现异常处理 37](#_Toc460619757)

[5.1.5 将API发布到在线网站 39](#_Toc460619758)

[第6章 基于注解的校验 43](#_Toc460619759)

[6.1 依赖的组件 43](#_Toc460619760)

[~~6.2~~ ~~自定义校验规则~~ 43](#_Toc460619761)

[~~6.2.1~~ ~~定义规则的Annotation~~ 43](#_Toc460619762)

[~~6.2.2~~ ~~定义规则的Validator~~ 44](#_Toc460619763)

[~~6.2.1~~ ~~如何使用自定义规则~~ 45](#_Toc460619764)

[6.3 校验规则集合 47](#_Toc460619765)

[6.4 说明 48](#_Toc460619766)

[第7章 核心工程构建过程 48](#_Toc460619767)

[7.1 主工程build脚本 48](#_Toc460619768)

[7.2 API工程build脚本 52](#_Toc460619769)

[7.3 Core工程build脚本 56](#_Toc460619770)

[7.4 DubboAPI构建发布到NEXUS 56](#_Toc460619771)

[7.5 DubboAPI构建发布到服务在线 57](#_Toc460619772)

[7.6 Project构建输出 58](#_Toc460619773)

[7.7 如何让版本支持不同环境？ 58](#_Toc460619774)

[7.7.1 服务端工程处理方法 59](#_Toc460619775)

[7.7.2 WEB工程处理方法 60](#_Toc460619776)

[第8章 打包部署 61](#_Toc460619777)

[8.1 开发环境部署目录规范 62](#_Toc460619778)

[8.2 从git获取最新代码 63](#_Toc460619779)

[8.3 执行编译 64](#_Toc460619780)

[8.4 从编译目录拷贝到部署目录 64](#_Toc460619781)

[8.5 启动dubbo服务 65](#_Toc460619782)

[8.6 一键部署(onekey.sh) 67](#_Toc460619783)

[第9章 服务在线测试 68](#_Toc460619784)

[9.1 自动化的服务文档 68](#_Toc460619785)

[9.2 服务在线测试 69](#_Toc460619786)

[第10章 Rest服务改造升级 72](#_Toc460619787)

[10.1 Rest服务改造方法 72](#_Toc460619788)

[10.2 Dubbo方式调用Rest服务 74](#_Toc460619789)

[10.3 HttpClient方式调用Rest服务 74](#_Toc460619790)

[10.4 Jar包冲突问题排除示例 75](#_Toc460619791)

[第11章 opt-sdk组件SDK模式支撑 76](#_Toc460619792)

[11.1 opt-sdk两种模式支撑 77](#_Toc460619793)

[11.2 opt-sdk ipaas组件service模式 77](#_Toc460619794)

[11.3 opt-sdk ipaas组件sdk模式 77](#_Toc460619795)

[11.4 opt-sdk ipaas组件两种模式切换 77](#_Toc460619796)

[11.5 CCS SDK模式配置信息 78](#_Toc460619797)

[11.6 MCS SDK模式配置信息 79](#_Toc460619798)

[11.7 MDS SDK模式 Kafka部署事项 79](#_Toc460619799)

[11.8 MDS SDK模式 Sender配置信息 79](#_Toc460619800)

[11.9 MDS SDK模式 Consumer配置信息 80](#_Toc460619801)

[11.10 DSS SDK模式配置信息 81](#_Toc460619802)

[第12章 Docker环境下工程规范 81](#_Toc460619803)

[12.1 Shell文件格式要求 81](#_Toc460619804)

[12.1.1 Shell文件格式 81](#_Toc460619805)

[12.1.2 Shell文件格式调整方法 84](#_Toc460619806)

[12.2 后场工程规范 86](#_Toc460619807)

[12.2.1 后场父工程规范 86](#_Toc460619808)

[12.2.2 配置文件备份规范 88](#_Toc460619809)

[12.2.3 后场镜像打包步骤 90](#_Toc460619810)

[12.3 前端web工程规范 91](#_Toc460619811)

[12.3.1 web工程规范 91](#_Toc460619812)

[12.3.2 配置文件规范 92](#_Toc460619813)

[12.3.3 前端web打包步骤 94](#_Toc460619814)

[第13章 前端Web UI编写规范 95](#_Toc460619815)

# 引言

## 目的

本文档是基于多年产品研发过程中产生技术体系、标准、约定而编写的技术规范手册，旨在指导原有产品团队进行改进，促进新的产品团队能更好的开展研发工作。

## 阅读对象

阅读对象：软件研发工程师，测试工程师，运维工程师

## 参考资料

暂无

## 名词解释

# 编码规范及工程结构设计

## 工程编码规范

### Web工程编码规范

1）web工程包定义规范：(以商品中心为例)

com.x.centra.product.web.contants 常量包

com.x.centra.product.web.controller.XXX XXX模块对应的controller包

com.x.centra.product.web.filter 过滤器包

com.x.centra.product.web.model 模型包

com.x.centra.product.web.util 工具包

2）jsp及js层次规范：

/webapp/jsp/XXX XXX模块的jsp文件

/webapp/resources/spm\_modules/app/jsp /XXX XXX模块的js文件

3）文件命名规范

|  |  |  |  |
| --- | --- | --- | --- |
| 描述 | jsp文件 | seajs文件 | SpringMVC |
| 查询列表页 | xxxList.jsp | xxxList.js | /xxx/list |
| 新增和修改复用的表单 | xxxForm.jsp | xxxForm.js | 页面：  /xxx/toadd  /xxx/toUpdate  ajax数据处理：  /xxx/save |
| 新增表单 | xxxAdd.jsp | xxxAdd.js | 页面：  /xxx/toadd  ajax数据处理：  /xxx/save |
| 修改表单 | xxxUpdate.jsp | xxxUpdate.js | 页面：  /xxx/toUpdate  ajax数据处理：  /xxx/save |
| 详情页面 | xxxDetail.jsp | xxxDetail.js | 页面：  /xxx/toDetail  ajax数据处理：  /xxx/detail |
| 过渡—修改成功 | xxxSuccess.jsp | xxxSuccess.js | 页面：  /xxx/toSucess  ajax数据处理：  /xxx/success |
| 过渡—修改失败 | xxxFailure.jsp | xxxFailure.js | 页面：  /xxx/toFailure  ajax数据处理：  /xxx/failure |
| 步骤1 | xxxStep1.jsp | xxxStep1.js | 页面：  /xxx/toStep1  ajax数据处理：  /xxx/Step1 |
| 步骤2 | xxxStep2.jsp | xxxStep2.js | 页面：  /xxx/toStep2  ajax数据处理：  /xxx/Step2 |
| 步骤3 | xxxStep3.jsp | xxxStep3.js | 页面：  /xxx/toStep3  ajax数据处理：  /xxx/Step3 |

4)SeaJS、AraleJS方法名及写法规范

目前前端JS开发基于AraleJS，而AraleJS依赖于SeaJS进行模块化管理。

查询列表的js样例：

|  |
| --- |
| define('app/jsp/standardfee/standPostage', **function** (require, exports, module) {  'use strict';  **var** $=require('jquery'),  Widget = require('arale-widget/1.2.0/widget'),  Dialog = require("artDialog/src/dialog"),  AjaxController = require('opt-ajax/1.0.0/index');    require("jsviews/jsrender");  require("jsviews/jsviews");  require("twbs-pagination/jquery.twbsPagination.min");  require("opt-paging/aiopt.pagination");  **var** SendMessageUtil = require("app/util/sendMessage");    //实例化AJAX控制处理对象  // var ajaxController = new AjaxController();  //定义页面组件类  **var** StandPostagePager = Widget.extend({    Implements:SendMessageUtil,  //属性，使用时由类的构造函数传入  attrs: {  },  Statics: {  DEFAULT\_PAGE\_SIZE: 10  },  //事件代理  events: {  //查询  "click #BTN\_SEARCH":"\_searchBtnClick",  "click #PHONE\_IDENTIFY":"\_getPhoneVerifyCode",  "click #CLOSE\_POP":"\_closeMessage"  },  //重写父类  setup: **function** () {  StandPostagePager.superclass.setup.call(**this**);  **this**.\_bindEvents();  **this**.\_setMenu();  //初始化执行搜索  **this**.\_searchBtnClick();  **this**.\_bindDeleteStandFee();  **this**.\_showSetValidDiv();  $("#PHONE\_IDENTIFY").removeAttr("disabled"); //移除disabled属性  **this**.\_getServiceType();  **this**.\_getStandardStatus();    **this**.\_bindSendMessage();  },  \_setMenu:**function** () {  setBreadCrumb("标准资费管理", "标准资费查询");  $("#mnu\_bmc\_config").addClass("current");  },  \_bindSendMessage:**function**(){  **var** \_this = **this**;  **var** func = **function**(){  $.ajax({  type : "POST",  url : \_base + "/standardFee/updateStatus",  data: {  standardId : $("input[name='targetId']").val(),  status : $("input[name='feeStatus']:checked").val()  },  processing: **true**,  message : "正在处理中，请稍候...",  success : **function**(data) {  $("#tcc\_div").hide();  **var** tip = Dialog({  title: '提示',  width: '200px',  height: '50px',  content: data.statusInfo  }).show();    $("#PHONE\_IDENTIFY").removeAttr("disabled"); //移除disabled属性  $('#PHONE\_IDENTIFY').val('获取验证码');  $('#phoneVerifyCode').val('');    window.setTimeout(**function**(){  tip.close();  \_this.\_searchBtnClick();  } ,1000);  }  });  };  $('#BTN\_UPDATE\_STATUS').bind("click", **function**(){  \_this.\_verifyPhoneCode(func);  });  },  \_bindDeleteStandFee:**function**(){  **var** pThis = **this**;  $(".del-button").bind("click", **function**(){  **var** \_this = **this**;  **var** status = $(\_this).attr('status');  **if**(status=='ACTIVE'){  **var** da = Dialog({  content:"生效状态下的产品无法删除",  okValue:"确定",  ok:**function** () {  **this**.close();  }  });  da.showModal();  **return**;  }  **var** d = Dialog({  content:"确定删除？",  ok:**function**(){  $.ajax({  method: "POST",  url: \_base+"/standardFee/delete",  dataType: "json",  showWait: **true**,  async: **false**,  data: {  "standardId": $(\_this).attr('standardId'),  "comments": $(\_this).attr('comments')  },  message: "正在加载数据..",  success: **function** (data) {  **var** d = Dialog({  content: data.statusInfo,  ok:**function**(){  **this**.close();  }  });  d.showModal();  pThis.\_searchBtnClick();  }  });  **this**.close;  },  okValue:"确定",  cancel:**function**(){  **this**.close;  },  cancelValue:"取消"  });  d.showModal();  });  },  \_bindEditStandFee:**function**(){  $(".edit-button").bind("click", **function**(){  **var** \_this = **this**;  **var** status = $(\_this).attr('status');  **if**(status=='ACTIVE'){  **var** da = Dialog({  content:"生效状态下的产品无法编辑",  okValue:"确定",  ok:**function** () {  **this**.close();  }  });  da.showModal();  **return**;  }  window.location.href = \_base+"/standardFee/toUpdate?standardId="+$(\_this).attr('standardId');  });  },  \_bindRelateStandFee:**function**(){  $(".relate-button").bind("click", **function**(){  **var** \_this = **this**;  **var** status = $(\_this).attr('status');  **if**(status=='ACTIVE'){  **var** da = Dialog({  content:"生效状态下的产品无法关联详单科目",  okValue:"确定",  ok:**function** () {  **this**.close();  }  });  da.showModal();  **return**;  }  window.location.href = \_base+"/standardFee/toRelateSubject?standardId="+$(\_this).attr('standardId');  });  },  \_showSetValidDiv:**function**(){  $(".sp-setvalid").bind("click", **function**(){    **var** status = $(**this**).attr('status');  $("input[name='targetId']").val($(**this**).attr('sid'));  $("#tcc\_div input:radio").removeAttr("checked");  $("#valid").removeAttr("disabled");  $("#invalid").removeAttr("disabled");  **if**(status=='ACTIVE'){  $("#invalid").prop("checked","checked");  $("#valid").attr("disabled","disabled");  }**else**{  $("#valid").prop("checked","checked");  $("#invalid").attr("disabled","disabled");  }  $('#tcc\_div').show();  });  },  \_searchBtnClick: **function**(){  **var** \_this = **this**;  **var** url = \_base+"/standardFee/getList";  $("#pagination-ul").runnerPagination({  url: url,  method: "POST",  dataType: "json",  processing: **true**,  data : $('#queryForm').serializeArray(),  pageSize: StandPostagePager.DEFAULT\_PAGE\_SIZE,  visiblePages:5,  message: "正在为您查询数据..",  render: **function** (data) {  **if**(data != **null** && data != 'undefined' && data.length>0){  **var** template = $.templates("#standardListTemple");  **var** htmlOutput = template.render(data);  $("#standardData").html(htmlOutput);  \_this.\_bindDeleteStandFee();  \_this.\_bindEditStandFee();  \_this.\_bindRelateStandFee();  \_this.\_showSetValidDiv();  }**else**{  $("#standardData").html("没有搜索到相关信息");  }  }  });  },    \_bindEvents: **function**(){  **var** \_this = **this**;    },  \_getServiceType: **function**() {  **var** \_this = **this**;  **this**.setSelectValue(\_base + '/param/getServiceType', **function**(data){  **var** json = eval(data);  $  .each(  json,  **function**(index, item) {  // 循环获取数据  **var** paramName = json[index].paramName;  **var** paramCode = json[index].paramCode;  $("#serviceType")  .append('<option cid="'+json[index].id+'" value="'+paramCode+'">'+paramName+'</option>');  });  $("#serviceType")  .append("<label id='accesstype\_error'></label>");  \_this.\_getServiceDetail();  });  },  \_getServiceDetail: **function**() {  **var** serverId = $("#serviceType").find("option:selected").attr('cid');  **if**(serverId==**undefined** || serverId==''){  $("#serviceTypeDetail").val("");  $("#serviceTypeDetail").attr("disabled",**true**);  }**else**{  $("#serviceTypeDetail").removeAttr("disabled",**true**);  **this**.setSelectValue(\_base + '/param/getServiceDetail?serverId='+serverId, **function**(data){  $("#serviceTypeDetail").html('<option value="">请选择</option>');  **var** json = eval(data);  $  .each(  json,  **function**(index, item) {  // 循环获取数据  **var** paramName = json[index].paramName;  **var** paramCode = json[index].paramCode;  $("#serviceTypeDetail")  .append('<option value="'+paramCode+'">'+paramName+'</option>');  });  $("#serviceTypeDetail")  .append("<label id='accesstype\_error'></label>");  });  }  },  \_getStandardStatus: **function**() {  **this**.setSelectValue(\_base + '/param/getStandardStatus', **function**(data){  **var** json = eval(data);  $  .each(  json,  **function**(index, item) {  // 循环获取数据  **var** paramName = json[index].paramName;  **var** paramCode = json[index].paramCode;  $("#priceState")  .append('<option value="'+paramCode+'">'+paramName+'</option>');  });  $("#priceState")  .append("<label id='accesstype\_error'></label>");  });  },  setSelectValue: **function**(url, func){  $  .ajax({  url : url,  type : "post",  async : **true**,  dataType : "html",  timeout : "10000",  error : **function**() {  alert("服务加载出错");  },  success : **function**(data) {  func(data);  }  });  }    });    module.exports = StandPostagePager  }); |

### 后场服务工程编码规范

1）每个dubbo接口定义都要抛出BusinessException和SystemException

2）接口中的方法名按驼峰命名法，如registerByPhone

3）方法中的请求参数名称都以Request结尾，如注册的请求参数命名为PhoneRegisterRequest。另外，如果接口中的请求参数中包含tenantId，则请求参数需继承BaseInfo，否则不需继承，但需要实现Serializable接口

4）方法中的返回值参数名称都以Reponse结尾，如注册的返回参数命名为PhoneRegisterResponse，PhoneRegisterResponse必须继承BaseResponse。另外，如果方法没有返回值，则直接返回BaseResponse。

5）api工程接口包定义规范：（以UAC工程为例）

com.x.centra.uac.api.XXX.interfaces 接口包

com.x.centra.uac.api.XXX.param 接口参数包

com.x.centra.uac.api.XXX 包下面需要有version.ini文件

XXX示例：注册包：register,登录：sso，账户中心：account，账户安全：security

6）core工程包定义规范：

com.x.centra.uac.api.XXX.impl 接口实现包

com.x.centra.uac.constants 常量包

com.x.centra.uac.dao.mapper 数据库映射

com.x.centra.uac.service.atom.interfaces 原子服务接口

com.x.centra.uac.service.atom.impl原子服务接口

com.x.centra.uac.service.busi.interfaces 业务服务接口

com.x.centra.uac.service.busi.impl 业务服务接口（带事务）

com.x.centra.uac.util 工具类

7）配置文件规范

src/main/resources/context/core-context.xml 核心bean文件

src/main/resources/dubbo/dubbo.properties dubbo属性文件

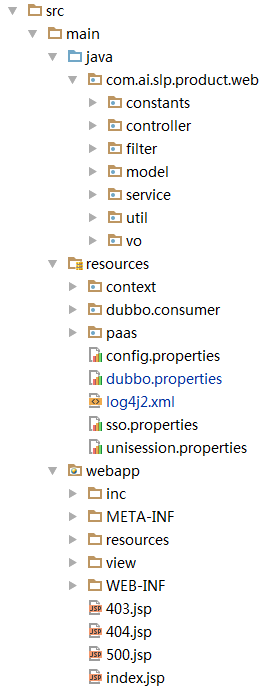
src/main/resources/dubbo/provider/dubbo-provider.xml dubbo provider xml文件

src/main/resources/mybatis/mapper/XXX mybatis映射文件

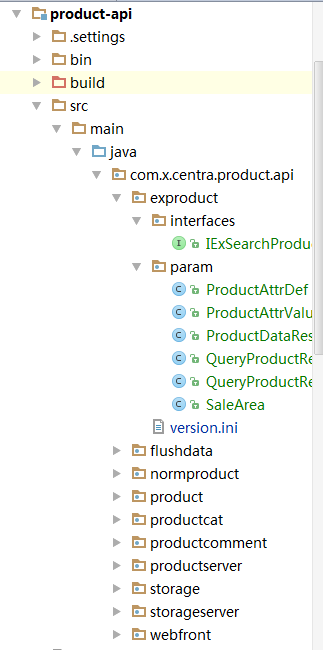
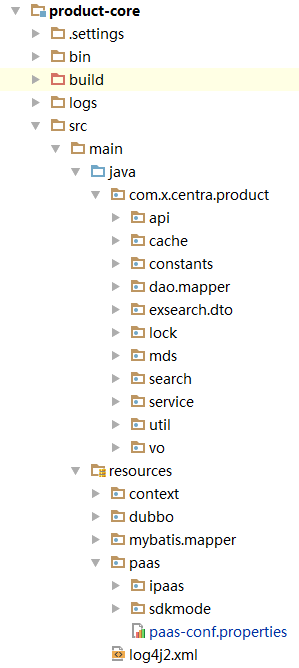
src/main/resources/paas/paas-conf.properties paas配置文件

src/main/resources/log4j2.xml 日志配置文件

## WEB工程结构



## 后场服务工程结构

# SDK

## sdk配置文件

sdk约定读取classpath下paas/paas-conf.properties文件，每个使用sdk的工程都应提供该配置文件。文件内容示例配置如下：

|  |
| --- |
| *# 是否使用远程配置文件 # true(默认)会从远程获取配置 false则直接获取本地配置* **enable.remote.conf**=**true** *# 配置服务器的 HOST,用逗号分隔 127.0.0.1:8000,127.0.0.1:8000* **conf\_server\_host**=**http://172.16.13.121:5002** *# 版本, 请采用 X\_X\_X\_X 格式* **version**=**1\_0\_0\_0** *# APP 请采用 产品线\_服务名 格式* **app**=**sdk** *# 环境* **env**=**rd** *# debug* **debug**=**true** *# 忽略哪些分布式配置，用逗号分隔* **ignore**= *# 获取远程配置 重试次数，默认是3次* **conf\_server\_url\_retry\_times**=**1** *# 获取远程配置 重试时休眠时间，默认是5秒* **conf\_server\_url\_retry\_sleep\_seconds**=**1** |

## sdk引用方法

工程都以gradle形式引用jar包，opt-sdk的引用方法：

|  |
| --- |
| compile 'com.x: sdk:**1.0-SNAPSHOT**' |

## 配置中心使用方法

地址 <http://172.16.13.121:5002/login.html>

admin/admin

### 公共配置路径

目前规划的公共配置路径如下：

|  |  |  |
| --- | --- | --- |
| **序号** | **配置项** | **配置路径** |
| 1 | paas服务密码配置 | /com/x/paas-service-pwd-mapped |
| 2 | 缓存命名空间 | /com/x/paas-cachens-mcs-mapped |
| 3 | 消息服务命名空间 | /com/x/paas-mdsns-mds-mapped |
| 4 | 消息与topic映射 | /com/x/paas-mds-topic-mapped |
| 5 | 文档服务命名空间 | /com/x/paas-dssns-dss-mapped |
| 6 | 数据源配置 | /com/x/db-conf |

### PaaS服务与密码映射

PaaS服务密码映射配置示例：

|  |
| --- |
| 配置路径：/com/x/paas-service-pwd-mapped |
| 配置值示例：  {  "MCS001": "123456",//其中，MCS001为缓存服务的ID，123456为服务的密码。  "MCS002": "123456",  "MDS001": "123456",  "MDS002": "123456",  "DSS001": "123456"  } |

调用代码示例（注：由于此节点为）：

|  |
| --- |
| //PaaS 服务与密码映射配置方法  @Test  **public** **void** addServiceIdPwdMap() **throws** ConfigException {  String cachesnsConfig = "{\"MCS001\":\"" + "123456"  + "\",\"MCS002\":\"" + "123456"  + "\",\"MDS001\":\"" + "123456"  + "\",\"DSS001\":\"" + "123456"  + "\"}";    // paas serviceid password 映射配置  **if** (!client.exists(SDKConstants.***PAAS\_SERVICE\_PWD\_MAPPED\_PATH***))  client.add(SDKConstants.***PAAS\_SERVICE\_PWD\_MAPPED\_PATH***,  cachesnsConfig);  **else** {  client.modify(SDKConstants.***PAAS\_SERVICE\_PWD\_MAPPED\_PATH***,  cachesnsConfig);  }  } |

### 数据源配置

数据源配置示例：

|  |
| --- |
| 配置路径：/com/x/db-conf |
| 配置值示例：  {  "opt-uac-db": **//此处的名称要与core-context.xml中的dataSource名称一致**  {  "driverClassName":"com.mysql.jdbc.Driver", "jdbcUrl":  "jdbc:mysql:// 10.1.235.245:31306/dev\_baas\_uacdb1?useUnicode=true&characterEncoding=UTF-8",  "username":" uacusr01", "password":" uacusr01\_123", "autoCommit":"true",  "connectionTimeout":"30000", "idleTimeout":"600000", "maxLifetime":"1800000",  "maximumPoolSize":"10"  },  "opt-sys-db":  {  "driverClassName":"com.mysql.jdbc.Driver",  "jdbcUrl":"jdbc:mysql://10.1.228.222:39306/devibsscmdb1?useUnicode=true&characterEncoding=UTF-8",  "username":"devibsscmusr1", "password":"devibsscmusr1", "autoCommit":"true",  "connectionTimeout":"30000", "idleTimeout":"600000", "maxLifetime":"1800000",  "maximumPoolSize":"10"  }  } |

调用代码示例：

|  |
| --- |
| @Test  **public** **void** addDbConfInfo() {  System.***out***.println("DBConf config ... start");  StringBuilder sb = **new** StringBuilder();  sb.append("{ ");  sb.append(" \"opt-uac-db\": ");  sb.append(" { ");  sb.append(" \"driverClassName\":\"com.mysql.jdbc.Driver\", ");  sb.append(" \"jdbcUrl\":\"jdbc:mysql://10.1.235.245:31306/dev\_baas\_uacdb1?useUnicode=true&characterEncoding=UTF-8\", ");  sb.append(" \"username\":\"uacusr01\", ");  sb.append(" \"password\":\"uacusr01\_123\", ");  sb.append(" \"autoCommit\":\"true\", ");  sb.append(" \"connectionTimeout\":\"30000\", ");  sb.append(" \"idleTimeout\":\"600000\", ");  sb.append(" \"maxLifetime\":\"1800000\", ");  sb.append(" \"maximumPoolSize\":\"10\" ");  sb.append(" } ");  sb.append("} ");  **if** (!client.exists(SDKConstants.***DB\_CONF\_PATH***)) {  client.add(SDKConstants.***DB\_CONF\_PATH***, sb.toString());  } **else** {  client.modify(SDKConstants.***DB\_CONF\_PATH***, sb.toString());  }  System.***out***.println("DBConf config ... end");  } |

Spring core-context.xml配置示例：

|  |
| --- |
| <!-- 配置数据源 -->  <bean id=*"dataSource"* class=*"com.x.sdk.datasource.OptHikariDataSource"*>  <constructor-arg name=*"dataSourceName"* value=*"opt-uac-db"*/>  </bean> |

注意：上面的opt-uac-db 要与配置中心的数据源配置信息一致

### 缓存MCS配置

|  |
| --- |
| 配置路径：/com/x/paas-cachens-mcs-mapped |
| 配置值：  {"com.ai.opt.xxx.xxx":"MCS001", "com.ai.opt.xxx.yyy":"MCS002", " com.ai.opt.xxx.zzz ":"MCS003"} |

调用代码示例：

|  |
| --- |
| @Test  **public** **void** addMcsConfig() **throws** ConfigException {  // 缓存服务主机  String mcs002 = "MCS002";  // 缓存空间  String cachesnsConfig = "{\"com.ai.opt.test.mcs\":\"" + mcs002  + "\",\"com.x.centra.common.cache.gncfgproperties\":\"" + mcs002  + "\",\"com.x.centra.common.cache.gnservicerouteconfig\":\"" + mcs002  + "\",\"com.x.centra.common.cache.gndepart\":\"" + mcs002  + "\",\"com.x.centra.common.cache.gnsubject\":\"" + mcs002  + "\",\"com.x.centra.cache.test\":\"" + mcs002 + "\"}";      // 缓存空间配置  **if** (!client.exists(SDKConstants.***PAAS\_CACHENS\_MCS\_MAPPED\_PATH***))  client.add(SDKConstants.***PAAS\_CACHENS\_MCS\_MAPPED\_PATH***,  cachesnsConfig);  **else** {  client.modify(SDKConstants.***PAAS\_CACHENS\_MCS\_MAPPED\_PATH***,  cachesnsConfig);  }  } |

### 消息MDS配置

|  |
| --- |
| 配置路径：/com/x/paas-mdsns-mds-mapped |
| 配置值：  {"baasSmcCheckTopic":"MDS001","baasAmcTopic":"MDS002","baasOmcTopic":"MDS003"} |
|  |
| 配置路径：/com/x/paas-mds-topic-mapped |
| 配置值：  {"MDS001":"BCA976731EF24B899B143755A3AF5794\_MDS001\_1743120261","MDS002":"BCA976731EF24B899B143755A3AF5794\_MDS001\_1743120261"} |

调用代码：

|  |
| --- |
| @Test  **public** **void** addMdsConfig() **throws** ConfigException {  //  String mds001 = "MDS001";  // 空间  String mdssnsConfig = "{\"baas-bmc-topic\":\"" + mds001  + "\",\"baas-amc-topic\":\"" + mds001  + "\",\"baas-omc-topic\":\"" + mds001  + "\",\"baas-smc-topic\":\"" + mds001 + "\"}";  // MDS空间配置  **if** (!client.exists(SDKConstants.***PAAS\_MDSNS\_MDS\_MAPPED\_PATH***))  client.add(SDKConstants.***PAAS\_MDSNS\_MDS\_MAPPED\_PATH***,  mdssnsConfig);  **else** {  client.modify(SDKConstants.***PAAS\_MDSNS\_MDS\_MAPPED\_PATH***,  mdssnsConfig);  }    //  String mdsRealTopic = "BCA976731EF24B899B143755A3AF5794\_MDS001\_1743120261";  // 空间  String mdstopicConfig = "{\"MDS001\":\"" + mdsRealTopic  + "\"}";  // MDS空间配置  **if** (!client.exists(SDKConstants.***PAAS\_MDS\_TOPIC\_MAPPED\_PATH***))  client.add(SDKConstants.***PAAS\_MDS\_TOPIC\_MAPPED\_PATH***,  mdstopicConfig);  **else** {  client.modify(SDKConstants.***PAAS\_MDS\_TOPIC\_MAPPED\_PATH***,  mdstopicConfig);  }    } |

### 文档存储DSS配置

|  |
| --- |
| 配置路径：/com/x/paas-dssns-dss-mapped |
| 配置值：  {"com.ai.opt.xxx.xxx":"DSS001","com.ai.opt.xxx.yyy":"DSS002","com.ai.opt.xxx.zzz":"DSS003"} |

调用代码：

|  |
| --- |
| @Test  **public** **void** addDssConfig() **throws** ConfigException {  // 缓存服务主机  String dssId = "DSS001";  // 缓存空间  String dssnsConfig = "{\"baas-bmc-dss\":\"" + dssId  + "\",\"baas-amc-dss\":\"" + dssId  + "\",\"baas-omc-dss\":\"" + dssId  + "\",\"baas-smc-dss\":\"" + dssId + "\"}";      // 缓存空间配置  **if** (!client.exists(SDKConstants.***PAAS\_DSSNS\_DSS\_MAPPED\_PATH***))  client.add(SDKConstants.***PAAS\_DSSNS\_DSS\_MAPPED\_PATH***,  dssnsConfig);  **else** {  client.modify(SDKConstants.***PAAS\_DSSNS\_DSS\_MAPPED\_PATH***,  dssnsConfig);  }  } |

## 缓存中心MCS使用方法

调用代码示例

|  |
| --- |
| //获取cache客户端  ICacheClient iCacheClient = MCSClientFactory.*getCacheClient*(Register.***CACHE\_NAMESPACE***);  //从cache中读取信息  String pictureCode = iCacheClient.get(Register.***CACHE\_KEY\_VERIFY\_PICTURE***+session.getId());  //往cache中写数据  iCacheClient.setex(accountIdKey, Register.CACHE\_REGISTER\_ACCOUNT\_ID\_TIME, accountId); |

## 消息中心MDS使用方法

消息发送调用代码示例：

|  |
| --- |
| @Test  **public** **void** sendMsgTest(){  String mdsns = "baas-bmc-topic";//  IMessageSender msgSender = MDSClientFactory.*getSenderClient*(mdsns);  //msgSender.getParititions() 方法废弃，以后不再调用  **for**(**int** i=0;i<5;i++){  //send方法的第二个参数为随机数，依据该随机数均匀往各个片区发送消息  msgSender.send("[test-baas-bmc-topic-msg:"+i+"]This is a test message……", **new** Random(1000).nextLong());  //System.out.println("sender---[test-baas-bmc-topic-msg:"+i+"]This is a test message……");  }  } |

消息接收调用代码示例：

|  |
| --- |
| @Test  **public** **void** recvMsgTest(){  String mdsns = "baas-bmc-topic";//  IMsgProcessorHandler msgProcessorHandler=**new** IMsgProcessorHandler() {    @Override  **public** IMessageProcessor[] createInstances(**int** paramInt) {  // **TODO** Auto-generated method stub    List<IMessageProcessor> processors = **new** ArrayList<>();  IMessageProcessor processor = **null**;  **for** (**int** i = 0; i < paramInt; i++) {  processor = **new** MessageProcessorImpl();  processors.add(processor);  }  **return** processors.toArray(**new** IMessageProcessor[processors.size()]);  }  };  IMessageConsumer msgConsumer= MDSClientFactory.*getConsumerClient*(mdsns, msgProcessorHandler);  msgConsumer.start();  **synchronized** (MdsTest.**class**) {  **while** (**true**) {  **try** {  MdsTest.**class**.wait();  } **catch** (Exception e) {  System.***out***.println("MDS 消费错误，具体信息为：" + e.getMessage());  }  }  }    }  //消息处理接口  **package** com.x.sdk.test.paas.mds;  **import** com.ai.paas.ipaas.mds.IMessageProcessor;  **import** com.ai.paas.ipaas.mds.vo.MessageAndMetadata;  **public** **class** MessageProcessorImpl **implements** IMessageProcessor{  @Override  **public** **void** process(MessageAndMetadata message) **throws** Exception {  // **TODO** Auto-generated method stub  **if** (**null** != message) {  System.***out***.println("------Topic:" + message.getTopic() + ",key:"  + **new** String(message.getKey(), "UTF-8") + ",content:"  + **new** String(message.getMessage(), "UTF-8"));  }  }  } |

## 文档存储DSS使用方法

调用代码示例：

|  |
| --- |
| @Test  **public** **void** saveAndRead() **throws** Exception {  IDSSClient client=DSSClientFactory.*getDSSClient*(dssns);  String ss = "002test\t123456\t6666\t200902190002\t";  // 客户资料上传的场景命名空间  // 根据命名空间，从配置中心获取对应的DSS服务ID  String fileid = "";  **for** (**int** k = 0; k < 10; k++) {  fileid = client.save((ss+"---"+k).getBytes(), "test");  System.***out***.println(fileid);  **byte**[] filebytes = client.read(fileid);  System.***out***.println(**new** String(filebytes));  }  } |

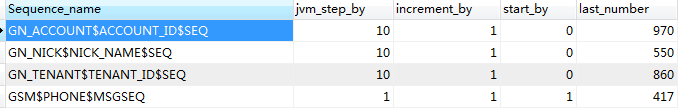
## 序列使用方法

使用配置中心的前提是配置好paas-conf.properties文件。具体配置方法见：[sdk配置文件](#_opt-sdk配置文件)

1）在core-context中配置序列所需的数据源：

|  |
| --- |
| <!-- 配置数据源 -->  <bean id=*"dataSource"* class=*"com.x.sdk.datasource.OptHikariDataSource"*>  <constructor-arg name=*"dataSourceName"* value=*"opt-uac-db"*/>  </bean>    <bean id=*"seqDataSourceLoader"* class=*"com.x.sdk.components.sequence.datasource.SeqDataSourceLoader"* init-method=*"init"*>  <property name=*"ds"* ref=*"dataSource"* />  </bean> |

2）在对应数据库中创建sys\_sequences表，并添加需要的数据，如下图：



其中，sequence\_name为序列的名称，jvm\_step\_by为jvm的缓存步长，increment\_by为序列每次增加的步长，start\_by为起始序列号，last\_number为上一次使用的序列号

**sequence\_name命名规则：表名+“$”+字段名+“$SEQ”**

3）调用代码示例：

|  |
| --- |
| //获取下一个可用的序列号  String newId = SeqUtil.getNewId(“GSM$PHONE$MSGSEQ”); |
| //获取下一个可用的序列号（指定位数，不够的前面补0）  String newId = SeqUtil.getNewId(“GSM$PHONE$MSGSEQ”, 7); |

## SDK工具包

### 中心产品DUBBO服务启动

功能：提供统一的DUBBO服务启动主函数。要求DUBBO服务加载的context文件路径与名称必须固定

|  |  |  |
| --- | --- | --- |
| 类路径 | 方法 | 说明 |
| com.x.sdk.appserver.DubboServiceStart | **public static void main(String[] args)** | 主函数不需要传入任何参数 |

说明：

* 主函数加载的DUBBO配置文件必须存放在资源目录下的dubbo/provider/dubbo-provider.xml。文件名称与路径都必须固定
* 通过sh脚本调度主函数即可
* DUBBO服务可以同时启动多个进程来实现负载均衡策略。
* 每个中心产品的启动隔离，互相不影响。 进程需要常驻内存。

### DUBBO消费端接口获取

功能：在客户端调用DUBBO服务时候，获取DUBBO服务的客户端。

|  |  |  |
| --- | --- | --- |
| 类路径 | 方法 | 说明 |
| com.x.sdk.dubbo.util.DubboConsumerFactory | public static <T> T getService(String beanId, Class<T> clazz) | 获取DUBBO服务实例 |
| public static <T> T getService(String beanId) | 根据BEAN\_ID获取DUBBO服务实例 |
| public static <T> T getService(Class<T> clazz) | 根据类获取DUBBO服务，优先采用此方法。特殊情况除外 |

说明：

需要加载DUBBO消费端引用文件dubbo/ consumer /dubbo-consumer.xml 路径与文件名必须在资源目录下固定

### 常用工具类

功能：提供一组基础的工具类

|  |  |  |
| --- | --- | --- |
| 类路径 | 方法 | 说明 |
| com.x.sdk.util.BeanUtils | **public** **static** **void** copyVO(Object destSVO, Object orignSVO)  **public** **static** **void** copyProperties(Object destSVO, Object orignSVO) | 属性对拷贝 |
| com.x.sdk.util.CollectionUtil | **public** **static** **boolean** isEmpty(Collection<?> collection) | 判断集合是否空 |
| com.opt.sdk.util.DateUtil | **public** **static** Timestamp getSysDate() | 获取系统时间。其它方法省略 |
| com.x.sdk.util. StringUtil | **public** **static** **boolean** isBlank(String str)  **public** **static** String toString(Object obj)  **public** **static** String restrictLength(String strSrc, **int** iMaxLength) | 字符串处理函数 |

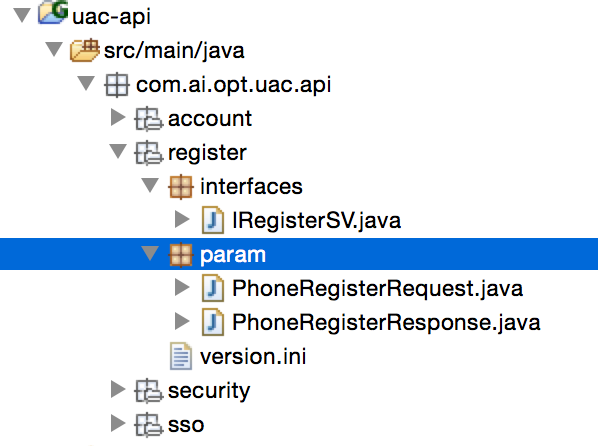
# 服务开发

新的研发体系中，RPC服务主要采用开源的dubbo框架实现，需要定义JAVA接口类和可序列化的参数，在通过dubbo注解或者配置文件的方式完成服务注册。本章主要规定服务研发过程中的主要注意点

## 服务开发

### 服务包结构

为了对服务进行规范的约束管理，要求在各个业务子工程(xx-api)中对服务的包结构进行统一的约束。参考下面图示进行说明



|  |
| --- |
| 主包路径如 com.ai.opt.uac.api  服务包 如 register 包名就是将来发布到nexus上的artifactId  服务包下分为interfaces 定义接口类 param 定义可序列化的参数类  version.ini 是版本文件，非常重要，gradle 是通过这个来识别服务包构件的 |

### 服务接口签名要求

为了对服务进行智能发布到服务在线管控，有效的处理服务调用过程中的异常，特别对服务方法的的定义做出如下要求：

* 接口方法签名中必须显示声明BusinessException,SystemException 这两个异常类来自opt-base的基础包
* 必须添加注解，如方法说明，参数说明。@Author @ApiCode等信息。服务采集插件将智能获取服务元信息PUSH到服务在线，方便服务在线或者离线测试。
* 有必要增加 @interface MethodName{} 通过声明同名服务方法内部接口的方式，实现基本参数约束的分组绑定。MethodName{} 这个名字dubbo的validator扩展中要求必须是与服务方法同名并且首字母大写。

|  |
| --- |
| /\*\*  \* 用户注册<br>  \* Date: 2016年3月25日 <br>  \* Copyright (c) 2016 asiainfo.com <br>  \*  \* **@author** zhanglh  \* **@ApiCode** UAC\_0001  \*/  PhoneRegisterResponse registerByPhone(PhoneRegisterRequest request) **throws** BusinessException,SystemException;  **@interface** RegisterByPhone {} //一定要放到方法名的下面 |

### 服务参数要求

**服务的入参要求**是可序列化的，并且针对每个属性要规范注解说明，同时按照服务规范绑定基本的约束条件。具体的校验规则及其用法如下

|  |
| --- |
| **public** **class** PhoneRegisterRequest **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  /\*\*  \* 手机号码，必填  \*/  @NotNull(message = "手机号码不能为空", groups = { IRegisterSV.RegisterByPhone.**class** })  @Pattern(regexp = "^((13[0-9])|(14[0-9])|(15[^4,\\D])|(18[0,5-9])|(17[0-9]))\\d{8}$", message = "手机号码格式不正确", groups = { IRegisterSV.RegisterByPhone.**class** })  @Size(min = 11, max = 11, message = "手机号码长度不是11位", groups = { IRegisterSV.RegisterByPhone.**class** })  **private** String phone;  /\*\*  \* 密码，必填  \*/  @NotNull(message = "密码不能为空", groups = { IRegisterSV.RegisterByPhone.**class** })  **private** String accountPassword;  /\*\*  \* 手机验证码，必填  \*/  @NotNull(message = "手机验证码不能为空", groups = { IRegisterSV.RegisterByPhone.**class** })  **private** String phoneVerifyCode;  /\*\*  \* 图片验证码，必填  \*/  @NotNull(message = "图片验证码不能为空", groups = { IRegisterSV.RegisterByPhone.**class** })  **private** String pictureVerifyCode; |

**服务的出参要求：实现序列化，属性注释要按照要求写清楚。并且必须要继承BaseResponse**

**具体参考**

|  |
| --- |
| /\*\*  \* 注册返回参数 <br>  \* Date: 2016年3月16日 <br>  \* Copyright (c) 2016 asiainfo.com <br>  \*  \* **@author** zhanglh  \*/  **public** **class** PhoneRegisterResponse **extends** BaseResponse {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  /\*\*  \* 账号ID  \*/  **private** **long** accountId;  **public** **long** getAccountId() {  **return** accountId;  }  **public** **void** setAccountId(**long** accountId) {  **this**.accountId = accountId;  }  } |

### 服务实现异常处理

在服务接口签名要求必须显示的声明BusinessException和SystemException，主要是为了配合服务实现层对于业务级别异常和系统级别异常的处理。因为这两个是运行时异常，在服务实现过程中不必要进行频繁的try..catch逻辑。

SDK包中的dubbo扩展类DubboRequestTrackFilter会自动对服务端进行激活，这里会将服务端实现中抛出的BusinessException或通过javax.validator注解定义的参数属性基本规则校验失败的信息认为是业务级别异常，自动转换成ResponseHeader对象。对于SystemException则认为是系统级别异常，直接向客户端传递。

因此，实现过程中，应该区分什么时候抛出BusinessException（例如：客户资料不存在，参数不能为空等）；什么时候抛出SystemException（比如try..catch(SQLException) throw new SystemException(“sql error”)）。

关于异常的处理，详细参考…

示例代码

|  |
| --- |
| @Service(validation = "true")  @Component  **public** **class** RegisterSVImpl **implements** IRegisterSV {  @Autowired  **private** IRegisterBusiSV iRegisterBusiSV;  @Autowired  IVoValidateSV iVoValidateSV;  @Override  **public** PhoneRegisterResponse registerByPhone(PhoneRegisterRequest request)  **throws** BusinessException,SystemException {  iVoValidateSV.validateRegister(request);  // 设置默认值  GnAccount account = **new** GnAccount();  BeanUtils.*copyProperties*(account, request);  account.setState(AccountConstants.Account.***ACCOUNT\_STATE***);  account.setNickName(AccountSeqUtil.*createNickName*());  account.setAccountType(AccountConstants.Account.***ACCOUNT\_TYPE***);  account.setAccountLevel(AccountConstants.Account.***ACCOUNT\_LEVEL***);  account.setActiveTime(DateUtil.*getSysDate*());  account.setInactiveTime(DateUtil.*getTimestamp*(AccountConstants.Account.***INACTIVE\_DATE***,  DateUtil.***DATETIME\_FORMAT***));  account.setCreateTime(DateUtil.*getSysDate*());  account.setTenantId(AccountConstants.Account.***INIT\_TENANT\_ID***);  **long** accountId = iRegisterBusiSV.registerByPhone(account);  PhoneRegisterResponse response = **new** PhoneRegisterResponse();  ResponseHeader responseHeader = **new** ResponseHeader(**true**, ResultCode.***SUCCESS\_CODE***, "成功");  response.setAccountId(accountId);  response.setResponseHeader(responseHeader);  **return** response;  }  } |

### 将API发布到在线网站

开发人员在定义好接口后，直接集成gradle插件，完成文档自动发布。

首先，将GRADLE插件集成到项目的build.gradle中。这里以中心产品的uac-api工程为例。执行api2doc任务，即可将服务信息发布至在线服务网站

|  |
| --- |
| **apply** plugin: "APICollector"  **apply** plugin: "APIJarProcess"  **configurations** {  jaxDoclet  optBaseSource  }  **dependencies** {  jaxDoclet 'com.x.plugin:apicollector:2.0-SNAPSHOT'  optBaseSource "com.x: base:1.0-SNAPSHOT:sources"  compile "javax.validation:validation-api:1.0.0.GA"  compile "org.hibernate:hibernate-validator:4.2.0.Final"    }  **buildscript** {  **repositories** {  maven { url mavenPublicURL }  }  **dependencies** {  classpath **group**: 'com.x.plugin', **name**: 'apicollectorplugin', **version**: '1.0'  classpath **group**: 'com.x.plugin', **name**: 'apijarprocessplugin', **version**: '1.0'  }  }  **def** apiJarProcessTask=**task**("apiJarProcessTask",type:com.ai.runner.gradle.plugin.jar.APIJarProcessTask){  jarPaths=**configurations**.optBaseSource.files.asType(List)  destinationDir file("$buildDir/opt-base")  }  **def** List<Map<String,String>> apiList = **new** ArrayList<Map<String,String>>();  Set<File> fileSet = fileTree(apiDir) { include '\*\*/\*.ini' }.getFiles();  for(File file:fileSet){  File f1 = **new** File(file.getParent());  String apiName = f1.getName();  String **version** = file.getText('UTF-8');  Map<String,String> apiMap = **new** HashMap<String,String>();  apiMap.put("apiName", apiName);  apiMap.put("apiArtifactId", apiName);  apiMap.put("apiTaskName", "A"+apiName);  apiMap.put("apiVersion", **version**);  apiList.add(apiMap);  }  **def** Map<String,Object> apiDocTaskMap = **new** HashMap<String,Object>();  **def** Map<String,Object> apiTaskMap = **new** HashMap<String,Object>();  for(Map<String,String> apiMap:apiList){  String apiName = apiMap.get("apiName");  String apiVersion = apiMap.get("apiVersion");  String apiArtifactId = apiMap.get("apiArtifactId");  String apiTaskName = apiMap.get("apiTaskName");  **def** jarTask = **task**("${apiTaskName}Jar", type: Jar){  **version** apiVersion  baseName apiArtifactId  from **sourceSets**.**main**.output  destinationDir file("$buildDir/api-libs")  include 'com/x/centrauac/api/'+apiName+'/\*\*'  **manifest** {  attributes 'artifactId': apiArtifactId, 'Built-By': builtBy,'Built-date': **new** Date().format('yyyy-MM-dd HH:mm:ss'),'Manifest-Version':apiVersion  }  }  apiTaskMap.put(apiName, jarTask);  **def** apidocTask=**task**("${apiArtifactId}APIDoc",type:com.ai.runner.gradle.plugin.APICollectorTask){  dependsOn apiJarProcessTask  esconfig = "{\"clusterName\":\"elasticsearch\_for\_sol\",\"ip\":\"10.1.234.160\", \"port\":14999}"  source = fileTree(dir: 'src/main/java', include: 'com/x/centrauac/api/'+apiName+'/\*\*/\*.java')+fileTree(dir:"$buildDir/opt-base", include:"\*\*/\*.java")  docletpath = **configurations**.jaxDoclet.files.asType(List)  docletClass = 'com.ai.runner.apicollector.doclet.APIDoclet'  memberLevel = 'private'  encoding = 'utf-8'  belong='baas-uac'  ownerType='baas'  owner = 'baas-uac'  groupId = 'com.ai.opt.uac'  artifactId = apiArtifactId  **version** = apiVersion  }  apiDocTaskMap.put(apiName, apidocTask);  }  **task** api2doc{  dependsOn {  tasks.findAll { **task** -> **task**.**name**.endsWith('APIDoc') }  }  }  **publishing** {  publications {  for(Map<String,String> apiMap:apiList){  String apiName = apiMap.get("apiName");  String apiVersion = apiMap.get("apiVersion");  String apiArtifactId = apiMap.get("apiArtifactId");  **publishing**.publications.create(apiName, MavenPublication) {  groupId **group**  artifactId apiArtifactId  **version** apiVersion  artifact apiTaskMap.get(apiName)  pom.withXml {  asNode().children().last() + {  delegate.**dependencies** {  delegate.dependency {  delegate.groupId("com.ai.opt")  delegate.artifactId("opt-base")  delegate.**version**(optBaseVersion)  }  delegate.dependency {  delegate.groupId("org.jboss.resteasy")  delegate.artifactId("resteasy-jaxrs")  delegate.**version**("3.0.11.Final")  }  }  }  }  }  }  }  **repositories** {  maven {  url publishURL  credentials {  username = publishUserName  password = publishUserPassword  }  }  }  } |

# 核心工程构建过程

核心工程构建的过程依赖于标准的build.gradle脚本的编写。工程的结构一般采用父子工程结构。

## 主工程build脚本

主工程包含2个脚本。

Build.gradle 控制核心构建逻辑的

Settings.gradle 规定包含哪些子模块（子工程）

下面以客户中心为例说明构建脚本

Build.gradle脚本

|  |
| --- |
| **allprojects** {  **apply** plugin: 'java'  **apply** plugin: 'eclipse'  **apply** plugin: 'maven-publish'  **ext** {  profile=**project**.hasProperty('profile')? (**project**['profile']==""?"aic-dev":**project**['profile']) : "aic-dev"  moduleVersion = "1.0-SNAPSHOT"  optBaseVersion = "1.0-SNAPSHOT"  builtBy = "gradle 2.2"  publishUserName = "runnerdev"  publishUserPassword = "runnerdev"  publishURL = "http://10.1.228.199:18081/nexus/content/repositories/snapshots/"  mavenPublicURL = "http://10.1.228.199:18081/nexus/content/groups/public/"  }  compileJava.dependsOn clean  **sourceCompatibility** = 1.7  **configurations**.all { resolutionStrategy.cacheChangingModulesFor 0, 'seconds' }  **group**="com.ai.opt.uac"  }  **subprojects** {  [compileJava, compileTestJava]\*.options\*.encoding = 'UTF-8'  **repositories** { maven { url mavenPublicURL } }  **sourceSets** {  **main** {  java {  **srcDirs** = [  'src/main/java'  ]  }  resources { **srcDirs** = ['src/main/resources']}  }  **test** {  java {  **srcDirs** = [  'src/test/java'  ]  }  resources { **srcDirs** = ['src/test/resources']}  }  }  **dependencies** {  compile "com.ai.opt:opt-base:${optBaseVersion}"  compile 'org.jboss.resteasy:resteasy-jaxrs:3.0.11.Final'  compile 'org.jboss.resteasy:resteasy-client:3.0.11.Final'  compile 'org.jboss.resteasy:resteasy-jackson-provider:3.0.11.Final'  compile 'org.mortbay.jetty:jetty:6.1.26'  compile 'javax.validation:validation-api:1.1.0.Final'  compile 'javax.ws.rs:javax.ws.rs-api:2.0.1'  }  }  **project**("uac-api") {  **project**.**ext** {  appName = "uac-api"  apiDir = "src/main/java/com/x/centrauac/api/"  }  jar {  baseName appName  **version** moduleVersion  **manifest** {  attributes 'packageName': appName, 'Built-By': builtBy, 'create-date': **new** Date().format('yyyy-MM-dd HH:mm:ss')  }  }  publish.dependsOn build  }  **project**("uac-core") {  **project**.**ext** { appName = "uac-core" }  jar {  baseName appName  **version** moduleVersion  include "com/x/centrauac/\*/\*\*"  **manifest** {  attributes 'packageName': appName, 'Built-By': builtBy, 'create-date': **new** Date().format('yyyy-MM-dd HH:mm:ss')  }  }  }  jar {  baseName baseName  **version** moduleVersion  destinationDir file("$buildDir/m-libs/")  **manifest** {  attributes 'packageName': baseName, 'Built-By': builtBy, 'create-date': **new** Date().format('yyyy-MM-dd HH:mm:ss')  }  }  configure(**subprojects**) { subproject ->  **task** sourcesJar(type: Jar, dependsOn:classes) { from **sourceSets**.**main**.output }  **artifacts** { archives sourcesJar }  }  **task** copySubProjectsJar(type: Copy) {  **subprojects**.each { subproject ->  from subproject.jar  destinationDir file("$buildDir/libs/core-libs")  }  }  **task** copy3rdJars(type: Copy) {  **subprojects**.each { subproject ->  from subproject.**configurations**.runtime  exclude "uac-api-\*.\*"  destinationDir file("$buildDir/libs/3rd-libs")  }  }  **task** copySubProjectsConfig(type: Copy) {  **subprojects**.each { subproject ->  from subproject.fileTree(dir: 'src/main/resources')+subproject.fileTree(dir: 'profile/'+profile)  destinationDir file("$buildDir/config")  }  }  **task** zipLibs(type: Zip) {  baseName "libs"  from "$buildDir/libs"  }  **task** zipConfig(type: Zip) {  baseName "config"  from "$buildDir/config"  }  build.dependsOn copySubProjectsJar,copy3rdJars,copySubProjectsConfig |

未来可以加入代码检查的pmd,fingbugs,checkstyle等插件体系

Settings.gradle的脚本

|  |
| --- |
| include 'uac-api','uac-core' |

## API工程build脚本

API工程只包含一个Build.gradle脚本，负责接口定义后的接口构建发布到NEXUS和发布到文档体系中。

脚本参考如下

|  |
| --- |
| **apply** plugin: "APICollector"  **apply** plugin: "APIJarProcess"  **configurations** {  jaxDoclet  optBaseSource  }  **dependencies** {  jaxDoclet 'com.ai.runner.plugin:apicollector:1.4-SNAPSHOT'  optBaseSource "com.ai.opt:opt-base:1.0-SNAPSHOT:sources"  compile "javax.validation:validation-api:1.0.0.GA"  compile "org.hibernate:hibernate-validator:4.2.0.Final"    }  **buildscript** {  **repositories** {  maven { url mavenPublicURL }  }  **dependencies** {  classpath **group**: 'com.ai.runner.plugin', **name**: 'apicollectorplugin', **version**: '1.0'  classpath **group**: 'com.ai.runner.plugin', **name**: 'apijarprocessplugin', **version**: '1.0'  }  }  **def** apiJarProcessTask=**task**("apiJarProcessTask",type:com.ai.runner.gradle.plugin.jar.APIJarProcessTask){  jarPaths=**configurations**.optBaseSource.files.asType(List)  destinationDir file("$buildDir/opt-base")  }  **def** List<Map<String,String>> apiList = **new** ArrayList<Map<String,String>>();  Set<File> fileSet = fileTree(apiDir) { include '\*\*/\*.ini' }.getFiles();  for(File file:fileSet){  File f1 = **new** File(file.getParent());  String apiName = f1.getName();  String **version** = file.getText('UTF-8');  Map<String,String> apiMap = **new** HashMap<String,String>();  apiMap.put("apiName", apiName);  apiMap.put("apiArtifactId", apiName);  apiMap.put("apiTaskName", "A"+apiName);  apiMap.put("apiVersion", **version**);  apiList.add(apiMap);  }  **def** Map<String,Object> apiDocTaskMap = **new** HashMap<String,Object>();  **def** Map<String,Object> apiTaskMap = **new** HashMap<String,Object>();  for(Map<String,String> apiMap:apiList){  String apiName = apiMap.get("apiName");  String apiVersion = apiMap.get("apiVersion");  String apiArtifactId = apiMap.get("apiArtifactId");  String apiTaskName = apiMap.get("apiTaskName");  **def** jarTask = **task**("${apiTaskName}Jar", type: Jar){  **version** apiVersion  baseName apiArtifactId  from **sourceSets**.**main**.output  destinationDir file("$buildDir/api-libs")  include 'com/x/centrauac/api/'+apiName+'/\*\*'  **manifest** {  attributes 'artifactId': apiArtifactId, 'Built-By': builtBy,'Built-date': **new** Date().format('yyyy-MM-dd HH:mm:ss'),'Manifest-Version':apiVersion  }  }  apiTaskMap.put(apiName, jarTask);  **def** apidocTask=**task**("${apiArtifactId}APIDoc",type:com.ai.runner.gradle.plugin.APICollectorTask){  dependsOn apiJarProcessTask  esconfig = "[{ip:10.1.234.160, port:14999}]"  source = fileTree(dir: 'src/main/java', include: 'com/x/centrauac/api/'+apiName+'/\*\*/\*.java')+fileTree(dir:"$buildDir/opt-base", include:"\*\*/\*.java")  docletpath = **configurations**.jaxDoclet.files.asType(List)  docletClass = 'com.ai.runner.apicollector.doclet.APIDoclet'  memberLevel = 'private'  encoding = 'utf-8'  belong='baas-uac'  ownerType='baas'  owner = 'baas-uac'  groupId = 'com.ai.opt.uac'  artifactId = apiArtifactId  **version** = apiVersion  }  apiDocTaskMap.put(apiName, apidocTask);  }  **task** api2doc{  dependsOn {  tasks.findAll { **task** -> **task**.**name**.endsWith('APIDoc') }  }  }  **publishing** {  publications {  for(Map<String,String> apiMap:apiList){  String apiName = apiMap.get("apiName");  String apiVersion = apiMap.get("apiVersion");  String apiArtifactId = apiMap.get("apiArtifactId");  **publishing**.publications.create(apiName, MavenPublication) {  groupId **group**  artifactId apiArtifactId  **version** apiVersion  artifact apiTaskMap.get(apiName)  pom.withXml {  asNode().children().last() + {  delegate.**dependencies** {  delegate.dependency {  delegate.groupId("com.ai.opt")  delegate.artifactId("opt-base")  delegate.**version**(optBaseVersion)  }  delegate.dependency {  delegate.groupId("org.jboss.resteasy")  delegate.artifactId("resteasy-jaxrs")  delegate.**version**("3.0.11.Final")  }  }  }  }  }  }  }  **repositories** {  maven {  url publishURL  credentials {  username = publishUserName  password = publishUserPassword  }  }  }  } |

## Core工程build脚本

主要管理CORE工程依赖的三方JAR

|  |
| --- |
| **dependencies** {  //compile project(":sso-api")  compile **project**(":uac-api")  compile 'org.mybatis:mybatis:3.3.0'  compile 'org.mybatis:mybatis-spring:1.2.3'  compile 'mysql:mysql-connector-java:5.1.36'  compile 'org.springframework:spring-tx:4.1.7.RELEASE'  compile 'org.springframework:spring-jdbc:4.1.7.RELEASE'  compile 'com.x.sdk:opt-sdk:1.0.1-SNAPSHOT'  //compile 'com.ai.runner.util:runner-util:1.0.7-SNAPSHOT'  testCompile(  "org.springframework:spring-test:3.2.13.RELEASE",  "junit:junit:4.11"  )  } |

## DubboAPI构建发布到NEXUS

参考[服务接口发布要求](#_服务接口发布要求)，避免往NEXUS发布很多无用的快照或版本，要求一般情况下每次只发布特定的接口。

首先执行任务查看

|  |
| --- |
| E:\AI\_Works\BaaS\opt-uac>gradle tasks |

能看到以下的Publish命令

|  |
| --- |
| Publishing tasks  ----------------  publish - Publishes all publications produced by this project.  publishAccountPublicationToMavenLocal - Publishes Maven publication 'account' to the local Maven repository.  publishAccountPublicationToMavenRepository - Publishes Maven publication 'account' to Maven repository 'maven'.  publishRegisterPublicationToMavenLocal - Publishes Maven publication 'register' to the local Maven repository.  publishRegisterPublicationToMavenRepository - Publishes Maven publication 'register' to Maven repository 'maven'  publishSecurityPublicationToMavenLocal - Publishes Maven publication 'security' to the local Maven repository.  publishSecurityPublicationToMavenRepository - Publishes Maven publication 'security' to Maven repository 'maven'  publishSeqPublicationToMavenLocal - Publishes Maven publication 'seq' to the local Maven repository.  publishSeqPublicationToMavenRepository - Publishes Maven publication 'seq' to Maven repository 'maven'.  publishSsoPublicationToMavenLocal - Publishes Maven publication 'sso' to the local Maven repository.  publishSsoPublicationToMavenRepository - Publishes Maven publication 'sso' to Maven repository 'maven'.  publishTenantPublicationToMavenLocal - Publishes Maven publication 'tenant' to the local Maven repository.  publishTenantPublicationToMavenRepository - Publishes Maven publication 'tenant' to Maven repository 'maven'.  publishToMavenLocal - Publishes all Maven publications produced by this project to the local Maven cache. |

单个发布任务的命令如下publishAccountPublicationToMavenRepository

执行 gradle publishAccountPublicationToMavenRepository 可以将接口包发布到NEXUS中

## DubboAPI构建发布到服务在线

DUBBO接口的文档采用自动化文档体系构建

首先执行任务查看

|  |
| --- |
| E:\AI\_Works\BaaS\opt-uac>gradle tasks |

能看到以下的ApiDoc命令

|  |
| --- |
| Other tasks  -----------  api2doc  APICollect  APIJarProcess  zipConfig  zipLibs |

发布任务的命令如下：api2doc

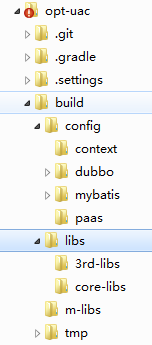
执行 gradle api2doc 可以将接口的文档发布到Web Online中

## Project构建输出

在主工程目录下，执行gradle build

如果不想执行TEST CASE，则在执行命令时候增加 –x test 如： gradle build –x test

将在build目录下输入如下结构



Config—源码目录中src/main/resources 下的资源文件

Libs/3rd-libs 所有依赖的第三方jar包

Libs/core-libs 包含子工程的源码的Jar包。比如uac-api.jar uac-core.jar

## 如何让版本支持不同环境？

程序源代码经过构建后，理论上要达到针对某个环境的构建。比如你给AWS云生产版本出个程序包，或者给BYD私有云生产出个包，或者给AIC研发云开发环境出个包。

一般而言，代码可能不需要改，但是部分配置文件可能里面需要配置环境的IP和端口，面对标准版本在不同的云上部署时候，手工修改是非常吃力不讨好的事情，出错概率极大。

提前预设，减少人工干预。

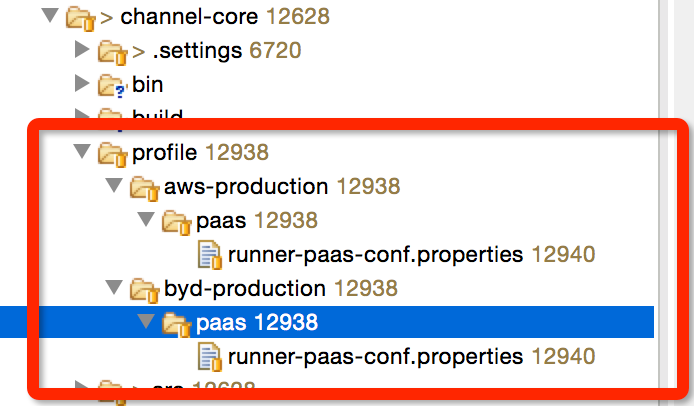
产品体系中，要达到上述目的，通过两个组合拳完成：

1. 配置中心化：通过程序优化，将不需要放在配置文件的里面的配置，全部转移到配置中心，通过SDK的方式加载。这样，能做到大部分配置与程序包分离。
2. 设置profile目录：对于不能放到配置中心的配置文件，可能需要提前预设profile的配置文件了。这里以paas-config.properties为例。

### 服务端工程处理方法

下面重点说明下profile模式的设置，以服务端工程为例

1. Core工程根目录，比如Runner-Channel\Channel-Core下面。建好profile/ENV\_PROFILE/配置文件



备注：ENV\_PROFILE的命名规则：云定义+“-”+版本类型

|  |  |
| --- | --- |
| 云定义 | 说明 |
| aws | AWS云 |
| byd | BYD私有云 |
| aic | 亚信研发云 |

|  |  |
| --- | --- |
| 版本定义 | 说明 |
| production | 生产版本 |
| qa | 测试版本 |
| dev | 开发版本 |

默认情况下，不需要在profile中增加dev目录。程序src/main/resources中默认为dev的配置

1. 修改主工程Build.gradle

|  |
| --- |
| ext {  profile=project.hasProperty('profile')? (project['profile']==""?"aic-dev":project['profile']) : "aic-dev"  }  说明：增加的这个属性，用来接收gradle命令中的参数。如果命令中没有传，默认为aic-dev 表示采用AIC亚信云开发的环境  下面的红色配置，在拷贝配置文件时候，加上红色部分的。Profile的传入必须要与上述文件夹中的aws-production类似的名称一致，否则打包后还是获取的开发配置  **task** copySubProjectsConfig(type: Copy) {  **subprojects**.each { subproject ->  from subproject.fileTree(dir: 'src/main/resources')+subproject.fileTree(dir: 'profile/'+profile)  destinationDir file("$buildDir/config")  }  } |

1. Gradle运行

要打包时候，你只需要按照上述规范在命令中传递参数

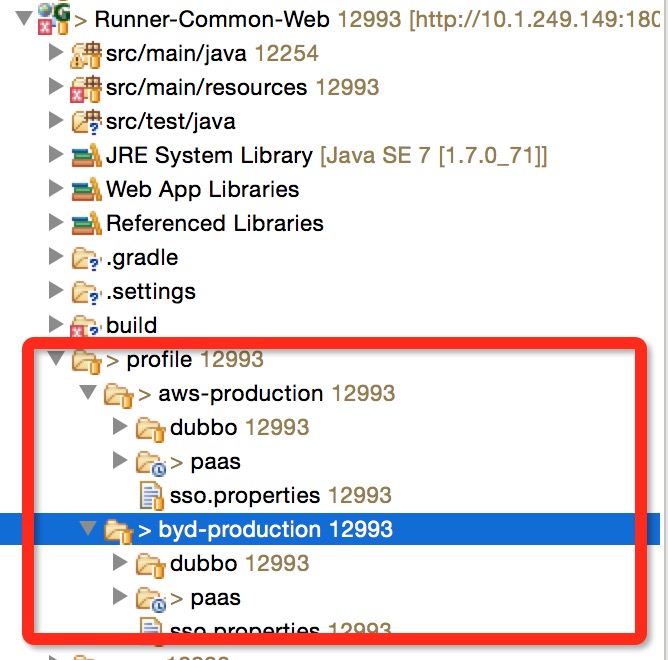
|  |
| --- |
| 比如，你想给AIC研发云打包。你的命令和之前是一样的  gradle build -x check  比如，你想给AWS生产打包。你的命令是这样的  gradle build -x check -Pprofile=aws-production |

1. OK，您最后输出的程序，可以直接部署到对应的环境了。

### WEB工程处理方法

以Gradle构建为核心的WEB工程的处理方法非常简单。

1. 同服务端工程设置一样，设置profile目录。源代码路径中的配置文件默认为开发分支的。



放置的路径必须要和src/main/resources 里面的路径文件名称完全一样。

建议只放置与环境配置有关系的

1. 修改build.gradle，2处红色部分的内容是新增的

|  |
| --- |
| apply plugin: 'war'  apply plugin: 'java'  apply plugin: 'eclipse'  apply plugin: 'eclipse-wtp'  apply plugin: "APIJarProcess"  ext {  profile=project.hasProperty('profile')? (project['profile']==""?"aic-dev":project['profile']) : "aic-dev"  }  "  ……….省略…  processResources {  from('profile/'+profile) {  include '\*\*/\*'  }  } |

1. 执行命令

|  |
| --- |
| 比如，你想给AIC研发云打包。你的命令和之前是一样的  gradle build -x check  比如，你想给AWS生产打包。你的命令是这样的  gradle build -x check -Pprofile=aws-production |

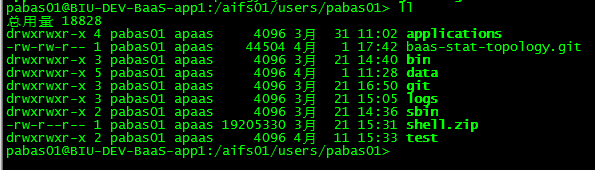
未来如果采用CAAS容器服务后，程序代码与配置要绝对隔离。

# 打包部署【启动dubbo服务】

以开发环境为例子，说明单个中心产品如何从git获取代码，如何编译，如何部署到指定目录，如何启动服务。

## 开发环境部署目录规范

以中心产品后场开发环境为例，主机IP为10.1.130.84，用户为pabas01，登陆主机后，用户主目录下的文件如下：



关键目录为：applications, bin,git,logs,sbin,zookeeper,说明如下：

|  |  |  |
| --- | --- | --- |
| **目录** | **用途** | **备注** |
| **${HOME}** | 用户登录主目录 | 如，devrun01登录后的主目录为：  /aifs01/users/pabas01 |
| **git** | 存放各中心产品源码 | 所有代码都放在git/opt-baas-projects下面 |
| **applications** | 存放各中心产品编译好的文件 | 所有编译好的中心产品文件都放在applications/ opt-baas-services |
| **bin** | 存放启动脚本 | 启动脚本主要包含：  setEnv.sh ： 用于设置环境变量，加载类及配置文件；  copyLibs.sh ： 用于拷贝编译好的文件到applications/ opt-baas-services对应的中心产品目录下；  opt-uac-dubbo-server.sh ： dubbo服务启动脚本，命名规范为“中心产品”+“-dubbo-server.sh” |
| **sbin** | 存启动公共脚本 | monitor\_process.sh ： 所有启动脚本的公共脚本。 |
| **logs** | 存放日志 | 存放dubbo服务启动脚本的日志，放在logs/opt-baas-logs/目录下 |

|  |
| --- |
| **git目录规范示例：**  |--${HOME}/git  | |-- opt-baas-projects //所有git工程代码都放在该目录下，没有则创建  | |--opt-uac //账户体系产品工程代码目录  | |--baas-bmc //bmc产品工程代码目录  | ………… |

|  |
| --- |
| **applications目录规范示例：**  |--${HOME}/applications  | |-- opt-baas-services //所有工程编译包都放在该目录下，没有则创建  | |--opt-uac-services //账户体系编译包目录  | |--baas-bmc-services //bmc编译包目录  | ………… |

|  |
| --- |
| **bin目录规范示例：**  |--${HOME}/bin  | |-- opt-baas //所有工程启动脚本都放在该目录下，没有则创建  | |--opt-uac //账户体系启动脚本目录  | |--baas-bmc //bmc启动脚本目录  | ………… |

|  |  |
| --- | --- |
| **sbin目录规范示例：**  |--${HOME}/sbin/monitor\_process.sh //公共脚本 |  |

## 从git获取最新代码

在从git下载源码前，必须先安装并配置git

下载源代码方法如下：

|  |
| --- |
| 切换到**${HOME}/git /opt-baas-projects**目录下面，执行如下命令：  **git clone https://github.com/AI-OPT/opt-uac.git**  即可在**${HOME}/git /opt-baas-projects**下面生成**${HOME}/git /opt-baas-projects/opt-uac**工程代码。 |

**${HOME}/git /opt-baas-projects/ opt-uac**工程代码。

更新代码的方法如下：

|  |
| --- |
| 切换到**${HOME}/git /opt-baas-projects/ opt-uac**目录，执行如下命令：  **git pull**  //拉取并合并远程代码 |

## 执行编译

编译代码的方法如下：

|  |
| --- |
| 切换到**${HOME}/git /opt-baas-projects/ opt-uac**目录，执行如下命令：  **gradle build –x test**  //编译代码 |

## 从编译目录拷贝到部署目录

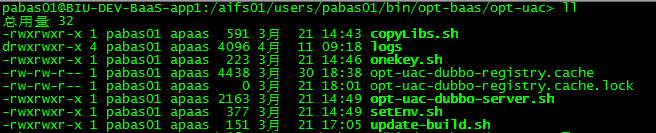
|  |  |
| --- | --- |
| 切换到**${HOME}/bin /opt-baas /opt-uac**目录，执行如下命令：  **./copyLibs.sh**  //执行拷贝  其中，**copyLibs.sh**内容如下（红色部分，是各项目要调整的信息）：  #!/bin/sh  SRC\_BASIC\_BUILD\_HOME=${HOME}/git/opt-baas-projects/opt-uac/build  SRC\_CONFIG\_HOME=${SRC\_BASIC\_BUILD\_HOME}/config  SRC\_HOME\_LIB=${SRC\_BASIC\_BUILD\_HOME}/libs  TARGET\_HOME=${HOME}/applications/opt-baas-services/opt-uac-services  echo $SRC\_HOME\_LIB  echo $TARGET\_HOME  echo $UTIL\_HOME\_LIB  echo "delete old"  cd $TARGET\_HOME  rm -f libs/3rd-libs/\*  rm -f libs/core-libs/\*  echo "delete old config"  rm -rf config/\*  echo "copy new"  cp $SRC\_HOME\_LIB/3rd-libs/\* libs/3rd-libs/.  cp $SRC\_HOME\_LIB/core-libs/\* libs/core-libs/.  echo "copy new config"  cp -r $SRC\_CONFIG\_HOME/\* config/  echo "ok" |  |

## 启动dubbo服务

|  |
| --- |
| 切换到**${HOME}/bin /opt-baas/opt-uac**目录，编写setEnv.sh文件，内容如下：  #!/bin/sh  DEPOLY\_PATH=$HOME/applications/opt-baas-services/opt-uac-services  export DEPOLY\_PATH  for file in ${DEPOLY\_PATH}/libs/\*\*/\*.jar;  do CP=${CP}:$file;  done  CLASSPATH="${CP}"  export CLASSPATH  CLASSPATH="${CLASSPATH}:${DEPOLY\_PATH}/config"  JAVA\_OPTIONS=" -Dfile.encoding=UTF-8 -Djava.net.preferIPv4Stack=true -Dsun.net.inetaddr.ttl=10"  MEM\_ARGS="-Xms128m -Xmx512m"    切换到**${HOME}/bin /opt-baas/opt-uac**目录，执行如下命令：  **./** **opt-uac-dubbo-server.sh start**  //启动账户体系dubbo服务  **./** **opt-uac -dubbo-server.sh stop**  //停止账户体系dubbo服务  **./** **opt-uac -dubbo-server.sh monitor** //监控账户体系dubbo服务  其中，**opt-uac-dubbo-server.sh** 内容如下（红色部分，是各项目要调整的信息）：  #!/bin/sh  DUBBO\_PORT="10881" //各服务的端口不能重复  PROCESS\_NAME="DubboServiceStart"  PROCESS\_PARM="opt.uac.dubbo.port=$DUBBO\_PORT"  CUR\_USER=`whoami`  RUNNER\_PRODUCT\_NAME=opt.uac  BIN\_PATH=${HOME}/bin/opt-baas/opt-uac  LOG\_PATH=$HOME/logs/opt-baas-logs/opt-uac-dubbo-$DUBBO\_PORT.log  CLASSPATH="${CLASSPATH}"  MEM\_ARGS="-Xms256m -Xmx512m -XX:PermSize=64M -XX:MaxPermSize=128M"  . "${BIN\_PATH}/setEnv.sh"  if [ "$1" = "start" ]; then  PROCESS\_ALIVE\_STATUS=$($HOME/sbin/monitor\_process.sh $PROCESS\_NAME $PROCESS\_PARM)  if [ "$PROCESS\_ALIVE\_STATUS" = "PROCESS\_EXIST" ];  then  echo " $RUNNER\_PRODUCT\_NAME dubbo server had already started!! process param is [$PROCESS\_PARM]"  exit 0;  fi    ${JAVA\_HOME}/bin/java ${MEM\_ARGS} -D${PROCESS\_PARM} ${JAVA\_OPTIONS} com.x.sdk.appserver.DubboServiceStart >> $LOG\_PATH & 2>&1&  echo "$RUNNER\_PRODUCT\_NAME dubbo server started!! logs at $LOG\_PATH"  else if [ "$1" = "stop" ]; then  PROCESS\_NUMBER=0  ps -ef|grep $PROCESS\_NAME |grep ${CUR\_USER} | grep $PROCESS\_PARM | grep java | grep -v grep | awk '{print $2}' |while read pid  do  PROCESS\_NUMBER=PROCESS\_NUMBER+1  kill ${pid} 2>&1 >/dev/null  echo "stopped $RUNNER\_PRODUCT\_NAME dubbo server process name :${PROCESS\_NAME},process param:${PROCESS\_PARM},PID:${pid} "  done  if [ "$PROCESS\_NUMBER" = 0 ];  then  echo "$RUNNER\_PRODUCT\_NAME dubbo server not exists. process name :${PROCESS\_NAME},process param:${PROCESS\_PARM}"  fi  exit 0;  else if [ "$1" = "monitor" ]; then  PROCESS\_ALIVE\_STATUS=$($HOME/sbin/monitor\_process.sh $PROCESS\_NAME $PROCESS\_PARM)  if [ "$PROCESS\_ALIVE\_STATUS" = "PROCESS\_EXIST" ];  then  echo "$RUNNER\_PRODUCT\_NAME dubbo server started!! process param is [$PROCESS\_PARM]"  exit 0;  else  echo "$RUNNER\_PRODUCT\_NAME dubbo server not exists. process name :${PROCESS\_NAME},process param:${PROCESS\_PARM}"  exit 0;  fi  else  echo "ERROR: Please input a correct argument,such as: start or stop or monitor"  exit 1  fi  fi  fi |

## 一键部署(onekey.sh)

opt-uac的shell脚本



从git下载代码后，可通过一键部署脚本onekey.sh来更新代码、编译代码、停止服务、更换jar包、启动服务等工作。

onekey.sh脚本如下：

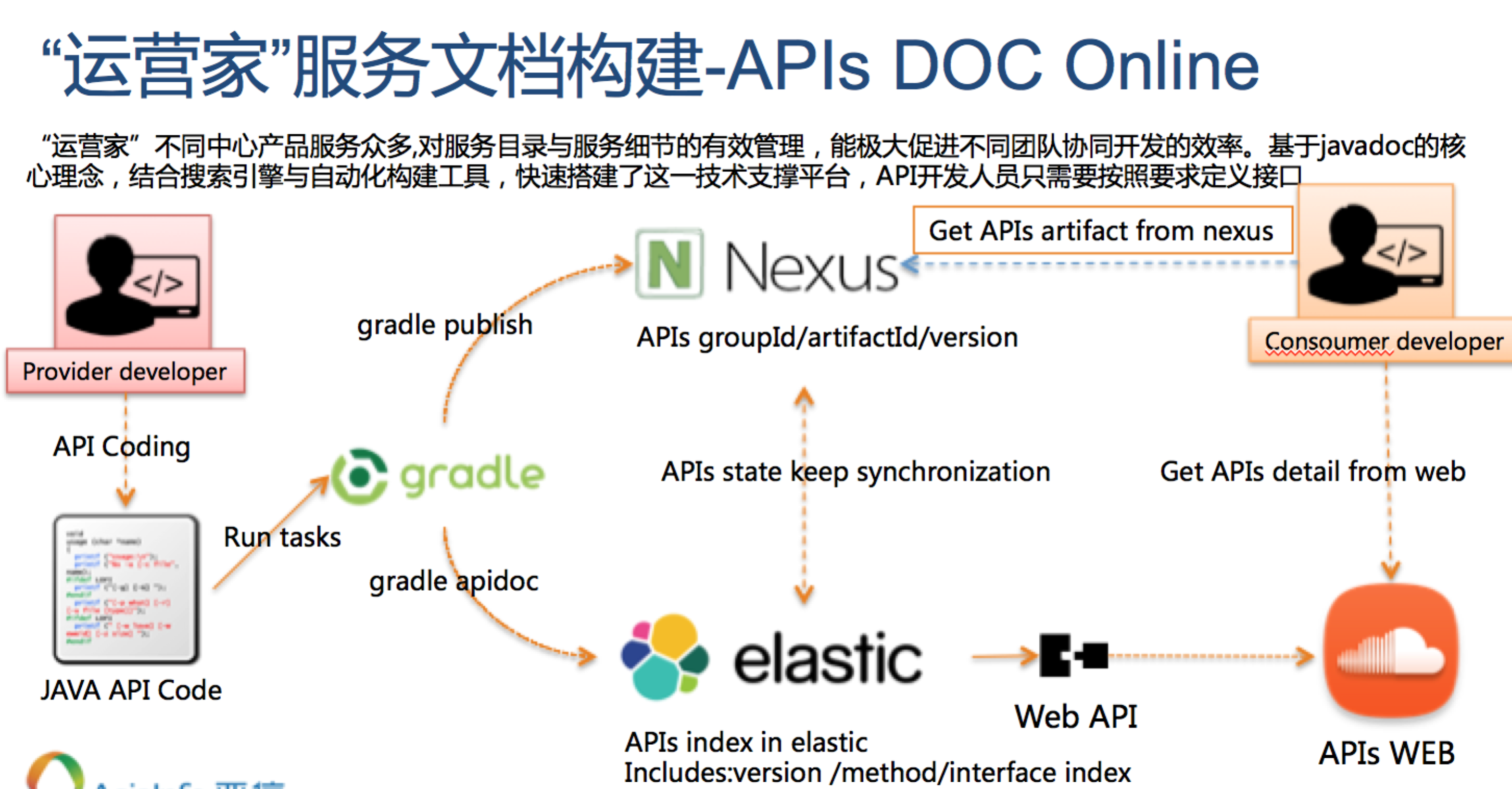
|  |
| --- |
| #!/bin/sh  ONEKEY\_BIN\_PATH=${HOME}/bin/opt-baas/opt-uac  ${ONEKEY\_BIN\_PATH}/update-build.sh  ${ONEKEY\_BIN\_PATH}/opt-uac-dubbo-server.sh stop  ${ONEKEY\_BIN\_PATH}/copyLibs.sh  ${ONEKEY\_BIN\_PATH}/opt-uac-dubbo-server.sh start |

其中，update-build.sh负责更新代码和打包，具体如下：

|  |
| --- |
| SRC\_BASIC\_HOME=${HOME}/git/opt-baas-projects/opt-uac  //此处使用的master分支  pushd ${SRC\_BASIC\_HOME} && git pull && popd  # 使用feature的示例  # pushd ${SRC\_BASIC\_HOME} && git pull && git checkout feature && popd  gradle build -x test -x check -p ${SRC\_BASIC\_HOME} |

# 服务在线测试【废弃】

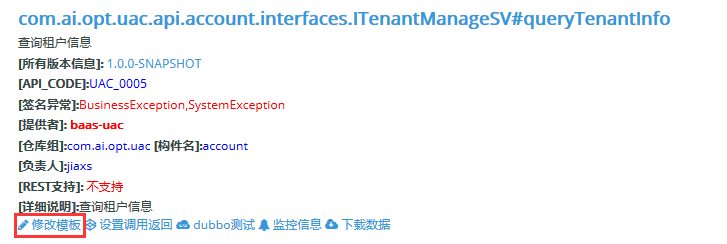
## 自动化的服务文档



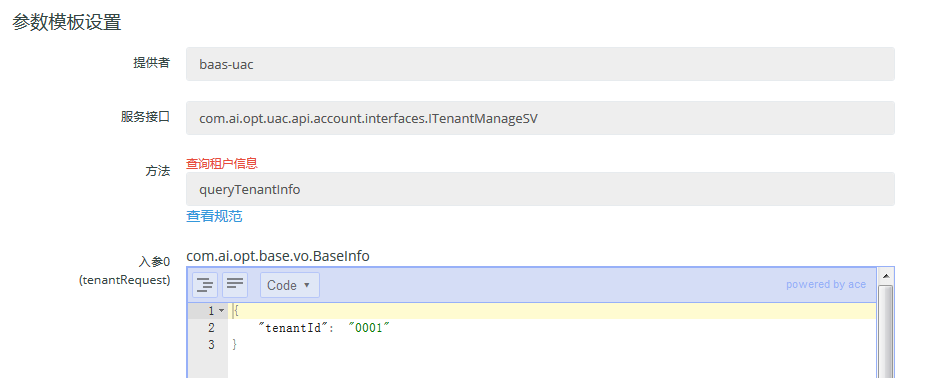
## 服务在线测试

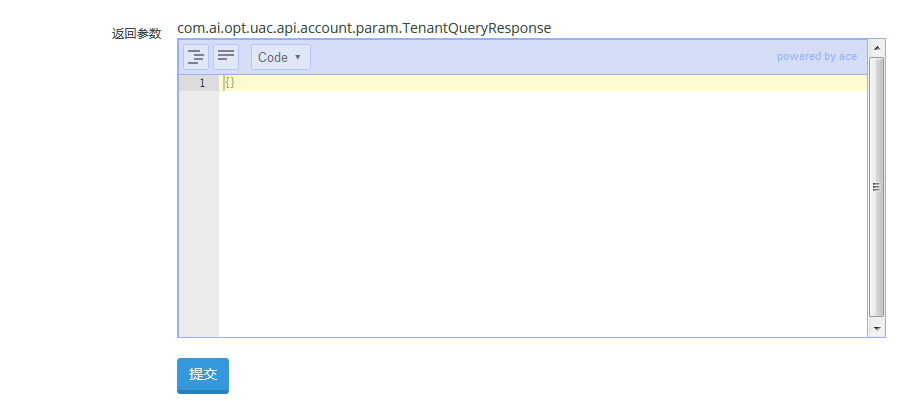
登录在线服务网站：



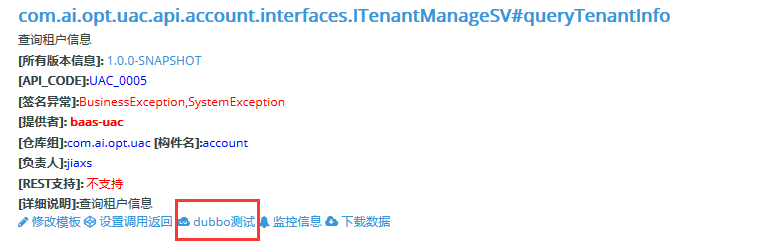


找到待测试的方法，点击设置模板(修改模板)，进入模板设置页面，如下：

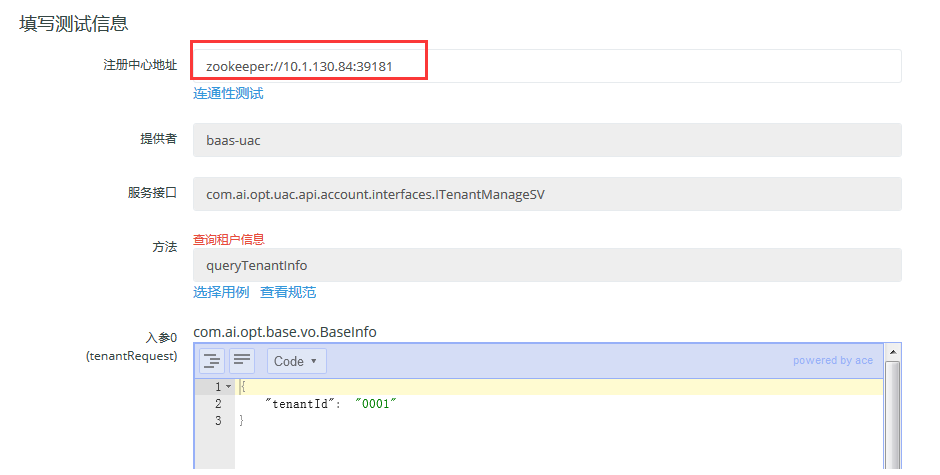


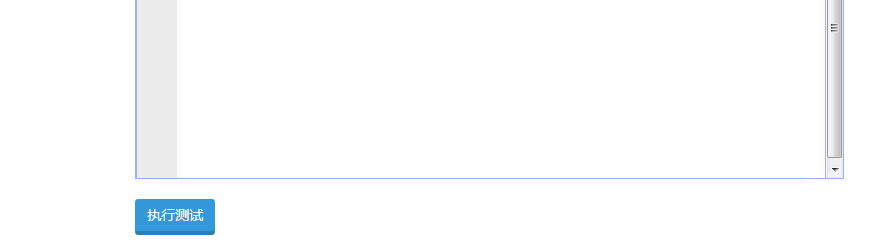


点击提交，设置模板完毕

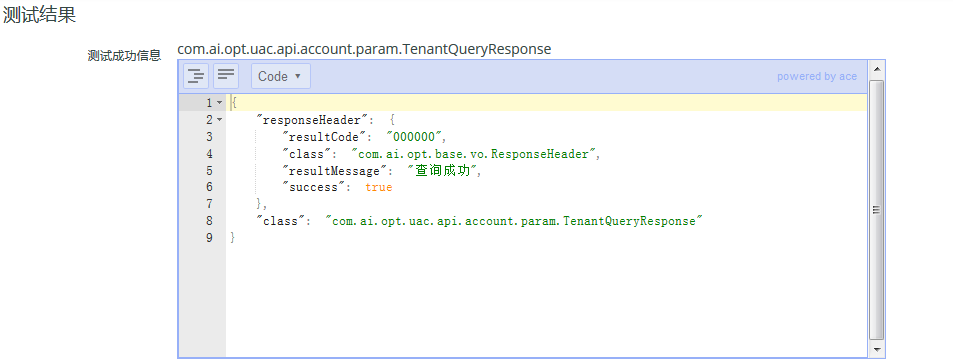


点击“dubbo测试”，进入服务测试页面，如下：

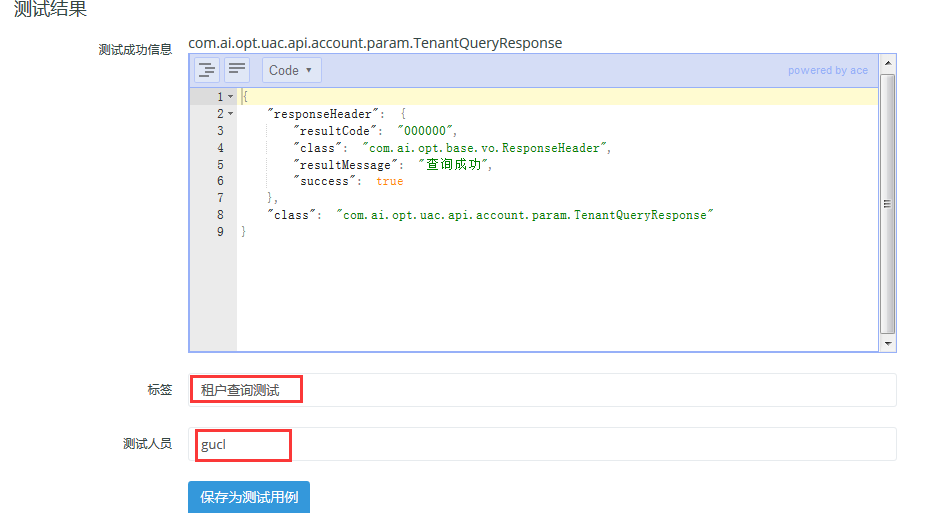




点击执行测试按钮，显示测试结果



测试结束后，可将测试结果保存为测试用例，以便以后测试。

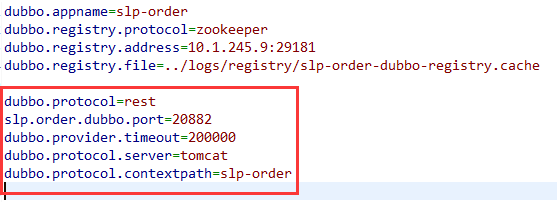


# Rest服务改造升级

## Rest服务改造方法

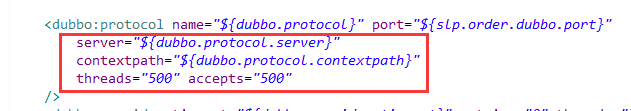
以Rest方式发布服务后，既可以通过rest来请求服务，也可以通过dubbo的方式来调用服务，并且服务只需要部署一份即可。改造方法如下：

1）修改classpath:/dubbo/dubbo.properties文件，如下：



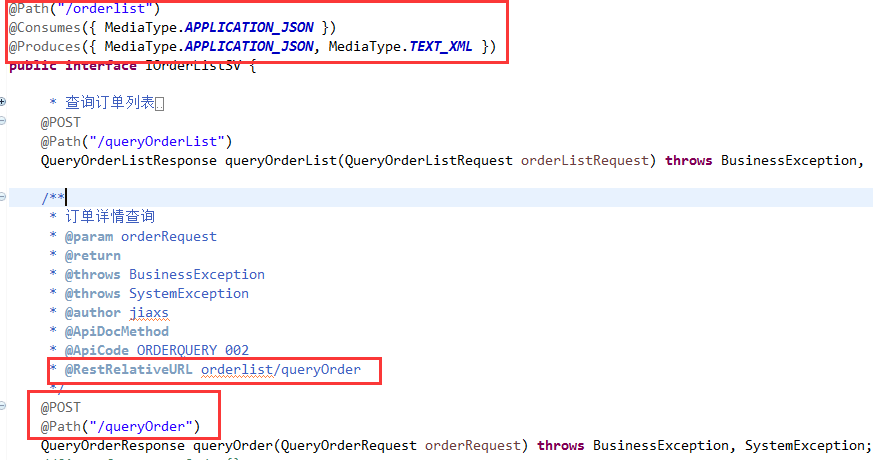
2）dubbo.protocol=dubbo改为rest，另外需添加dubbo.protocol.server和dubbo.protocol.contextpath选项。

其次，需修改classpath:dubbo/provider/dubbo-provider.xml文件，如下：



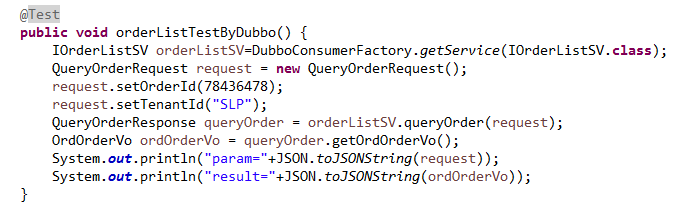
红框处是需要修改的添加的配置项，与上面dubbo.properties的值对应

3）修改服务接口的定义



## Dubbo方式调用Rest服务

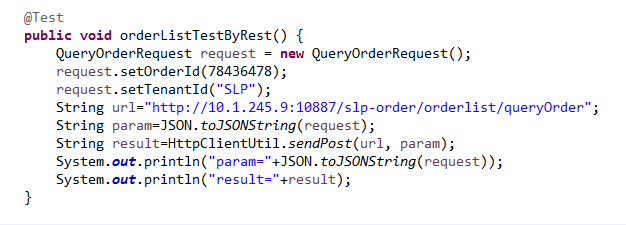
1）junit单元测试调用示例，与普通的dubbo方式调用一样：



2）在线服务测试方法：

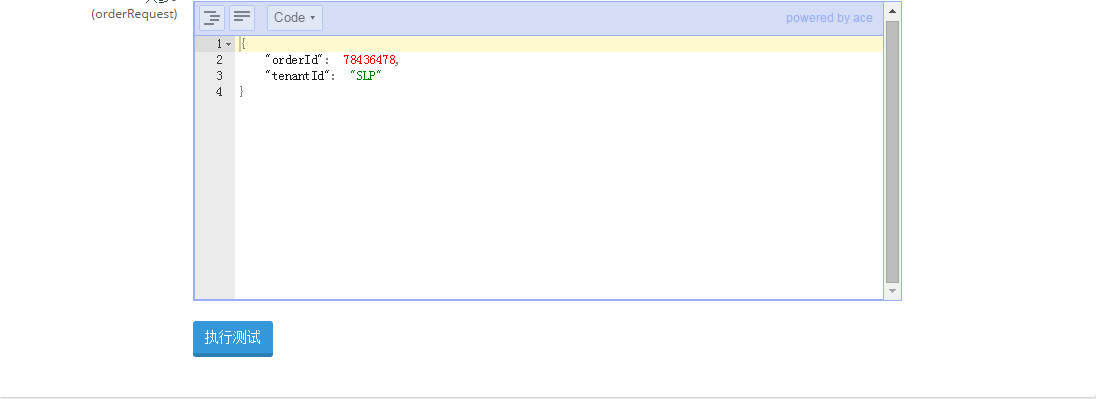
## HttpClient方式调用Rest服务

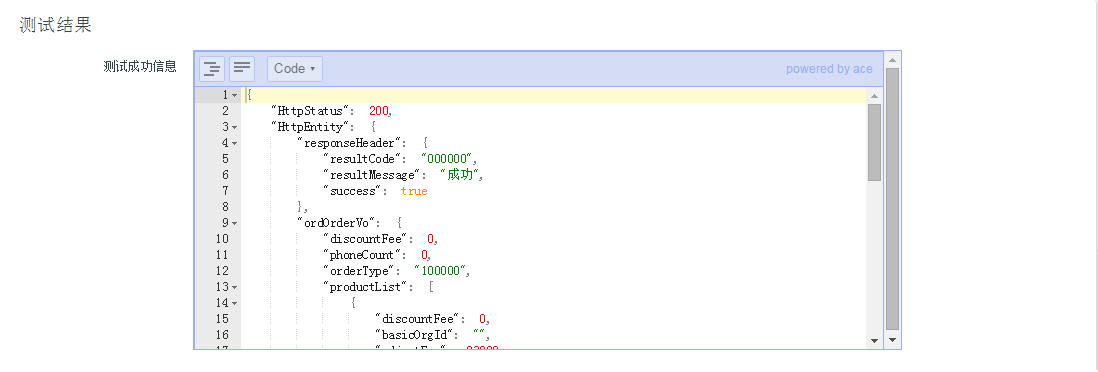
1）junit单元测试调用示例



2）在线服务测试方法：

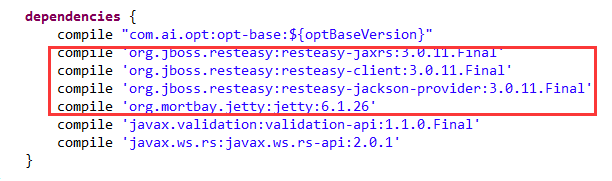




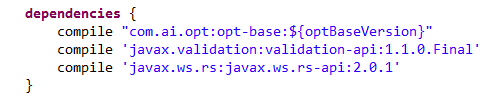


## Jar包冲突问题排除示例

1）在后场dubbo的父工程中，找到build.gradle文件：



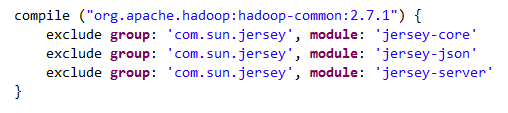
红框处的依赖要删除，防止客户端调用时，调用HttpClient出错。改为如下：



2）Web工程中，若有依赖hadoop-common包的，如下：



需改为（若不改，则无法以dubbo的方式调用rest服务）：

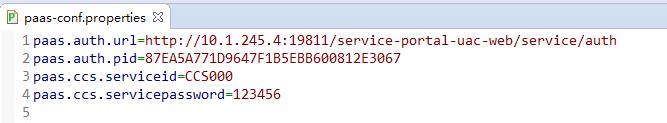


# opt-sdk组件SDK模式支撑

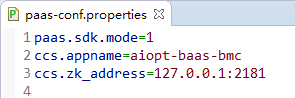
## opt-sdk两种模式支撑



## opt-sdk ipaas组件service模式



## opt-sdk ipaas组件sdk模式



paas.sdk.mode=1代表启动sdk模式

ccs.appname 规则：aiopt-产品线-模块名称，如aiopt-baas-bmc

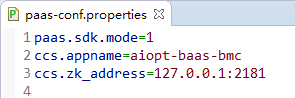
## opt-sdk ipaas组件两种模式切换

iPaaS 组件两种模式切换



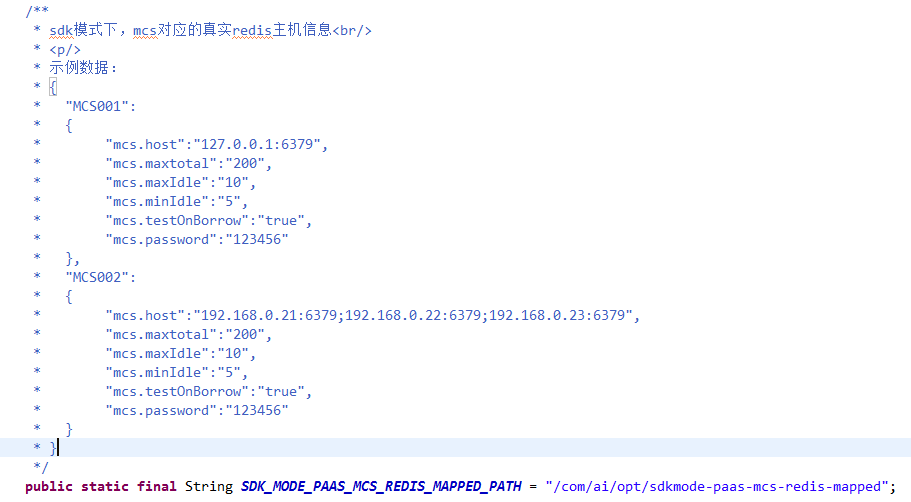
* opt-sdk屏蔽了两种模式下的差异，切换模式时，程序代码无需做任何改动
* 模式开关：paas.sdk.mode=1 启用sdk模式，否则为service模式。
* Servcie模式🡪SDK模式，改动相应的服务配置即可，代码不需改动
* SDK模式🡪Service模式，增加各ipaas组件对应的实际开源组件配置信息即可，代码不需改动。
* 目前支持SDK模式的组件：CCS、MCS、MDS、DSS

## CCS SDK模式配置信息



* 指定ccs.appname
* 指定Zookeeper地址，zk地址可以为一个或多个，多个之间英文逗号分隔
* 通过代码的方式把配置信息刷到zk里
* 后期通过界面的形式，维护各个产品模块的配置中心

## MCS SDK模式配置信息



在CCS的${ccs.appname}/com/x/sdkmode-paas-mcs-redis-mapped 路径，添加MDS对应的实际Redis配置

## MDS SDK模式 Kafka部署事项

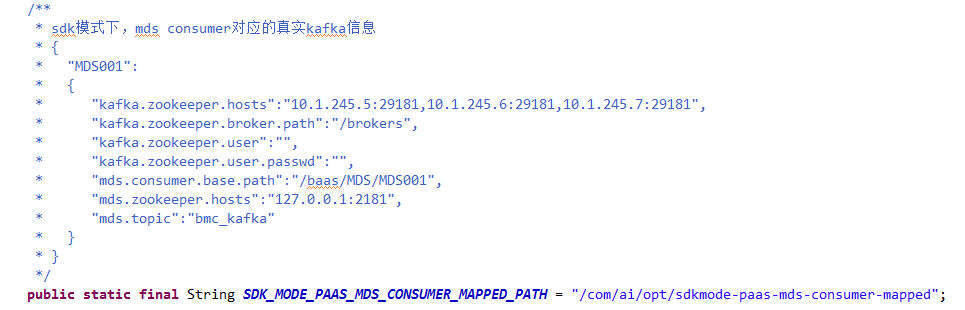
* 因iPaaS组件引用的kafka依赖为0.9.0.1，Kafka版本必须是kafka\_2.11-0.9.0.1
* Kafka服务端server.properties修改（单机示例）
* host.name=192.168.0.15
* advertised.host.name=192.168.0.15
* zookeeper.connect=192.168.0.15:2181

## MDS SDK模式 Sender配置信息



在CCS的${ccs.appname}/com/x/sdkmode-paas-mds-sender-mapped 路径，添加MDS对应的实际Kafka配置

## MDS SDK模式 Consumer配置信息



在CCS的${ccs.appname}/com/x/sdkmode-paas-mds-consumer-mapped 路径，添加MDS对应的实际Kafka配置

MDS消息多订阅方法：

|  |
| --- |
| //不指定consumerId的情况下，默认为"consumer"  IMessageConsumer msgConsumer= MDSClientFactory.*getConsumerClient*(mdsns, msgProcessorHandler);  //指定consumerId为"mds-consumer-mytopic1"  IMessageConsumer msgConsumer2= MDSClientFactory.*getConsumerClient*(mdsns, msgProcessorHandler,"mds-consumer-mytopic1"); |

## DSS SDK模式配置信息



在CCS的${ccs.appname}/com/x/sdkmode-paas-dss-mongodb-mapped 路径，添加MDS对应的实际MongoDB配置

# Docker环境下工程规范

## Shell文件格式要求

### Shell文件格式

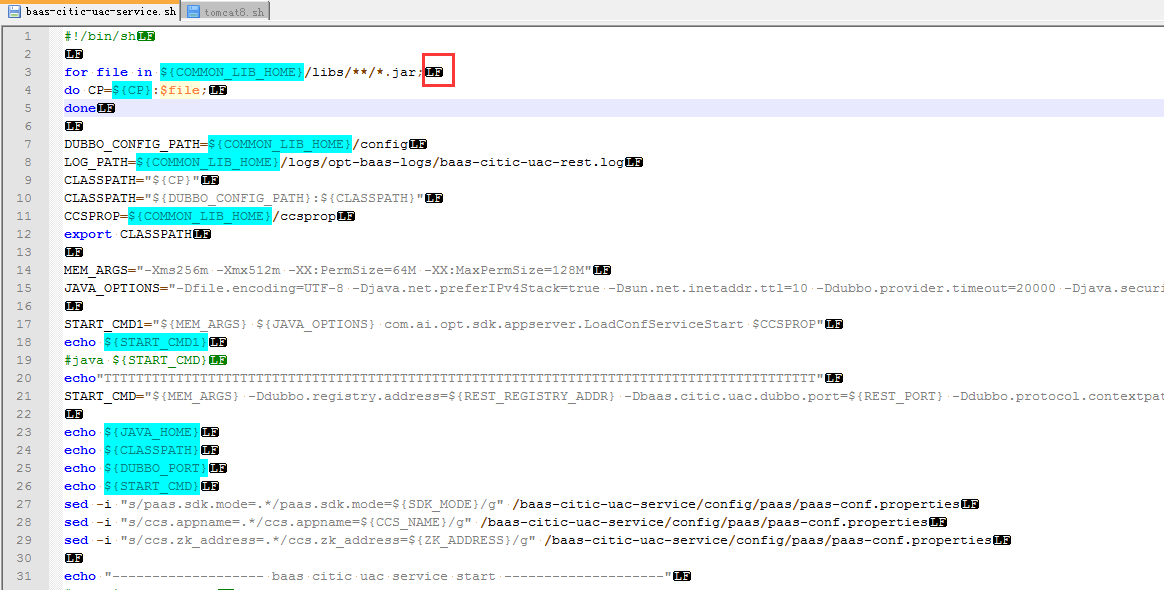
所有的shell脚本，必须是unix格式，不能是dos格式。如果是dos格式，运行时会报以下错误：

|  |
| --- |
| /bin/sh^M: bad interpreter: 没有那个文件或目录 |

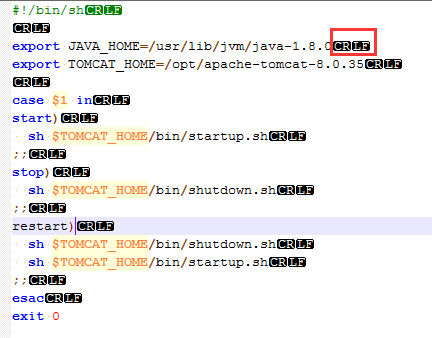
DOS格式的脚本文件时无法解释执行的，因为脚本文件的第一行是用来指定解释器的，Linux系统认为解释器是/bin/sh^M，而不是/bin/sh。

**判断方法一**，在windows环境下，用Notepad++打开shell脚本，点击菜单 视图🡪显示符号🡪显示所有符号，会显示隐藏的回车换行符，unix格式的回车换行符为 ，window格式的换行符为。必须确保shell脚本的回车换行符为，如果为 ，启动脚本会报错。

Unix格式的文件，如下图，行尾为“LF”：



DOS格式的文件：



**判断方法二**，在Linux下，执行如下命令：

|  |
| --- |
| *file 文件名称*  如下图：    执行结果有CRLF说明是DOS格式的，正常unix格式的结果显示如下： |

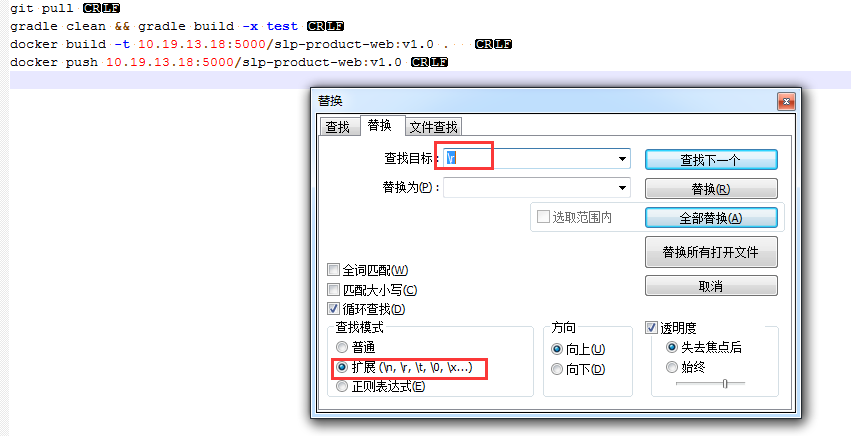
**判断方法三**：用vi编辑器打开文件查看

|  |
| --- |
| 1、编辑出错文件          vi filename  2、查看该格式（报错文件格式是DOS）          :set ff |

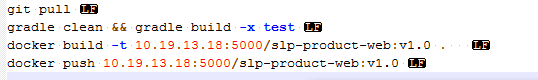
### Shell文件格式调整方法

Shell文件格式，如果为dos格式，调整为unix格式的方法主要有三种

**调整方法一：**在windows下，用Notepad++打开文件，点击视图🡪显示符号🡪显示所有字符，按下图进行替换操作



替换后



**调整方法二**：在linux环境下 用vi命令

|  |
| --- |
| 1、编辑出错文件  vi filename    2、查看该格式（报错文件格式是DOS）  :set ff    3、修改格式  :set ff=unix    4、保存退出  :wq!    记得输入“:wq!”保存退出 |

调整方法三：在linux环境下 用dos2unix命令

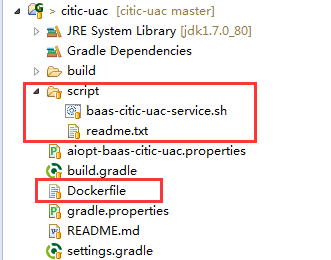
|  |
| --- |
| *dos2unix filename*  如果提示无dos2unix命令，请先安装 安装命令：  *yum install dos2unix* |

## 后场工程规范

### 后场父工程规范

后场工程以citic-uac为例。

父工程调整后目录：



如上图，在父工程中，添加如下目录和文件：

1）scirpt目录 主要存放docker镜像中的启动脚本、缓存刷新脚本、及readme.txt等

2）Dockerfile文件 Dockerfile文件为打镜像的文件

Dockerfile文件示例如下：

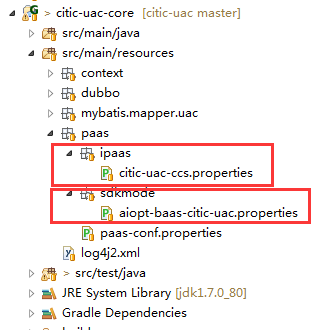
|  |
| --- |
| # Pull base image  FROM centos:7  MAINTAINER gucl<gucl@asiainfo.com>  #FROM docker.io/mysql:5.7  WORKDIR /  RUN yum install -y java-1.8.0-openjdk  # deploy user dubbo service  COPY ./build/libs /baas-citic-uac-service/libs/  COPY ./build/config /baas-citic-uac-service/config/  #mkdir logs path  RUN cd /baas-citic-uac-service && mkdir logs && cd /baas-citic-uac-service/logs && mkdir opt-baas-logs  ## copy start script  COPY ./script/baas-citic-uac-service.sh /baas-citic-uac-service.sh  RUN chmod 755 /baas-citic-uac-service.sh  # set start parameter for dubbo service  ENV COMMON\_LIB\_HOME /baas-citic-uac-service  # Expose ports.  EXPOSE 18881  # Define default command.  CMD ["./baas-citic-uac-service.sh"] |

其中readme.txt格式要求：

|  |
| --- |
| 1.编译打包  #先clean  gradle clean  #再打包  gradle build -x test  2.生成镜像  #在本地生成镜像 Dockerfile 默认采用的是openjdk8  docker build -t citic-uac:1.0 .  #在本地生成带私服前缀的镜像 每次打镜像前版本号要更新  docker build -t 10.1.245.4:5000/citic-uac:1.0 .  #将镜像推送到私服  docker push 10.1.245.4:5000/citic-uac:1.0  3. 运行镜像  #--net=host 表示为主机(host)模式 去掉该配置，默认为桥接(bridge)模式  #-e 代表需要设置的环境变量  docker run -d --name citic-uac-1.0 --net=host -p 18881:18881 -e "REST\_REGISTRY\_ADDR=10.1.130.84:39181" -e "REST\_PORT=18881" -e "CONTEXT\_PATH=citic-uac" -e "PROTOCOL=rest" -e "SDK\_MODE=1" -e "CCS\_NAME=aiopt-baas-citic-uac" -e "ZK\_ADDRESS=10.1.130.84:39181" 10.1.245.4:5000/citic-uac:1.0  #查看镜像启动日志  docker logs citic-uac-1.0  #进入容器，查看镜像内部的情况  docker exec -it citic-uac-1.0 /bin/bash  #删除运行的容器  docker rm -fv citic-uac-1.0  #=============更新日志===============#  \*2016-08-24 1.0版本  1）按需求，增加新模块xxx的开发  \*2016-07-04 0.9版本  1）opt-sdk统一升级为2.1-SNAPSHOT。  2）为满足中信项目的需要，CCS、MCS、MDS、DSS组件 以iPaaS SDK的方式进行调用，不采用ipaas申请服务的方式调用。其他项目仍按照ipaas服务申请的方式调用  3）升级后，原先的代码无需做任何更改，只需更改paas-conf.properties，并往CCS中添加相应的配置即可。中信项目 kafka的版本必须是kafka\_2.11-0.9.0.1，以保持和ipaas sdk的兼容  4）加入分布式任务DTS组件 |

### 配置文件备份规范

配置文件放到子工程src/main/resources/paas目录下，如下图：



|  |  |  |
| --- | --- | --- |
| ipaas模式 | 配置文件备份路径 | 说明 |
| ipaas平台模式 | src/main/resources/paas/ipaas/xxx-ccs.properties | xxx为对应的产品线+模块名称，如citic-uac-ccs.properties |
| SDK模式 | src/main/resources/paas/sdkmode/xxx.properties | xxx与paas-conf.properties中的ccs.appname对应 |

ipaas平台模式下的配置文件（从ipaas门户导出）示例：

|  |
| --- |
| #Configurationcenterconfigurationinformation  #FriAug2608:13:42UTC2016  /com/ai/opt/db-conf={"opt-uac-db"\:{"driverClassName"\:"com.mysql.jdbc.Driver","jdbcUrl"\:"jdbc\:mysql\://10.1.235.245\:31306/dev\_baas\_uacdb1?useUnicode\=true&characterEncoding\=UTF-8","username"\:"uacusr01","password"\:"uacusr01\_123","autoCommit"\:"true","connectionTimeout"\:"30000","idleTimeout"\:"600000","maxLifetime"\:"1800000","maximumPoolSize"\:"10"}} |

sdkmode模式下的配置文件（从运维平台导出）示例

|  |
| --- |
| /com/x/sdkmode-paas-mcs-redis-mapped={"MCS005": {"mcs.host":"10.1.130.84:46379", "mcs.maxtotal":"200","mcs.maxIdle":"10","mcs.minIdle":"5", "mcs.testOnBorrow":"true", "mcs.password":""} }  /com/ai/opt/db-conf={"opt-uac-db": { "driverClassName":"com.mysql.jdbc.Driver","jdbcUrl":"jdbc:mysql://10.1.130.84:3306/dev\_baas\_uacdb1?useUnicode=true&characterEncoding=UTF-8", "username":"root", "password":"123456", "autoCommit":"true","connectionTimeout":"30000","idleTimeout":"600000", "maxLifetime":"1800000","maximumPoolSize":"10"} }  /com/ai/opt/paas-cachens-mcs-mapped={"com.ai.opt.uac.sso.unicache":"MCS005","com.ai.opt.uac.register.cache":"MCS005","com.ai.opt.uac.retakepassword.cache":"MCS005","com.ai.opt.uac.updateemail.cache":"MCS005","com.ai.opt.uac.updatephone.cache":"MCS005","com.ai.opt.uac.updatepassword.cache":"MCS005","com.ai.opt.uac.login.cache":"MCS005","com.ai.opt.uac.bandemail.cache":"MCS005","com.ai.opt.uni.session.sessionclient.uacweb":"MCS005"}  /opt\_uac\_web\_index\_url=http://10.1.235.245:14101/baas-pt  /phone\_send\_ip\_maxno=10  /citic\_help\_index\_url=http://10.1.235.245:14125/citic-uac  /baas\_op\_index\_url=http://10.1.235.245:14105/baas-op  /email\_verifycode\_send\_maxtime=60  /email\_send\_ip\_maxno=10  /phone\_ip\_send\_overtime=300  /citic\_yun\_mall\_index\_url=http://10.1.235.245:14125/citic-uac  /citic\_index\_url=http://10.1.235.245:14125/citic-uac  /picture\_verifycode\_overtime=600  /phone\_verifycode\_send\_maxtime=60  /phone\_verifycode\_overtime=300  /email\_ip\_send\_overtime=300  /email\_verifycode\_overtime=1800 |

### 后场镜像打包步骤

参照readme.txt中的步骤，打包镜像

|  |
| --- |
| 1.编译打包  #一定要先clean  gradle clean  #然后再打包  gradle build -x test  2.生成镜像  #在本地生成镜像 Dockerfile 默认采用的是openjdk8  docker build -t citic-uac:1.0 .  #在本地生成带私服前缀的镜像  docker build -t 10.1.245.4:5000/citic-uac:1.0 . (每次打镜像前版本号要更新)  #将镜像推送到私服  docker push 10.1.245.4:5000/citic-uac:1.0  3. 运行镜像  #--net=host 表示为主机(host)模式 去掉该配置，默认为桥接(bridge)模式  #-e 代表需要设置的环境变量  docker run -d --name citic-uac-1.0 --net=host -p 18881:18881 -e "REST\_REGISTRY\_ADDR=10.1.130.84:39181" -e "REST\_PORT=18881" -e "CONTEXT\_PATH=citic-uac" -e "PROTOCOL=rest" -e "SDK\_MODE=1" -e "CCS\_NAME=aiopt-baas-citic-uac" -e "ZK\_ADDRESS=10.1.130.84:39181" 10.1.245.4:5000/citic-uac:1.0  #查看镜像启动日志  docker logs citic-uac-1.0  #进入容器，查看镜像内部的情况  docker exec -it citic-uac-1.0 /bin/bash  #删除运行的容器  docker rm -fv citic-uac-1.0  #=============更新日志===============#  \*2016-08-24 1.0版本  1）按需求，增加新模块xxx的开发  \*2016-07-04 0.9版本  1）opt-sdk统一升级为2.1-SNAPSHOT。  2）为满足中信项目的需要，CCS、MCS、MDS、DSS组件 以iPaaS SDK的方式进行调用，不采用ipaas申请服务的方式调用。其他项目仍按照ipaas服务申请的方式调用  3）升级后，原先的代码无需做任何更改，只需更改paas-conf.properties，并往CCS中添加相应的配置即可。中信项目 kafka的版本必须是kafka\_2.11-0.9.0.1，以保持和ipaas sdk的兼容  4）加入分布式任务DTS组件 |

### Script目录新增文件

1. Service.json

对于自己的工程，黄色的部分是需要修改的部分

{

"kind": "Service",

"apiVersion": "v1",

"metadata": {

"name": "amc",

"namespace": "unit-test"

},

"spec": {

"ports": [{

"port": 10774,

"targetPort": 10774,

"nodePort":30774

}],

"selector": {

"app": "amc-pod"

},

"type":"NodePort"

}

}

2、ReplicationController.json

对于自己的工程，黄色的部分是需要修改的部分

{

"apiVersion": "v1",

"kind": "ReplicationController",

"metadata":{

"name": "amc-rc",

"namespace": "unit-test"

},

"spec":{

"replicas": 1,

"selector":{

"app": "amc-pod"

},

"template":{

"metadata":{

"name": "amc",

"labels":{

"app": "amc-pod"

}

},

"spec":{

"containers":[{

"name": "amc",

"image": "10.1.234.164:5000/baas-amc:0.1",

"imagePullPolicy": "IfNotPresent",

"ports":[{

"containerPort": 10774

}],

"env":[{

"name": "REST\_REGISTRY\_ADDR",

"value":"10.1.130.84:39181"

},{

"name": "REST\_PORT",

"value":"10774"

},{

"name": "CONTEXT\_PATH",

"value":"baas-amc"

},{

"name": "PROTOCOL",

"value":"rest"

},{

"name": "CCS\_NAME",

"value":"aiopt-baas-amc"

},{

"name": "ZK\_ADDRESS",

"value":"10.1.130.84:39181"

},{

"name": "SDK\_MODE",

"value":"1"

}]

}]

}

}

}

}

3、deploy shell

对于自己的工程，黄色的部分是需要修改的部分

#!/bin/sh

echo "create amc service on k8s cluster...."

exist=`kubectl get svc --namespace=unit-test | grep amc |grep -v amc-query`

if [ "$exist" = "" ]

then

echo "amc service not exist! NOT need delete "

else

kubectl delete svc amc --namespace=unit-test

fi

kubectl create -f ./script/amc-service.json

echo "amc service done"

echo "create amc ReplicationController on k8s cluster...."

exist=`kubectl get ReplicationController --namespace=unit-test | grep "amc-rc" `

if [ "$exist" = "" ]

then

echo "amc ReplicationController not exist! NOT need delete"

else

kubectl delete ReplicationController amc-rc --namespace=unit-test

fi

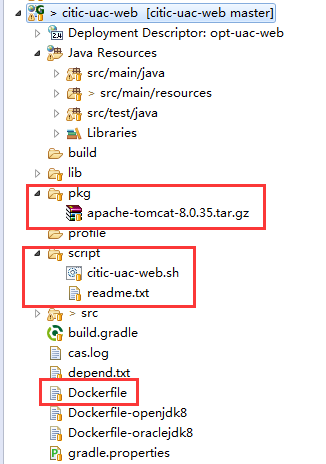
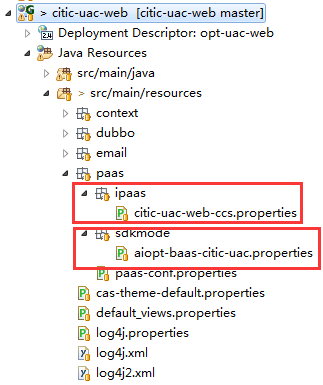
kubectl create -f ./script/amc-ReplicationController.json

echo "amc ReplicationController done"

## 前端web工程规范

### web工程规范

前端web工程以citic-uac-web为例：

如上图，web工程中，添加如下目录和文件

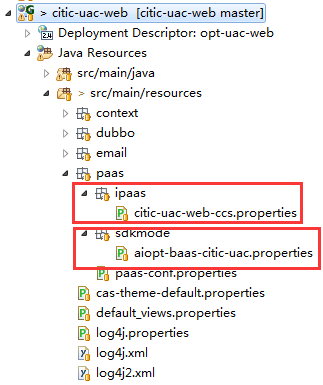
1）pkg目录 主要存放开源软件，如tomcat、jdk等

2）scirpt目录 主要存放docker镜像中的启动脚本及readme.txt等

3）Dockerfile文件 Dockerfile文件为打镜像的文件

### 配置文件规范

配置文件放到子工程src/main/resources/paas目录下，如下图：



|  |  |  |
| --- | --- | --- |
| ipaas模式 | 配置文件备份路径 | 说明 |
| ipaas平台模式 | src/main/resources/paas/ipaas/xxx-ccs.properties | xxx为对应的产品线+模块名称，如citic-uac-web-ccs.properties |
| SDK模式 | src/main/resources/paas/sdkmode/xxx.properties | xxx与paas-conf.properties中的ccs.appname对应 |

ipaas平台模式下的配置文件（从ipaas门户导出）示例：

|  |
| --- |
| #Configurationcenterconfigurationinformation  #FriAug2608:13:58UTC2016  /picture\_verifycode\_overtime=600  /phone\_verifycode\_overtime=300  /slp\_uac\_web\_index\_url=http\://10.1.245.8\:14105/slp-mall  /opt\_uac\_web\_index\_url=http\://10.1.235.245\:14101/baas-pt  /com/ai/opt/paas-service-pwd-mapped={"MCS005"\:"123456"}  /email\_verifycode\_overtime=1800  /phone\_verifycode\_send\_maxtime=60  /email\_verifycode\_send\_maxtime=60  /email\_ip\_send\_overtime=300  /com/ai/opt/paas-cachens-mcs-mapped={"com.ai.opt.uac.sso.unicache"\:"MCS005","com.ai.opt.uac.register.cache"\:"MCS005","com.ai.opt.uac.retakepassword.cache"\:"MCS005","com.ai.opt.uac.updateemail.cache"\:"MCS005","com.ai.opt.uac.updatephone.cache"\:"MCS005","com.ai.opt.uac.updatepassword.cache"\:"MCS005","com.ai.opt.uac.login.cache"\:"MCS005","com.ai.opt.uac.bandemail.cache"\:"MCS005","com.ai.opt.uni.session.sessionclient.uacweb"\:"MCS005"}  /email\_send\_ip\_maxno=10  /baas\_op\_index\_url=http\://10.1.235.245\:14105/baas-op  /phone\_ip\_send\_overtime=300  /com/x/sdkmode-paas-mcs-redis-mapped={"MCS005"\:{"mcsHost"\:"10.1.130.84\:16379","mcsMaxtotal"\:"200","mcsMaxIdle"\:"10","mcsMinIdle"\:"5","mcsTestOnBorrow"\:"true","mcsPassword"\:""}}  /phone\_send\_ip\_maxno=10 |

sdkmode模式下的配置文件（从运维平台导出）示例

|  |
| --- |
| /com/x/sdkmode-paas-mcs-redis-mapped={"MCS005": {"mcs.host":"10.1.130.84:46379", "mcs.maxtotal":"200","mcs.maxIdle":"10","mcs.minIdle":"5", "mcs.testOnBorrow":"true", "mcs.password":""} }  /com/ai/opt/db-conf={"opt-uac-db": { "driverClassName":"com.mysql.jdbc.Driver","jdbcUrl":"jdbc:mysql://10.1.130.84:3306/dev\_baas\_uacdb1?useUnicode=true&characterEncoding=UTF-8", "username":"root", "password":"123456", "autoCommit":"true","connectionTimeout":"30000","idleTimeout":"600000", "maxLifetime":"1800000","maximumPoolSize":"10"} }  /com/ai/opt/paas-cachens-mcs-mapped={"com.ai.opt.uac.sso.unicache":"MCS005","com.ai.opt.uac.register.cache":"MCS005","com.ai.opt.uac.retakepassword.cache":"MCS005","com.ai.opt.uac.updateemail.cache":"MCS005","com.ai.opt.uac.updatephone.cache":"MCS005","com.ai.opt.uac.updatepassword.cache":"MCS005","com.ai.opt.uac.login.cache":"MCS005","com.ai.opt.uac.bandemail.cache":"MCS005","com.ai.opt.uni.session.sessionclient.uacweb":"MCS005"}  /opt\_uac\_web\_index\_url=http://10.1.235.245:14101/baas-pt  /phone\_send\_ip\_maxno=10  /citic\_help\_index\_url=http://10.1.235.245:14125/citic-uac  /baas\_op\_index\_url=http://10.1.235.245:14105/baas-op  /email\_verifycode\_send\_maxtime=60  /email\_send\_ip\_maxno=10  /phone\_ip\_send\_overtime=300  /citic\_yun\_mall\_index\_url=http://10.1.235.245:14125/citic-uac  /citic\_index\_url=http://10.1.235.245:14125/citic-uac  /picture\_verifycode\_overtime=600  /phone\_verifycode\_send\_maxtime=60  /phone\_verifycode\_overtime=300  /email\_ip\_send\_overtime=300  /email\_verifycode\_overtime=1800 |

### 前端web打包步骤

参照readme.txt中的步骤，打包镜像

|  |
| --- |
| 1.编译打包  #一定要先clean  gradle clean  #然后再打包  gradle build -x test  2.生成镜像  #Dockerfile 默认采用的是openjdk8，  #如果要用官方的orackejdk8 需手工将jdk-8u101-linux-x64.tar.gz 拷贝到pkg目录下。  docker build -t citic-uac-web:12.2 .  #在本地生成带私服前缀的镜像 (每次打镜像前版本号要更新)  docker build -t 10.1.234.164:5000/citic-uac-web:12.2 .  #将镜像推送到私服  docker push 10.1.234.164:5000/citic-uac-web:12.2  3. 运行镜像  #--net=host 表示为主机(host)模式 去掉该配置，默认为桥接(bridge)模式  #-e 代表需要设置的环境变量  docker run -d --name citic-uac-web-12.2 --net=host -p 14125:8080 -e "casServerLoginUrl=http://10.1.234.163:14125/login" -e "casServerUrlPrefix=http://10.1.234.163:14125" -e "serverName=http://10.1.234.163:14125" -e "logOutServerUrl=http://10.1.234.163:14125/logout" -e "logOutBackUrl=http://10.1.234.163:14125" -e "casServerLoginUrl\_Inner=http://10.1.234.163:14125/login" -e "casServerUrlPrefix\_Inner=http://10.1.234.163:14125" -e "serverName\_Inner=http://10.1.234.163:14125" -e "logOutServerUrl\_Inner=http://10.1.234.163:14125/logout" -e "logOutBackUrl\_Inner=http://10.1.234.163:14125" -e "innerDomains=sso.citicdao.com,billingwebui.citicdao.com,billing.citicdao.com,user.citicdao.com,webui.citicdao.com,aliapt.citicdao.com,scapt.cticdao.com,service.citicdao.com" -e "ZK\_ADDR=10.1.130.84:39181" -e "DUBBO\_REGISTRY=10.1.130.84:39181" -e "DUBBO\_PROTOCOL=dubbo" -e "DUBBO\_PORT=10881" 10.1.234.164:5000/citic-uac-web:12.2  #查看镜像启动日志  docker logs citic-uac-web-12.2  #进入容器，查看镜像内部的情况  docker exec -it citic-uac-web-12.2 /bin/bash  #删除运行的容器  docker rm -fv citic-uac-web-12.2  #=============更新日志===============#  \*2016-08-24 12.2版本  1）按需求，增加新模块xxx的开发  \*2016-07-04 12.1版本  1）opt-sdk统一升级为2.1-SNAPSHOT。  2）为满足中信项目的需要，CCS、MCS、MDS、DSS组件 以iPaaS SDK的方式进行调用，不采用ipaas申请服务的方式调用。其他项目仍按照ipaas服务申请的方式调用  3）升级后，原先的代码无需做任何更改，只需更改paas-conf.properties，并往CCS中添加相应的配置即可。中信项目 kafka的版本必须是kafka\_2.11-0.9.0.1，以保持和ipaas sdk的兼容  4）加入分布式任务DTS组件 |

# 前端Web UI编写规范

1）所有字体统一为“微软雅黑”

2）所有的按钮，统一为浅蓝色（以长虹物联网为例）

3）所有的查询页面，查询条件和查询结果分开。



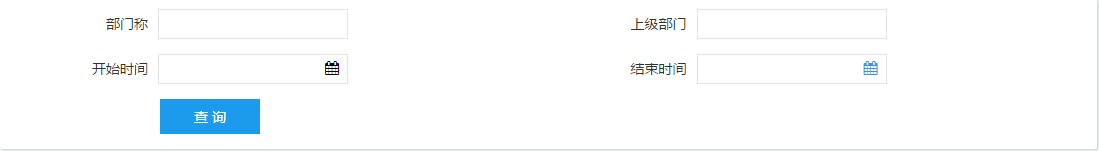
4）查询页面的按钮名称统一为“查询”。

查询按钮位置约定：

如果查询条件少于等于两个时，查询按钮放置在右边；



如果查询条件多于两个时，查询按钮放在最后一行查询条件的下方，居左。



如果查询条件有动态显示的情况，查询按钮放在最后一行查询条件的下方，居左。

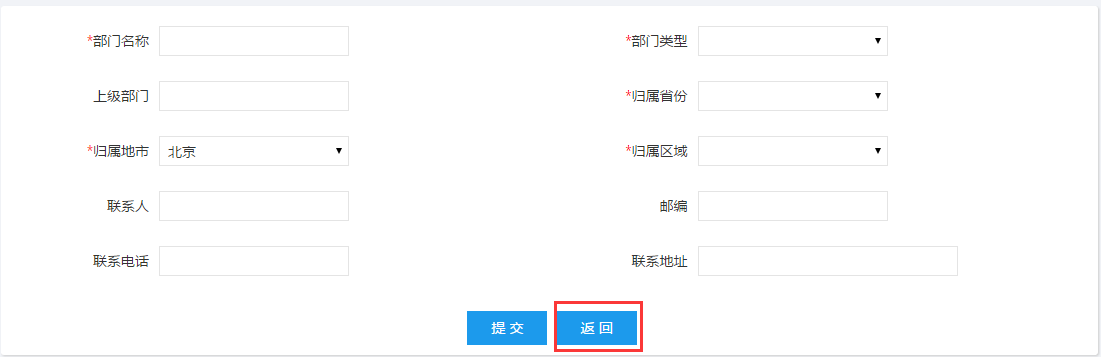


高级搜索展开时：

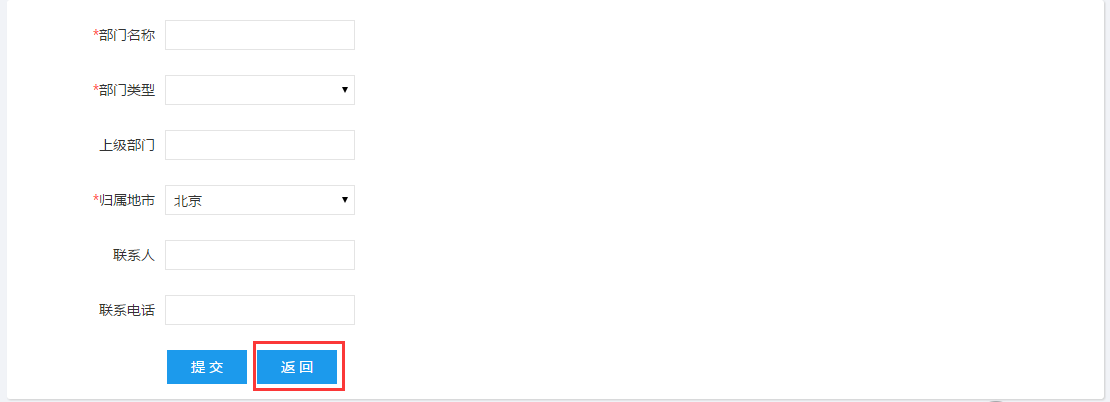


5）所有的“返回”按钮，居页面下方。

如果页面分两列，则位于下方居中；



如果页面只有一列，则位于下方居左。



6）所有的新增和删除按钮，居页面右部：



7）“：”的问题，所有的查询页面和编辑框之间，不要“：”号

8）所有的数据校验，采用统一的校验框架。

校验效果要求：

a)鼠标失去焦点时，自动校验当前元素

b)提交表单时，自动校验所有的校验项目

c)所有的校验信息为红色字体



页面分两列的情况，注意，每行所在的li，要加class=“col-md-6”



9）表头内字体和表内字体，由UED设计的统一风格，不得擅自更改样式。

10）表格内的操作列，统一用超链接的方式，不使用button，如下图：



11）所有的查询条件，如果不是必填项，前面不要加“\*”

12）表格中某一列过宽时，采用UED设计的写法，超出部分”…”，鼠标悬浮时，会有提示

13）数据校验时，关于字符长度的问题，由于mysql5.0以上的版本，varchar(6)的数据类型，既可以放6个英文字母，也可以放6个汉字。因此，前台数据校验时，一个英文字符或一个中文汉字，长度都视为1即可。用校验规则中的minlength和maxlength控制即可