

SMARTAS 架构文档

BY 烽火云创平台组

本文件及其附件含有公司的保密信息，仅限于发送给项目中相关开发相关个人或群组。禁止任何其他人以任何形式使用（包括但不限于全部或部分地泄露、复制、或散发）本文档中的信息。

目录

Introduction	1.1
SmartAs框架介绍	1.2
环境篇	1.3
Java环境	1.3.1
Git安装	1.3.2
启动应用	1.3.3
小结	1.3.4
基础篇	1.4
Annotation(注解)	1.4.1
Model(实体模型)	1.4.2
SQL(sql扩展)	1.4.3
Dao(数据访问接口)	1.4.4
Service(服务访问接口)	1.4.5
Resource(Rest资源)	1.4.6
Exception(异常)	1.4.7
Util(工具类)	1.4.8
小结	1.4.9
开发篇	1.5
Dao	1.5.1
Service	1.5.2
UI	1.5.3
小结	1.5.4
配置篇	1.6
基础配置	1.6.1
环境配置	1.6.2
缓存配置	1.6.3
小结	1.6.4
进阶篇	1.7
5.1. 通用功能	1.7.1
批处理	1.7.2
数据字典	1.7.3
定时任务	1.7.4
异步服务框架	1.7.5
Lookup管理	1.7.6
菜单管理	1.7.7
国际化信息管理	1.7.8
导入管理	1.7.9

导出管理	1.7.10
日志服务	1.7.11
富文本	1.7.12
规则引擎	1.7.13
附件上传下载	1.7.14
组织机构	1.7.15
首页	1.7.16
Session管理	1.7.17
消息总线	1.7.18
DMN	1.7.19
注册表	1.7.20
5.2. 用户和权限管理	1.7.21
用户管理	1.7.22
群组管理	1.7.23
角色管理	1.7.24
属性组管理	1.7.25
权限验证	1.7.26
数据权限管理	1.7.27
5.3. 工作流	1.7.28
流程业务接入规范	1.7.29
流程部署	1.7.30
流程创建	1.7.31
流程审批	1.7.32
待办列表	1.7.33
已办列表	1.7.34
草稿功能规范定义	1.7.35
草稿箱	1.7.36
功能事件	1.7.37
待办邮箱集成	1.7.38
待办对外集成API	1.7.39
5.4. DevOps	1.7.40
骨架代码自动生成	1.7.41
项目创建打包	1.7.42
服务API在线文档	1.7.43
UI在线文档	1.7.44
业务操作在线文档	1.7.45
服务器管理	1.7.46
运维工具	1.7.47
缓存管理	1.7.48
版本信息	1.7.49

Dubbo集成	1.7.50
集群管理	1.7.51
小结	1.7.52
扩展篇	1.8
首页定制	1.8.1
用户管理扩展	1.8.2
登陆页定制	1.8.3
组织机构定制	1.8.4
小结	1.8.5
Web篇	1.9
xxx	1.9.1
xxx	1.9.2
xx	1.9.3
小结	1.9.4
FAQ	1.10
撰写	1.11
参考资料	1.12

SmartAs-Doc

SmartAs是集成业界流行和成熟的技术框架和良好用户体验的应用快熟开发框架

1.引言

不是简单的技术框架堆砌而成的脚手架, 而是提供高效的业务开发框架。
把应用系统从纷繁复杂的框架选型, 基础功能开发等重复劳动中解放出来, 从而聚焦业务本身的功能。
它的优秀之处并非原创, 它的原创之处并不优秀, 所以SmartAs横空出世。

2.设计原则

2.1 轻量

技术选型及架构上避免选择重型的框架和架构

2.2 前后端分离

前端完成UI相关的动作, 服务端只负责数据处理

2.3 Rest风格

标准的restful风格架构

2.4 约定俗成

约定优先, 规则优先(接口定义, 命名等等)。统一代码风格。

2.5 移动互联

2.6 支持分布式部署

2.7 面向接口

2.8 高内聚低耦合

SmartAs是集成业界流行和成熟的技术框架和良好用户体验的应用快熟开发框架。它具有以下特点：

- 轻量 : 技术选型及架构上避免选择重型的框架和架构
- 前后端分离 : 前端完成UI相关的动作, 服务端只负责数据处理
- Rest风格 : 标准的restful风格架构
- 约定俗成 : 约定优先, 规则优先(接口定义, 命名等等)。统一代码风格。
- 移动互联 :
- 支持分布式部署 :
- 面向接口 :
- 高内聚低耦合 :

1 背景

随着传统互联网, 移动互联网, 互联网+等互联网形态演进, 新技术层出不穷, 都为一个目标, 为适应瞬息万变的市
场。传统的企业应用业务越来越庞大和复杂, 而交付周期却越来越快。国内各大企业都在打造适应自己业务需求的应用框
架和平台。构建成熟合适的平台, 有利于整个系统的快速开发和良好实施, 也有利于企业内部技术体系的构建和资源的共
享。不是简单的技术框架堆砌而成的脚手架, 而是提供高效的业务开发平台。把应用系统从纷繁复杂的框架选型, 基础功
能开发, 系统集成等重复劳动中解放出来, 从而聚焦业务本身的功能。

2 简介

整个架构体系以轻量为指导原则, 大道至简的架构才能更顺应技术变革和满足不同行业应用的实施。传统企业框架UI
组件大多都枚举方式构建组件库, 在高速迭代快速变化的应用开发中, 将会使UI组件库逐渐变得臃肿和难以维护, UI组件
是受具体业务场景约束的, SmartAs框架采用React技术, 以组建组合方式开发新组建来构建组件库, 业务系统能够比较
容易开发定义组件, 并在行业应用复制。前后端分离的设计, 能够让业务系统比较容易的微服务化, 云服务化, 同一套业务
逻辑能够比较容易支持多终端。通过严格定义的API, 前后端可以独立的做技术, 业务的迭代升级。

links

- [目录](#)
- [下一章: 环境篇](#)

1 环境安装

欢迎来到SmartAs的世界, 让我们开始探索吧!

在本章中, 我们将讲述基于SmartAs的应用本地环境配置(包括JDK, Tomcat, Eclipse等), 以及如何配置项目信息。

第一步, 让我们先设置本地hosts (C:\Windows\System32\drivers\etc) 文件。

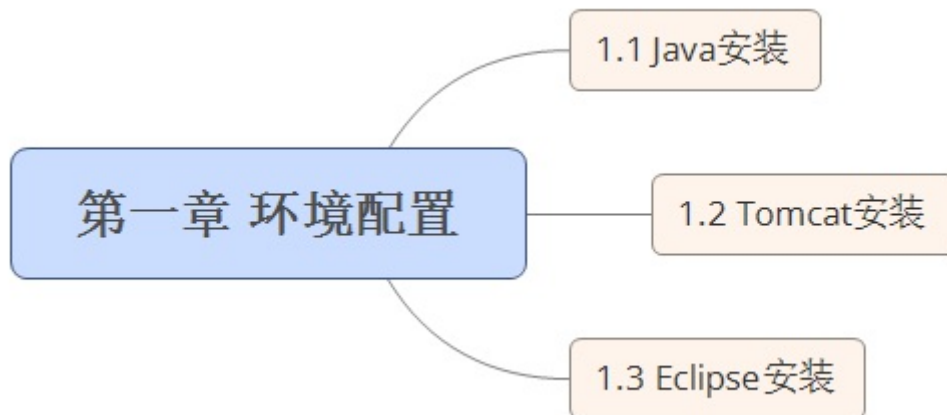
添加如下类容:

```
127.0.0.1    localhost.smartas.com
10.121.4.78  78.smartas.com
10.121.4.79  79.smartas.com
10.121.4.80  80.smartas.com
```

该域名源自于web工程中app.properties中session.domainName的设置 session.domainName将sessionid写入根域名

在项目开发环境下,开发人员可仍然遵循上述配置, 项目打包到测试或生产环境时, 可将session.domainName的内容设置为目标服务器的ip地址。

目录



links

- [目录](#)
- 下一节: [Java环境](#)

1 JAVA安装

1.要安装 JDK(JDK8+), 请转至 <http://java.sun.com/javase/downloads/index.jsp>。

安装JDK 选择安装目录 安装过程中会出现两次 安装提示 。第一次是安装 jdk , 第二次是安装 jre 。建议两个都安装在同一个java文件夹中的不同文件夹中,推荐路径 d:\Java\jdk8 下 (注意安装路径不要有空格)。一旦安装了 JDK, 您即可设置 JAVA_HOME。

3.要设置 JDK, 请右键单击“我的电脑”, 然后选择“属性”。

4.在“高级”选项卡上, 选择“环境变量”, 然后编辑 JAVA_HOME 以指向 JDK 所在的位置(例如:d:\Java\jdk8)。

5.添加Path路径, 系统变量→寻找 Path 变量→编辑

在变量值最后输入 d:\Java\jdk8\bin; (注意原来Path的变量值末尾有没有;号, 如果没有先输入;号再输入上面的代码)

2 Tomcat8安装

1.要安装Tomcat(8.x), 请转至 <http://tomcat.apache.org/>。

2.解压Tomcat,拷贝DevLoader-2.0.0.jar到tomcat里面的\lib\下面

3.Tomcat下的conf\server.xml 两处Connector 节点加上URIEncoding="UTF-8"。解决tomcat下的URL中文乱码问题

4.在Tomcat的conf\catalina.properties文件中添加本地代码路径:

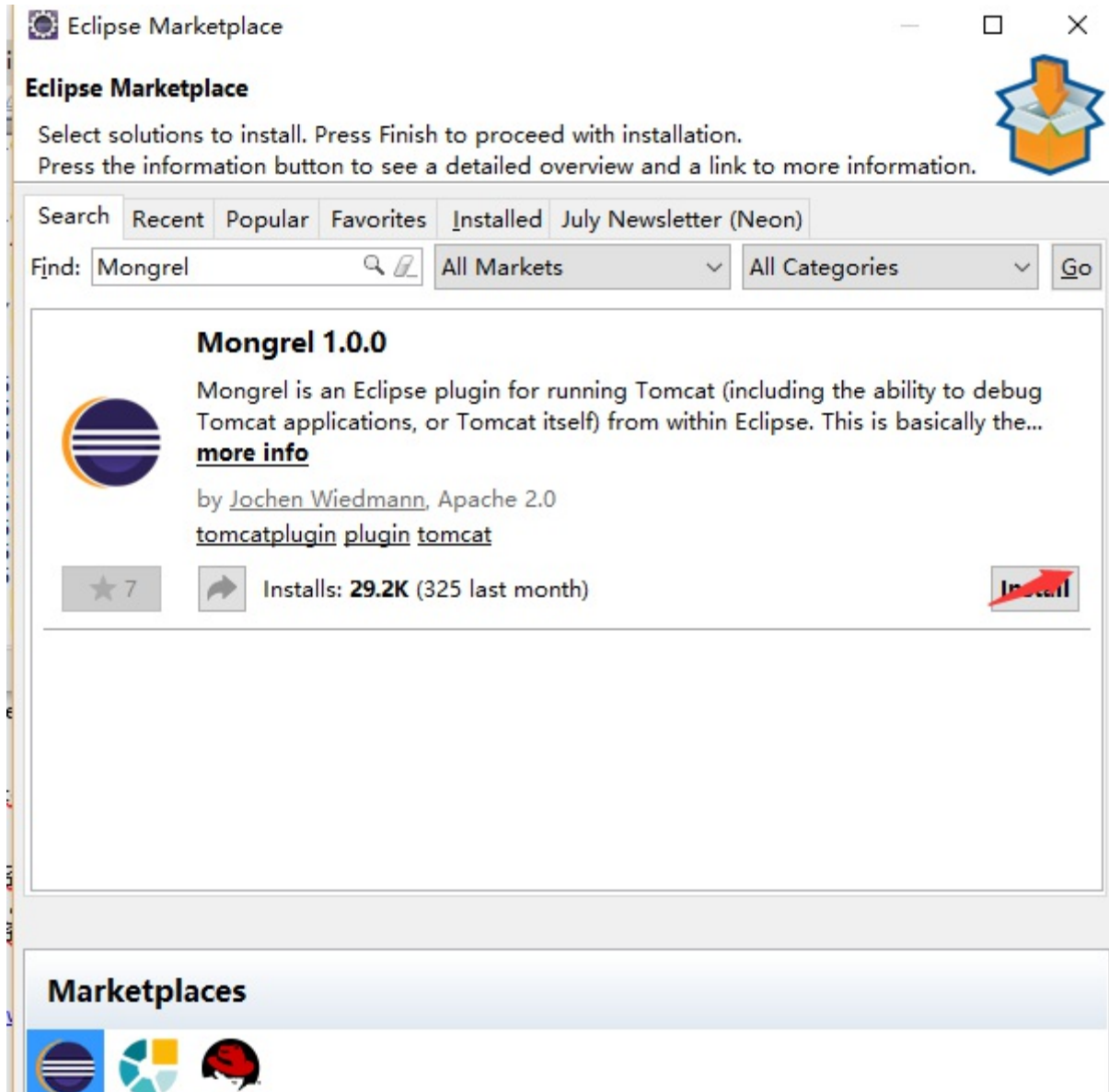
project_dir=F:\Projects (项目源码根目录, 注意反斜杠转义)

注:这个目录不要放在盘的跟目录下, 最好和示例中一致, 将来所有基于SmartAs的项目源码放在此目录下

Eclipse安装

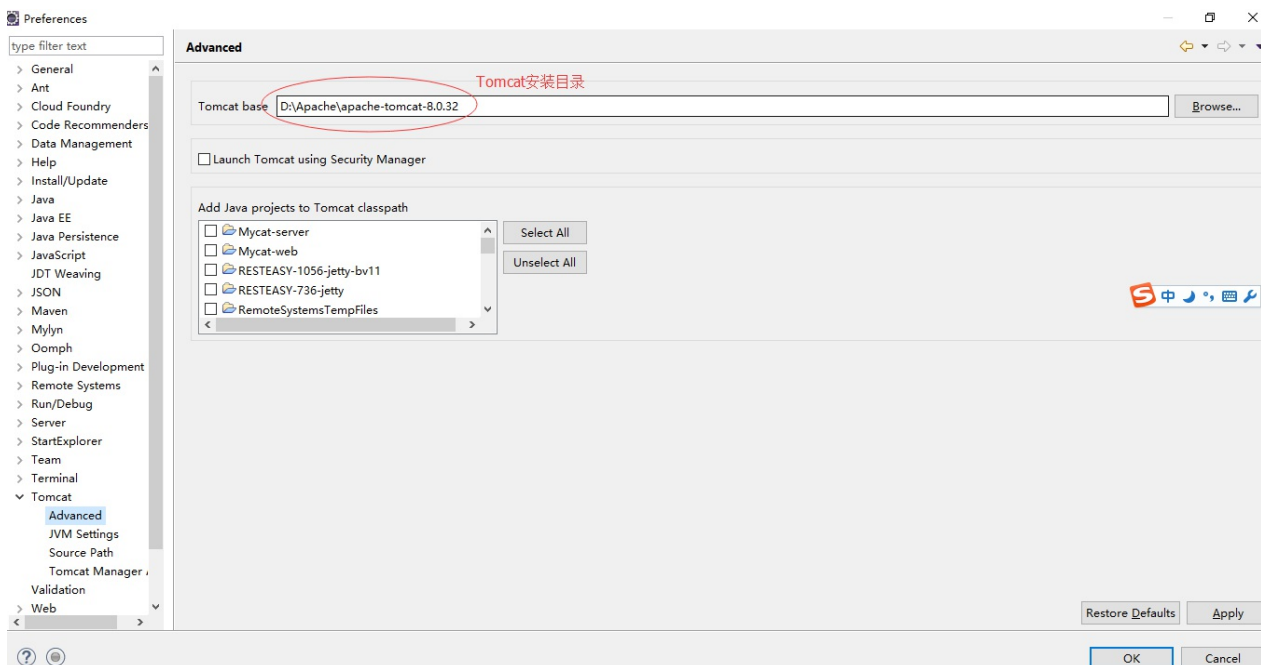
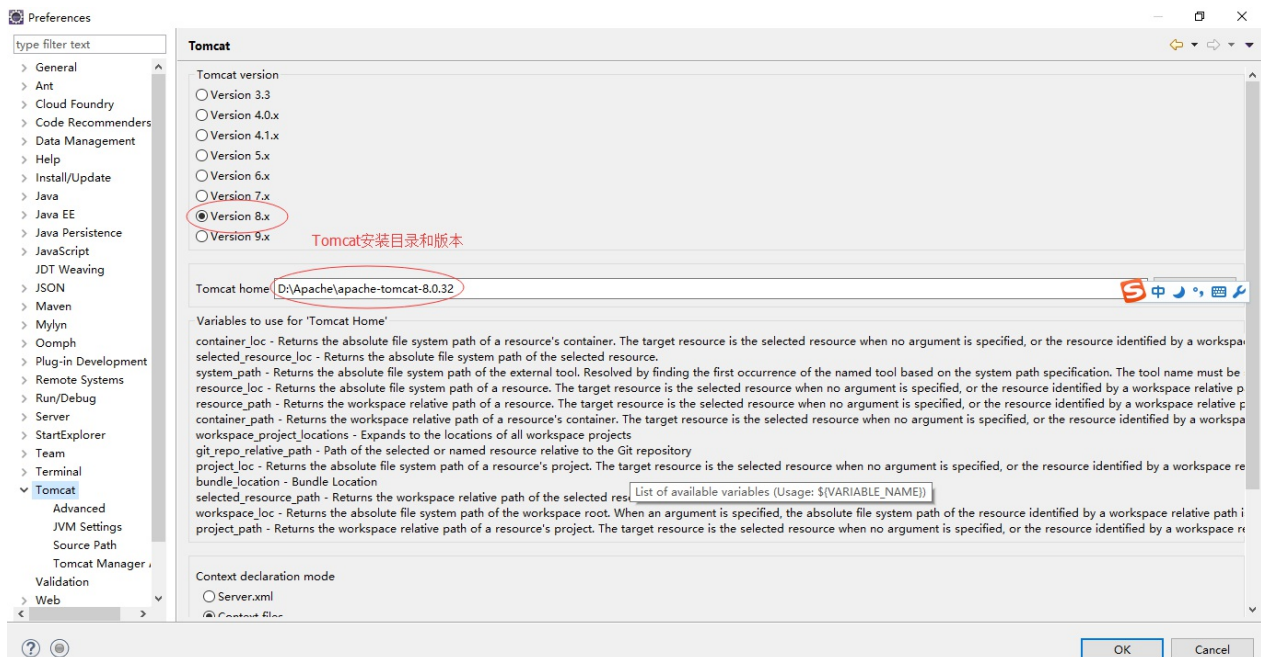
1.要安装Eclipse(Neon), 请转至<http://www.eclipse.org/downloads/>。

2.启动Eclipse, 配置Tomcat插件(Help->Eclipse Marketplace),搜索Mongrel插件



安装并重启Eclipse.

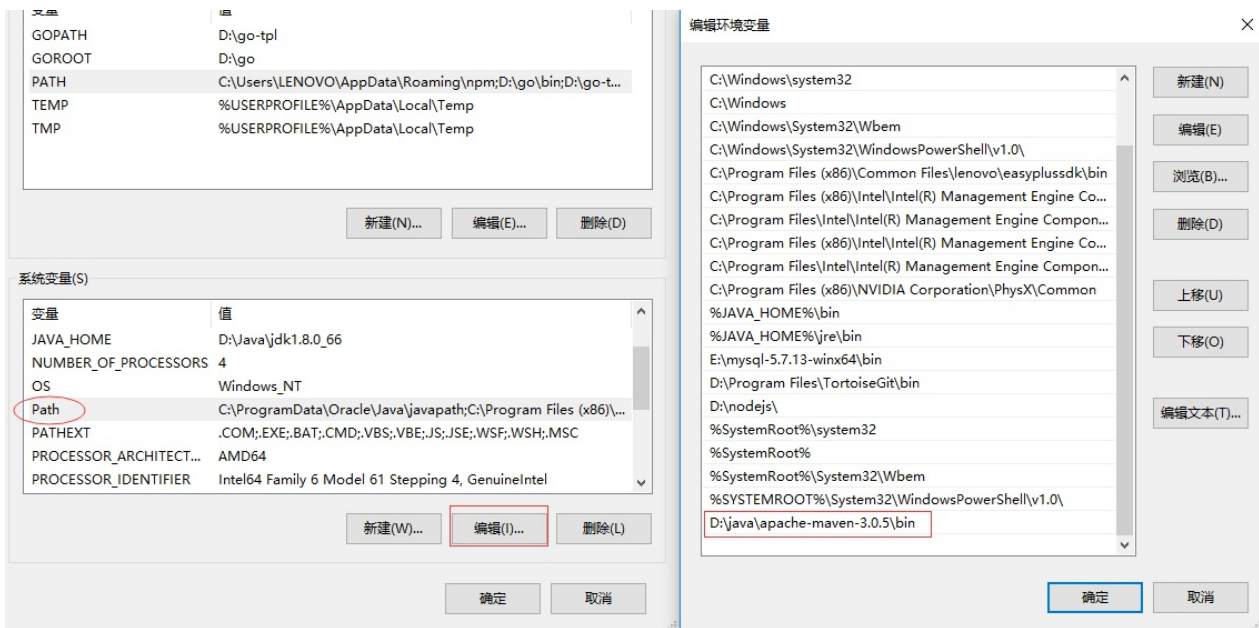
3.配置Tomcat插件(Window->Preferences)



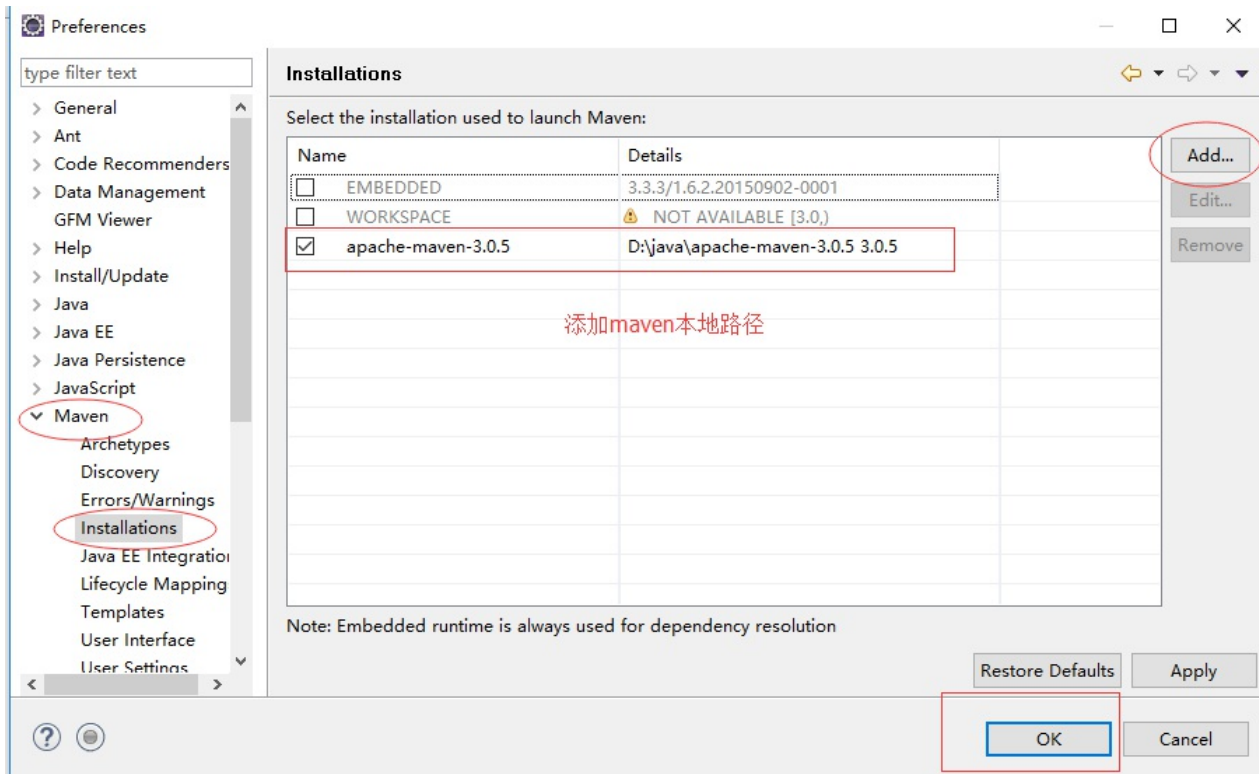
点击工具栏猫图标，启动Tomcat。

Maven安装

1. 安装Maven3.0.5版本，安装路径为<https://archive.apache.org/dist/maven/maven-3/3.0.5/binaries/>
2. 配置环境变量，请右键单击“我的电脑”，然后选择“属性”，在“高级”选项卡上，选择“环境变量”。



3.在Eclipse中配置Maven的安装路径。



links

- [目录](#)
- [上一节: 环境篇](#)
- [下一节: Git安装](#)

1 Git安装

1.要安装 Git, 请转至<https://git-scm.com/download/win>。

2 TortoiseGit安装

1.要安装 TortoiseGit, 请转至 <https://tortoisegit.org/download/>。

links

- [目录](#)
- 上一节: [Java环境](#)
- 下一节: [启动应用](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

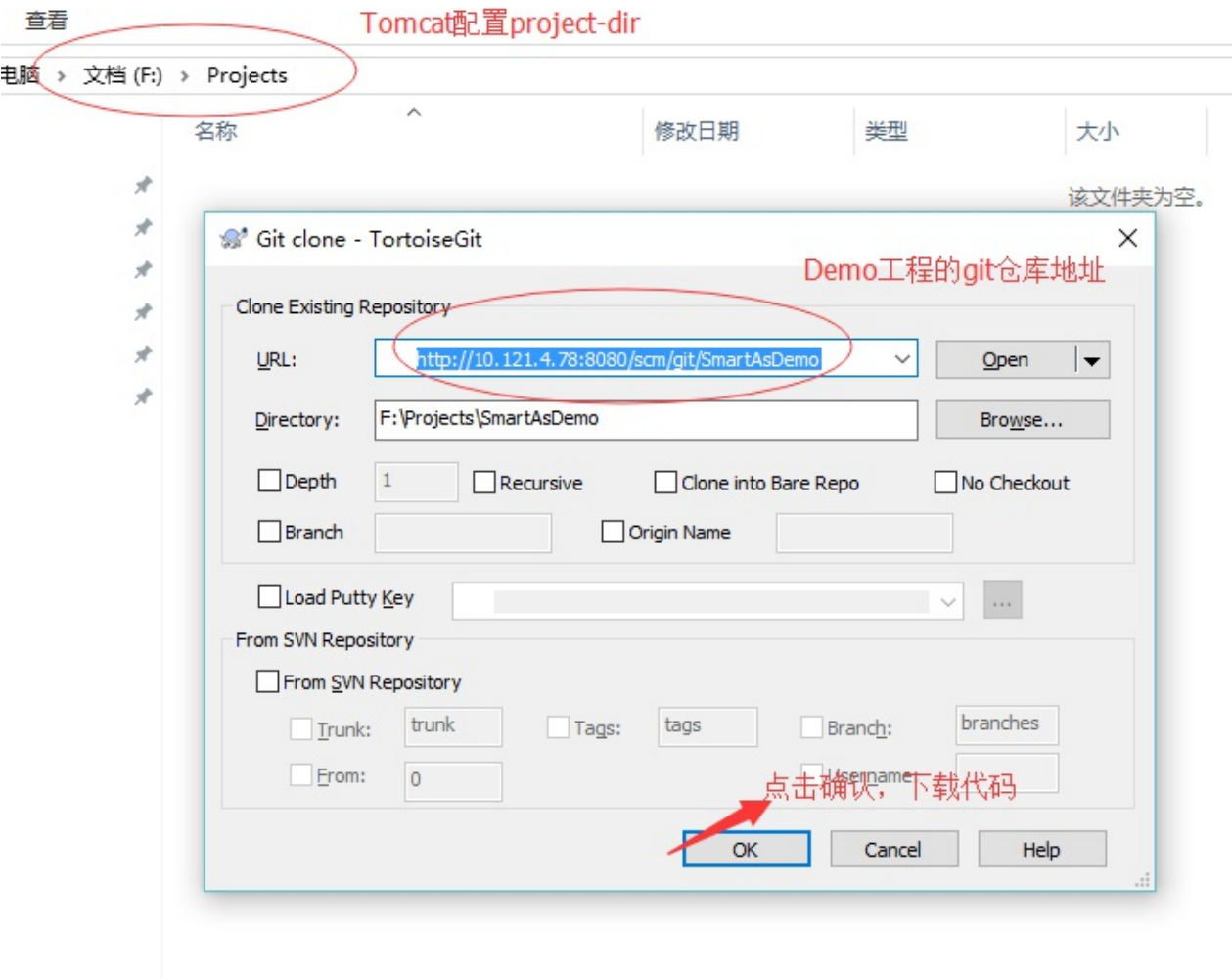
1 下载应用

应用列表

- 1.SmartAs
- 2.SmartAsDemo
- 3.FitInq
- 4.xxx项目

通过TortoiseGit下载应用到【F:\Projects】目录(Tomcat环境配置的project-dir目录)

1.下面以SmartAsDemo工程为例, 拷贝 `http://10.121.4.78/fit/SmartAsDemo` 工程地址, 2.在F:\Projects目录下右键鼠标菜单栏中点击 Git Clone ... ,显示如下图

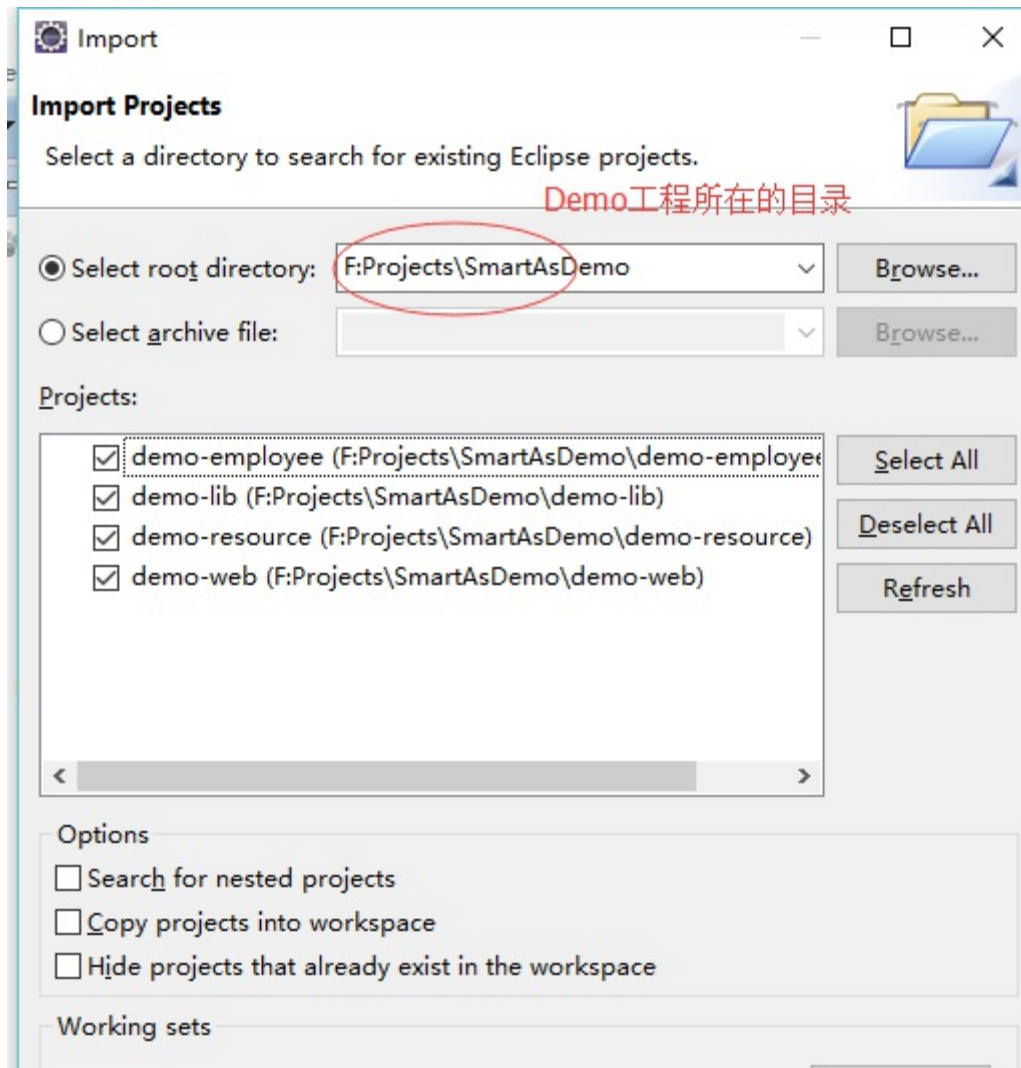


注意: 如遇到提示输入用户名账号信息情况。

gogs上的所有工程。可以自行注册账号

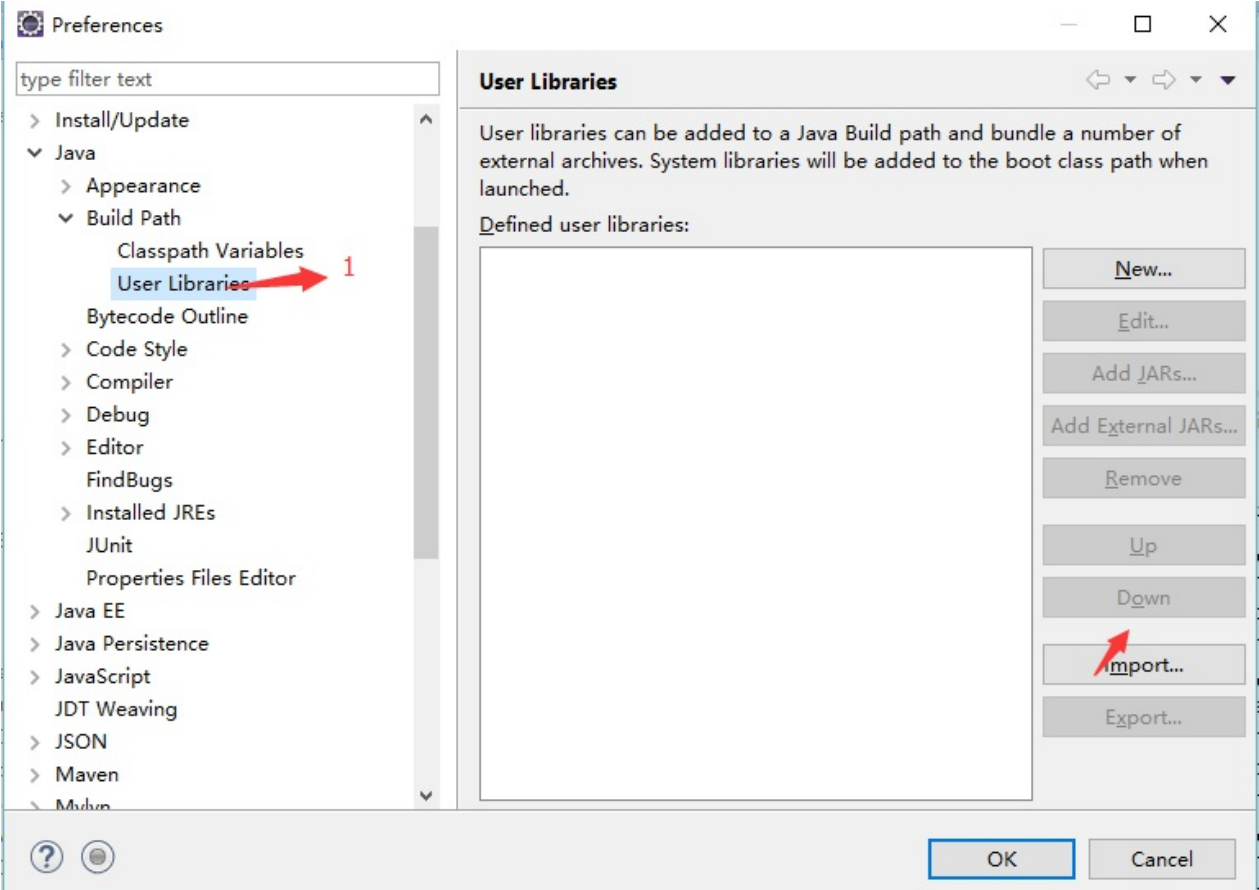
2 导入工程

启动Eclipse, 选择对应的工程导入, 选择General->Existing Projects into Workspace, 显示如下图

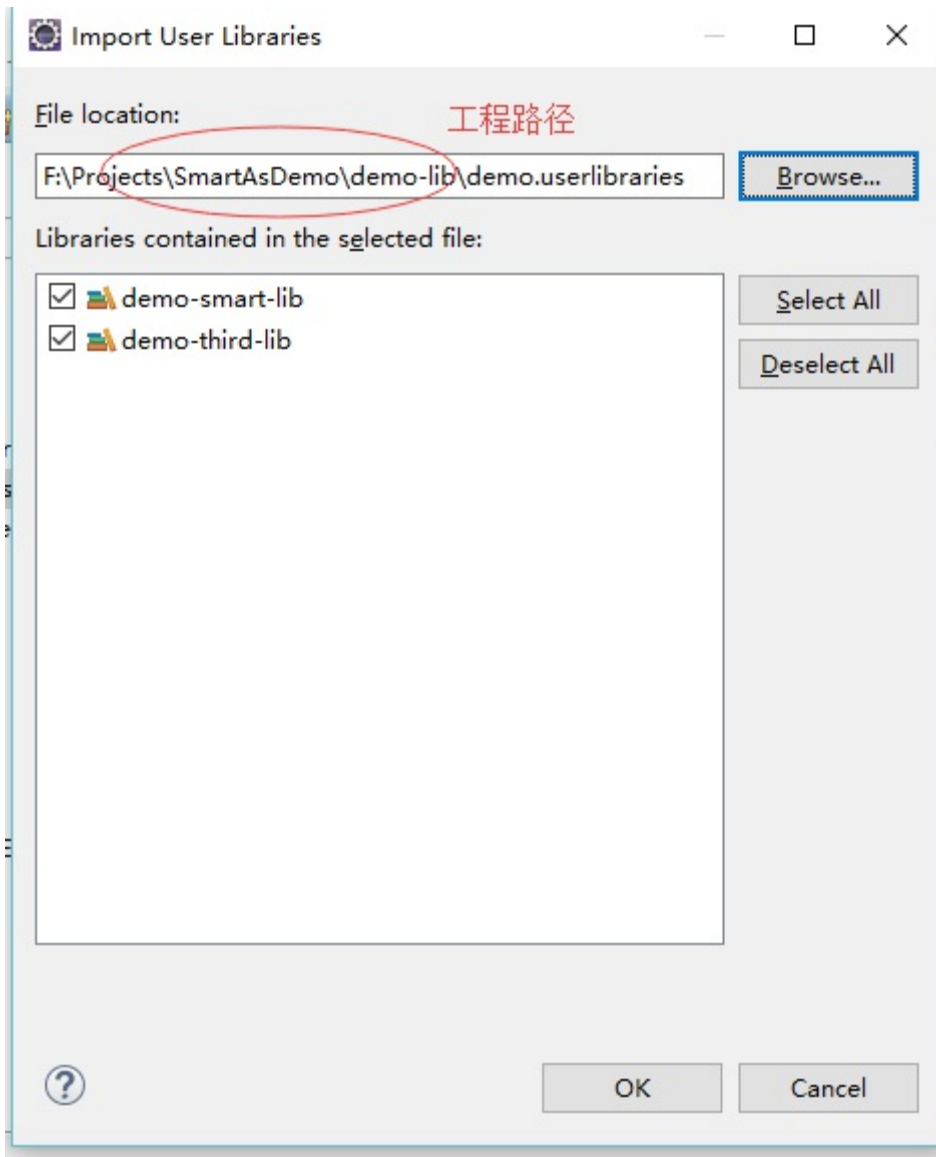


如果代码提示错误, 还需要导入用户自定义的 User Libraries(文件在工程目录的下xxx-lib下, xxx依据不同的项目可能不一样, SmartAsDemo为demo-lib)

找到导入用户自定义libraries



选择对应的libraries文件，导入所有



3 启动应用

- 1.定位到%TOMCAT_HOME%\conf\Catalina\localhost目录，不存此文件夹自行创建。
- 2.添加xml配置文件(以SmartAsDemo工程为例，文件名为demo.xml,demo为该工程的启动后应用上下文根)，类容如下

```
<Context reloadable="true" docBase="${project_dir}\SmartAsDemo\demo-web\src\main\webapp">
  <Loader className="org.apache.catalina.loader.DevLoader" reloadable="true" debug="1" />
</Context>
```

注意修改docBase的类容为实际项目的webapp目录，**如果不想启动某一个应用，可以修改文件的后缀名，让文件类型变成非xml格式**

- 3.配置JNDI 框架从2.0版本开始，框架默认支持数据源jndi配置方式，在上面XML文件中配置，Resource 标签放到Context 标签中间:特别注意在url属性中配置自己项目的url路径和数据库名称，不要再用默认框架组的数据库smartas，username属性配置自己数据库的用户名，password属性配置自己数据库的密码


```

<!-- MySQL-->
<Resource
    name="jdbc/SmartAsDS"
    factory="com.alibaba.druid.pool.DruidDataSourceFactory"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://**.*.*.*.*/**?*useUnicode=true&characterEncoding=UTF-8&useFastDateParsing=false&
;allowMultiQueries"
    username="****"
    password="****"
    maxActive="50"
    maxWait="10000"
    removeabandoned="true"
    removeabandonedtimeout="60"
    logabandoned="false"
    poolPreparedStatements="false"
    maxPoolPreparedStatementPerConnectionSize="20"
    filters="wall,stat,log4j"/>

```

4.更改项目配置文件web.xml中webAppRootKey, 如下所示中knowledge_ct为项目名称

```

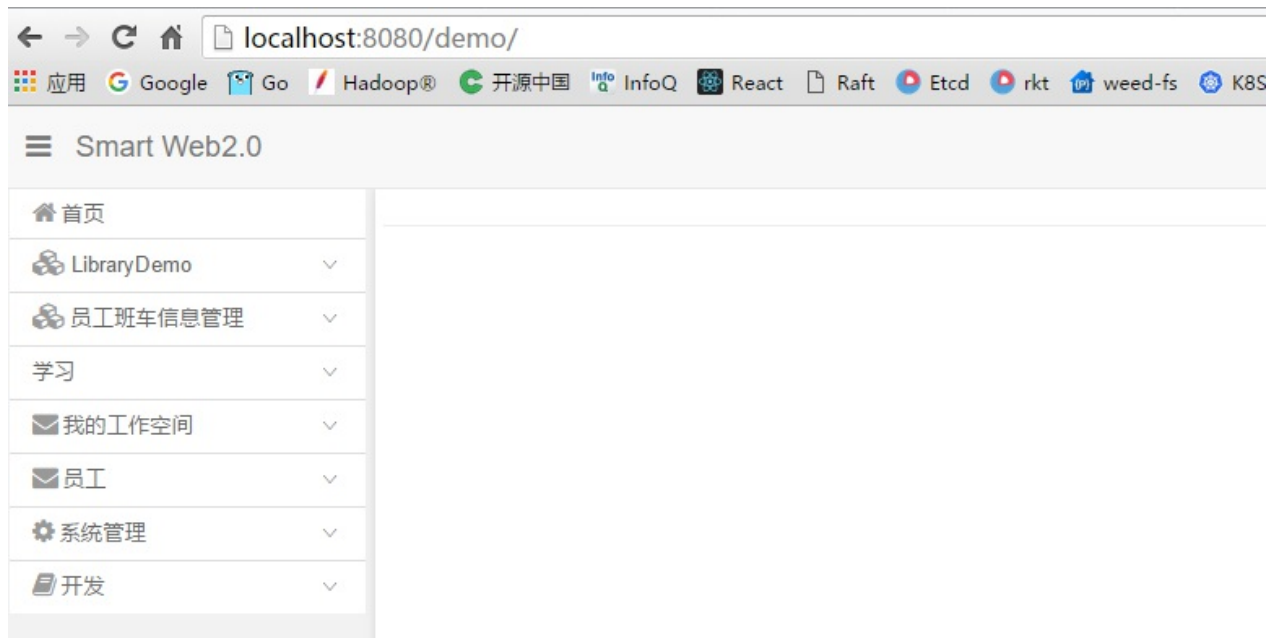
<context-param>
    <param-name>webAppRootKey</param-name>
    <param-value>knowledge_ct.root</param-value>
</context-param>

```

5.点击Eclipse工具栏中的猫图标启动Tomcat, 成功后访问

<http://localhost.smartas.com:8080/demo/>

进入登录页面, 输入测试账号(test1到test9), 密码(123), 成功后显示如下



links

- [目录](#)
- 上一节: [Git安装](#)
- 下一节: [小结](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

1.4 总结

这一章我们主要介绍了基于框架的开发环境搭建, 包括JAVA, Git, eclipse, Tomcat。安装环境后还需要配置开发环境, 以及通过Git导入项目。然后介绍了如何启动应用, 希望通过统一的工具和环境配置能快熟的开发基于SmartAs的应用。

links

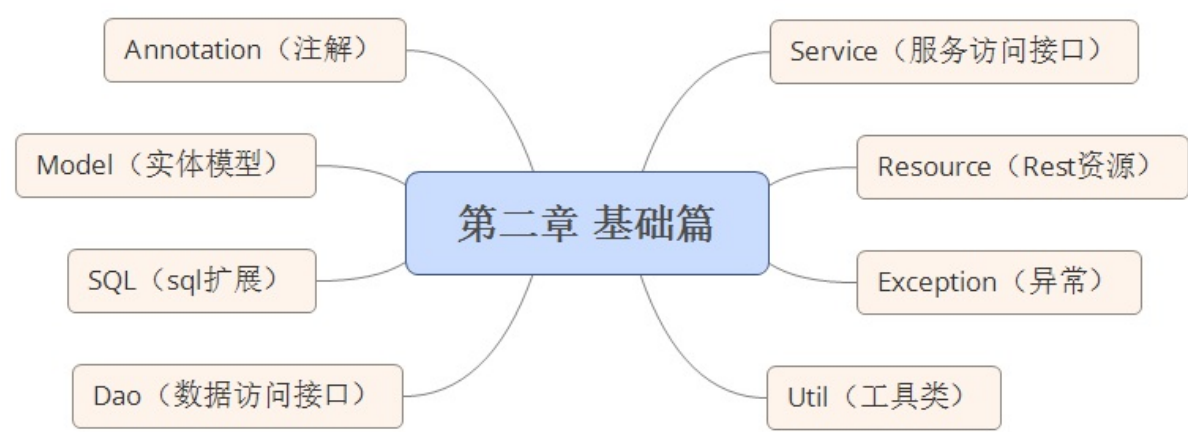
- [目录](#)
- 上一节: [启动应用](#)
- 下一章: [基础篇](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

1 框架基础介绍

在本章中, 我们将讲述框架注解, 基础类, 接口等核心功能主城要素。通过本章的学习, 希望大家能掌握基于 SmartAs 框架的后端 Rest 服务开发。

目录



links

- [目录](#)
- 下一节: [Annotation\(注解\)](#)

注解

java 注解, 从名字上看是注释, 解释, 但功能却不仅仅是注释那么简单。注解(Annotation) 为我们在代码中添加信息提供了一种形式化的方法。通过注解机制, 我们可以更好的实现一些特性。比如: JSON序列化增强, 工作流的启动及审批, 及配置信息等。接下来, 我们会详细介绍框架提供的注解及作用和用法。

1 Lookup(查找表)

Lookup模块为用户提供了查找表的前后端实现, 可以快熟的实现系统中的一些类型, 分类, 业务字典等信息的配置。通过提供的UI控件, 不需要编写任何代码就可以实现查找表的功能, 业务表中记录对应查找表的key值即可。如果我们需要在gird列表, Lookup控件中显示对应的描述(Label)的时候, 我们只需要在对应的业务字段上添加Lookup注解, JSON序列化框架会在返回的最终结果集中提供 `业务字段名Desp` 增强的属性, 如果查询不到对应的type指定的值, 不做任何处理

样例:

```
public class XXX extends LongIdVO {
    //其他业务字段
    @Lookup(type = "core.status")
    private String status;
    //其他业务字段
}
```

增强后的结果:

成员	说明	类型	默认值
status	lookup对应的key	String	
statusDesp	描述	String	

注意事项: 业务字段建议用 **String** 类型, type值根据业务模块分类定义, 并必须全局唯一。

2 User(用户)

通常业务模型中, 根据不同的业务需求都会需要记录用户的信息(直接记录用户ID, 外键关联)。当我们需要查询用户的其他信息包括用户名, 账号, 邮件地址的时候, 通常做法是通过SQL连表操作这, 这样不仅影响性能, 而却和微服务化的设计理念也是冲突的。通过User注解, 可以让用户不在关注具体的用户表, 在需要访问用户其他信息的时候只需要在业务字段添加User注解, JSON序列化框架会在返回的最终结果集中增强对应的属性, 如果查询不到对应的id指定的用户, 不做任何处理

样例:

```
public class XXX extends LongIdVO {
    //其他业务字段
    @User
    protected String createUser;
    //其他业务字段
}
```

增强后的结果:

成员	说明	类型	默认值
createUser	id	String	
createUserAccount	账号	String	
createUserLastName	姓名	String	
createUserEmail	邮件地址	String	

3 Dep(组织, 版本2.1启用) (原@Org 2.1版本开始弃用)

实体模型对象根据业务场景需要记录组织机构的信息(用组织机构的ID做关联, 业务处理很多时候直接使用该编码)。当我们需要查询组织的中文名称时候, 我们可以直接通过添加Dep注解的方式实现, 默认的住址机构解析类型为ID JSON序列化框架会在返回的最终结果集中增强对应的属性, 如果查询不到对应的id指定的组织机构, 不做任何处理

样例:

```
public class XXX extends LongIdVO {
    //其他业务字段
    @Dep
    private String orginfo;
}
```

增强后的结果:

成员	说明	类型	默认值
orginfo	id	String	
orginfoCode	部门编码	String	
orginfoName	部门名称	String	

4 Group(群组) (2.1版本新增)

用法同@Dep

5 Operation(操作)

该注解只能应用于方法(一般是jax-rs中rest方法), 代表对资源的一种操作, 资源操作定义 结合资源定义, 完成权限控制 .注意: *Operation*提供000到006 一共7个标准动作权限, 200以下为平台权限, 不要使用。它有两个属性和一个内部注解 *@Login*。@Operation注解提供的信息将会作为操作日志的一部分。

5.1 @Operation 的属性

注解有两个属性:code和desc,code是操作码, desc是操作描述,这两个属性是必选的。

1) 注解内置的几组属性值

code(键 / 对应的int类型值)	desc(键 / 对应的String类型值)	含义
Operation.MENU / 0	MENU_DESC / "menu"	菜单
Operation.CREATE / 1	CREATE_DESC / "create"	创建
Operation.UPDATE / 2	UPDATE_DESC / "update"	更新
Operation.READ / 3	READ_DESC / "read"	读
Operation.DELETE / 4	DELETE_DESC / "delete"	删除
Operation.CREATE_PROCESS / 5	CREATE_PROCESS_DESC / "create process"	创建流程
Operation.APPROVE_PROCESS / 6	APPROVE_PROCESS_DESC / "approve process"	审批流程
Operation.FILE_UPLOAD / 7	FILE_UPLOAD_DESC / "file upload"	文件上传
Operation.FILE_DOWNLOAD / 8	FILE_DOWNLOAD_DESC / "file download"	文件下载
Operation.TODO / 101	TODO_DESC / "to-do list"	待办列表
Operation.DONE / 102	DONE_DESC / "done list"	已办列表
Operation.HANDIN / 103	HANDIN_DESC / "handin list"	参与列表
Operation.SPOON / 104	SPOON_DESC / "spoon list"	发起列表
Operation.DRAFT / 105	DRAFT_DESC / "drafts"	草稿箱

如

```

@Path("/skills/DemoSkills")
@Resource(code = 69000, model = "demo", desc = "skills")
public class DemoSkillsUI extends BaseResource<DemoSkills> {

    ...

    @GET
    @Path(value = "/index")
    @Operation(code = Operation.READ, desc = Operation.READ_DESC)
    public String index() {
        return null;
    }

    ....

}

```

2) 自定义属性值

```

如
@Path("/skills/DemoSkills")
@Resource(code = 69000, model = "demo", desc = "skills")
public class DemoSkillsUI extends BaseResource<DemoSkills> {

    ...

    @GET
    @Path(value = "/index")
    @Operation(code = 10001, desc = "view user detail info")
    public String index() {
        return null;
    }

    ....
}

```

5.2 @Operation.Login（登陆）

该注解代表的某个方法（操作）是需要登陆以后才能进行的。

使用：@Path("/skills/DemoSkills") @Resource(code = 69000, model = "demo", desc = "skills") public class DemoSkillsUI extends BaseResource {

```

....

@GET
@Path(value = "/index")
@Operation.Login
public String index() {
    return null;
}

...
}

```

@Resource（资源定义注释）

该注解只能作用于类class,在需要进行权限控制的资源类上添加该注释。注意:模块的代码不能用20000以下, 20000以下为系统层使用。该注解提供的信息将会作为操作日志的一部分。属性如下：

属性名	属性类型	属性说明
code	int	资源编码
model	String	模块名
desc	String	描述

使用方式:

```

@Path("/demo2")
@Resource(code = 99002, model = "Smart", desc = "Demo2 Resource")
public class Demo2UI extends BaseResource<Demo2> {
    ....
}

```

@Resource.Bind

暂不使用

@Menu（菜单）

该注解的值是@Operation类型的, 代表了用户对某个菜单的操作定义。与@Resource注解一起使用来决定。由于@Menu值的类型是@Operation, 所以可以使用框架内置设定值, 在业务需要的时候, 可以进一步自定义,并且同一个类上可以重复声明该注解。

```
使用方式:
@Path("/demo/workflow")
@Resource(code = 19004, model = "Smart", desc = "DemoWorkflow Resource")
@Menu(@Operation(code = 800, desc = "test1"))
@Menu(@Operation(code = 900, desc = "test2"))
public class DemoWorkflowUI extends BaseFlowResource<DemoWorkflow>

...

}
```

@Menus（菜单集合）

该注解的取值为@Menu数组, 代表了对一组菜单资源的操作定义, 默认值为空数组。需要注意的是当同时 在类上使用@Menu和@Menus注解时, @Menu注解不能重复。

```
使用
@Path("/demo/workflow")
@Resource(code = 19004, model = "Smart", desc = "DemoWorkflow Resource")
@Menu(@Operation(code = 800, desc = "test1"))
@Menus(value={@Menu(@Operation(code = 801, desc = "test2")),@Menu(@Operation(code = 900, desc = "test3"))})
public class DemoWorkflowUI extends BaseFlowResource<DemoWorkflow> {

    ...

}
```

@Cache（缓存注解）

该注解标记的类可以自定义缓存策略, 注解有以下属性:

属性名	属性类型	属性说明	默认值	是否必须
expire	int	过期时间	0	否
name	String	缓存模块名	无	是
keyGenerator	String	负责缓存键生成策略的bean名称	已内置	否
cacheManager	String	负责缓存管理发的bean名称	已内置	否
cacheResolver	String	缓存解析处理的bean名称	已内置	否

该注解可配合spring的缓存注解来使用, spring缓存注解主要有三个, 具体参考官网说明:

- @Cacheable

负责将方法的返回值加入到缓存中。当方法第一次调用之后, 就会创建缓存, 当再次调用的时候, 会先去缓存查看有没有, 如果有对应的缓存直接返回结果, 不会去执行实际方法, 否则会去执行实际方法, 并将结果存入缓存。

- @CacheEvict 负责清除缓存
- @CachePut 主要针对方法配置, 能够根据方法的请求参数对其结果进行缓存

使用:

```

@Service
@Cache(name = "Smart:Lookup", expire = 24 * 60 * 60)
public class LookupServiceImpl extends BaseServiceImpl<Lookup> implements LookupService {

    ...
    //删除缓存
    @CacheEvict(cacheNames = {"id","list", "other"}, allEntries = true)
    public void modifyAccount(User user) {
        dao.modifyAccount(user);
    }
}

```

@HttpCache (rest请求响应缓存注解)

这个注解用于控制rest请求响应的Cache-Control header。他有以下属性：

属性名	属性类型	说明	默认值
maxAge	int	用于指示缓存的有效期;如果max-age和Expires同时指定, 则max-age有效	-1
sMaxAge	int	用于指定缓存存在一个共享的, 中间节点的最大生命期	-1
noStore	boolean	通过缓存数据都被缓存了硬盘中, 以使用得下次可以; 这个指令表示不要将缓存存在本地	false
noTransform	boolean	有时缓存会被自动进行转换以节省内存或者带宽, 例如压缩图像。no-transform用于指定不允许进行数据转换	false
mustRevalidate	boolean		false
proxyRevalidate	boolean		false
isPrivate	boolean		false

引申知识：

网页的缓存是由HTTP消息头中的“Cache-control”来控制的, 常见的取值有private、no-cache、max-age、must-revalidate等, 默认为private。其作用根据不同的重新浏览方式分为以下几种情况：

- 打开新窗口 如果指定cache-control的值为private、no-cache、must-revalidate, 那么打开新窗口访问时都会重新访问服务器。而如果指定了max-age值, 那么在此值内的时间里就不会重新访问服务器, 例如: Cache-control: max-age=5 表示当访问此网页后的5秒内再次访问不会去服务器
- 在地址栏回车 如果值为private或must-revalidate (和网上说的不一样), 则只有第一次访问时会访问服务器, 以后就不再访问。如果值为no-cache, 那么每次都会访问。如果值为max-age, 则在过期之前不会重复访问。
- 按后退按钮 如果值为private、must-revalidate、max-age, 则不会重访问, 而如果为no-cache, 则每次都重复访问
- 按刷新按钮 无论为何值, 都会重复访问

注解用于类上

```

@Path("/test")
@HttpCache(maxAge=5000)
public Myclass {
    ...
}

```

注解用于方法上

```

@Path("/test")
@HttpCache(noTransform=true)
public MyClass1 {
    ...
    @GET
    @Path(value = "/index")
    @HttpCache(maxAge=5000)
    public String index(){
        ...
    }
}

```

@HttpNoCache

该注解用来设置http响应头中Cache-Control的nocache属性, 用来注解哪些数据内容不被缓存。它的属性是一个String数组, 默认为空, 注解可放置于类或者方法上,例如:

```

@Path("/test")
@HttpNoCache(fields={"fld1","fld2"})
public MyClass {
    ...
}

@Path("/test")
public MyClass1 {
    ...
    @GET
    @Path(value = "/index")
    @HttpNoCache(fields={"fld1","fld2"})
    public String index(){
        ...
    }
}

```

@StartProcess（流程启动注解）

该注解是基于spring的aop机制实现的,用于发起一个流程实例, 它只有一个String类型的 value属性。被该注解作用的方法, 必定要包含一个FlowAware接口实例参数, 注解将从中 获取流程发起所需要的流程标识、业务编码等参数。

使用

```

@Service
public class DemoWorkflowServiceImpl extends BaseFlowServiceImpl<DemoWorkflow>{
    ...

    @StartProcess()
    public Serializable start(DemoWorkflow vo) {
        vo.setProcessName(vo.getName());
        return super.start(vo);
    }
    ...
}

```

@StartProcess.Draft（流程草稿箱）

该注解只能在方法上使用, 基于spring aop实现, 该注解没有属性, 被该注解标注的方法含有 一个FlowAware实例的参数, 并且返回Serializable类型。这个注解可以将流程相关业务信 息进行草稿箱备份。

使用

```

@Service
public class DemoWorkflowServiceImpl extends BaseFlowServiceImpl<DemoWorkflow>{
    ...

    @StartProcess.Draft()
    public Serializable draft(DemoWorkflow vo) {
        // 判断是否为修改的情况
        if (vo.getId() != null) {
            update(vo);
            return vo.getId();
        } else {
            return save(vo);
        }
    }
    ...
}

```

@ApproveTask(流程审批注解)

该注解只能用于方法, 基于spring aop机制, 注解作用的方法必须含有一个FlowAware类型的 参数, 否则会抛出空指针异常, 该方法将提供流程审批所需要的信息。

使用

```

@Service
public class DemoWorkflowServiceImpl extends BaseFlowServiceImpl<DemoWorkflow>{
    ...

    @ApproveTask
    public void approve(DemoWorkflow vo) {
        update(vo);
    }
    ...
}

```

@Compress

该注解可用于类、方法和参数,用于给被标注的内容启用gzip压缩。

使用

```

@GET
@Path(value = "/list")
@Operation(code = Operation.READ, desc = Operation.READ_DESC)
@Compress
public List<Rule> list() throws BusinessException {
    return ruleHandler.getRules();
}

```

@PATCH

该注解只作用于方法, 被jax的@HttpMethod修饰, 用以表明使用该方法来响应HTTP PATCH 请求。

@Url

该注解用以自定义处理器映射，注解的属性值是个数组，当只有一个映射路径的时候可以简写为 `@Url("/test.do")`，匹配多个路径时使用 `@Url(value={"/text.html","/test1.html"})`。`@Url`注解只用于类，这个类可以是 `ResourceHandler`的子类，当请求路径匹配上 `@Url`注解定义的路径 时，就会把视图映射逻辑交给这个子类进行处理，然后在这个子类中覆盖 `getResource`方法，该方法的返回 值就是最终的视图资源。同时，为了让spring知道这个自定义映射器，需要将他声明在spring的配置 文件中或者注解的方式加入到spring的环境中。

新建一个 `ResourceHandler`的子类，覆盖父类的 `getResource` 方法

```
@Url(value={"/text.html","/test1.html"})
public class TestHandler extends ResourceHandler {

    //视图资源
    private Resource resource;
    ...

    @Override
    protected Resource getResource(HttpServletRequest request) {
        return resource;
    }

    //设置编码格式
    protected MediaType getMediaType(Resource resource) {
        return MediaType.parseMediaType("text/html;charset=UTF-8");
    }
    ...
}
```

通过xml注入spring, `TestHandler`的 `resource`属性指定了去哪里寻找视图

```
<bean class="com.fiberhome.smartas.demo.handler.TestHandler">
    <property name="resource" value="classpath:web/smartas/demo/index.html" />
</bean>
```

这样之后， `TestHandler`就会拦截 `xxx/text.html`,"`xxx/test1.html`"形式的请求路径，然后 返回对应的视图。

@Validate

该注解可以用于类和方法，通过该注解可以指定一个验证文件。该注解只有一个属性 `context`(`String` 类型)，通过这个属性指定验证文件的位置，当没有配置此属性时，有一个默认查找规则：它默认获取 当前类所在的包路径，然后把这个路径作为 `classpath`路径的子路径，最终的路径就是 `"classpath:包路径"`，然后在这个路径下查找名称为当前类名称加上 `"validate.xml"`后缀的文件，将它 作为当前类的验证文件。

看如下示例：

```
package com.fiberhome.smartas.security.ui;

@Path("/security/menu")
@Validate()
public class MenuResource extends BaseResource<Menu> {

    ...
}
```

上述例子中，使用了 `@Validate`的默认配置，根据默认查找规则：

- 它的包路径为 `com.fiberhome.smartas.security.ui`，对于的文件路径为 `com/fiberhome/smartas/security/ui/`

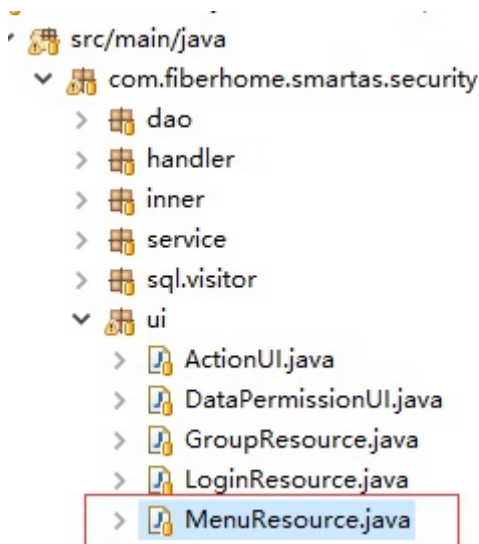
- 类名为MenuResource

那么它对应的验证配置文件就应该位于:

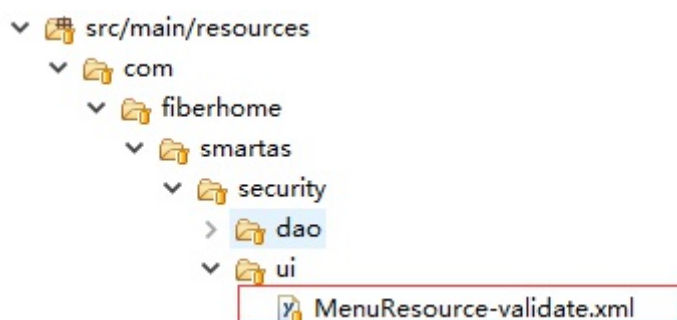
```
classpath:/com/fiberhome/smartas/security/ui/MenuResource-__validate.xml"
```

如图:

注解类位置



验证文件的位置位置



@Validate.Rule

该注解只能作用于方法的参数, 用于指定参数的校验规则和校验模式。它需要配合@Validate注解一起使用, 由@validate属性提供验证规则所在的文件。它有以下属性:

属性名	属性类型	属性说明	默认值	是否必填
value	String	校验规则名称	无	是
strict	boolean	是否严格校验模式	false	否

links

- [目录](#)
- [上一节: 基础篇](#)
- [下一节: Model\(数据模型\)](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

Model

Model数据实体层

- Model可以理解为面向对象的一个实体类, 这个类对应数据库中的一张表, 类的属性对应数据库表中字段
- SmartAs框架为每张表定义了一些公共字段, 例如以后框架扩展到集群的多租户、版本号、创建人ID等, 并对这些公共信息进行了封装 `BaseVO` :

```
protected String tenantId; // 多租户
protected int revision = 1; // 版本
@User
protected String createUserId;
protected Date createDate;
@User
protected String lastUpdateUserId;
protected Date lastUpdateDate;
```

- 框架同时对表的ID也进行了封装 `IntIdVO`、`LongIdVO`、`StringIdVO`, 在model实体类中, 都需要继承与数据库ID类型相对应的 `vo`, 例如某实体类的ID类型为long:

```
public class xxx extends LongIdVO {
    \\\
}
```

组织模型

.
..
..

links

- [目录](#)
- 上一节: [1.Annotation](#)
- 下一节: [3.SQL](#)

SQL

SmartAs框架提供动态SQL查询

1. QueryFilter对象

过滤的查询条件

- 过滤的查询参数名称格式必须为: `Q_field_T_OP/Q_field_OP`
- 其中 `Q` 表示该参数为查询的参数, `field` 查询的字段名称, `T` 代表该参数的类型(不出现则表示字符串类型S), `OP` 代表操作
- `field` 按驼峰命名, `firstName` 会编译成 `first_name`
- `T` 位置值有:

```
Z boolean
B byte
I int
L long
F float
J double
S String
D Date "yyyy-MM-dd" or "yyyy-MM-dd HH:mm:ss"
T Time "HH:mm:ss"
```

- `OP` 位置值有:

```
LT &lt;
LE &lt;=
GT &gt;
GE &gt;=
EQ =
NE !=
IN in
NI not in
LK like %aa%
ST like aa%
ED like %aa
NL null
NN not null
```

- 通用样例

```
Q_fieldName_T_LT=value -> field_name < value
Q_firstName_T_LE=value -> field_name <= value
Q_firstName_T_GT=value -> field_name > value
Q_firstName_T_GE=value -> field_name >= value
Q_firstName_T_EQ=value -> field_name = value
Q_firstName_T_NE=value -> field_name != value
Q_firstName_T_IN=v1,v2 -> field_name in (v1,v2)
Q_firstName_T_NI=v1,v2 -> field_name not in (v1,v2)
Q_firstName_LK=value   -> field_name like '%value%'
Q_firstName_ST=value   -> field_name like 'value%'
Q_firstName_ED=value   -> field_name like '%value'
Q_firstName_NL         -> field_name is null
Q_firstName_NN         -> field_name is not null
```

- 前端js页面代码样例

```
const {getFieldProps} = this.props.form;
const knowledgeName = getFieldProps('Q_knowledgeName_S_LK');
```

2.Filter标签

```
<!ATTLIST filter
open CDATA #IMPLIED
close CDATA #IMPLIED
name CDATA #IMPLIED default "T" //指定的数据库表别名
value CDATA #IMPLIED default "query" //查询对象的名称
>
```

- 样例如下:

```
//在dao层的getCount方法中添加过滤器
int getCount(@Param("query") QueryFilter filter);

//在对应的xml文件的SQL语句中
<select id="getCount" resultType="int">
    SELECT
        count(1)
    FROM TPL_REGISTRY_T T
    <!-- <filter open="where (" close=)" name="T" value="query"/> -->
    <filter open="where (" close=)" />
    //当原SQL语句中有条件where时, 应为<filter open="and (" close=)" />
</select>
```

3.Pageable分页

- SmartAs支持不同数据库的分页查询功能
- xml文件中的select标签通过扩展pageable属性实现分页

```
<select id="select" pageable="true" resultMap="RegistryResultMap">
    SELECT T.* FROM TPL_REGISTRY_T T
    <filter open="where (" close=)" />
</select>
```

links

- [目录](#)
- 上一节: [2.Model](#)
- 下一节: [4.Dao](#)

Dao

1. Dao简介

数据库持久层使用的是开源 Mybatis + 标签定制和扩展方式实现, Dao层只需要提供API, 通过mybatis-spring的扫描机制, 自动组装实现类. SmartAS框架提供Dao父接口 `BaseDao<T>`, 自定义的Dao接口继承该接口, 则继承了 `BaseDao<T>` 定义的全部方法。

2. 框架Dao接口

2.1 Dao父接口

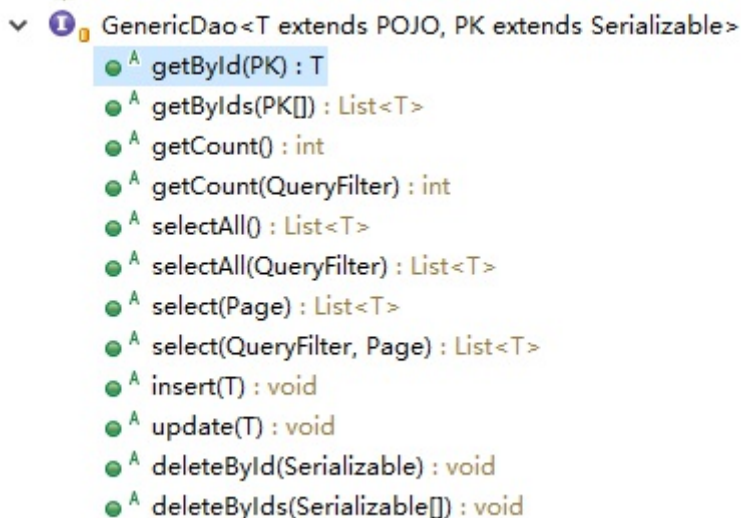
SmartAs框架提供Dao父接口 `BaseDao<T>`, 在框架内部提供组件类型为long的Dao接口, 即数据表中主键ID类型为long。

```
/**
 * 组件类型为long的抽象数据访问接口
 * @author chenb
 *
 * @param <T> 业务对象类型
 */
public interface BaseDao<T extends POJO> extends GenericDao<T, Long> {

}
```

2.2 数据访问接口API

SmartAs框架的Dao父接口 `BaseDao<T>` 继承了组件类型为long的抽象数据访问接口 `GenericDao<T, Long>` `GenericDao` 定义了数据访问最基础的方法, 其中包括八个查询的方法, 一个插入的方法, 一个更新的方法, 两个删除的方法, 接口API如下图所示:



```
GenericDao<T extends POJO, PK extends Serializable>
  A getById(PK) : T
  A getByIds(PK[]) : List<T>
  A getCount() : int
  A getCount(QueryFilter) : int
  A selectAll() : List<T>
  A selectAll(QueryFilter) : List<T>
  A select(Page) : List<T>
  A select(QueryFilter, Page) : List<T>
  A insert(T) : void
  A update(T) : void
  A deleteById(Serializable) : void
  A deleteByIds(Serializable[]) : void
```

2.2.1 查询数据方法

根据 `id` 从数据库查询满足条件的对象,这里我们不处理找不到结果的情况,因为存在业务要求 数据库表中是有这条记录的。

```
T getById(PK id);
List<T> getByIds(@Param("ids") PK[] ids);
```

自动分页或者需要得到返回某类对象在数据库中的记录数,分页用的比较多。 `getCount()` 方法重载分为是否包含查询条件两种情况。

```
@MapKey("count")
int getCount();
@MapKey("count")
int getCount(@Param("query") QueryFilter filter);
```

返回数据库中此类记录的所有数据,按id排序。 `selectAll()` 方法重载分为是否包含查询条件。注意:大数据集合影响效率。

```
List<T> selectAll();
List<T> selectAll(@Param("query") QueryFilter filter);
```

返回数据库中此类记录满足分页条件的记录集合的数据,满足查询边界。 `select()` 方法重载分为是否包含查询条件。

```
List<T> select(@Param("page") Page page);
List<T> select(@Param("query") QueryFilter filter, @Param("page") Page page);
```

2.2.2 保存数据方法

保存数据记录,根据具id是否为空来判断是否插入新记录还是更新操作, id为空时插入, id自增加1。增加一个实体对象 `T entity` 。

```
void insert(T entity);
```

2.2.3 更新数据方法

更新数据记录, 根据具id是否为空来判断是否插入新记录还是更新操作, `entity` 对象中包含id值, 更新数据表中该行对应的值。

```
void update(T entity);
```

2.2.4 删除数据方法

删除记录, 批量删除记录, 根据主键id值删除数据。

```
void deleteById(Serializable id);
void deleteByIds(@Param("ids") Serializable[] ids);
```

3. Dao扩展

3.1 Dao方法扩展

根据业务场景，框架数据访问接口中的方法不满足需求，可以对Dao接口进行扩展。具体描述为[Dao扩展](#)。

3.2 Dao接口主键类型扩展

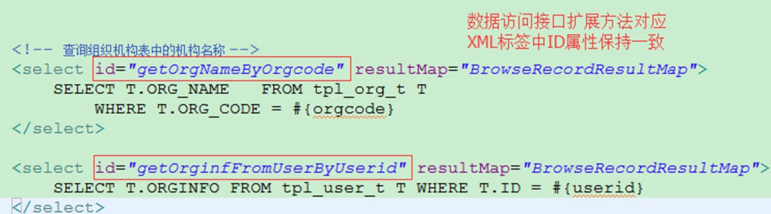
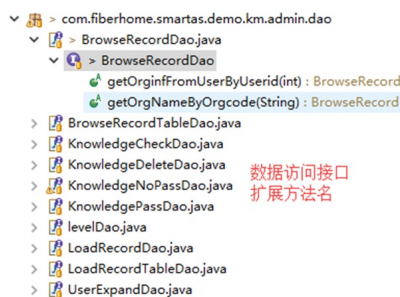
框架默认的组件数据访问接口主键类型为 `long`，框架保留了组件不同主键类型的扩展。`GenericDao<T,Long>`，可以根据自己的组件主键类型写基本数据访问接口，如组件类型为String的抽象数据访问接口。如下代码块所示：

```
public interface BaseStringDao<T extends POJO> extends GenericDao<T, String>{  
  
}
```

4. XML配置

在对应mapping XML文件中进行配置时：

- 如果是自动生成代码XML文件中会生成框架数据访问接口 `GenericDao` 定义的12个方法，如上述API所示。
- 如果不是自动生成代码，根据业务需求配好对应方法的标签。
- 如果需要重写框架数据访问接口提供的方法，在XML文件中也需要根据重写逻辑修改sql语句。
- 如果对数据访问接口进行了扩展，在XML文件中配置对应标签的ID属性应该与扩展的方法名称保持一致，如下图所示：



links

- [目录](#)
- [上一节: SQL\(sql扩展\)](#)
- [下一节: Service\(服务访问接口\)](#)

Service

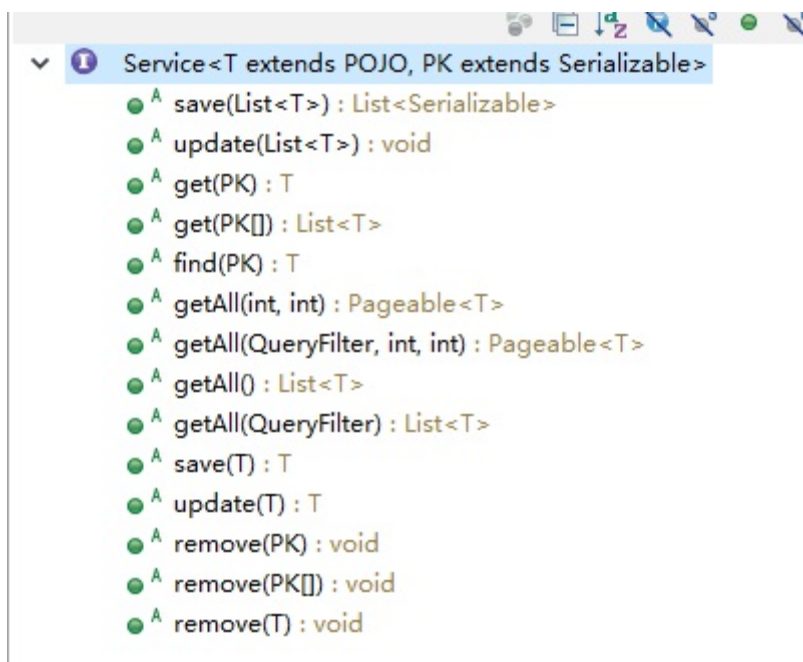
1 Service简介

Service层是SmartAs框架的服务层, 在该接口中定义服务相关的方法名。Service层提供事物支持, 对外提供原子操作的服务。框架提供父接口 `BaseService<T>` ,服务层接口继承该接口通用功能就不必再重写基础方法。只需根据业务需求对服务层接口方法进行扩展。

2 框架Service接口

2.1 Service接口API

SmartAs框架在服务层提供父接口 `BaseService<T>` , 该接口继承了服务通用接口 `Service<T extends POJO, PK extends Serializable>` 。API如下图所示:



在该接口中集成了通用的服务, 主要分为查询服务, 存储服务, 更新服务, 删除服务。且对外提供原子操作的服务, 在方法定义上一行加注解 `@Transactional(readonly=true)` 保证服务操作的原子性, 即要不服务执行成功否则回到之前没有任何改变。

2.2 查询服务

根据id号获得对象, 可以返回。如果返回null, 不作相应的异常处理。

```
@Transactional(readonly=true)
T get(PK id) throws BusinessException;
@Transactional(readonly=true)
List<T> get(PK[] id) throws BusinessException;
```

根据id号获得对象，可以返回。如果返回null，要做相应的异常处理。

```
@Transactional(readOnly=true)
T find(PK id) throws BusinessException;
```

根据指定范围，返回查询数据。

```
@Transactional(readOnly=true)
Pageable<T> getAll(int page, int pageSize) throws BusinessException;
@Transactional(readOnly=true)
Pageable<T> getAll(QueryFilter query,int page, int pageSize) throws BusinessException;
@Transactional(readOnly=true)
List<T> getAll() throws BusinessException;
@Transactional(readOnly=true)
List<T> getAll(QueryFilter query) throws BusinessException;
```

2.3 存储服务

应该检查id是否存在此记录的业务逻辑。根据id值是否为空判断是存储还是修改服务，id为空则是存储。id自增加1。

```
@Transactional()
Serializable save(T o) throws BusinessException;
```

2.3 修改服务

应该检查id是否存在此记录的业务逻辑。根据id值是否为空判断是存储还是修改服务，id不为空且存在于数据表中则是修改服务。

```
@Transactional()
void update(T o) throws BusinessException;
```

2.4 删除服务

应该检查id是否存在此记录的业务逻辑，根据id删除记录或者批量删除记录。

```
@Transactional()
void remove(PK id) throws BusinessException;
@Transactional()
void remove(PK[] id) throws BusinessException;
@Transactional()
void remove(T o) throws BusinessException;
```

3 ServiceImpl

框架的服务层，包括提供对外接口的 `Service` 层。还有实现服务接口的 `ServiceImpl` 层。框架的 `GenericServiceImpl` 通过对应数据访问层dao进行数据访问实现通用服务。在服务实现层框架提供了缓存机制，即多次用同样的参数执行同样的方法只执行第一次，接下来都是直接从缓存里读取结果，可以提高数据访问效率。

```

@Override
@CacheEvict(cacheNames = {"id", "list", "other"}, allEntries = true)
public void modifyAccount(User user) {
    dao.modifyAccount(user);
}

```

4 Service接口扩展

根据业务需求在服务层扩展方法。

4.1 服务方法扩展

与数据访问接口一样，服务接口也要根据对应数据访问接口的扩展而扩展服务层方法。，如下图所示：

```

public interface BrowseRecordDao extends BaseDao<BrowseRecord> {

    //根据id从km_knowledge表中查询得到对象其中的属性用户id      数据访问层扩展方法
    BrowseRecord getById(@Param("id") Long id);

    //根据用户id从表tpl_user_t中得到orging
    BrowseRecord getOrginfFromUserByUserid(@Param("userid") int userid);

    //根据orgcode从tpl_org_t表中得到机构名称orgname
    BrowseRecord getOrgNameByOrgcode(@Param("orgcode") String orgcode);

}

public interface BrowseRecordService extends BaseService<BrowseRecord> {

    @Transactional(readOnly=true)
    BrowseRecord getOrgnameByOrgcode(String orgcode) throws BusinessException;

    @Transactional(readOnly=true)
    BrowseRecord get(Long id) throws BusinessException;

    @Transactional(readOnly=true)
    BrowseRecord getOrginfByUserid(int userid) throws BusinessException;

}

```

同时在服务实现层，进行业务逻辑操作，如对得到的数据访问记录解析、遍历、修改等。

4.2 服务接口主键类型扩展

同数据访问接口一样，框架默认的组件服务接口主键类型为 `long`，框架保留了组件不同主键类型的扩展。 `interface Service<T extends POJO,PK extends Serializable>`，可以根据自己的组件主键类型写基本服务接口，如组件类型为String的抽象服务接口接口。如下代码块所示：

```

public interface BaseStringService<T extends POJO> extends Service<T, String>{

}

```

links

- [目录](#)
- 上一节: [Dao\(数据访问接口\)](#)

- 下一节: [Resource \(Rest资源\)](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

Resource层

Resource层提供rest风格的网络服务，支持权限验证，登录验证，Metracs审计。如果资源是需要权限验证，需要在类上添加Resource注解，在方法上添加Operation注解。如果方法需要验证登录，只需要添加Operation.Login注解

1.Resource层继承了SmartAs框架提供的基础资源实现父类 `BaseResource<T>`

```
public class xxxResource extends BaseResource<T>    // T为业务对象类型
```

2.基础资源父类 `BaseResource<T>` 又继承了框架提供的通用资源父类 `GenericResource<T, Long>`，主键类型为 `long`

```
public abstract class BaseResource<T extends Id> extends GenericResource<T, Long>
```

- 注:由于在绝大部分业务场景下，主键ID的类型都为long，框架在 `BaseResource` 中给通用资源父类 `GenericResource` 传递的主键ID类型为 `long`。
- 若业务中遇到主键ID类型不为long，而为其他类型时(如String类型)，开发人员可以重新写一个 `BaseStringResource<T>`，在继承通用资源父类 `GenericResource` 时给其传的主键ID类型改为String，如下：

```
public abstract class BaseStringResource<T extends Id> extends GenericResource<T, String>
```

3.通用资源父类 `GenericResource<T, Long>` 提供能满足大部分业务需求的数据库操作，API具体如下：

```
/**
 * 根据主键id查询业务对象 API: <pre>
 *   Method:   GET
 *   Url      :  serives/{module}/single/{id}
 * </pre>
 *
 * @param id 主键id
 * @return 业务对象
 * @throws BusinessException 业务异常
 */
@GET
@Path(value = "/single/{id}")
@Operation(code = Operation.READ, desc = Operation.READ_DESC)
public T get(@PathParam("id") PK id) throws BusinessException {
    return getService().get(id);
}

/**
 * 批量查询ids指定的业务对象 API: <pre>
 *   Method:   GET
 *   Url      :  serives/{module}/batch/{ids}
 * </pre>
 *
 * @param ids 逗号分隔的id字符串
 * @return ids指定的对象集合
 * @throws BusinessException 业务异常
 */
@GET
@Path(value = "/batch/{ids}")
@Operation(code = Operation.READ, desc = Operation.READ_DESC)
public List<T> gets(@PathParam("ids") String ids) throws BusinessException {
    PK[] pks = commaDelimitedListToArray(ids);
    if (ArrayUtils.isEmpty(pks)) {
        throw new BusinessException("error.core.batch.empty");
    }
}
```

```

    }
    return getService().get(pks);
}

/**
 * 分页查询 API: <pre>
 *   Method:   GET
 *   Url      :  serives/{module}/list/{page}/{pageSize}?{queryString}
 * </pre>
 *
 * @param page 页码
 * @param pageSize 页大小
 * @return 返回分页查询对象
 * @throws BusinessException 业务异常
 */
@GET
@Path(value = "/list/{page}/{pageSize}")
@Operation(code = Operation.READ, desc = Operation.READ_DESC)
@Compress
public Pageable<T> listPage(@PathParam("page") int page, @PathParam("pageSize") int pageSize)
    throws BusinessException {
    QueryFilter query = QueryUtils.parseMultiQuery(uriInfo.getQueryParameters());
    return getService().getAll(query, page, pageSize);
}

/**
 * 查询集合 API: <pre>
 *   Method:   GET
 *   Url      :  serives/{module}/list?{queryString}
 * </pre>
 *
 * @return 满足条件的对象集合
 * @throws BusinessException 业务异常
 */
@GET
@Path(value = "/list")
@Operation(code = Operation.READ, desc = Operation.READ_DESC)
@Compress
public List<T> list() throws BusinessException {
    QueryFilter query = QueryUtils.parseMultiQuery(uriInfo.getQueryParameters());
    return getService().getAll(query);
}

/**
 * 创建对象 API: <pre>
 *   Method:   POST
 *   Url      :  serives/{module}/single
 *   Body     :  json格式对象
 * </pre>
 *
 * @param entity 业务对象
 * @return 业务对象的唯一标识符
 * @throws BusinessException 业务异常
 */
@POST
@Path(value = "/single")
@Operation(code = Operation.CREATE, desc = Operation.CREATE_DESC)
public Serializable create(@Rule("save") T entity) throws BusinessException {
    getService().save(entity);
    return entity.getId();
}

/**
 * 批量创建对象 API: <pre>
 *   Method:   POST
 *   Url      :  serives/{module}/batch
 *   Body     :  json格式对象
 * </pre>

```

```

*
* @param list 业务对象集合
* @return 返回标识符集合
* @throws BusinessException 业务异常
*/
@POST
@Path(value = "/batch")
@Operation(code = Operation.CREATE, desc = Operation.CREATE_DESC)
public List<Serializable> create(@Rule("save") List<T> list) throws BusinessException {
    if (CollectionUtils.isEmpty(list)) {
        throw new BusinessException("error.core.batch.empty");
    }
    return getService().save(list);
}

/**
* 修改对象 API: <pre>
*   Method:  PUT
*   Url    :  serives/{module}/single
*   Body   :  json格式对象
* </pre>
*
* @param o 业务对象
* @return 业务对象唯一标识符
* @throws BusinessException 业务异常
*/
@PUT
@Path(value = "/single")
@Operation(code = Operation.UPDATE, desc = Operation.UPDATE_DESC)
public Serializable update(@Rule("update") T o) throws BusinessException {
    getService().update(o);
    return o.getId();
}

/**
* 批量修改对象 API: <pre>
*   Method:  PUT
*   Url    :  serives/{module}/batch
*   Body   :  json格式对象
* </pre>
*
* @param list 业务对象集合
* @throws BusinessException 业务异常
*/
@PUT
@Path(value = "/batch")
@Operation(code = Operation.UPDATE, desc = Operation.UPDATE_DESC)
public void update(@Rule("update") List<T> list) throws BusinessException {
    if (CollectionUtils.isEmpty(list)) {
        throw new BusinessException("error.core.batch.empty");
    }
    getService().update(list);
}

/**
* 删除对象 API: <pre>
*   Method:  DELETE
*   Url    :  serives/{module}/single/{id}
* </pre>
*
* @param id 业务标识符
* @throws BusinessException 业务异常
*/
@DELETE
@Path(value = "/single/{id}")
@Operation(code = Operation.DELETE, desc = Operation.DELETE_DESC)
public void remove(@PathParam("id") PK id) throws BusinessException {

```

```
getService().remove(id);
}

/**
 * 批量删除 API: <pre>
 *      Method:  DELETE
 *      Url    : serives/{module}/batch/{ids}
 * </pre>
 *
 * @param ids 逗号分隔的业务标识符id
 * @throws BusinessException 业务异常
 */
@DELETE
@Path(value = "/batch/{ids}")
@Operation(code = Operation.DELETE, desc = Operation.DELETE_DESC)
public void removes(@PathParam("ids") String ids) throws BusinessException {
    PK[] pks = commaDelimitedListToArray(ids);
    if (ArrayUtils.isEmpty(pks)) {
        throw new BusinessException("error.core.batch.empty");
    }
    getService().remove(pks);
}
```

links

- [目录](#)
- 上一节: [5.Service](#)
- 下一节: [7.Exception](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

Exception

SmartAs框架对业务操作过程中的业务异常进行处理, 保证代码的健壮性。

1异常简介

框架提供业务异常处理类 `BusinessAccessException` , 它是所有业务操作异常结构体系的根对象。该异常类是一个抽象对象, 用户可以根据不同模块需要继承该类实现具体的业务操作异常。

框架的每个模块的异常处理类都继承该父类。



框架在model中创建了描述异常信息的 `ErrorVO` 类。包含描述异常信息的三个属性如下代码块所示:

```
/**
 * 编码
 */
private String code;

/**
 * 消息
 */
private String message;

/**
 * 异常栈
 */
private String stackTrace;
```

2处理异常

在每个模块的如下路径中, 框架对要抛出异常信息在如下路径文件中进行配置。

src/main/resources/META-INF/i18n/Messages.properties

配置的抛出异常信息如下模块内容示例:

```
#
error.core.sys=系统异常
error.core.rt=运行异常
error.core.batch.empty=业务对象集合为空
#
error.validate.form=表单验证失败

error.core.dp=数据权限异常
```

在XML中创建一个bean, 解析获取 `Messages.properties` 定义的异常编码和消息。

```
<bean id="messageSource" class="com.fiberhome.smartas.core.spring.MultipleMessageSource">
  <property name="basenames">
    <list>
      <value>classpath*:META-INF/i18n/Messages</value>
    </list>
  </property>
  <property name="fileEncodings" value="UTF-8" />
  <property name="defaultEncoding" value="UTF-8" />
</bean>
```

在数据访问层、服务实现层、UI层遇到要处理的异常则抛出异常, 如下代码所示:

```
throw new DownloadException("error.filedownload.io.core" , e , consumer);
```

links

- [目录](#)
- 上一节: [Resource \(Rest资源\)](#)
- 下一节: [Util \(工具类\)](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

Util

1. AuthInfoUtils

当前登录用户信息工具类，提供API如下：

- 登录用户信息

```
public static AuthInfo getAuthInfo()
```

- 当前登录回话

```
public static Subject getSubject()
```

- 登录用户信息(可以为null)

```
public static AuthInfo getNullableAuthInfo()
```

- 判断当前用户是否登录

```
public static boolean isLogin()
```

- 获取当前登录用户的id

```
public static Serializable getCurrentUserId()
```

2. DaoUtils

获取真实的当前页page

```
// page      当前页
// pageSize  页大小
// length    满足条件的对象数量

public static int realPage(int page, int pageSize, int length)
```

3. BeanContext

该类主要提供根据参数 `Class<T> requiredType` 获取beanFactory中的指定bean, 以及得到应用环境信息 `AppEnv` 对象, 进一步获取应用的配置参数:包括引用名称, 数据库名, 业务表前缀, 系统群。两个方法的API如下代码块所示

```
public static <T> T getBean(Class<T> requiredType) throws BeansException {

}

public static AppEnv getAppEnv() throws BeansException {

}
```

在框架其它模块或业务开发中若使用到该类的用法如下代码块所示：


```

public String getAppName() {
    return BeanContext.getAppEnv().getAppName();
}

public String getTenantId() {
    return BeanContext.getAppEnv().getTenantId();
}

public String getScope() {
    return BeanContext.getAppEnv().getScope();
}

```

4. BeansUtils

得到某个类型的所有bean, 类API如下所示:

```

public static <T> List<T> getBeansOfType(ApplicationContext context, Class<T> type) throws BeansException {

}

```

5. FlowUtils

FlowUtils 类中定义了一个 ThreadLocal<FlowAware> 类型对象,该对象是当前线程的局部变量, 将非线程安全的 FlowAware 变量存放在 ThreadLocal 中。

FlowAware 是流程关注接口, 具备流程操作的业务对象都需要集成该接口。

```

/**
 * 流程关注接口,
 *
 * 具备流程操作的业务对象都需要集成该接口
 * @author chenb
 *
 */
public interface FlowAware extends Id

```

该接口的API包括:

```

/**
 * 返回流程信息对象
 *
 * @return 流程对象
 */
Flow getFlow();

/**
 * 设置流程实例名称
 *
 * @param name 流程实例的名称或者标题
 */
void setProcessName(String name);

/**
 * 设置流程实例状态
 *
 * @param status 流程的业务状态
 */
void setProcessStatus(String status);

/**
 * 获取草稿ID
 *
 * @return 草稿id
 */
Long getDfId();

```

一般用法如下：

```

FlowAware flowAware = FlowUtils.getFlowAware();
FlowUtils.setFlowAware(flowAware);

```

6. StreamUtils

`StreamUtil1` 工具类提供了三个静态方法主要用于关闭流处理、获取资源URL、以及加载资源转换成流形式。API如下代码块所示：

```

public static void close(Closeable ...ins)
public static URL getResource(String resourceName, Class<?> callingClass)
public static InputStream getResourceAsStream(String resourceName, Class<?> callingClass)

```

7. XMLUtil

xml工具类主要用于XML文件的读取和解析、将node节点转化成xml字符串、通过xpath表达式查找节点，并返回第一个符合条件节点、saveXML。

```
//读取xml文件
public static Document parseXML(File file)
public static Document parseXML(String xmlFile)

//将node节点转化成xml字符串
public static String toXml(Node node)

//查找节点,并返回第一个符合条件节点 express 是 xpath表达式
public static Node selectSingleNode(String express, Object source)
public static NodeList selectNodes(String express, Object source)

//将Document输出到文件
public static void saveXML(String fileName, Document doc)
```

8.StringUtils

处理String类型的工具类,具体的应用场景为如下所示:

判断字符串是否是合法的java标识符

```
public static boolean isJavaIdentifier(String s)
```

java属性转换成数据字段,大写字前加下划线,即 "ammBmmCmm"--> "amm_Bmm_Cmm"

```
public static String convertPropertyNameToUnderscoreName(String name)
```

判断路径是否有效

```
public static boolean isValidPath(String path)
```

9.QueryUtils

用于在Resource层框架处理QueryFilter请求的工具类,调用UriInfo类得到一个查询的请求,返回一个Map类型的对象。QueryUtils提供解析该对象的静态方法,API如下:

```
public static QueryFilter parseMultiQuery(MultivaluedMap<String, String> query) throws BusinessException
public static QueryFilter parseMultiQuery(Map<String, String[] > query) throws BusinessException
public static QueryFilter parseQuery(Map<String, String> query) throws BusinessException
```

这些静态方法返回的是QueryFilter类型对象,接收该对象之后调用Service层对应方法,即可进行SQL过滤查询。

links

- [目录](#)
- 上一节: [Exception\(异常\)](#)
- 下一节: [小结](#)

小结

本章主要讲解了SmartAs框架的基础知识, 包括注解、Model、SQL扩展、DAO、Service、Resource、Util等小结。通过本章学习可以知道框架的核心模块是怎样组成的。

links

- [目录](#)
- 上一节: [Util\(工具类\)](#)
- 下一章: [开发篇](#)

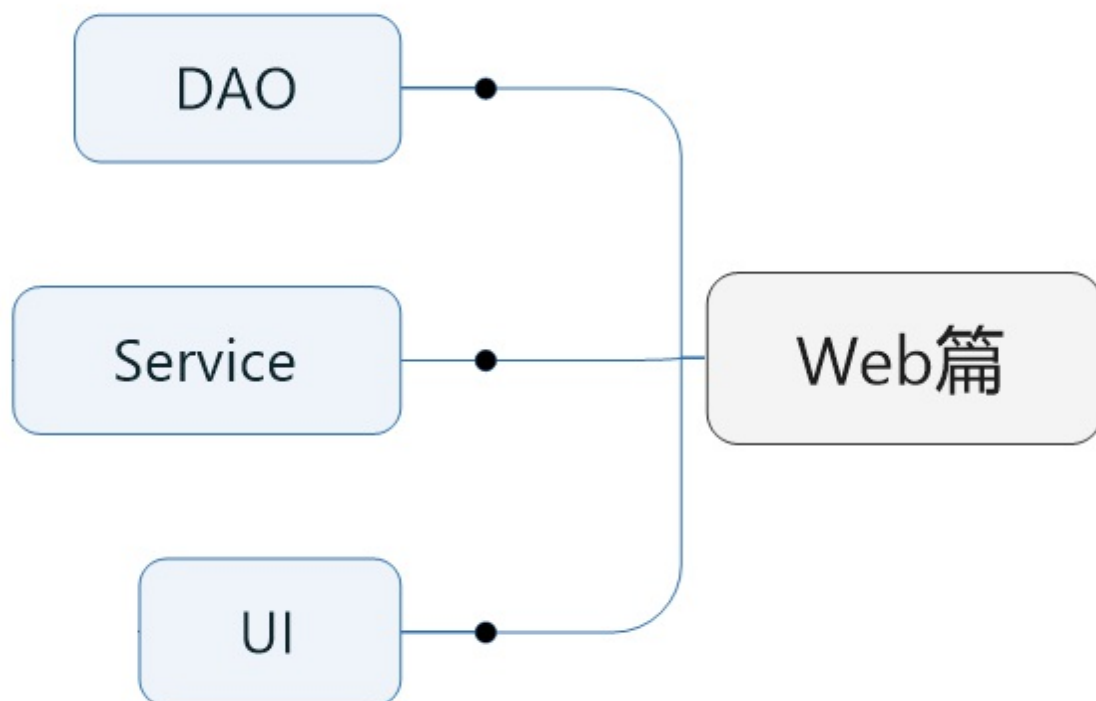
Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

1 开发篇

欢迎来到SmartAs的世界, 让我们开始探索吧!

在本章中, 我们将讲述基于SmartAs的应用本地环境配置(包括JDK, Tomcat, Eclipse等), 以及如何配置项目信息。

目录



links

- [目录](#)
- 下一节: [JAVA环境安装](#)

DAO

1 Dao描述

数据库持久层使用的是开源Mybatis + 标签定制和扩展方式实现, 通过mybatis-spring的扫描机制, 自动组装实现类。数据访问层只提供API, 且 SmartAS框架提供Dao父接口 `BaseDao<T>`, 自定义的Dao接口继承该接口, 则继承了 `BaseDao<T>` 定义的全部方法。在业务场景中如果继承方法不满足需求, 可以自定义方法提供数据访问接口。

2 自定义Dao

2.1 新建接口

在项目业务场景中, 如果Dao父接口API提供的方法不能满足数据库操作。则可以在自己的Dao接口下定义相应的方法名。如下代码块示例:

```
@Repository
public interface BrowseRecordDao extends BaseDao<BrowseRecord> {
    //根据id从km_knowledge表中查询得到对象其中的属性用户id
    BrowseRecord getById(@Param("id") Long id);
    //根据用户id从表tpl_user_t中得到orging
    BrowseRecord getOrginfoFromUserByUserId(@Param("userid") int userid);
    //根据orgcode从tpl_org_t表中得到机构名称orgname
    BrowseRecord getOrgNameByOrgcode(@Param("orgcode") String orgcode);
}
```

2.2 传递参数

传递参数用 `@Param("aaa") T aaa` 注解, 在mapping XML文件中通过 表达式语言 `#{aaa}` 获取该参数的值, 可以传递多个参数。

```
//根据用户id从表tpl_user_t中得到orging
BrowseRecord getOrginfoFromUserByUserId(@Param("userid") int userid);
```

在mapping xml文件中接收参数值如下代码块所示:

```
<select id="getOrginfoFromUserByUserId" resultMap="BrowseRecordResultMap">
    SELECT T.ORGINFO FROM tpl_user_t T WHERE T.ID = #{userid}
</select>
```

2.3 XML配置

Dao接口中的方法名必须和mapping XML文件的SQL语句标签例如 `Select、insert、update、delete` 的 `id` 属性值相一致。

2.4 注解

在自定义Dao接口, 接口体前要加 `@Repository` 注解, 如下段代码所示:

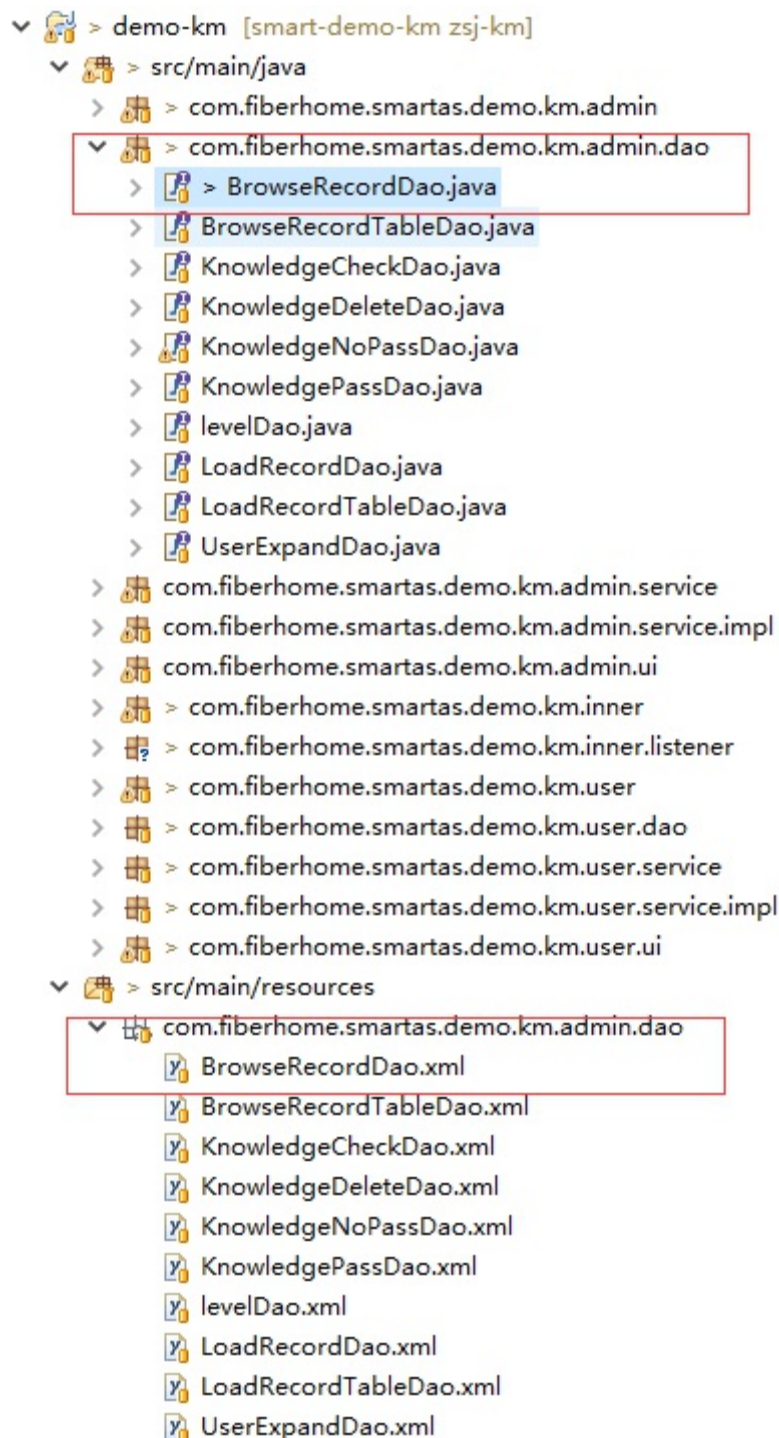
```
@Repository
public interface LoadRecordDao extends BaseDao<LoadRecord> {

}
```

3 映射XML标签

3.1 命名规范

对应的mybatis的mapping文件为resource目录下的(RegistryDao的packageName + RegistryDao.xml)方法名对应mapping文件中的statementName, 需要访问的参数直接由@Param注解指定的名称。



3.2 XML标签

根据自定义数据访问接口配置mapping XML文件, 该XML文件遵循的是mybatis标签结构。

<http://mybatis.org/dtd/mybatis-3-mapper.dtd>

3.3 mapper标签

首先配置 `mapper` 标签, `namespace` 属性与数据访问接口路径一致:


```
<mapper namespace="com.fiberhome.smartas.demo.km.admin.dao.BrowseRecordDao">
```

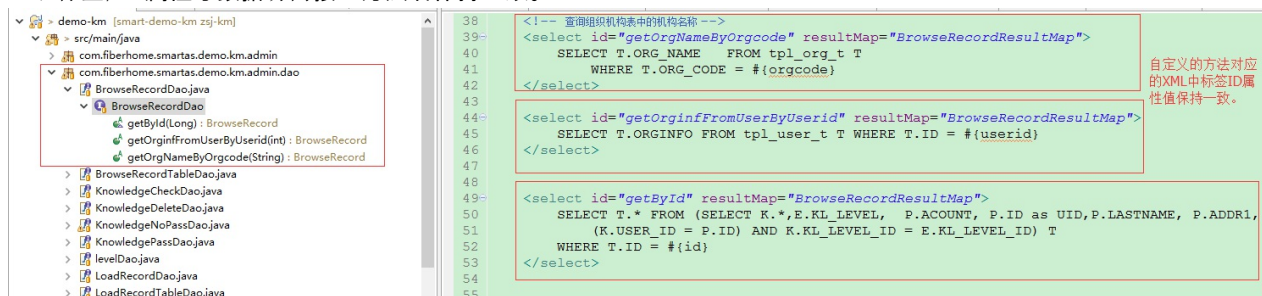
3.4 result标签

配置好 `resultMap` 以及 `result` 标签, 其中 `result` 标签对应的是数据表中的字段, 采用驼峰命名规则。

```
<result property="klId" column="KL_ID" jdbcType="INTEGER"/>
<result property="klName" column="KL_NAME" jdbcType="VARCHAR" />
```

3.5 SQL标签

SQL标签, ID属性与数据访问接口方法名保持一致。



自定义的方法对应的XML中标签ID属性值保持一致。

links

- [目录](#)
- 下一节: [Service](#)

Service

1 Service描述

Service层是SmartAs框架的服务层，提供事物支持，对外提供原子操作的服务。在该接口中定义服务相关的方法名，继承框架提供父接口 `BaseService<T>`，不必再重写基础方法。接下来介绍如何写模块的Service层以及实现层。

2 继承Service父接口

2.1 框架服务接口

SmartAs框架对外提供的接口 `BaseService<T>`，业务模块创建自己的Service接口时也要继承该父接口。

```
public interface BrowseRecordService extends BaseService<BrowseRecord>
```

2.2 框架服务接口API

框架提供的数据访问接口一共定义了14个数据访问方法，包括7个查找服务方法、2个保存服务方法、2个更新服务方法、3个删除服务方法。API如下图所示。

```
Service<T extends POJO, PK extends Serializable>  
  A get(PK) : T  
  A get(PK[]) : List<T>  
  A find(PK) : T  
  A getAll(int, int) : Pageable<T>  
  A getAll(QueryFilter, int, int) : Pageable<T>  
  A getAll() : List<T>  
  A getAll(QueryFilter) : List<T>  
  A save(T) : T  
  A save(List<T>) : List<Serializable>  
  A update(List<T>) : void  
  A update(T) : T  
  A remove(PK) : void  
  A remove(PK[]) : void  
  A remove(T) : void
```

2.3 处理异常

异常处理问题，业务异常如返回值为null则抛出 `BusinessAccessException` 异常。

2.4 事物注解

事物注解，加事物注解保持事物处理的原子性，可在方法、类上一行加该注解。

```
@Transactional(readOnly = true)
```

3 扩展Service接口方法

在业务管理接口中，根据业务需求自定义自己的方法。需要注意如需事物支持则用事物注解，处理业务异常抛出异常。

```
public interface BrowseRecordService extends BaseService<BrowseRecord> {  
    @Transactional(readOnly=true)  
    BrowseRecord getOrgnameByOrgcode(String orgcode) throws BusinessException;  
    @Transactional(readOnly=true)  
    BrowseRecord get(Long id) throws BusinessException;  
    @Transactional(readOnly=true)  
    BrowseRecord getOrginfByUserid(int userid) throws BusinessException;  
}
```

4 Service实现类Impl

4.1 新建服务接口

Service接口的实现类命名规范为 `ServiceImpl`，继承SmartAs框架提供的 `BaseServiceImpl<T>`，实现自定义的Service接口。类前面加注解 `@Service`，如下代码块所示：

```
public class BrowseRecordServiceImpl extends BaseServiceImpl<BrowseRecord> implements BrowseRecordService {
```

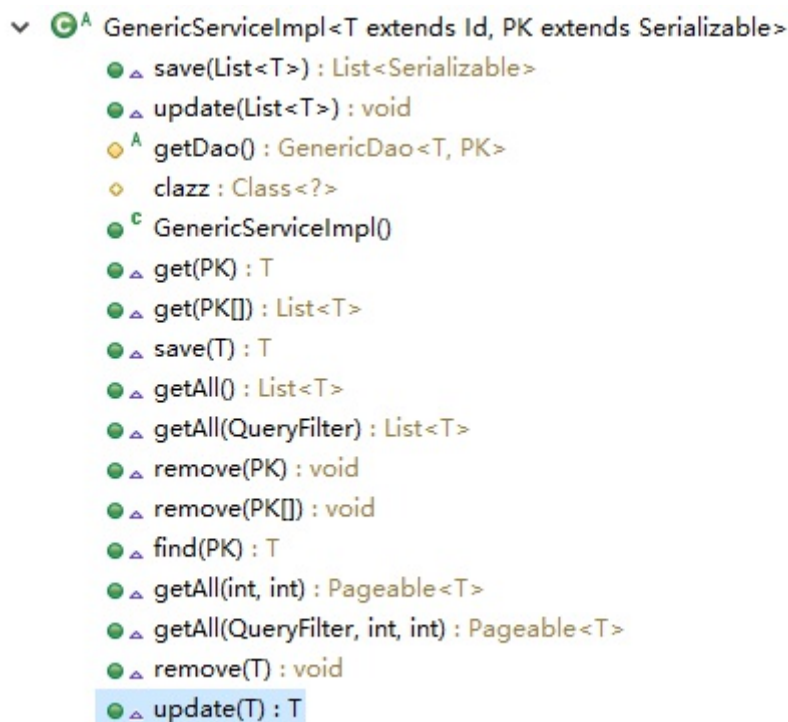
4.2 自动装配dao

用自动装配注解 `@Autowired` 注入对应dao对象，Spring容器会将其实例化。如下代码块所示：

```
@Service  
public class BrowseRecordServiceImpl extends BaseServiceImpl<BrowseRecord> implements BrowseRecordService {  
    @Autowired  
    private BrowseRecordDao dao;  
    protected BrowseRecordDao getDao() {  
        return dao;  
    }  
}
```

4.3 框架服务实现类API

SmartAs框架提供的ServiceImpl类封装了14个基本服务实现方法，API如下图所示：



4.4 自定义方法

写自己的服务实现方法，这和写一般的java类一样，主要是实现Service自定义的方法，示例代码如下所示：

```
private static int tempUserId;//设置一个用户id的全局变量
//根据ID从kl_knowledge表中得到userid
public BrowseRecord get(Long id) throws BusinessException {
    BrowseRecord browseRecord = new BrowseRecord();
    browseRecord = getDao().getById(id);
    tempUserId = browseRecord.getUserId();//得到根据id查询的browseRecord对象中的userid值
    return browseRecord;
}
//根据userid从tpl_user_t表中得到orginf("ROOT,3,31")
public BrowseRecord getOrginfByUserId(int userid) throws BusinessException {
    return dao.getOrginfFromUserByUserId(tempUserId);
}
//根据orginf, split(",")得到orgcode, 从表tpl_org_t得到机构名称
public BrowseRecord getOrgnameByOrgcode(String orgcode) throws BusinessException {
    return dao.getOrgNameByOrgcode(orgcode);
}
```

links

- [目录](#)
- [上一节: DAO](#)
- [下一节: UI](#)

UI层相当于SSH框架中的Action层，引用对应的Service层

- UI层继承了SmartAs框架提供的基础资源实现父类 `BaseResource<Upload>`

```
public class UploadUI extends BaseResource<Upload>
```

- 基础资源父类 `BaseResource<Upload>` 又继承了框架提供的通用资源父类 `GenericResource<T, Long>`

```
public abstract class BaseResource<T extends Id> extends GenericResource<T, Long>    //T为业务对象类型
```

- 通用资源父类 `GenericResource<T, Long>` 提供最基本的增删查改操作

现引用一个demo文件对UI层内部结构进行分析：

```
@Path("/web/demo/kmct/upload")
@Resource(code = 90016, model = "Kmct", desc = "Upload Resource")
public class UploadUI extends BaseResource<Upload> {
    @Autowired
    private UploadService service;

    protected UploadService getService() {
        return service;
    }

    //根据具体业务需要实现的UI方法
    @GET
    @Path(value = "/index")
    @Operation(code = Operation.READ, desc = Operation.READ_DESC)
    public String index() {
        return null;
    }

    @GET
    @Path(value = "/fileId")
    @Operation(code = Operation.READ, desc = Operation.READ_DESC)
    public Upload getFileId() {
        return getService().getFileId();
    }
}
```

```
@Path("/web/demo/kmct/upload")
```

- Path注解

整个UI文件的路径，与前端JSX页面中定义的service路径相对应

```
const service = Service.New("/web/demo/kmct/upload"); //前端jsx页面
```

```
@Resource(code = 90016, model = "Kmct", desc = "Upload Resource")
```

- Resource注解

定义对应功能模块的权限，code表示权限码，model表示模块名称，desc功能描述

```
@Autowired
private UploadService service;

protected UploadService getService() {
    return service;
}
```

- 装配对应的service服务

当**GenericResource**父类提供的API满足不了我们具体的业务需求时，就需要我们自己来实现，例如：

```
@GET
@Path(value = "/fileId")
@Operation(code = Operation.READ, desc = Operation.READ_DESC)
public Upload getFileId() {
    return getService().getFileId();
}
```

- @GET

表示为请求方法为GET，要向数据库拿数据。还有：

- @POST:插入数据时使用
- @GET:更新数据时使用
- 当业务复杂对这两者不确定时，建议用@POST

```
@Path(value = "/fileId")    //该方法的路径
@Operation(code = Operation.READ, desc = Operation.READ_DESC)    //该方法对应的操作权限
```

- 最后return service层对应的具体方法

links

- [目录](#)
- 上一节: [Service](#)

5.进阶篇

在本章中将会为您讲述SmartAs中的功能要点。

目录

- 5.1. [通用功能](#)
- 5.2. [用户和权限管理](#)
- 5.3. [工作流](#)
- 5.4. [DevOps](#)
- 5.5. [服务器管理](#)
- 5.6. [小结](#)

links

- [根目录](#)
- 下一节: [通用功能](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

1 任务管理

任务管理主要用于定时任务调度, 使用Spring task实现, 支持2种方式的定时任务。

1. cron表达式的方式
2. 间隔时间循环执行

2 任务设置

任务设置包含的内容有 名称、描述、Cron表达式、开始执行时间、结束执行时间、最大执行次数、完整类路径、执行类方法名、延迟时间、间隔时间

完整类路径+执行类方法名 决定了任务具体的执行内容, 使用反射机制调用

Cron表达式 和 间隔时间 2者只需设置其中之一即可, 如果都设置了, 以Cron表达式为准。

开始执行时间、结束执行时间、最大执行次数代表了任务的执行条件, 如果没有设置, 则没有约束条件。

开始执行时间: 当前时间达到开始执行时间, 任务才会进入等待执行状态, 按照Cron表达式或者间隔时间来执行任务。

结束执行时间: 当前时间达到结束执行时间, 任务立即进入完成状态, 不再执行任务。

最大执行次数: 当前执行次数如果达到最大执行次数, 任务立即进入完成状态, 不再执行任务。

结束执行时间 和 最大执行次数 两者只要满足其一, 任务就立刻进入完成状态, 不再执行任务。

最大执行次数 设置值如果小于1, 则代表没有执行次数的限制。延迟时间、间隔时间的单位为秒。

Cron表达式的具体使用方式, 可以自行网上查看相关资料。设置了延迟时间和间隔时间, 则代表 延迟多长时间之后, 每隔多少时间执行一次任务。

3 任务暂停与启动

用户可以通过管理界面, 手动控制任务的执行。

如果一个任务处于没有结束的状态, 用户可以手动暂停任务, 任务将不再执行, 直到用户点击启动任务才能继续执行任务。

links

- [目录](#)
- 上一节: [数据字典](#)
- 下一节: [异步服务框架](#)

1.新类型消息的开发，需要配置对应的Channel(type + 'Channel')，并开发对应的通道的处理器。在业务的配置文件 root-context.xml中，具体配置如下：

```
<bean id="exportExecutor" class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
    <property name="corePoolSize" value="5" />
    <property name="maxPoolSize" value="10" />
    <property name="queueCapacity" value="25" />
</bean>

<int:channel id="exportChannel" >
    <!--int:queue capacity="25"/-->
    <int:dispatcher task-executor="exportExecutor"/>
</int:channel>

<int:service-activator input-channel="exportChannel">
    <bean class="com.fiberhome.smartas.excel.handler.ExportExcelHandler"></bean>
</int:service-activator>
```

- 上述代码中的也可以放在 smartas-message 中的配置文件中统一配置，然后在这里只需要通过id调用就行

2.在业务service实现类中，装配MessageBus。调用消息总线MessageBus入口发送消息

```
@Autowired
private MessageBus messageBus;

messageBus.send(Message.Builder.with("mail", payload).build());
```

- mail 为消息类型
- payload 为业务对象

links

- [目录](#)
- 上一节: [定时任务](#)
- 下一节: [Lookup管理](#)

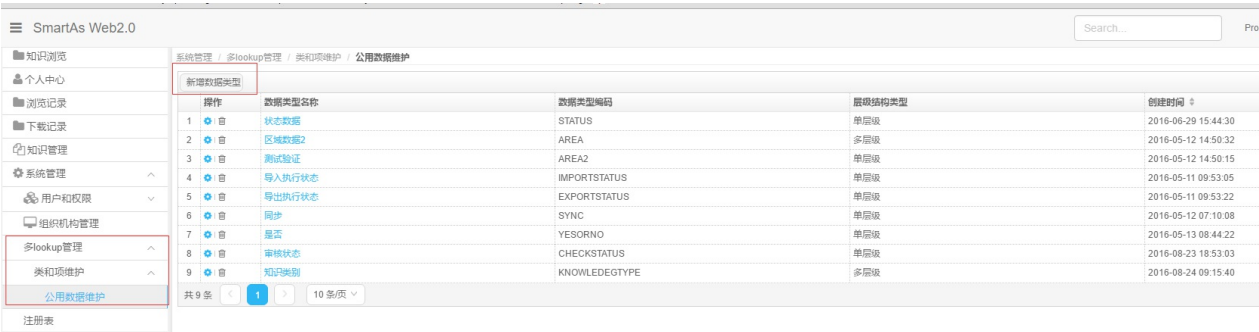
LookUp管理

1简介

LookUp是SmartAs提供的一个数据字典服务，在LooUp管理界面中对页面动态下拉框涉及的值进行定义如性别(男、女)。LookUp服务方便UI控件的实现。

2新增字典

进入系统管理->LookUp管理界面，点击“新增数据类型”按钮。填入对应信息，数据类型编码一般采用大写英文拼写，层次结构类型分为单层级和多层级，根据业务需求选择。



3配置Lookup

3.1 在功能对应的JAVABean下字段进行LookUp配置，如下面代码所示：

```
@Lookup(type = "STATUS")
private Short status;
```

3.2 如果字段为用户Id，字段的LookUp配置代码如下所示：

```
@User
private Integer userId;//添加User注解
```

3.3 如果字段为部门CODE，字段的LookUp配置代码如下所示：

```
@Org(Type.CODE)
private String orginfo;
```

4页面显示

4.1 在前端jsx文件中，在Grid或者Table组件显示遵循对应的命名规则。

```

{
  title: '状态',
  dataIndex: 'statusDesp',
},
{
  title: '用户ID',
  dataIndex: 'userId',
},
{
  title: '用户名称',
  dataIndex: 'userIdAccount',
},
{
  title: '姓名',
  dataIndex: 'userIdLastName',
},
{
  title: 'EMAIL',
  dataIndex: 'userIdEmail',
},
{
  title: '组织机构',
  dataIndex: 'orginfoName',
},

```

4.2 在下拉框显示LookUp字段

在Form中添加一个下拉框动态显示定义的LookUp字段，一般是类型等数据。用 `Lookup` 标签显示字段，示例如下代码块所示：

```

<Form inline onSubmit={this.props.querySubmit}>
  <FormItem label="审核状态:">
    <Lookup groupCode="CHECKSTATUS" type='array' placeholder="请选择审核状态" {...klStatus} />
  </FormItem>
  {this.props.children}
</Form>

```

links

- [目录](#)
- 上一节: [异步服务框架](#)
- 下一节: [菜单管理](#)

菜单管理

1简介

菜单管理服务是SmartAs框架提供给业务开发人员进行菜单管理功能页面的服务, 系统业务功能入口统一通过菜单服务进行配置。在菜单管理服务中用户可以从菜单目录树中合适的位置新增菜单, 通过配置菜单表单的属性对菜单的名称、路径、权限码、样式、图标、锚点、排序号进行配置。

2菜单配置

菜单属性表单如下图所示。

2.1 其中"名称"是必填项, 显示该菜单的名称。

2.2 "URL"是jsx文件的路径以"#!"开头。jsx文件位于 `src/main/resources/web` 下面, '#!'后面以web开始

```
web/demo/Demo.jsx
```

2.3 权限码是配置用户是否有权限操作该功能,权限码由资源码+操作码组成。资源码是用户是否对模块具有权限, 操作码是用户是否对该模块下的某些操作有权限。注意:模块的代码不能用20000以下, 20000以下为系统层使用。Operation提供000到006一共7个标准动作权限, 200以下为平台权限, 不要使用。在该处根据实际情况可以不输入、只输入资源码、输入资源码+操作码。在第3小结会对不同输入如何控制权限展开说明。

2.4 图标是显示菜单的一个图状标识。参考网址如下:

<http://www.fontawesome.com.cn/faicons/>

2.5 锚点是控制页面跳转的一个参数, 在 `UI.forward(url, '锚点')` 函数中调用。如下代码块所示:

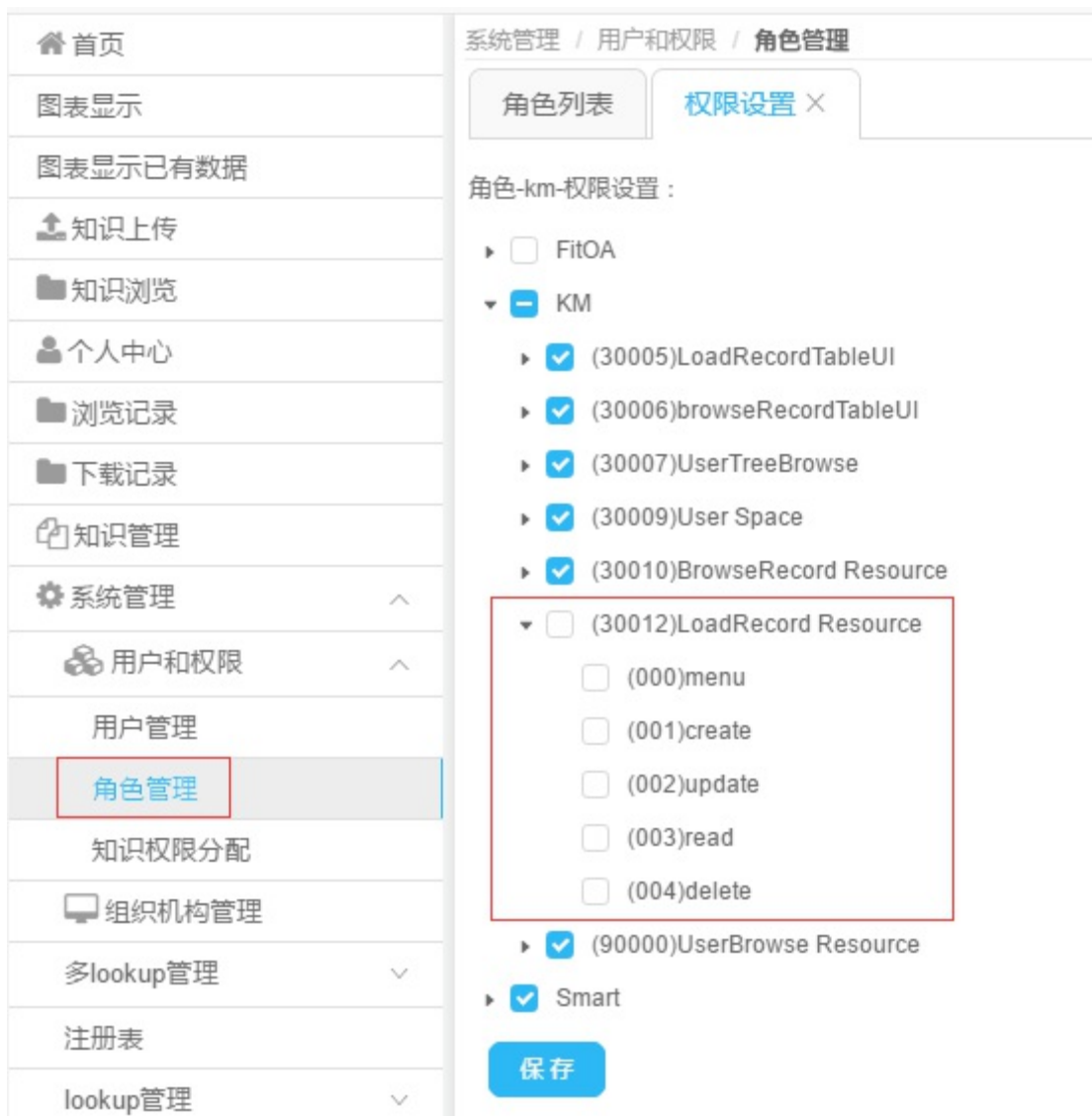
```
{
  text: '新增',
  icon: 'add',
  handler: function(){
    var url= "web/demo/librarySys/AddBook.jsx";
    UI.forward(url, 'book'); //UI.forward(锚点)
  }
}
```

2.6 排序号是该菜单排序的位置, 然后勾选是否发布点击确定按钮菜单配置完成。

* 名称:	用户管理
URL:	#/web/smartas/security/user/userList2.jsx
权限码:	10100.000
Class:	样式
Icon:	图标
锚点:	userList2
排序号:	0
<input checked="" type="checkbox"/> 是否发布	
<button>确定</button>	

3 菜单权限控制

3.1 如果不需要隐藏菜单名称，在配置菜单权限码时不输入权限码，只要发布成功菜单就会一直显示在菜单目录树上。在角色管理的时候如果没有给用户该菜单对应业务模块的权限，则点击菜单会出现"无权操作"。



3.2 如果需要对用户隐藏菜单名称, 在配置菜单权限码时应该输入权限码, 该权限码的值与对应业务模块UI层给的权限码值一致。如果用户的权限码集合包含该权限码, 则用户对该菜单有权操作, 否则无权操作且该菜单不显示对用户隐藏。

系统管理 / 菜单管理

新增 删除

Root

- 工具菜单
 - Dashboard
- 导航菜单
 - 首页
 - 图表显示
 - 图表显示已有数据
 - 账户设置
 - 知识上传
 - 知识浏览
 - 我的工作空间
 - 个人中心
 - 浏览记录
 - 下载记录
 - 知识管理
 - 系统管理
 - 开发

* 名称: 下载记录

URL: #/web/km/admin/LoadRecord.jsx

权限码: 30012

Class: 样式

Icon: folder

锚点: loadrecord

排序号: 7

☒ 是否发布

确定

3.3 在配置菜单权限码时输入带操作码的权限码，则对模块的对应操作码的操作进行了权限控制。

SmartAs Web2.0

系统管理 / 菜单管理

新增 删除

Root

- 工具菜单
 - Dashboard
- 导航菜单
 - 首页
 - 图表显示
 - 图表显示已有数据
 - 账户设置
 - 知识上传
 - 知识浏览
 - 我的工作空间
 - 我的待办
 - 我的参与
 - 我的发起
 - 我的已办
 - 我的导入
 - 我的导出
 - 草稿箱
 - 个人中心
 - 浏览记录
 - 下载记录
 - 知识管理
 - 系统管理
 - 开发

* 名称: 我的待办

URL: #/web/smartas/workflow/ToDo.jsx

权限码: 12001.101

Class: 样式

Icon: 图标

锚点: ToDo

排序号: 0

☒ 是否发布

确定

系统管理

用户和权限

用户管理

角色管理

知识权限分配

组织机构管理

多lookup管理

注册表

lookup管理

- ☒ (10120)I18N Resource
- ☒ (10201)Registry Resource
- ☒ (10301)Lookup UI
- ☒ (10302)Area UI
- ☒ (12001)Workspace Resource
 - ☒ (101)to-do list
 - ☒ (102)done list
 - ☒ (103)handin list
 - ☒ (104)spoon list
 - ☒ (110)drafts
- ☒ (12005)Draft Resource

links

- [目录](#)
- 上一节: [Lookup管理](#)
- 下一节: [国际化信息管理](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

国际化与本地化

1 简介

现阶段SmartAs并未实现完整的国际化, 仅提供前端组件支持, 该组件可针对文字或组合文字信息在不同语境下替换。

2 开发流程

以下示例页面可参考demo工程中的I18N.jsx

第一步 添加国际化数据

- 进入 系统管理 - 国际化信息管理
- 该页面可为需要国际化的数据提供语料, 单个数据填写时刻参考左侧的查询结果集, 如在项目实施过程中需要填入大量数据, 建议直接在数据库内导入。
- 如需增加语言, 可在【多lookup管理】的【语言】这项中添加。
- 添加相应的键值对后, 可在界面中调用这些内容。

第二步 修改页面

```
//声明UI
const {UI} = Smart;
...
//假设国际化数据为 key:test.abc value:This is a #a# and #b# test.
//调用
UI.I18N('test.abc',{a:'Hello',b:'123'});
```

- 将得到 This is a Hello and 123 test.
- 得到的内容可以文字形式组织在页面上, 也可以组件的形式渲染到页面

3 加载过程

1. 用户可设置自己的语言习惯, 菜单在 右上角设置 - 账户信息, 设置完毕后需重新登陆启用新的语言习惯设置
2. 加载流程为登录后加载主界面时, I18N.init.js将判断根据浏览器localStorage中的时间同I18N键值对更新的时间做对比, 并抓取差异部分(经gzip压缩)写入localStorage
3. 在调用UI.I18N时, 组件根据当前用户语言, 在localStorage中查询指定的key, 并将value及参数合并返回。
4. 返回的数据为字符串形式, 除文字信息外, 可根据react的特点实现动态组件, 但不建议将太复杂的内容写入国际化数据。
5. 注: 由于国际化在项目中数据量会比较大, 因此数据写入了浏览器localStorage中, 该功能为html5的一个扩展, 早期IE6,7,8不能提供支持, 其余版本已测试结果为准。

4 排查异常

- 正确录入国际化数据后, 如开发页面无法获取, 可检查浏览器的localStorage内是否能查到该数据。
- 如localStorage为空, 可检查I18N.init.js是否顺利加载。

其他

一些可能的扩展方向

- 如需实现完整国际化, 还需对SmartAs全部硬编码做提取properties, 以及标点符号位和组件搭配习惯的调整。

links

- [目录](#)
- 上一节: [菜单管理](#)
- 下一节: [导入管理](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

1 导入设置

用于事先设置一个导入的基本信息, 包括: 类型、描述、导入模板、同步、导入实现类全路径、是否分页导入、分页记录数

类型: 需要全局唯一

上传导入模板: 可以在导入界面, 让用户下载查看导入的文件格式

同步: 代表处理导入文件的数据是同步还是异步处理

导入实现类全路径: 必须实现接口 `ImportExcelForListString` (支持分页导入) 或者 `ImportExcelForInputStream` (不支持分页导入), 实际用来处理导出数据的地方

分页导入: 实现接口 `ImportExcelForListString`, 设置时才有效

分页记录数: 实现接口 `ImportExcelForListString`, 设置时才有效

2 导入监控

可以查询所有用户的导入信息, 查看导入执行情况, 以及下载用户上传的文件。

3 我的导入

可以查询当前用户的导入信息, 查看导入执行情况, 以及下载用户上传的文件。

4 导入演示

这里只是演示使用导入的一种情况,

在隐藏域中保存导入的类型ID, 设置一些导入的参数, 上传导入文件, 提交之后, 就会根据事先设置的导入信息, 对导入文件进行处理, 可以在我的导入或者导入监控中, 查看导入的执行结果。

links

- [目录](#)
- 上一节: [国际化信息管理](#)
- 下一节: [导出管理](#)

1 导出设置

用于事先设置好一个导出的信息, 包括:

类型、描述、导出模板、同步、导出实现类全路径、是否分页导出、分页记录数

类型:需要全局唯一

导出模板:用于在用户界面显示出导出文件的格式

同步:代表处理导出文件的数据是同步还是异步处理

导出实现类全路径:必须实现接口ExportExcelForListString, 实际用来处理导出数据的地方

2 导出监控

可以查询所有用户的导出信息, 查看导出执行情况, 以及下载用户导出的文件。

3 我的导出

可以查询当前用户的导出信息, 查看导出执行情况, 以及下载用户导出的文件。

4 导出演示

这里只是演示使用导出的一种情况,

在隐藏域中保存导出的类型ID, 设置一些导出的参数, 提交表单, 就会根据事先设置的导入信息, 对导入文件进行处理, 可以在我的导出或者导出监控中, 查看导出的执行结果, 并下载导出的文件。

links

- [目录](#)
- 上一节: [导入管理](#)
- 下一节: [日志服务](#)

日志服务

1 简介

为SmartAs提供日志记录, 将非读操作写入日志表, 并记录基本操作数据, 方便管理员及开发人员出现数据异常时检查。

2 查看记录

- 系统管理-服务器管理-操作日志
- 可根据条件查询相关记录

3 服务加载过程

- 用户发送REST请求, 权限检查器SecurityBinder内, 在检查用户权限后将操作相关数据写入日志表
- 注意: 该表不提供删除修改功能, 交由数据库运维人员自行制定定期删除策略

4 排查异常

- 业务表异常可提取相关日志记录, 检查输入参数判断异常来源
- 开发异常可检查服务器输出的log是否包含如下字段

```
can't write to tpl_action_t with ...
```

并根据相关操作代码查找异常的REST请求

links

- [目录](#)
- 上一节: [导出管理](#)
- 下一节: [富文本](#)

1 富文本编辑器

框架集成了百度的UEditor, 提供在线富文本编辑, 获取对应的代码html功能。

例子代码如下:

```
const {UI,Service} = Smart;
const {Ueditor} = UI;
const DemoApp = React.createClass({

  getValue : function(){
    return UE.getEditor("editor").getContent();
  },

  render(){
    return (
      <div>
        <Ueditor value="<h1>标题</h1>" id="editor" height="300" width="800"/>
      </div>
    );
  }
});
```

links

- [目录](#)
- 上一节: [日志服务](#)
- 下一节: [规则引擎](#)

Upload上传

附件上传功能用的是Ant Design的upload控件，该控件根据实际的功能需求，有几种表现形式，例如对上传的列表进行控制和编辑，拖拽上传等。由于上传后台实现方式都是一样的，在这里，我们只介绍基于SmartAs框架的附件上传功能的最基本的实现，如需其他实现方式，请参考(<http://ant.design/components/upload/>)

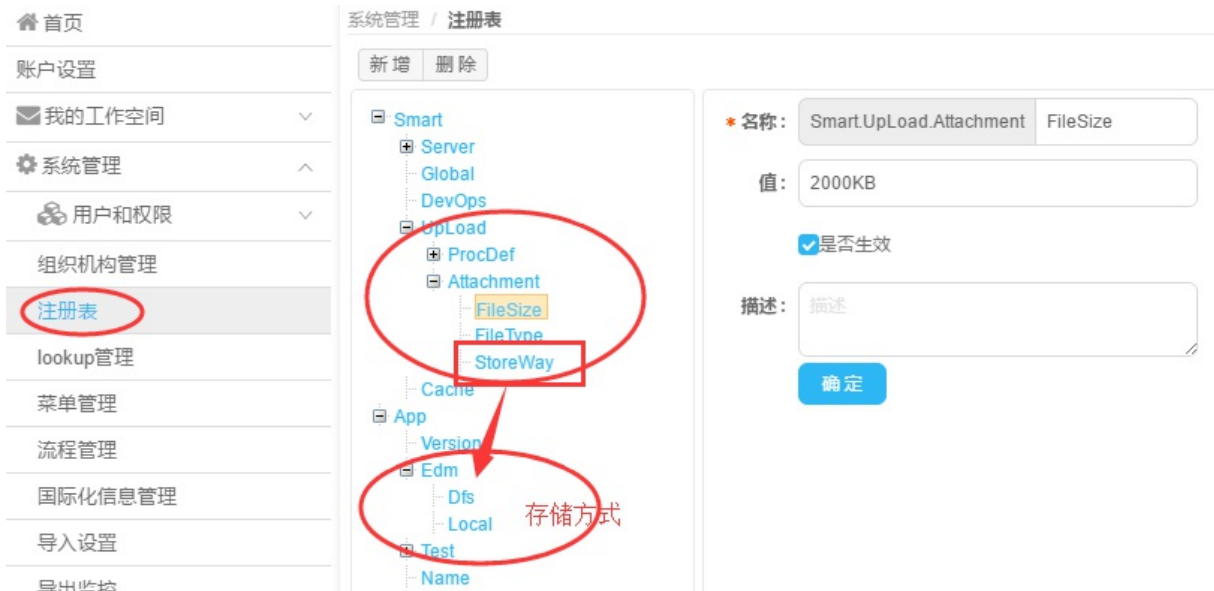
Jsx页面代码样式如下：

```
const { Upload, message, Button, Icon } UI;
const props = {
  name: 'file',
  multiple: false,
  action: 'services/file/upload/Attachment',
  data:{
    batchNo: batchNo,
  },
  onChange(info) {
    if (info.file.status !== 'uploading') {
      console.log(info.file, info.fileList);
    }
    if (info.file.status === 'done') {
      message.success(`${info.file.name} 上传成功。`);
    } else if (info.file.status === 'error') {
      message.error(`${info.file.name} 上传失败。`);
    }
  },
};
ReactDOM.render(
  <Upload {...props}>
    <Button type="ghost">
      <Icon type="upload" /> 点击上传
    </Button>
  </Upload>
, mountNode);
```

附：另一种上传控件Plupload(强烈推荐)

在实现上传功能前，首先在前端页面配置好注册表中关于上传的参数，包括附件大小、附件种类、附件存储方式以及附件存储路径。

- 首页菜单-->系统管理-->注册表



- 导出监控
- App/Edm/Dfs 文件分布式存储, 需指定服务器
- App/Edm/Local 文件本地存储, 需指定本地存储地址

jsx页面Plupload组件代码如下:

```
const props = {
```

```
  autoUpload:false,
  id:"plupload",
  runtimes:'html5,flash',
  multipart:true,
    chunk_size:'10mb',
  url:'services/file/upload/Attachment',
  flash_swf_url:'web/common/plupload/Moxie.swf',
  customParams:{
    projId:1
  }
}
```

```
};
```

Download下载

附件下载调用后台服务, 只需在jsx页面中提供文件名fileName和文件路径filePath即可:

```
const AttaColumns=[
  {
    title:'文件名称',
    dataIndex:'fileName',
  },
  {
    title:'文件路径',
    dataIndex:'filePath',
  },
  {
    title: '文件下载',
    dataIndex: 'fileDownload',
    render(text,record) {
      return <a href={'services/file/download/Attachment?fileName='+record.fileName+'&filePath='+record.filePath}>下载</a>;
    }
  },
];
```

links

- [目录](#)
- 上一节: [规则引擎](#)
- 下一节: [组织机构](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

首页

1 简介

首页页面集成了框架的一些元数据, 如框架名称、版本号、版权信息。加载这些数据的页面路径为:

```
smartas-web/src/main/resources/ftl/smartas/web/index.ftl
```

在SmartAs框架的数据表 `tpl_registry_t` 表中包含了首页集成的元数据。可以根据项目实际情况修改注册表中对应的数据。

在IndexHandler类中处理元数据信息然后传递数据, 在index.ftl中接收数据。

```
/smartas-web-support/src/main/java/com/fiberhome/smartas/web/support/handler/IndexHandler.java
```

加载 `css` `js`

```
modelAndView.addObject("css", css);
modelAndView.addObject("commonJs", commonJs);
modelAndView.addObject("coreJs", coreJs);
modelAndView.addObject("uiJs", uiJs);
```

```
加载框架元数据 modelAndView.addObject("profile", StringUtils.join(profiles, ","));
modelAndView.addObject("authInfo", AuthInfoUtils.getAuthInfo()); modelAndView.addObject("version", version);
modelAndView.addObject("name", name); modelAndView.addObject("homePage", homePage);
modelAndView.addObject("navStyles", navStyles);
```

在Web-context.xml 中配置bean的属性对应js文件的位置

```
/smartas-web/src/main/resources/config/web-context.xml
```

```
<bean id="indexHandler" parent="baseDevIndexHandler">
<property name="url" value="smartas/web/index" />
<property name="commonJs">
<list merge="true">
<value>web/common/react/browser.min</value>
</list>
</property>
<property name="uiJs">
<list merge="true">
<value>web/ui/react-debug</value>
</list>
</property>
</bean>
```

在index.ftl中对应项目首页的title、项目名称、项目版本号等对应的位置为

```
<title>${name}</title>

<#list css as c><#lt>
<link href="${c}.css?v=${version}" rel="stylesheet">
</#list><#lt>

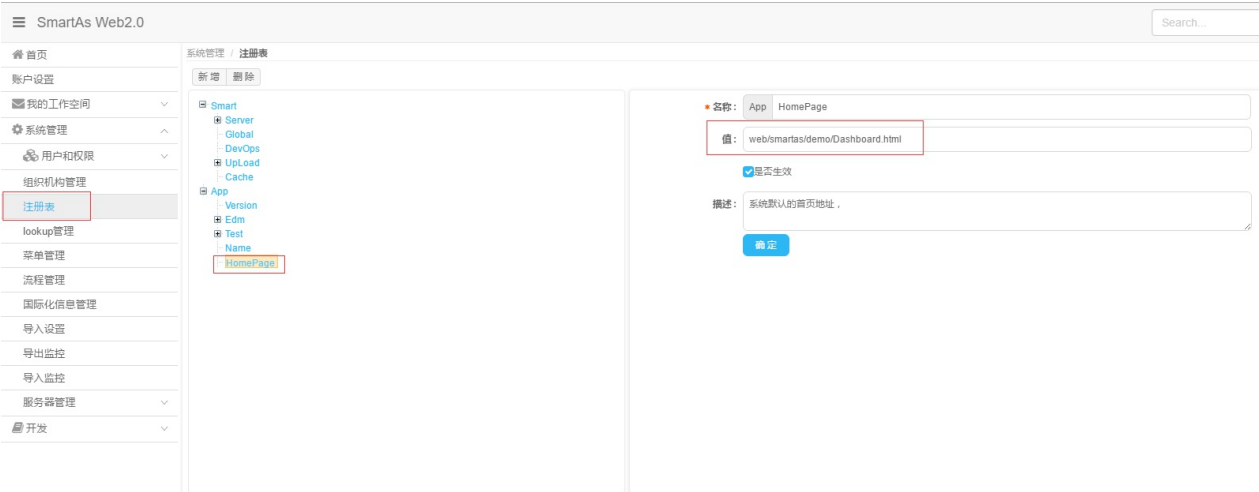
<a class="navbar-brand navbar-left" id="sidebar-toggle"><i class="fa fa-bars"></i></a>
<a class="navbar-brand" href="#!">${name}</a>
```

2 修改页面

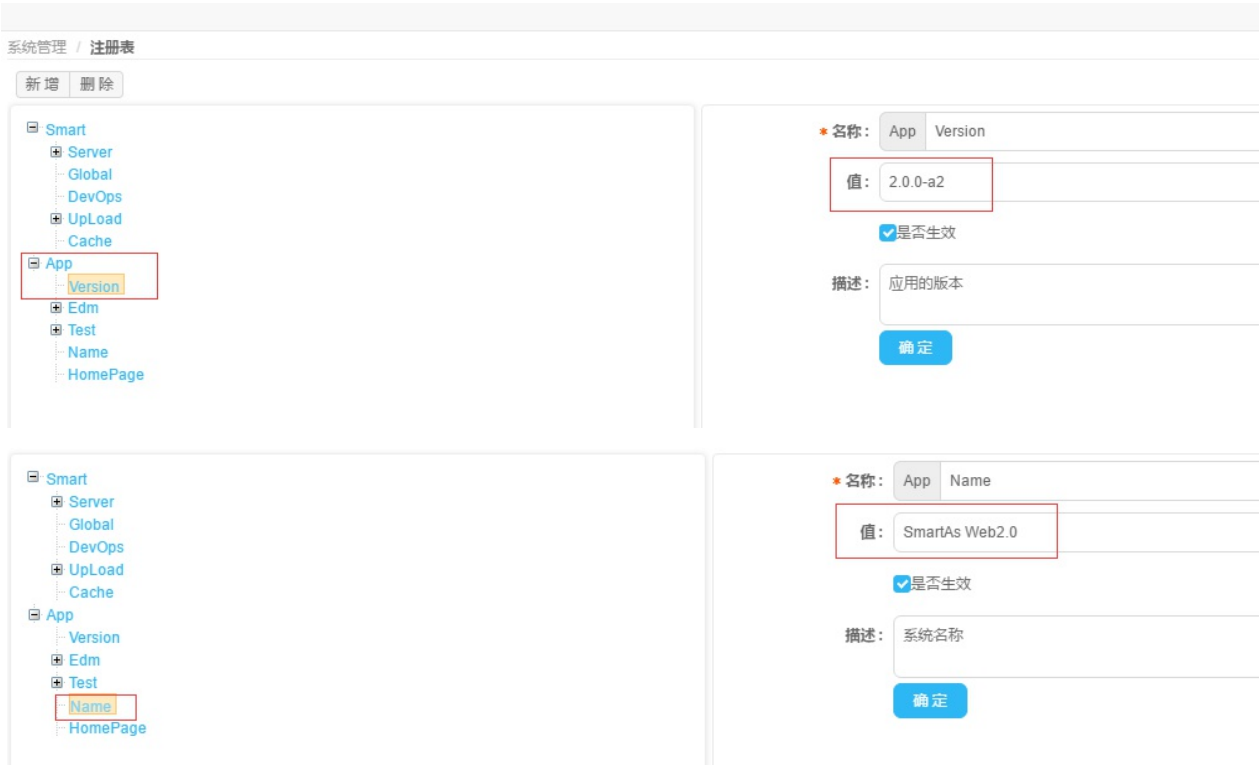
2.1 业务开发人员可以根据具体项目需求指定首页，点击界面首页目录出现如下的html页面。

!web/smartas/demo/Dashboard.html

在该页面下可以编辑呈现项目首页信息。如果需要重新指定首页的URL，可在注册表里进行配置如下图所示。



2.2 同理可以再注册表中根据项目需求配置项目名称name、项目版本号version，在注册表中修改如下图所示：



2.3 首页页面的样式文件.css路径为

smartas-web-resources/src/main/resources/web/ui/css

links

- [目录](#)
- 上一节: [组织机构](#)
- 下一节: [Session管理](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

SmartAs的缓存管理根据项目需求,分为本地缓存和redis缓存,具体配置如下:

本地缓存

在smartas-web工程下面的root-context.xml文件中import缓存配置文件

```
<!-- Root Context: defines shared resources visible to all other web components -->
<import resource="classpath:/META-INF/spring/root-context.xml" />

<!-- 本地缓存配置 -->
<import resource="classpath:/META-INF/spring/local-cache.xml" />

<!-- redis缓存配置 -->
<!-- <import resource="classpath:/META-INF/spring/redis-cache.xml" /> -->
```

redis缓存

1.在 smartas-web 工程下面的 app.properties 文件中配置 redis 的参数

```
#redis配置
#redis dev configuration
dev.redis.host=10.121.4.80
dev.redis.port=6379
dev.redis.pool=true
dev.redis.password=
dev.redis.timeout=2000

#redis configuration
redis.host=10.121.4.80
redis.port=6379
redis.pool=true
redis.password=
redis.timeout=2000
...

- redis.host 服务器地址
- redis.port redis的监听端口, 默认端口为6379
- redis.pool 是否使用连接池
- redis.password 授权密码
- redis.timeout 当客户端闲置多长时间后关闭连接
```

2.在 smartas-web 工程下面的 root-context.xml 文件中 import 缓存配置文件

```
<!-- Root Context: defines shared resources visible to all other web components -->
<import resource="classpath*:META-INF/spring/root-context.xml" />

<!-- 本地缓存配置 -->
<!-- <import resource="classpath:/META-INF/spring/local-cache.xml" /> -->

<!-- redis缓存配置 -->
<import resource="classpath:/META-INF/spring/redis-cache.xml" />
```

- 单应用用本地缓存
 - 多应用用redis
 - 两者只能import其一
-

links

- [目录](#)
- 上一节: [首页](#)
- 下一节: [消息总线](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

注册表主要是修改框架和应用的一些配置信息

前端页面-->系统管理-->注册表

系统管理 / 注册表

新增 删除

Smart

Server

list

localhost

node1

node2

Global

DevOps

UpLoad

ProcDef

FileType

Attachment

FileSize

FileType

StoreWay

Cache

App

Version

Edm

Dfs

Local

Test

estsssss

Name

HomePage

* 名称: Smart Smart

值: 值

☒ 是否生效

描述: 框架注册表

确定

- Smart 框架注册表

- Server
 - 服务配置相关, 扩展到集群时会用到

- Global
 - 全局配置信息

- DevOps
 - 开发运维相关配置

- UpLoad
 - 附件上传参数配置

- Cache
 - 服务缓存策略配置, 主要是国企时间

- App 应用注册表

87

- Version
应用的版本信息
 - Edm
附件的存储方式
 - Name
系统名称
 - HomePage
系统默认的首页地址
-

links

- [目录](#)
- 上一节: [批处理](#)
- 下一节: [定时任务](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

用户管理

1. 简介

用户表为security工程提供的系统基础表之一，为不同业务模型提供统一的用户管理安全等功能。

2. 操作流程

- 系统管理-用户和权限-用户管理
- 用户只能创建和修改状态，没有删除功能。因用户启用后会对多个业务表产生交互数据，如删除用户将导致业务表历史数据混乱，当用户发生离职转岗等变更时，可对状态或角色进行变更，不需删除用户

2.1. 新增用户

其中账户，姓名，组织机构，状态，语言为必填项



2.2. 修改用户

修改用户界面同新增用户界面相同，功能一致。需要注意的是一旦账户已产生业务数据，不建议修改姓名账户这两个属性。如业务表冗余这两个字段，可能会导致业务表数据不一致的情况。

2.3. 用户角色设置

- 点击用户列表中的齿轮图标，可进入角色设置界面
- 点击绑定角色可在角色列表中选择需要加入的角色进行绑定
- 勾选角色列表并点击解除绑定可解除相应角色的绑定
- 一个用户可绑定多个角色

2.4. 重置用户密码

- 点击用户列表中的钥匙图标，可重置用户密码。此功能用于用户遗失密码时为用户设置初始密码 111111

其他

一些可能的扩展方向

- 用户密码加密模式可配置
- 同步LDAP用户

links

- [目录](#)
- 下一节: [群组管理](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

群组管理

1. 简介

群组管理在框架中仅适用于 workflow 用户任务候选组

2. 操作流程

- 系统管理 - 用户和权限 - 群组管理
- 可对群组进行 新增, 修改, 删除, 绑定用户等操作

2.1. 新增群组

其中编码, 名称, 状态为必填项

创建群组

* 编码:

HR_MANAGER

* 名称:

人力资源经理

描述:

描述

* 状态:

有效

保存

返回

2.2. 修改群组

修改群组同新增群组界面相同, 在群组列表点击编码进入

2.3. 删除群组

在群组列表的操作一栏中, 点击垃圾桶图标可删除群组, 删除前会提示是否需要删除

2.4. 绑定用户

在群组列表的操作一栏中, 点击人群图标可进入绑定用户界面, 针对选中的群组绑定用户

links

- [目录](#)
- 上一节: [用户管理](#)
- 下一节: [角色管理](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.

All rights reserved.

角色管理












1. 简介

将用户按角色进行分组, 使角色可以赋予菜单权限, 数据权限。

2. 操作流程

- 系统管理 - 用户和权限 - 角色管理
- 可新增角色, 修改角色, 删除角色, 绑定用户, 设置菜单权限, 设置数据权限

新增角色

	操作	名称	描述	状态
1	  	Admin	系统管理员	有效
2	  	Public	公共权限	有效
3	  	Guest	访客	有效
4	   	Test64	测试64	有效
5	   	Test65	测试65	有效
6	   	Test70	测试70	有效
7	   	Test71	测试71	有效
8	   	Test72	测试72	有效
9	   	Test77	测试77	有效
10	   	Test78	测试7888888888888888	有效

共 47 条 < 1 2 3 4 5 > 10 条/页 v

2.1. 新增角色

名称, 描述, 状态为必填项

创建角色

* 名称:

Ach

* 描述:

架构组

* 状态:

有效

保存

返回

2.2. 修改角色

内容同新增角色一致, 在角色列表点击名称一栏可进入

2.3. 删除角色

对非系统角色 (Admin,Public,Guest)可执行删除操作。具体在角色列表操作栏中, 点击垃圾桶图标, 会提示是否执行删除角色。

2.4. 权限管理

详细内容参看 [权限验证](#)

2.5. 绑定用户

角色列表操作一栏, 点击人群图标可进入绑定用户界面, 根据选中的角色绑定或解除绑定用户。该功能可参考用户管理界面中的绑定角色。

2.6. 数据权限

详细内容参看 [数据权限管理](#)

links

- [目录](#)
- 上一节: [群组管理](#)
- 下一节: [属性组管理](#)

权限验证管理

1. 简介

以角色为基础设计, 实现针对Rest的菜单及其操作权限控制。

2. 开发流程

- UI开发详情参考 [UI开发](#)
- 菜单管理详情参考 [菜单管理](#)

3. 权限管理

- 系统管理 - 用户和权限 - 角色管理 - 角色列表中的权限设置

4. 权限验证

- 用户登录时, 权限数据将会写入到session缓存
- 前端发起Rest请求时, security中的SecurityBinder将会检查该用户是否拥有该权限
- 如权限满足, 则执行后续操作;如未包含该权限, 则写入403状态到Response

5. 排查异常

- 开发阶段重新建立smartas数据库脚本后有可能导致部分权限丢失的情况, 检查权限设置无异常

此时应检查菜单权限, 开发过程中@Operation code的变更导致菜单权限数据异常。后期数据库脚本统一管理后会解决这个问题, 如碰到类似情况可参考此做法。

- 修改数据权限后未能刷新

需用户重新登陆

- 新开发的Resource在数据权限中未能找到

可在项目启动时debug PermissionResource的afterPropertiesSet检查扫描失败的原因

links

- [目录](#)
- 上一节: [属性组管理](#)
- 下一节: [数据权限管理](#)

数据权限

1 简介

以角色为基础设计, 实现初步的数据访问权限规则, 让不同角色在访问同一功能时, 可获得不同的数据集。

2 开发流程

示例可参考demo工程

2.1. 建立规则文件

在业务工程 src/main/resources/META-INF/permission/ 下建立xml文件

2.2. xml节点示例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper SYSTEM "classpath:rule.dtd">
<rules>
  <rule name = "限制演示表时间" table = "tpl_demo_t" >
    <sql>
      <![CDATA[
        #alias#.CREATE_DATE>#date#
      ]]>
    </sql>
  </rule>
  <rule name = "限制演示表年龄" table = "tpl_demo_t" >
    <sql>
      <![CDATA[
        #alias#.AGE>#age#
      ]]>
    </sql>
  </rule>
</rules>
```

2.3. dao.xml中标注permissionable="true"

```
<select id="getCount" permissionable="true" resultType="int">
<select id="select" pageable="true" permissionable="true" resultMap="Demo2ResultMap">
```

为了将数据权限限制在开发可控范围内, 针对dao.xml增加此声明, 注意同select查询的还有getCount需一同声明, 否则总数不一致会导致前端分页异常

2.4. 设置数据权限

- 系统管理 - 用户和权限 - 角色管理 - 设置数据权限(角色列表内的写字图标)

- 加入规则 - 根据规则名称选取规则
- 添加或调整条件, 如sql规则为#age#, 则参数输入框需录入age, 值可以写 1

角色-Admin

资源:

* 标题:

规则(sql):

#alias#.AGE>#age#

条件

age

1

×

添加条件

保存

3 说明

- 规则的要义是对不同角色数据权限的抽象, 即不同的角色可使用同一个数据权限的规则, 但在设置数据权限时, 可传入具体参数。
- 以上文为例, 限制演示表时间的条件是创建时间 大于 传入参数。如角色1传入 2016-08-01 角色2 传入2016-09-01 那么这两个角色最终看到的数据将会有所差异。
- rule节点定义规则包, 需包含table属性, 用于dao处理sql时对物理表进行过滤, 此处也可以是视图;name属性为规则定义的名称, 方便在数据权限管理界面操作时区分规则。
- 规则体为<sql>节点

- **alias#**为表别名, 由于不能提前确定该规则插入的sql主体, 无法提前指定表别名, 因此用此参数代替用于组合sql的阶段替换

- **date#,#age#** 等参数可任意起名

- 由于数据权限的灵活性并未限制为通用sql语法, 因此根据业务设置数据权限时需注意不同数据库环境的语法

3 加载过程

3.1. 规则的加载

- security中配置了规则的控制规则RuleHandler, 该控制器将在工程加载容器时扫描classpath:/META-INF/permission.xml, 并解析其内容, 将解析后的规则写入内存
- 如规则较多, 可考虑将规则写入缓存
- 注: 规则数据不参与数据权限限制功能, 规则的具体sql将作为冗余字段保存在数据权限表中

3.2. 数据权限的加载

- 用户登录时, 加载用户全部数据权限, 并写入session
- 通过dao执行sql脚本时, 在mybatis解析阶段判断是否包含permissionable声明, 如果包含, 则转入sql解析器
- 分析当前sql中是否包含改用户数据权限的物理表, 如包含, 则汇总其条件, 将整理后的条件加入到where尾部
- 支持多表, 但未测试过过于复杂的情况, 不支持存储过程
- 暂时实现了查询, 修改, 删除的数据权限, 未实现增加数据的数据权限

4 排查异常

- SmartAs 1.0升级到SmartAs2.0的过程中可能会导致数据库查询异常的情况, 原因是数据权限扩展了mybatis的底层部分解析功能, 可重新构建core和security这两个工程, 刷新后再看是否有这个问题
- 调整过数据权限后, 用户需重新登陆才可以获得最新的数据权限内容, 调试时需注意

5 规则的格式

- 规则xml文件中选用了security工程中的rule.dtd文件作为格式验证, 如对规则xml语法不熟悉可找到该文件查看节点描述
- 在Eclipse开发过程中如无法验证, 可忽略掉Eclipse的验证, "classpath:"会在运行时读取其正确的路径

6 多数据库支持

- 已提供的数据库支持 MySQL, Oracle
- 未提供的数据库支持 MsSQL

其他

一些可能的扩展方向

- 字段权限
- 将角色关联改为组织(org)关系的关联

links

- [目录](#)
- 上一节: [权限验证](#)

项目打包

-
- 1. smartas2.1+提供了项目打包的基础模板, 即build.xml, 项目开发组可根据情况进行调整, 该模板为主要参照
-
- 1. 打包模板中包含了jsx编译的部分, 该部分需要nodejs环境支持, 如无此环境, 可修改ant脚本跳过此内容

在线创建项目模板

smartas未来会提供模板打包下载源码的功能, 该功能可定制开发环境配置(如数据库连接方式, 缓存地址等), 方便在项目初期构建基础开发环境。

暂无服务器资源, 搭建成功后会在此提供访问地址。

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

5.5. 服务管理

- 5.5.1. [运维工具](#)
- 5.5.2. [缓存管理](#)
- 5.5.3. [版本信息](#)
- 5.5.4. [Dubbo集成](#)
- 5.5.5. [集群管理](#)

links

- [根目录](#)
- [进阶篇](#)
- 下一节: [运维工具](#)

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

运维工具

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

缓存管理

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

last modified by Bruce , 2016/12/13 09:40:31

版本信息

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

last modified by Bruce , 2016/12/13 09:40:31

Dubbo集成

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

集群管理

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

6.扩展篇

在本章中将会为您讲述如何扩展框架已提供的功能, 满足业务需求。

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.

问答

主要内容请访问 <http://10.121.4.78/chenjpu/SmartAs-issue/issues>

下面是一些常见内容的问答

1. 如何搭建开发环境？

- [环境篇](#)
- [Java环境](#)
- [启动应用](#)

2. 如何开发？

- 熟悉smartas提供的注册表, lookup, 菜单, 角色权限机制及数据库表
- 设计好数据库, 并尽可能将字典数据设计未smartas提供的lookup或注册表中
- 用代码生成器生成前后台代码
- 调整dao.xml中增删改查与实际业务表中未涉及到的字段, 如tenantId等
- 定制内容 [登陆页定制](#) [首页定制](#) [用户管理定制](#)

3. 前端有哪些文档？

- 前端组件大部分文档可参照 <http://ant.design>
- Icon快速查找 <http://fontawesome.io/icons/>
- 组件rules规则 <https://github.com/yiminghe/async-validator>
- 工具方法 <http://lodashjs.com/docs/>
- smartas封装的组件 [Lookup管理](#) [注册表](#)

4. 如何反馈SmartAs架构方面的问题？

- 可联系架构组工作人员或在 <http://10.121.4.78/chenjpu/SmartAs-issue/issues> 提交工单

撰写

模块	节	子 节	负 责 人	完 成 情 况
环境篇				
	Java环境		陈兵	100%
	Git安装		陈兵	100%
	启动应用		陈兵	100%
基础篇				
	Annotation		陈兵	80%
	Model		陈兵	100%
	SQL		陈兵	100%
	Dao		陈兵	100%
	Service		陈兵	100%
	Resource		陈兵	100%
	Exception		陈兵	100%
	Util		陈兵	100%
开发篇				
	Dao		陈兵	100%
	Service		陈兵	100%
	UI		陈兵	100%
配置篇				
	基础配置		陈兵	0%
	环境配置		陈兵	0%
	缓存配置		陈兵	0%
进阶篇				
	通用功能			
		批处理	计划	
		注册表	陈兵	100%
		定时任务	曹哲军	100%
		异步服务框架	陈涛	100%
		Lookup管理	张时俊	100%
		菜单管理	张时俊	100%
		国际化信息管理	张东升	100%

		导入管理	曹哲军	100%
		导出管理	曹哲军	100%
		日志服务	张东升	100%
		富文本	曹哲军	100%
		规则引擎		0%
		附件上传下载	陈涛	100%
		组织机构	李少能	0%
		首页	张时俊	100%
		Session管理	陈涛	100%
		消息总线		0%
		DMN		0%
	用户和权限管理			
		用户管理	张东升	100%
		群组管理	张东升	100%
		角色管理	张东升	100%
		属性组管理	张东升	100%
		权限验证	张东升	100%
		数据权限管理	张东升	100%
	工作流			
		流程业务接入规范	李少能	0%
		流程部署	李少能	0%
		流程创建	李少能	0%
		流程审批	李少能	0%
		待办列表	李少能	0%
		已办列表	李少能	0%
		草稿功能规范定义	李少能	0%
		草稿箱	李少能	0%
		功能事件	李少能	0%
		待办邮箱集成	李少能	0%
		待办对外集成API	李少能	0%
	DevOps			
		骨架代码自动生成		0%
		项目创建	曹哲军	100%
		项目打包	曹哲军	100%
		服务API在线文档		0%

		UI在线文档		0%
		业务操作在线文档		0%
	服务器管理			
		运维工具		
		缓存管理		
		版本信息		
		Dubbo集成		
		集群管理		
扩展篇				
	登录页定制			100%
	首页定制			100%
	用户管理定制			100%
	组织机构定制			100%
Web篇				
	Service.js-API			90%
	Resource.js-API			90%
	namespace.js-api			50%
	smartui.js-api			30%

Copyright © 2016 www.skyinno.com 内部资料禁止外传.
All rights reserved.