

Venus Framework Reference Documentation

1. 3. 8

Copyright # 2014-2024

VIP.com

Table of Contents

I.	Venus Framework简介	1
1.	Venus是什么	3
2.	Venus目标	4
3.	Venus在产品生命周期中的位置	5
4.	Venus生态系统	6
II.	Venus快速入门	7
5.	Venus项目的运行环境	9
5.1.	环境变量的意义	9
5.2.	如何设置运行环境变量	9
5.3.	如何设置默认环境变量	10
6.	java开发规范（Formatter的使用）	11
7.	WEBAPP项目示例演示	12
7.1.	Venus开发环境	12
7.2.	Venus codegen工具的使用	12
7.3.	webapp项目的简单说明	18
7.4.	webapp项目的编译发布	19
7.5.	webapp项目的调试	21
8.	OSP项目示例演示	33
8.1.	Venus开发环境	33
8.2.	Venus codegen工具的使用	33
8.3.	OSP项目的文件说明	38
8.4.	OSP项目的本地运行	39
8.5.	OSP项目的调试	40
8.6.	进一步阅读	43
III.	Venus framework下的开发实践	44
9.	配置文件的变量定义	47
9.1.	properties下的变量定义	47
9.2.	applicationContext.xml下的变量定义	48
10.	使用venus-context读取配置文件	50
10.1.	属性配置	50
11.	使用venus-data进行分库/读写分离/分表	54
11.1.	添加依赖	54
11.2.	spring引入venus-datasource	54
11.3.	spring添加数据库操作相关bean	55
11.4.	properties文件配置	55
11.5.	分库操作	56
11.6.	读写分离操作	59
11.7.	分表操作	60
12.	使用venus-jdbc进行分库/读写分离/分表	63
12.1.	添加依赖	63
12.2.	spring引入venus-jdbc	63
12.3.	spring添加数据库操作相关bean	63
12.4.	spring配置	64
12.5.	properties文件配置	64
13.	使用venus-cache进行缓存操作	66
13.1.	spring引入venus-cache	66
13.2.	venus-cache配置和使用	66

13. 3. cache cluster配置	67
14. 使用配置中心配置channel和queue	69
14. 1. 配置要使用的配置中心地址	69
14. 2. 然后在配置中心创建channel，选择相应的集群	69
14. 3. 创建queue，选择相应的集群	70
14. 4. 如果需要设置routingkey，在下面绑定routingkey	70
15. 使用venus-test进行单元测试	71
15. 1. 添加依赖	71
15. 2. 配置文件说明	71
15. 3. 测试类说明	72
15. 4. 注解说明	72
16. 使用venus-security进行认证和授权	74
16. 1. 添加依赖	74
16. 2. 使用venus-security接入OA流程	74
17. 使用配置中心读取配置信息和监控配置变更	78
17. 1. 配置中心的zk集群	78
17. 2. venus-context配置	78
17. 3. 从配置中心读取cache信息	79
17. 4. 从配置中心监控配置变更	80
18. 使用Mercury	88
18. 1. Mercury trace接入步骤（开发篇）	88
18. 2. Mercury trace接入步骤（部署篇）	93
19. 使用Hermes	96
19. 1. 添加依赖	96
19. 2. XML配置	96
19. 3. 添加log4j/logback配置	96
19. 4. 定义Item	98
19. 5. 注册Item	99
20. 使用中央文档系统	100
20. 1. 系统简介	100
20. 2. 主要功能	100
20. 3. 使用步骤	101
21. 使用git管理源码	102
21. 1. git配置	102
21. 2. git分支模型	102
21. 3. 使用git上传项目流程	104
22. 使用jenkins持续集成	106
IV. Venus Framework模块	111
23. Venus概览	113
24. Venus Core	115
24. 1. Lombok	115
24. 2. Bean Mapping	115
24. 3. Exception	116
24. 4. Venus Context	117
25. Venus Data Access	119
25. 1. 关系数据库	119
25. 2. HBase	128
25. 3. MongoDB	128
26. Venus Cache	129
26. 1. venus-cache典型配置	129

27.	Venus MQ	133
27.1.	venus-mq典型配置	133
27.2.	venus-mq配置详细说明	133
28.	Venus Web	135
28.1.	venus web中的注解与类	135
28.2.	controller代码示例	135
29.	Venus Security	137
29.1.	venus-security典型配置	137
29.2.	venus-security配置详细说明	138
30.	Venus Test	141
30.1.	venus test中的注解	141
V.	Venus framework下gradle的使用	142
31.	为什么选择gradle作为项目构建工具	144
32.	gradle的安装与运行	145
33.	eclipse安装gradle插件	146
34.	gradle项目添加依赖	147
35.	常用的gradle命令	149
35.1.	gradle基本命令	149
35.2.	venus下的gradle命令	149
35.3.	使用gradle部署文件到maven仓库	149
36.	新建gradle project	151
36.1.	gradle与Jekins	152
VI.	配置中心	154
37.	配置中心设计	156
38.	配置中心介绍	157
39.	配置定义文件	158
VII.	OSP	160
40.	OSP入门	163
40.1.	OSP概述	163
40.2.	OSP详解	163
40.3.	OSP的未来	169
41.	OSP服务提供方用户手册	170
41.1.	快速开始	170
41.2.	服务定义	170
41.3.	服务实现	181
41.4.	服务测试与发布	185
41.5.	服务部署	187
42.	OSP服务接入方使用手册	192
42.1.	Java客户端接入	192
42.2.	PHP客户端接入	196
42.3.	OSP客户端配置	200
43.	OSP公共知识	202
43.1.	ZooKeeper及配置中心配置	202
43.2.	OSP-Proxy	203
43.3.	OSP服务的配置	206
43.4.	OSP的异常处理	207
43.5.	OSP路由功能	208
43.6.	静态路由表	210
43.7.	自定义OspFilter	211
43.8.	常见问题	212

VIII. Tools	214
44. 代码生成器 (Venus Codegen)	216
44.1. Codegen功能介绍	216
44.2. 命令行版本使用说明	216
44.3. UI版本版本使用说明	220
44.4. codegen使用注意事项	223
IX. 注意事项	225
45. properties中编辑json格式	227
46. 项目依赖	228
47. 配置编译后的代码兼容JDK1.6	229

Part#1. #Venus Framework简介

1. Venus是什么	3
2. Venus目标	4
3. Venus在产品生命周期中的位置	5
4. Venus生态系统	6

1. #Venus是什么

Venus是唯品会开发的一款基于spring的java开发框架，以降低开发的复杂度，提高开发人员的开发效率，提升代码质量，规范开发流程。

Venus框架涵盖了WEB, Rest Doc, SOA(osp), Data Access, Cache, Message Queue, Security, Test, Log, Code Generator等各方面的内容。

开发人员可以利用venus-codegen工具，根据设计好的数据库和表，自动生成venus风格的项目代码。

目前venus-codegen支持生成simple service, simple service+webapp, osp, osp+webapp四种类型的项目。

simple service项目可以打成jar包供别的项目使用，osp项目可以独立运行发布服务供别人使用。



2. #Venus 目标

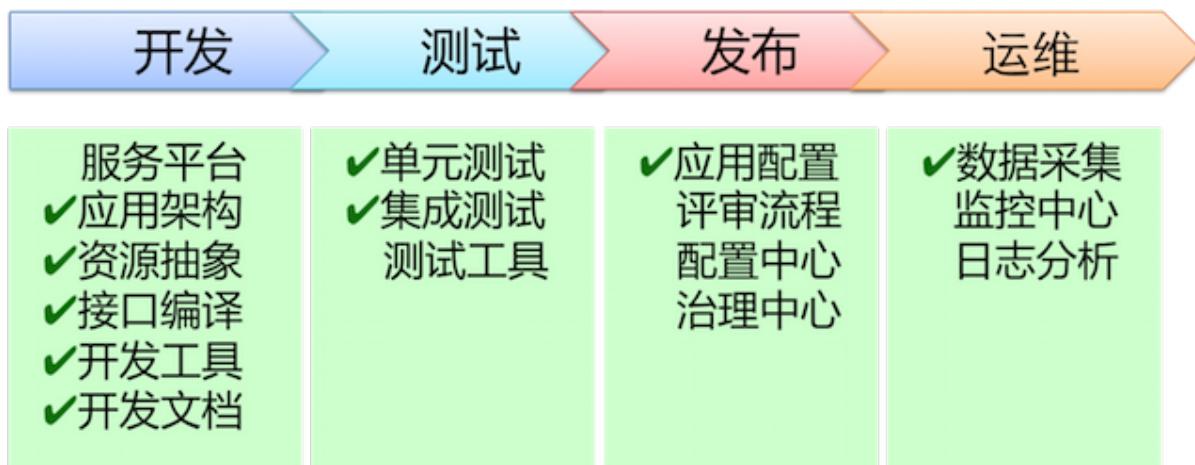
开发人员不用考虑数据库的连接和cache等这些模块代码，只需要专注于业务设计，只要设计好需要的数据库和数据表，就可以利用venus-codegen工具自动生成venus风格的项目代码，

Note

根据文档中的说明修改配置文件和代码，就可轻松的实现数据库中的表分离操作，分库操作，读写分离和缓存操作等

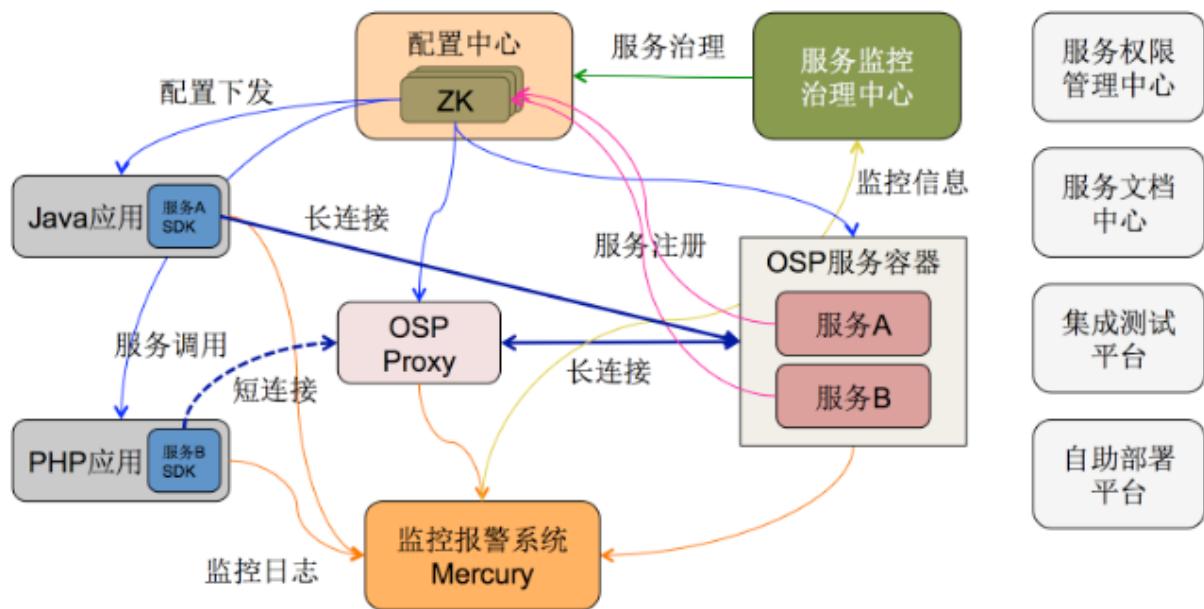


3. #Venus在产品生命周期中的位置



4. #Venus生态系统

Venus生态系统



Note

本文中所提到的环境变量是指全局的，比如`export`这种方式。 系统变量是针对某个应用的，通过`java`的`-D`参数设置

Part#11. #Venus快速入门

我们分别以一个webapp项目和一个OSP项目为例，在windows操作系统下示范如何利用venus-codegen工具快速生成一个venus风格的项目代码。

5. Venus项目的运行环境	9
5.1. 环境变量的意义	9
5.2. 如何设置运行环境变量	9
5.3. 如何设置默认环境变量	10
6. java开发规范（Formatter的使用）	11
7. WEBAPP项目示例演示	12
7.1. Venus开发环境	12
7.2. Venus codegen工具的使用	12
7.3. webapp项目的简单说明	18
7.4. webapp项目的编译发布	19
7.5. webapp项目的调试	21
8. OSP项目示例演示	33
8.1. Venus开发环境	33
8.2. Venus codegen工具的使用	33
8.3. OSP项目的文件说明	38
8.4. OSP项目的本地运行	39
8.5. OSP项目的调试	40
8.6. 进一步阅读	43

5. #Venus项目的运行环境

Venus分离三种运行环境：development, integratetest, production，分离三种运行环境可以使开发者的开发更为灵活。

5. 1#环境变量的意义

- 项目打出的发布包本身可以适用于任何环境，开发者可以通过设置spring.profiles.active来控制当前的运行环境。
- 通过环境变量来加载不同的properties配置文件

如果当前的环境变量是development模式，那么就会从webapp项目下的src/main/resources/properties/development下读取application-development.properties文件。

如果当前的环境变量是integratetest模式，那么就会从webapp项目下的src/main/resources/properties/integratetest下读取application-integratetest.properties文件。

如果当前的环境变量是production模式，那么就会从webapp项目下的src/main/resources/properties/production下读取application-production.properties文件。

- 开发者还可以通过设置环境变量来使用不同的spring bean，

如果在applicationContext.xml中这样定义<beans profile="development">…</beans>，

那么这个文件中的bean只能用于development这个运行环境

5. 2#如何设置运行环境变量

webapp项目：

运行环境变量的设置有三种方式：

- 直接在gradle.properties中配置，这种配置仅对gradlew tomcatRun这种启动方式有效，配置如下：

```
servlecontainer_jvmArgs=-Dspring.profiles.active=development
```

- 修改tomcat中bin/catalina.bat或者bin/catalina.sh文件：

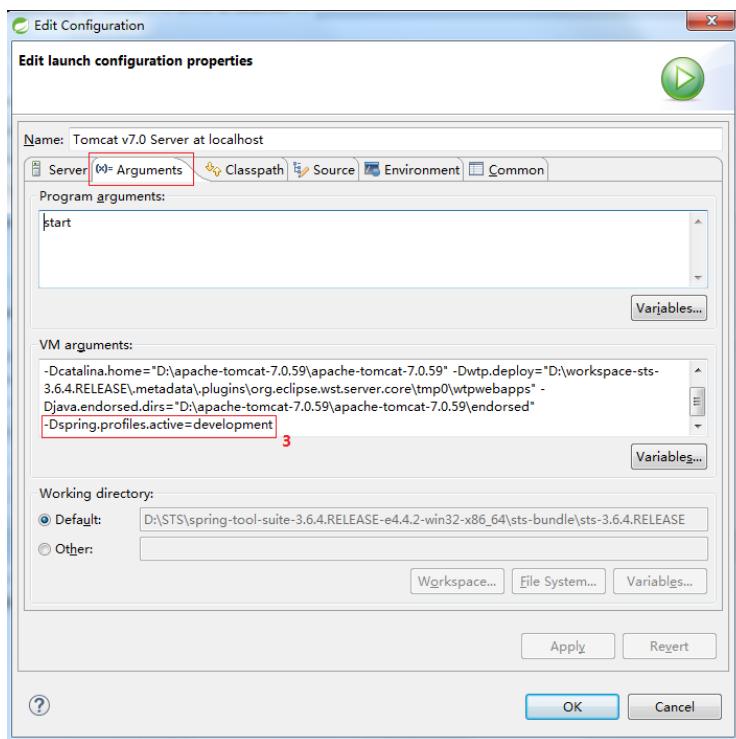
catalina.bat: 增加 set JAVA_OPTS="-Dspring.profiles.active=development"

catalina.sh: 增加 export JAVA_OPTS="-Dspring.profiles.active=development"

- 修改eclipse中tomcat的jvm参数

双击下图中的框1tomcat server→双击框2open launch configuration→Argument tab页，

添加框3参数：-Dspring.profiles.active=development



Note

如果没有设置运行环境变量，那么就从web.xml中读取默认环境变量当做当前的运行环境。

在使用gradlew tomcatRun来启动tomcat时，默认就是development模式；

OSP服务项目：

在使用gradlew run0sp来运行OSP服务的时候，
默认就是development模式，但也可以在
ospServiceProjects.gradle中设置运行环境：

```
ospService {
    ospVersion = project.ospVersion
    profile='production'
}
```

在集成环境，通过对OSP启动脚本增加 -Dspring.profiles.active=integratetest 来做配置

5. 3#如何设置默认环境变量

webapp项目：

webapp没有默认production环境变量，生产环境需要手动设置
spring.profiles.active=production

OSP服务项目：

已经内置了默认 spring.profiles.default=production，生产环境下无需再做配置

我们设置运行环境变量有3种方式：jvm参数，catalina脚本和web.xml，

优先级是jvm参数 > catalina脚本 > web.xml，运行环境变量以高优先级中的变量为准

6. #java开发规范（Formatter的使用）

Eclipse:

step1. 下载Eclipse的formatter文件: [vipshop-code-conventions](#)

step2. 导入eclipse: Window → Preferences → Java → Code Style → Formatter → Import → 选择文件

Note

编辑代码时我们可以设置自动格式化: Window → Preferences → Java → Editor → Save Actions,

先勾选“Perform the selected actions on save”，再勾选“Format source code”。

也可以手动格式化: 右键Java文件选择Source→Format或者通过快捷键 Ctrl + Shift + F来对文件进行格式化

IntelliJ:

Note

编码开发时可参考: [唯品会java编码规范](#)

7. #WEBAPP项目示例演示

我们必须先设计好要发布的业务，这里以student为例，student有id和name。

我们要先在mysql中设计好数据库qiyu和数据表student, 才可以利用codegen工具生成venus项目代码，代码涵盖service-api, service和webapp。

项目打包发布到tomcat上，我们就可以通过web往student表中增删改查数据了。

7. 1#Venus开发环境

venus项目要求spring 4.0.5+, JDK 1.7+, Tomcat 7+, 建议使用jdk1.7和tomcat7

7. 2#Venus codegen工具的使用

step1. [下载codegen—1.3.8](#), 解压缩至本地磁盘指定位置。

进入codegen/bin文件夹，打开config.properties配置文件，编辑此文件，然后运行codegen.bat文件。

```
#####
#   project configuration
#####
project.exportPath=E:/venuspractice ①
project.name=venuspractice ②
project.packageName=com.vip.venus ③
project.moduleName=userservice ④
project.type=simpleservice,webapp ⑤
project.buildTool=gradle ⑥

#####
#   database configuration
#####

db.driver=com.mysql.jdbc.Driver ⑦
db.url=jdbc:mysql://10.101.18.72:3306/qiyu ⑧
db.userId=vipuser ⑨
db.pwd=xR54PUU8GWia ⑩
db.name=qiyu ⑪
db.tables=student:Student ⑫
db.shardingTables= ⑬
```

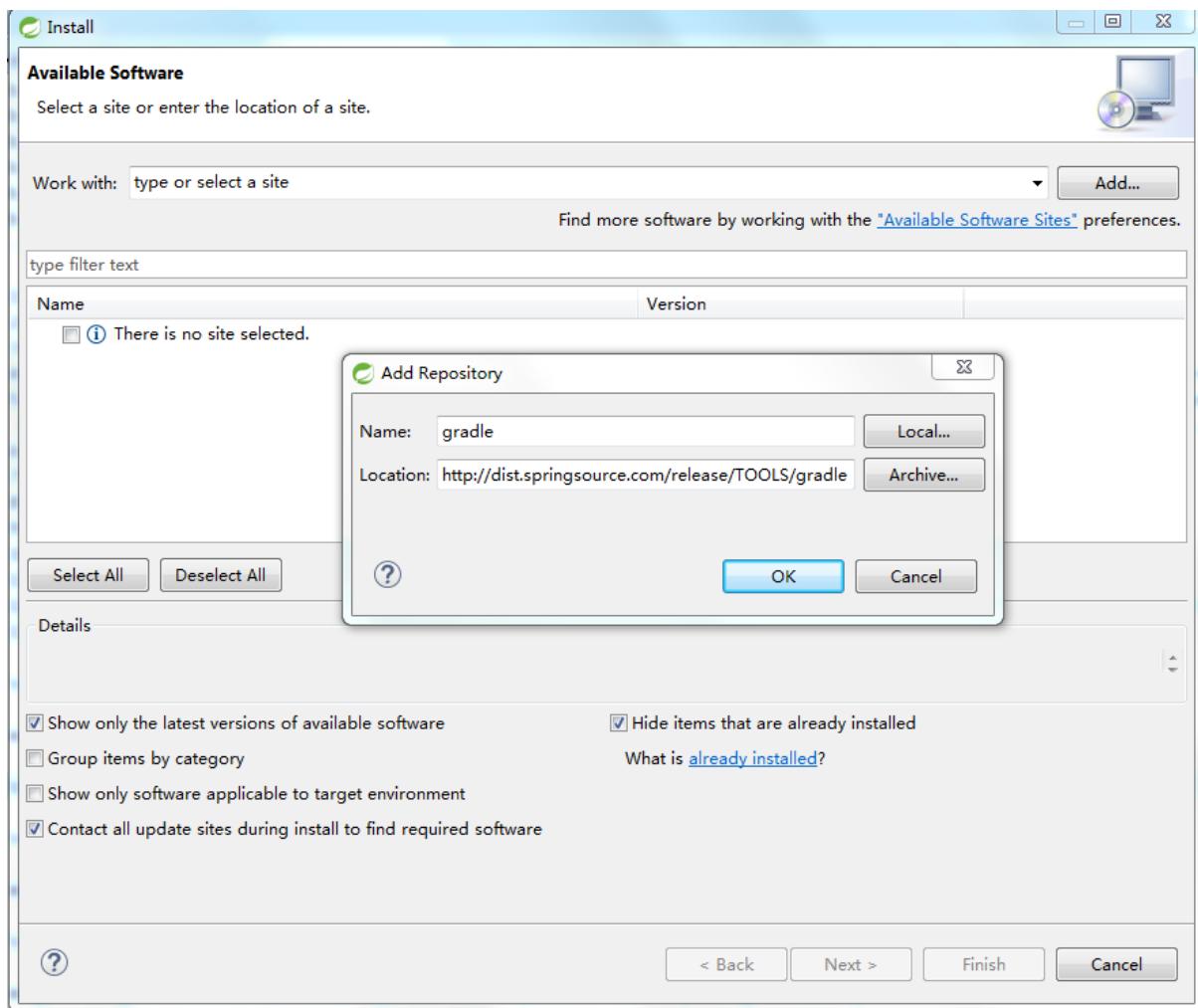
- ① project.exportPath 表示生成项目的路径，windows操作系统，可以指定路径为D:/XX文件夹，生成的项目就位于该路径文件夹下面；
- ② project.name 表示生成的项目名称；
- ③ project.packageName 表示生成项目的包名前缀部分（按照Java的文件包命名规范填写），例如：com.vip.venus
- ④ project.moduleName 表示生成项目的模块名称（按照Java的模块命名规范填写，仅为字母，不可含有特殊字符），生成的包名为：<包名前缀部分>+<模块名称>
- ⑤ project.type 表示生成项目的类型，可以不指定（如果为空则默认生成OSPServices项目），指定为simpleservice，生成独立的SimpleService项目；指定为ospService，生成独立的OSPServices项目，不可同时指定为simpleservice和ospService；指定simpleservice和webapp组合（用逗号分隔），生成Simple Service和Webapp项目；指定ospService和webapp组合（用逗号分隔），生成OSPServices和Webapp项目，不可单独指定生成Webapp项目；
- ⑥ project.buildTool 表示项目使用的构建工具，目前只支持gradle；
- ⑦ db.driver 数据库驱动类名称，根据不同的数据库配置对应的驱动类名称；

- ⑧ db.url 数据库的JDBC链接路径;
- ⑨ db.userId 连接数据库的用户账号, 即用户名;
- ⑩ db.pwd 连接数据库的密码;
- ⑪ db.name 连接数据库的schema, 即指定连接数据库的名称;
- ⑫ db.tables 指定表名称和生成的对象名称, 表名和对象名之间用冒号分隔, 多个表名对象名组之间用逗号分隔 (table1:Object1, table2:Object2), 如上所示;
- ⑬ db.shardingTables 待升级!

step2. 运行codegen.bat文件后, 可以在E:\venuspractice\venuspractice下看到:

名称	修改日期	类型	大小
gradle	2015/3/30 10:58	文件夹	
venuspractice-service	2015/3/30 10:58	文件夹	
venuspractice-service-api	2015/3/30 10:58	文件夹	
venuspractice-webapp	2015/3/30 10:58	文件夹	
build.gradle	2015/3/30 10:58	GRADLE 文件	5 KB
gradle.properties	2015/3/30 10:58	PROPERTIES 文件	2 KB
gradlew	2015/3/30 10:58	文件	5 KB
gradlew.bat	2015/3/30 10:58	Windows 批处理...	3 KB
libraries.gradle	2015/3/30 10:58	GRADLE 文件	6 KB
README.md	2015/3/30 10:58	MD 文件	0 KB
settings.gradle	2015/3/30 10:58	GRADLE 文件	1 KB

step3. eclipse安装gradle插件: 可以通过Eclipse Marketplace安装: help→Eclipse Marketplace→find框中输入 gradle→go



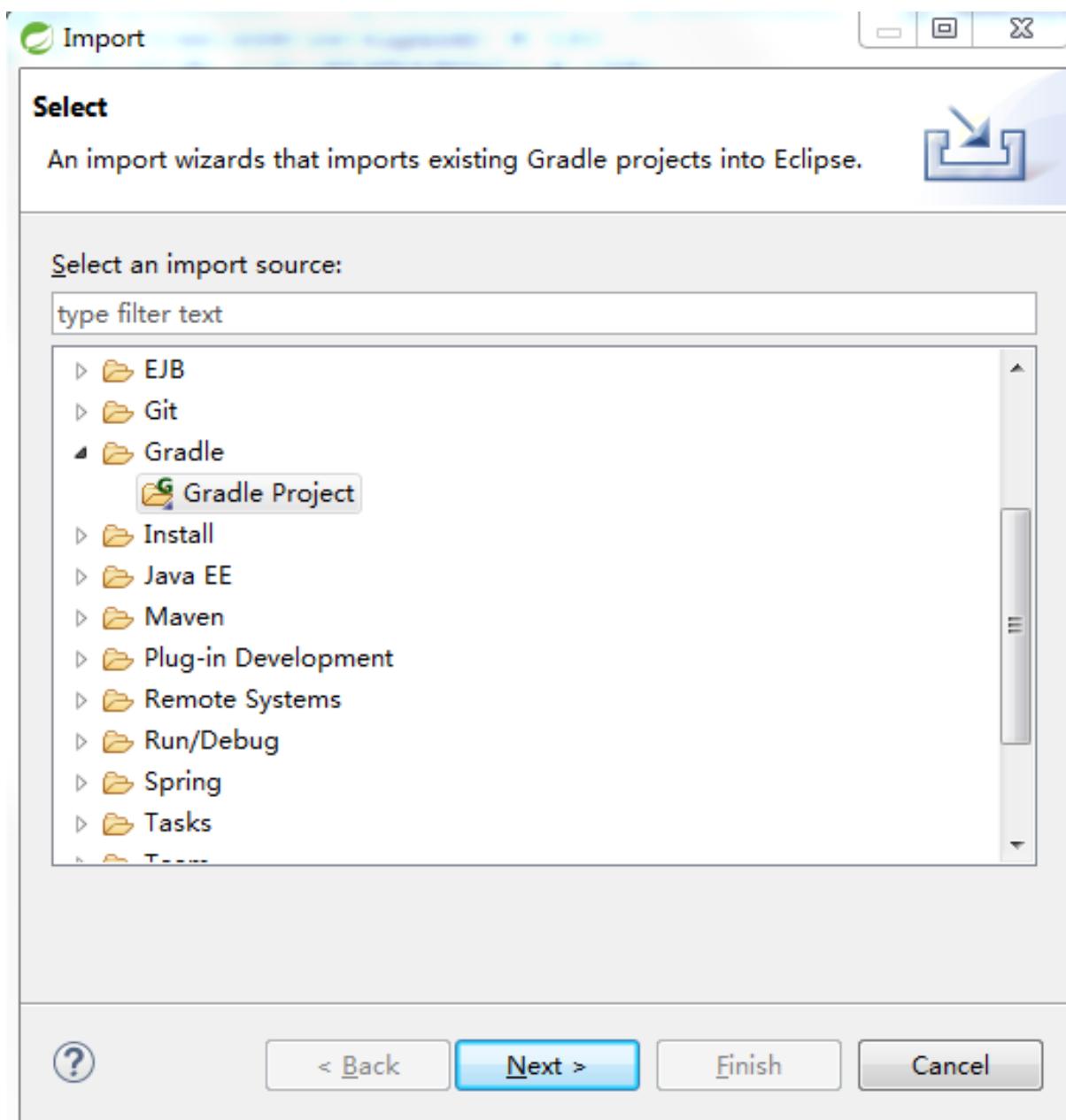
Note

如果没有eclipse Marketplace可以直接在线安装: help→install New Software→add,

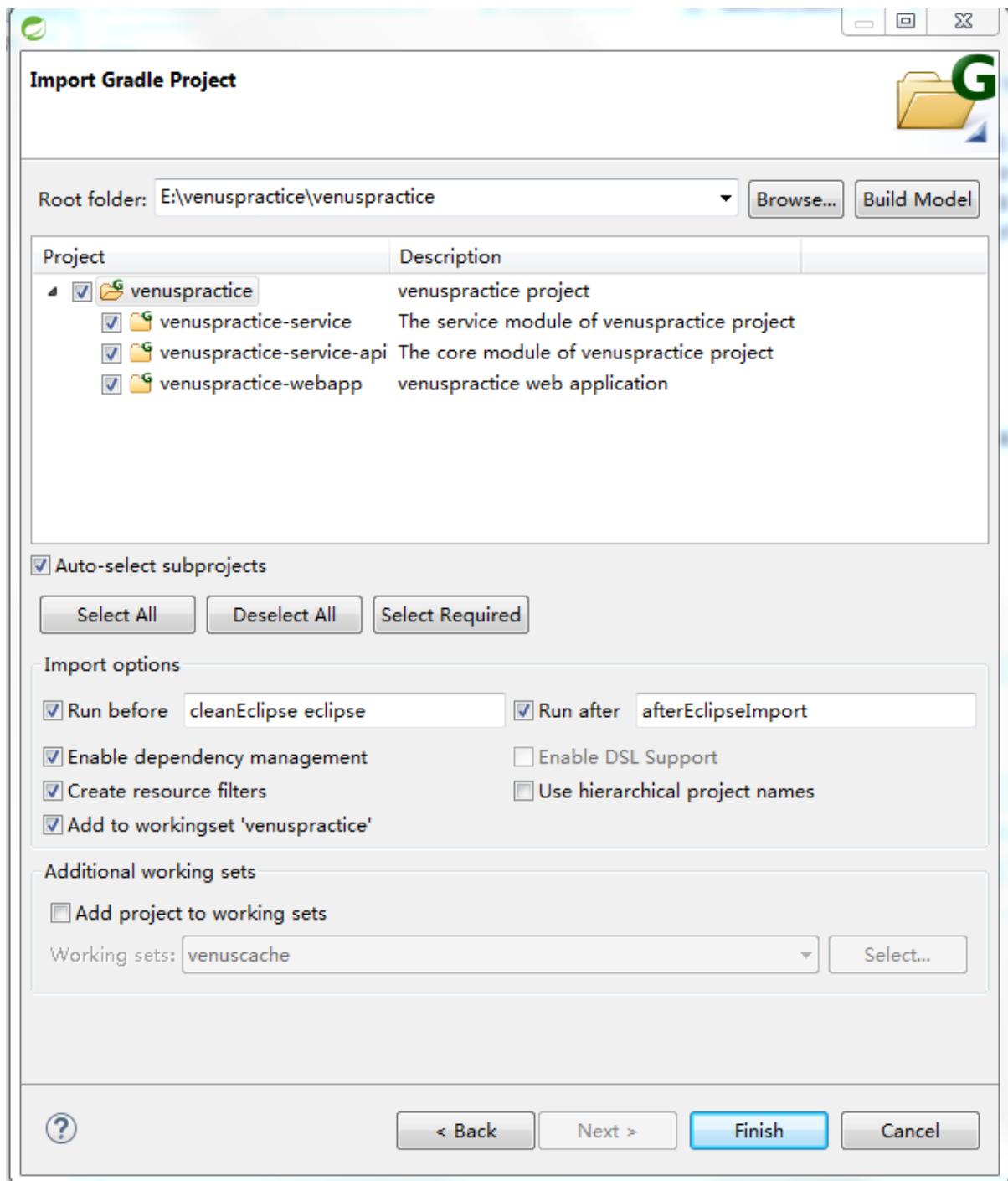
在Add Repository中输入Name和Location, 点击ok就可以了

gradle插件Location: <http://dist.springsource.com/release/TOOLS/gradle/>

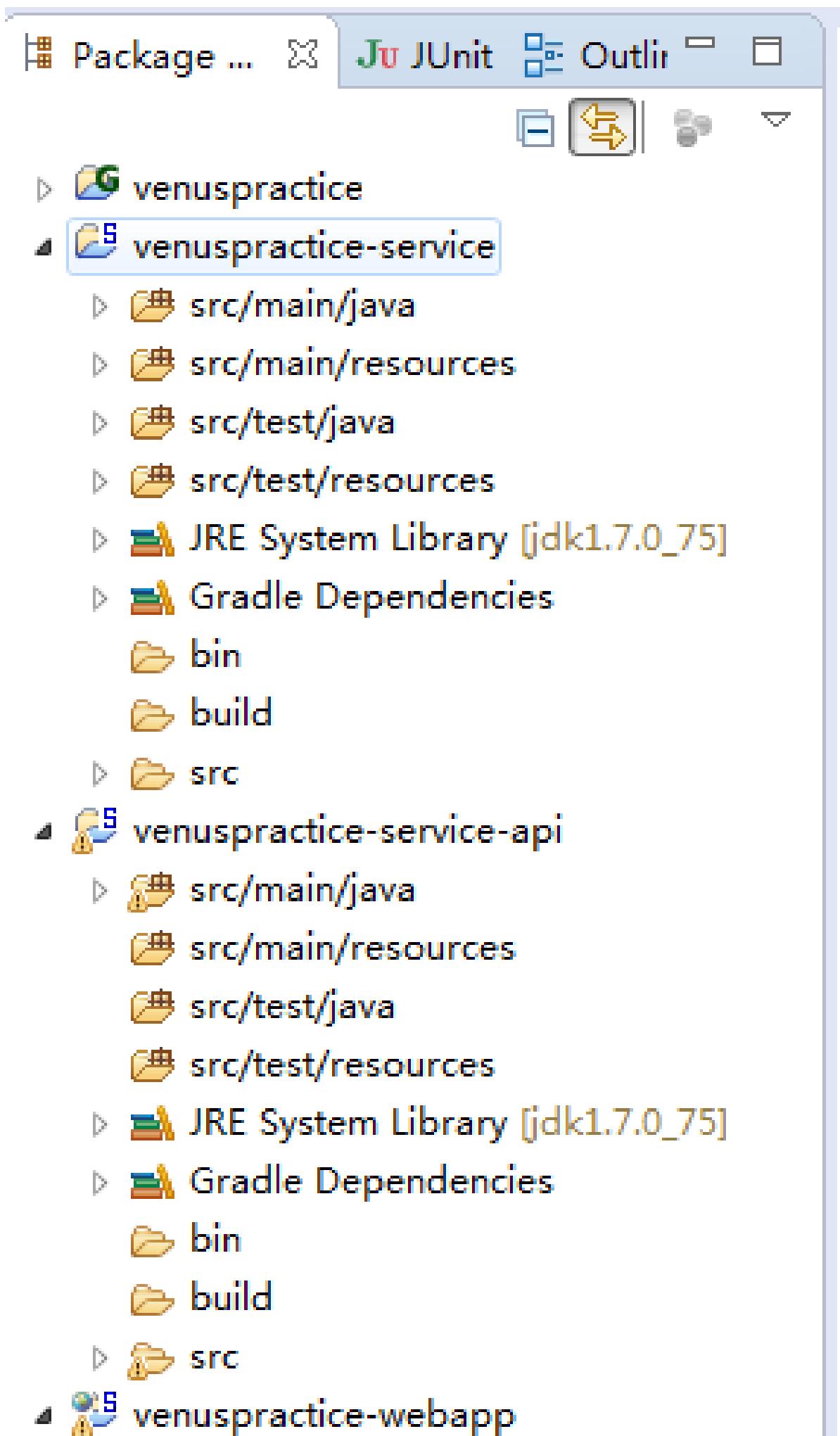
step4. 代码导入eclipse, 右键import Gradle Project:



step5. 选择代码所在目录E:\venuspractice\venuspractice→Build Module→项目全选→finish



step6. 在eclipse中能看到如下项目：



Note

当一个venus项目生成后，如果增加/修改数据库和数据表，还可以利用venus-codegen只生成关于修改的数据库/表的代码，具体使用参考Chapter#44，代码生成器(Venus Codegen)。

现实开发中如果项目A依赖项目B，然后B可能很频繁的出SNAPSHOT版本的jar给A依赖使用，这种情况请参考Chapter#46，项目依赖

7. 3#webapp项目的简单说明

venuspractice项目：gradle的相关配置文件

文件	描述
build.gradle	gradle标准build配置文件
gradle.properties	gradle相关的配置项，里面的配置项都可通过gradle命令行-P同名参数覆盖
gradlew	linux下的gradle wrapper执行文件
gradlew.bat	windows下的gradle wrapper执行文件
libraries.gradle	所有的依赖包的定义，用于在dependencies里引入依赖变量
setting.gradle	gradle项目的定义文件

venuspractice-service-api项目：服务的API项目，提供接口和模型

文件	描述
StudentModel.java	service接口提供的数据结构
StudentService.java	service的接口

venuspractice-service项目：服务实现的项目，对service-api接口的实现

文件	描述
Student.java	DAL层数据的映射类
StudentRepository.java	DAL层的接口类
StudentServiceImpl.java	service接口的实现类
StudentMapper.xml	myBatis的mapping文件

venuspractice-webapp项目： web层的实现，对外提供数据库表的增删改查操作

文件	描述
StudentRestApiController.java	用于提供REST接口

StudentVO.java	REST接口对外提供的数据结构
applicationCfgDef.xml	配置定义文件，具体请查看Chapter 18, 配置定义文件
applicationContext.xml	spring bean的配置文件
mvc-servlet.xml	spring bean的配置文件，主要是web层的bean配置

7. 4#webapp项目的编译发布

webapp的发布有三种方式：第一种是运行命令gradlew tomcatRun；第二种是build成war包部署到tomcat的webapps下；第三种是直接在eclipse下的tomcat中启动。

第一种方式：运行命令gradlew tomcatRun，这种发布方式默认是development模式（因为gradlew tomcatRun的方式只允许在开发模式下使用）

cmd到根目录下，直接运行命令gradlew tomcatRun。这种方式支持热部署，即修改代码即刻生效，不用重启服务器，但这种方式只能用于development下。

第二种方式：build成war包部署到tomcat的webapps下

step1. cmd到项目根目录下，执行gradlew build命令，打成war包

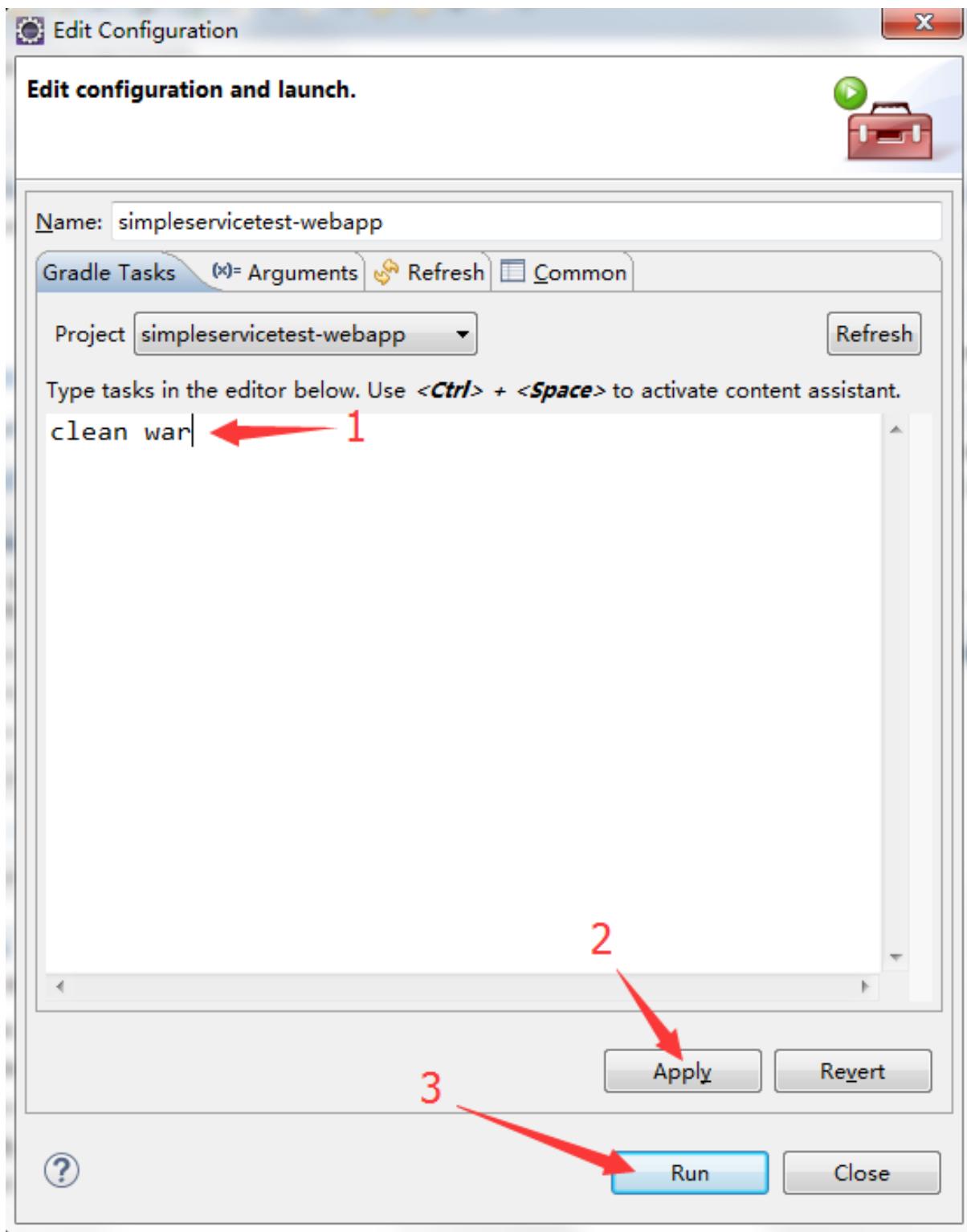
step2. 把war包copy到\$TOMCAT_HOME\$\webapps目录下

step3. cmd到\$TOMCAT_HOME\$\bin目录下，运行startup.bat/startup.sh，启动tomcat

第三种方式：直接在eclipse下的tomcat中启动

step1. 打包：eclipse中选中venuspractice工程，右键→RunAs→Gradle build，

```
#####clean war###Apply ###Run###
```



step2. 发布：在venuspractice-webapp工程下build/libs下有war包生成，把war包copy到本地安装tomcat

的webapps目录下，修改war包名venuspractice-webapp-0.0.1-SNAPSHOT.war

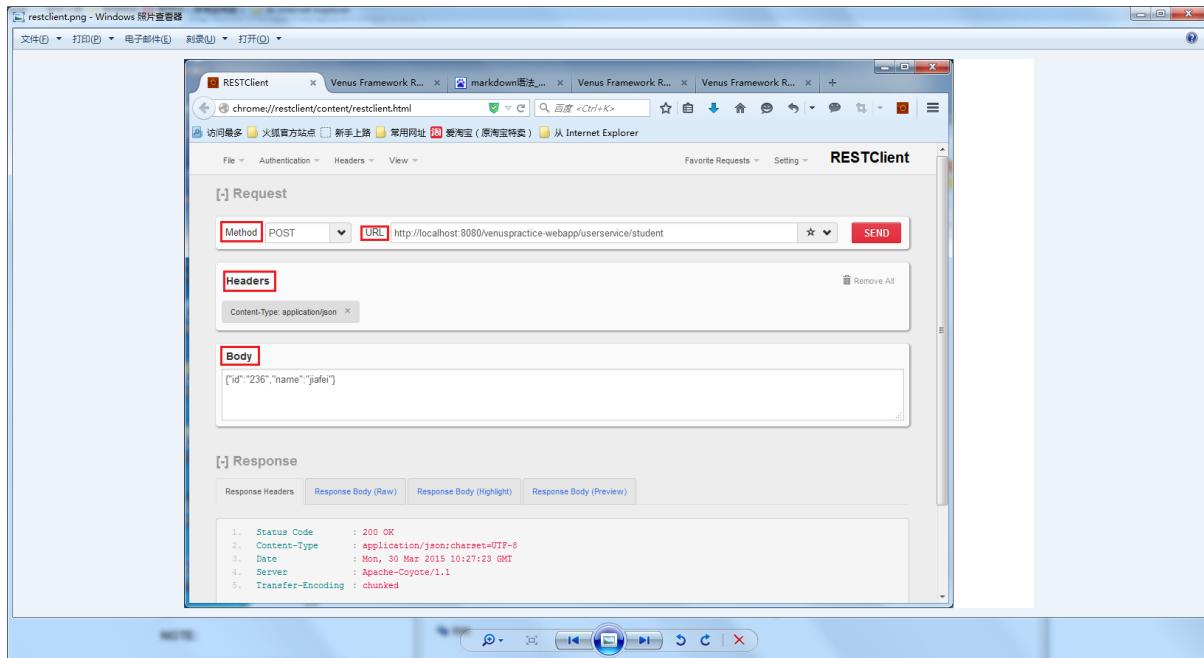
为venuspractice-webapp.war，启动tomcat就可以了。

step3. 测试：firefox浏览器中我们可以用RESTClient进行测试（需要先在安装RESTClient插件），如下图中返回200ok，

说明我们操作成功，就可以再mysql数据库student表中查询到此数据了（可以下载mysql客户端navicat方便查询）

Note

chrome浏览器下可以用postman插件测试



Note

method: 如果往表中增/删/改/查数据，分别选择post/delete/put/get方法. post/put方法还需要设置Headers

Header: 如果选择post/put方法，需要设置Headers设置Content-Type, 方法如下：
Headers→Customer Header, 弹出Request Header窗口中Name设置为Content-Type, Value设置为application/json, 点击ok

URL: 是根据ip, 端口, webapp名称和StudentRestApiController中的注解拼接成的，格式为：

<http://IP:8080/webapp名称/类前@RequestMapping中value/方法前@RequestMapping中value>

Body: 待插入表的数据，格式为 {"属性1": "value1", "属性2": "value2" ...}

7. 5#webapp项目的调试

webapp项目的调试分为两种：第一种是执行gradlew tomcatRunDebug，然后在Eclipse中新建远程debug；第二种是在eclipse通过在tomcat中添加webapp应用调试；

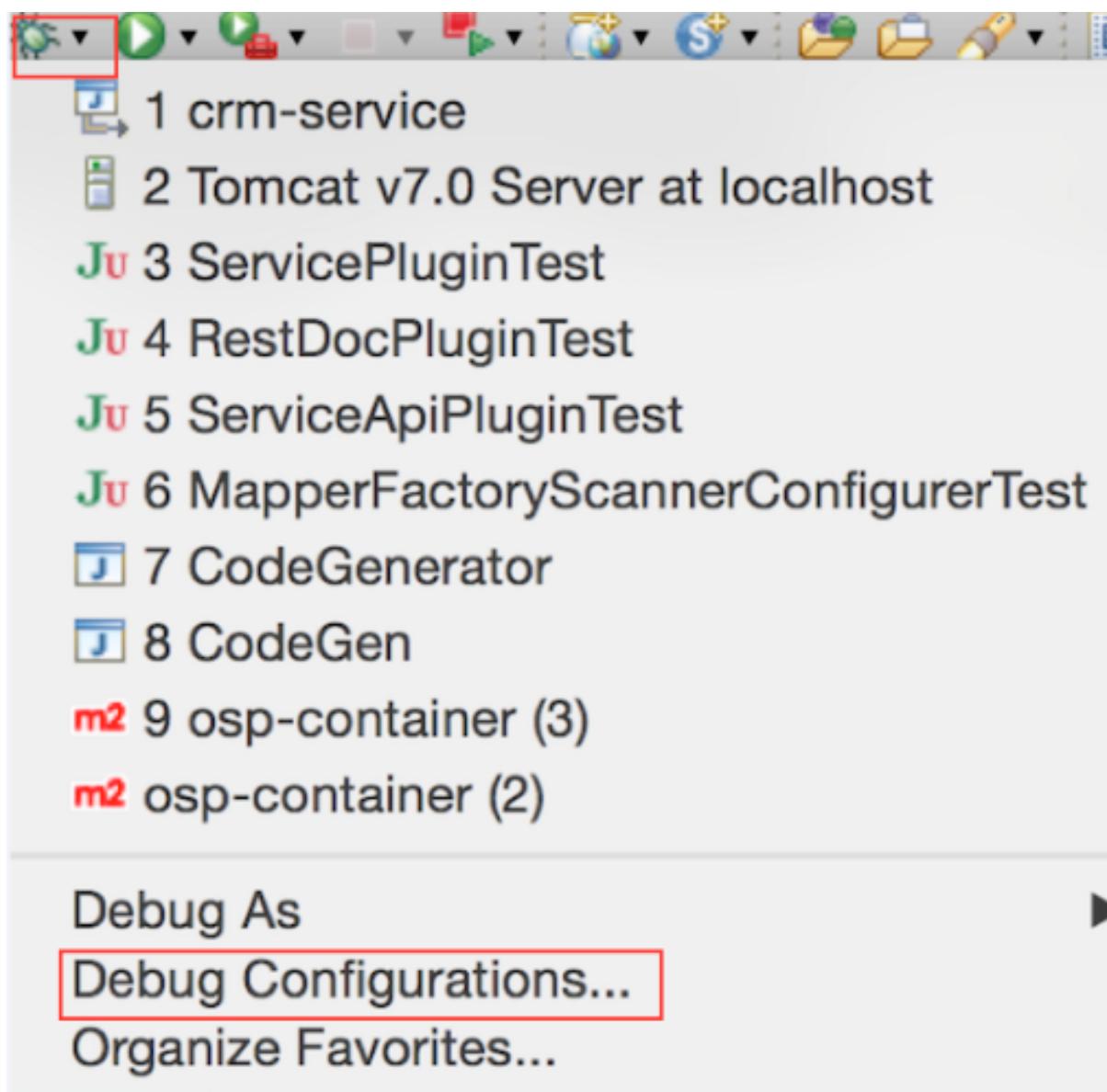
第一种方式支持热部署，修改代码即刻生效，不用重启服务器，因此推荐第一种调试方式

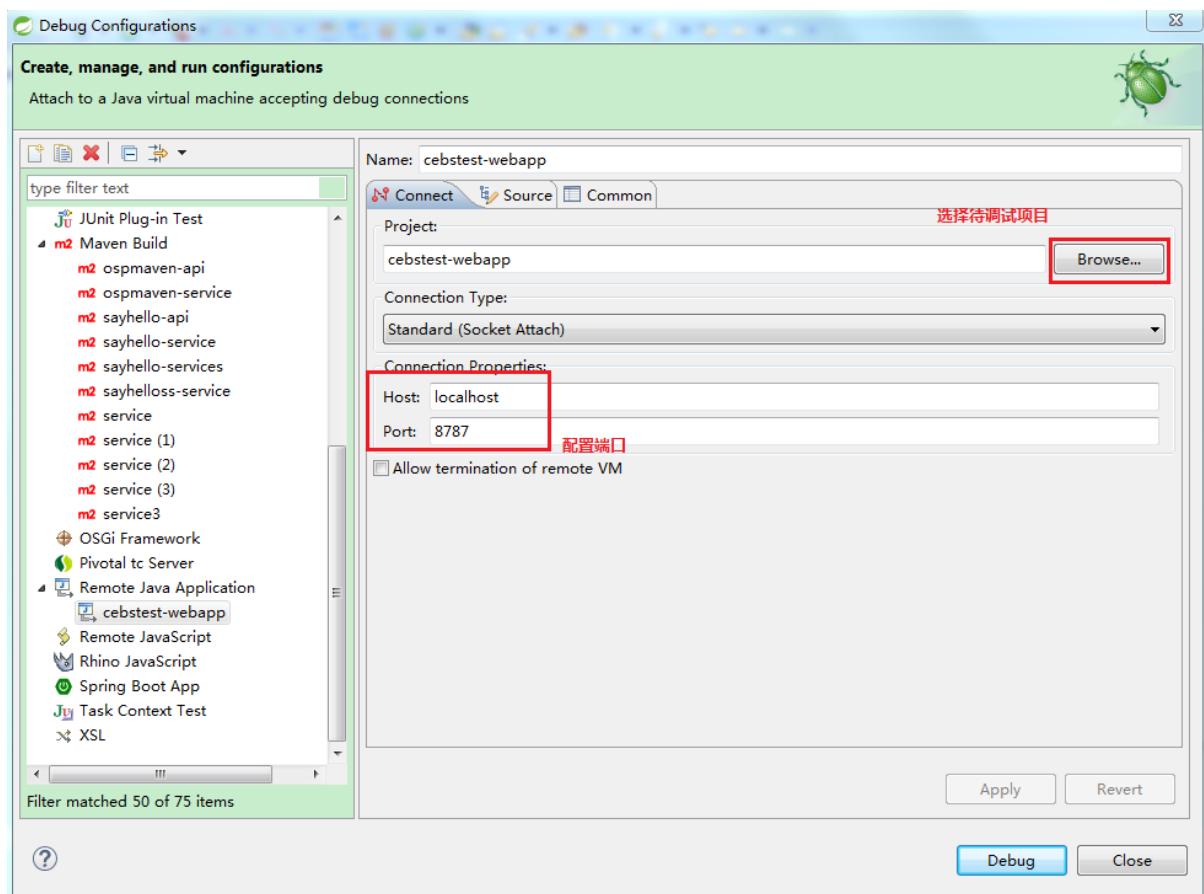
第一种方式：执行gradlew tomcatRunDebug，然后在Eclipse中新建远程debug

step1. cmd到项目根目录，执行命令gradlew tomcatRunDebug，默认端口是8787，可以在properties文件中通过servletcontainer_debug_port指定端口。

step2. Eclipse中新建远程debug

如下图配置：配置Host和port，引进java工程，port必须和step1中配置的port一致



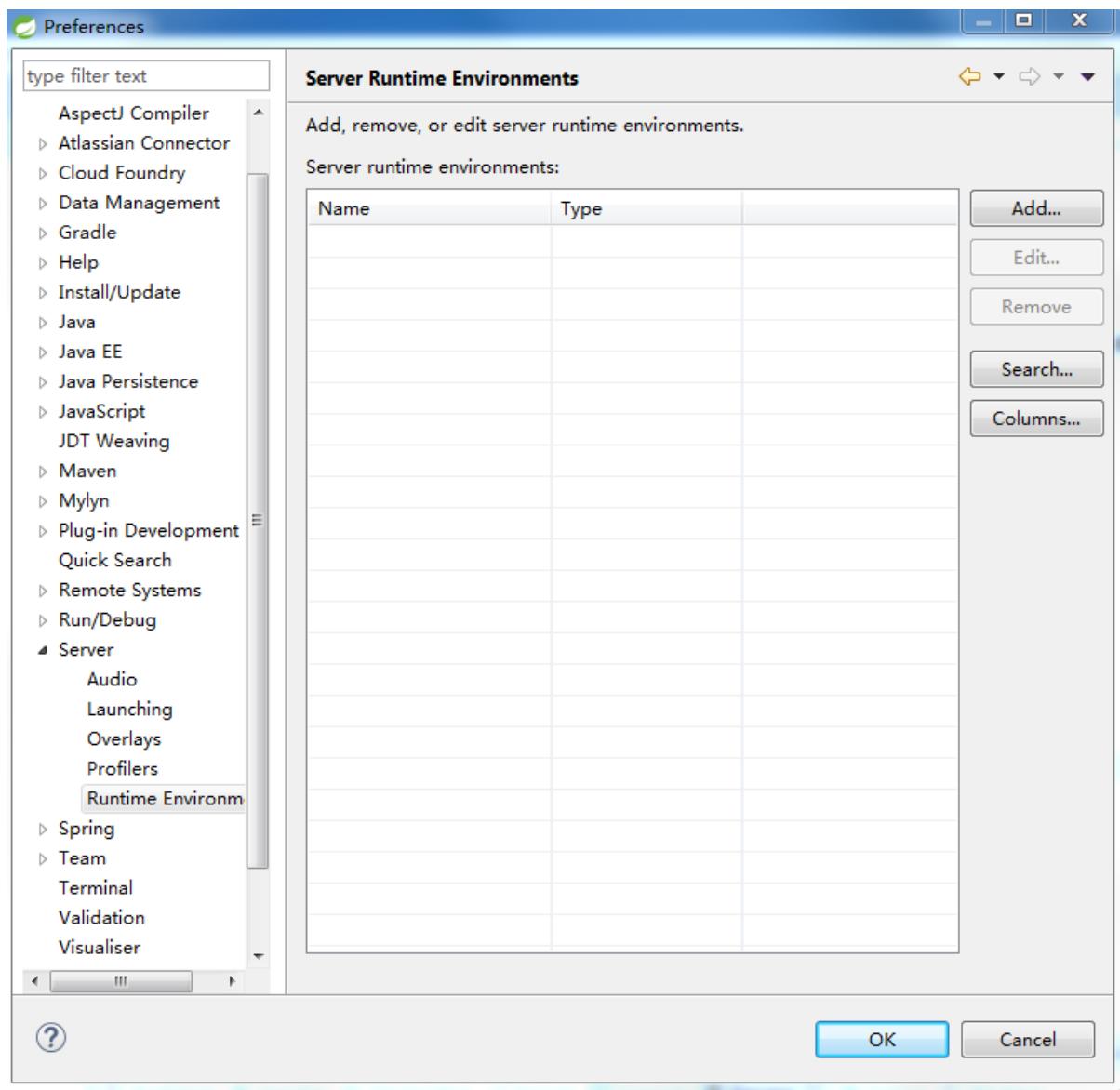


step3. Eclipse中设置断点，然后在RESTClient中执行就可以了

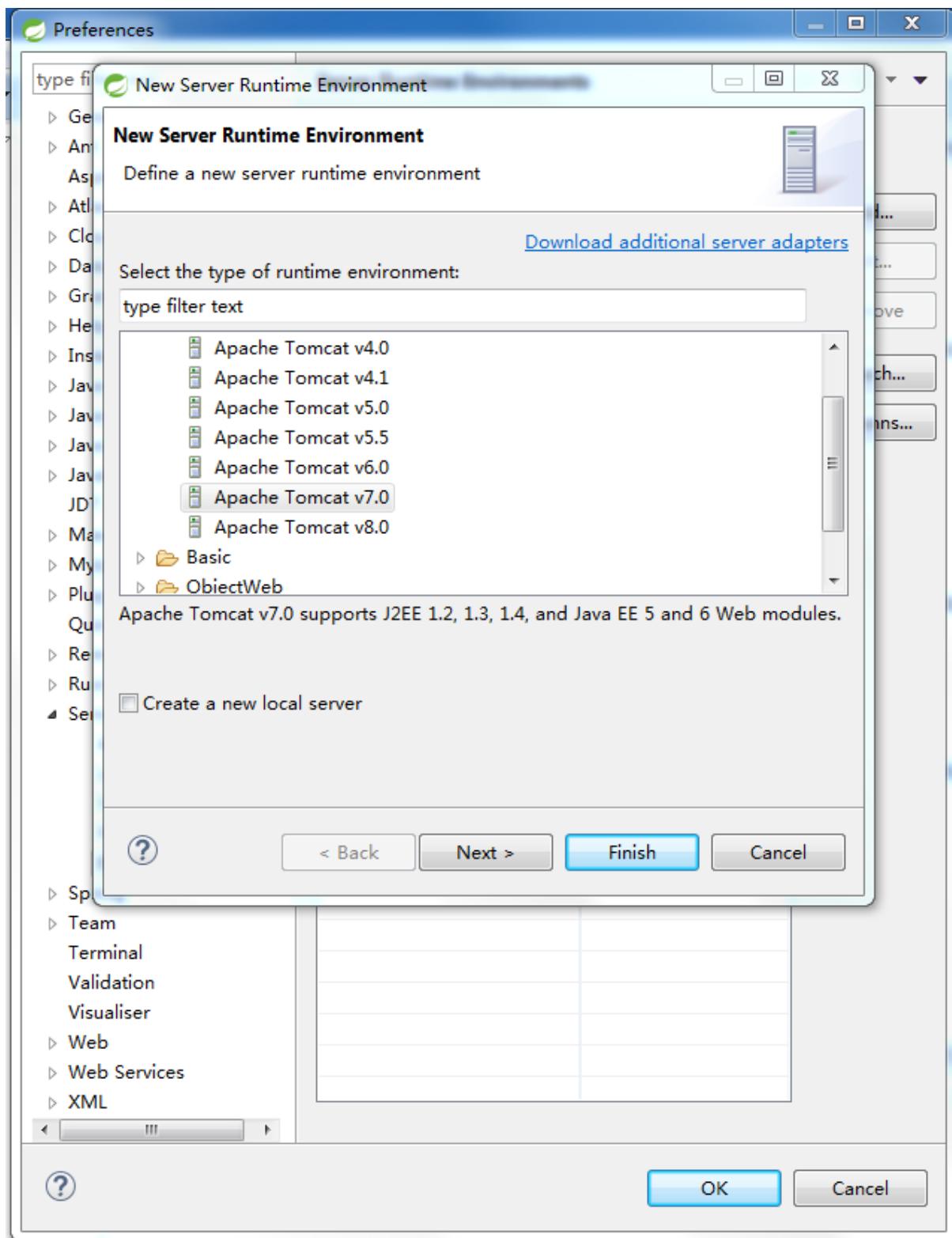
第二种方式：在eclipse通过在tomcat中添加webapp应用调试

eclipse中添加tomcat:

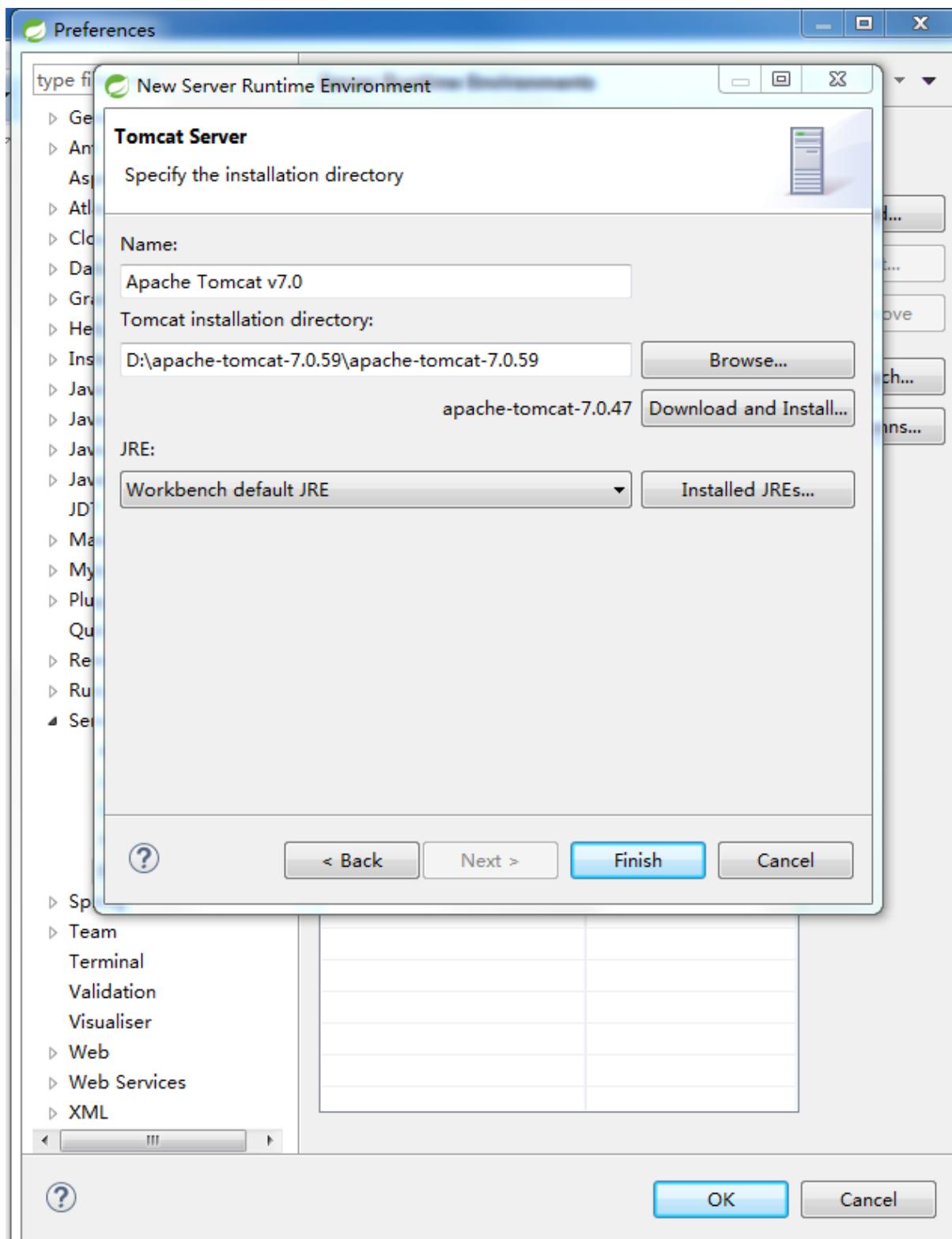
step1. 在eclipse先把tomcat加到servers上: Window→Preferences→Server→Runtime Environment, 如下截图



step2. 点击add, 选择本地安装的tomcat版本, 如下截图

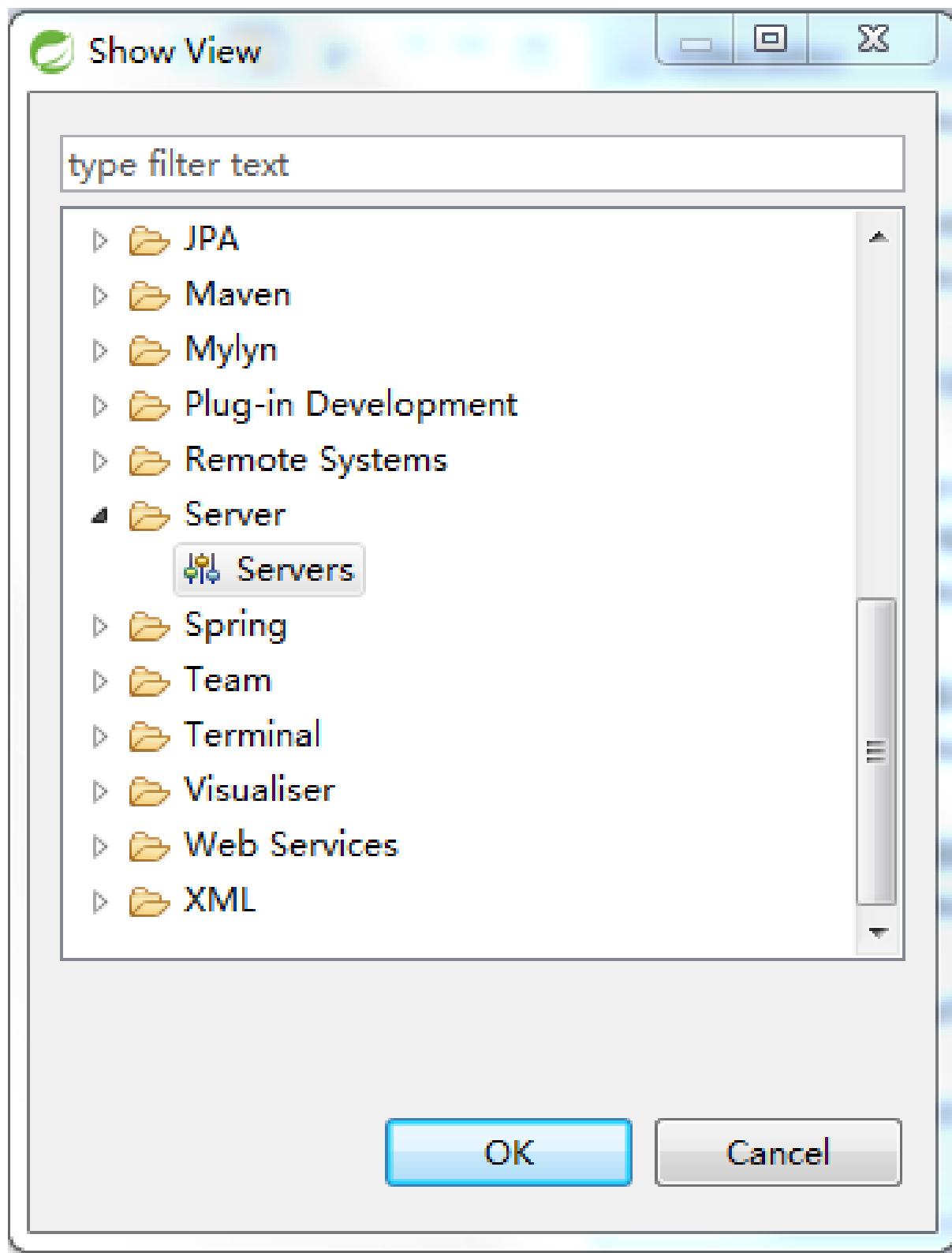


step3. 点击next, 选择本地tomcat的安装目录, 点击finish, 如下截图:

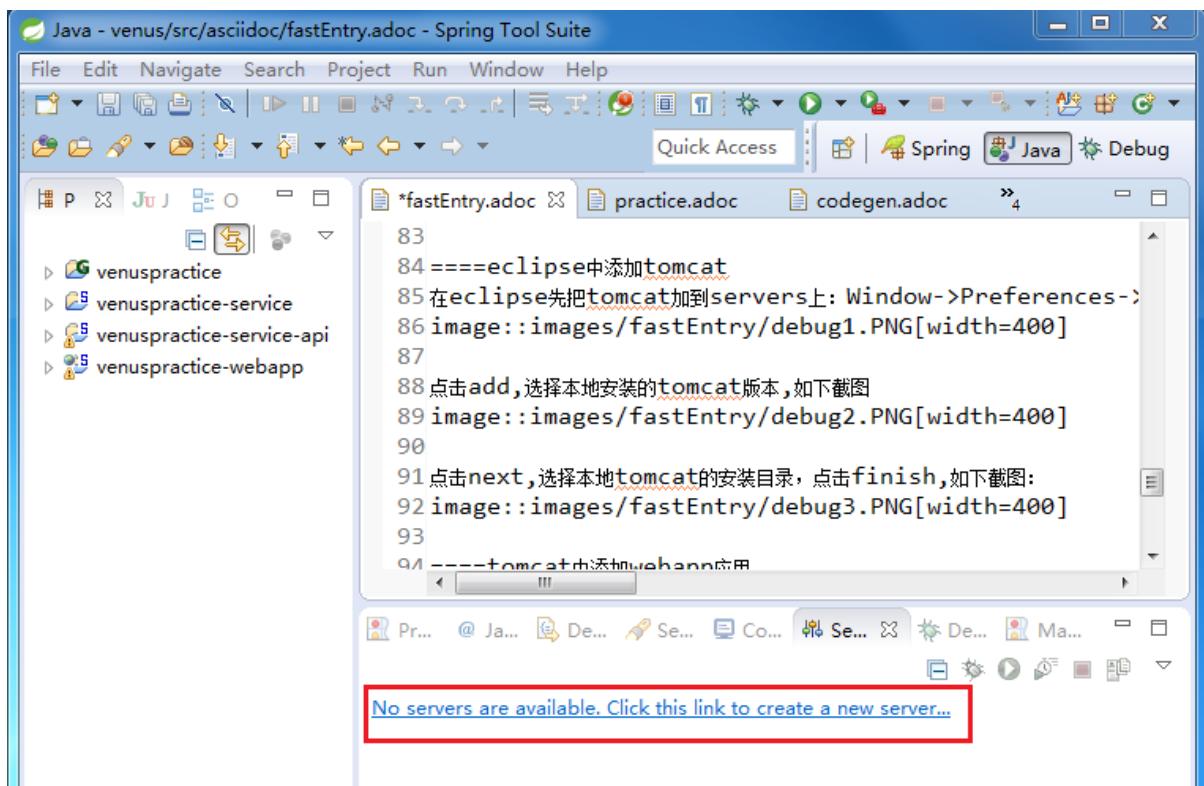


tomcat中添加webapp应用：

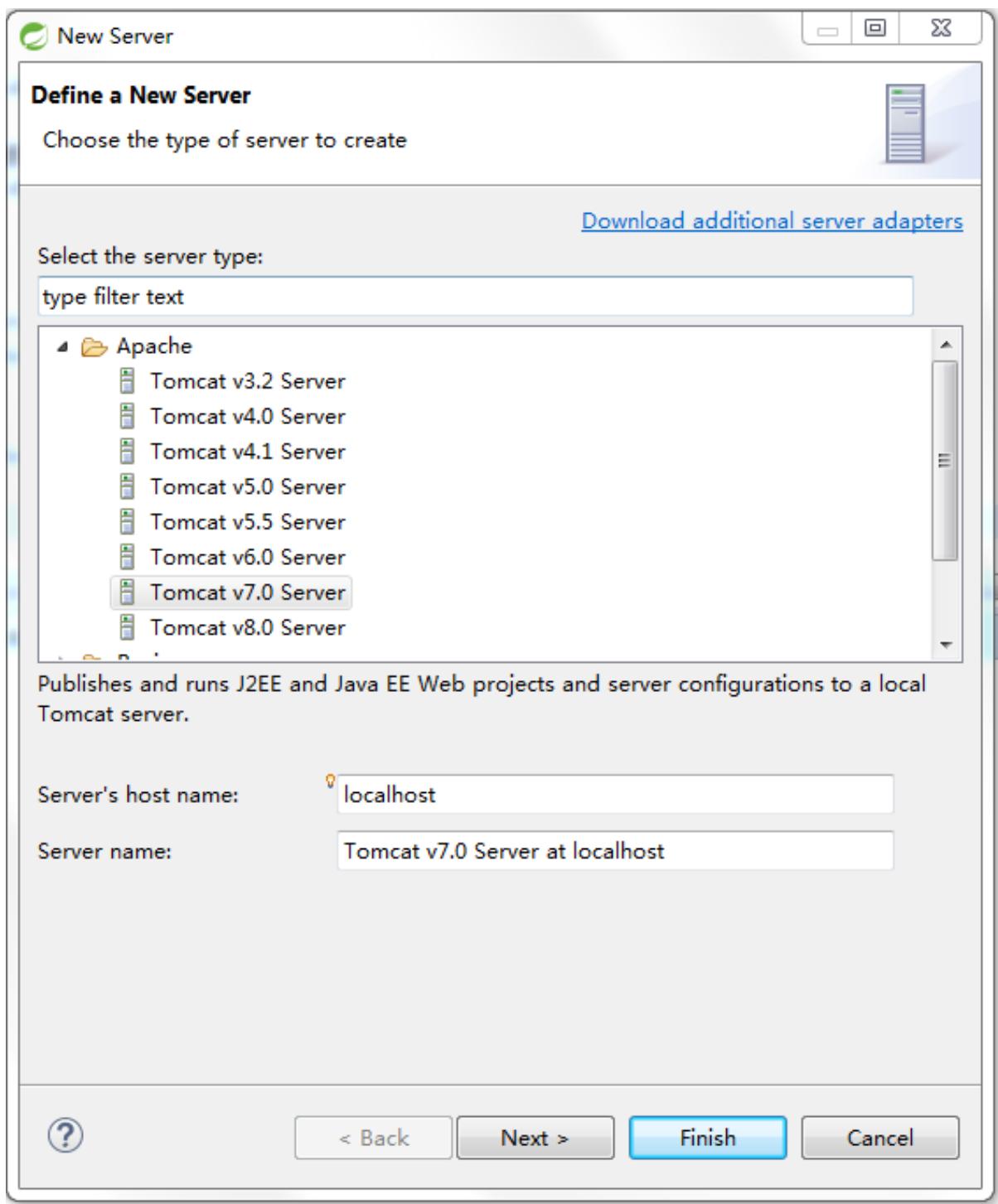
step1. 如果eclipse中没有servers窗口，先向eclipse中添加Servers窗口，Window→ShowView→Others→Server, 点击OK，如下截图：



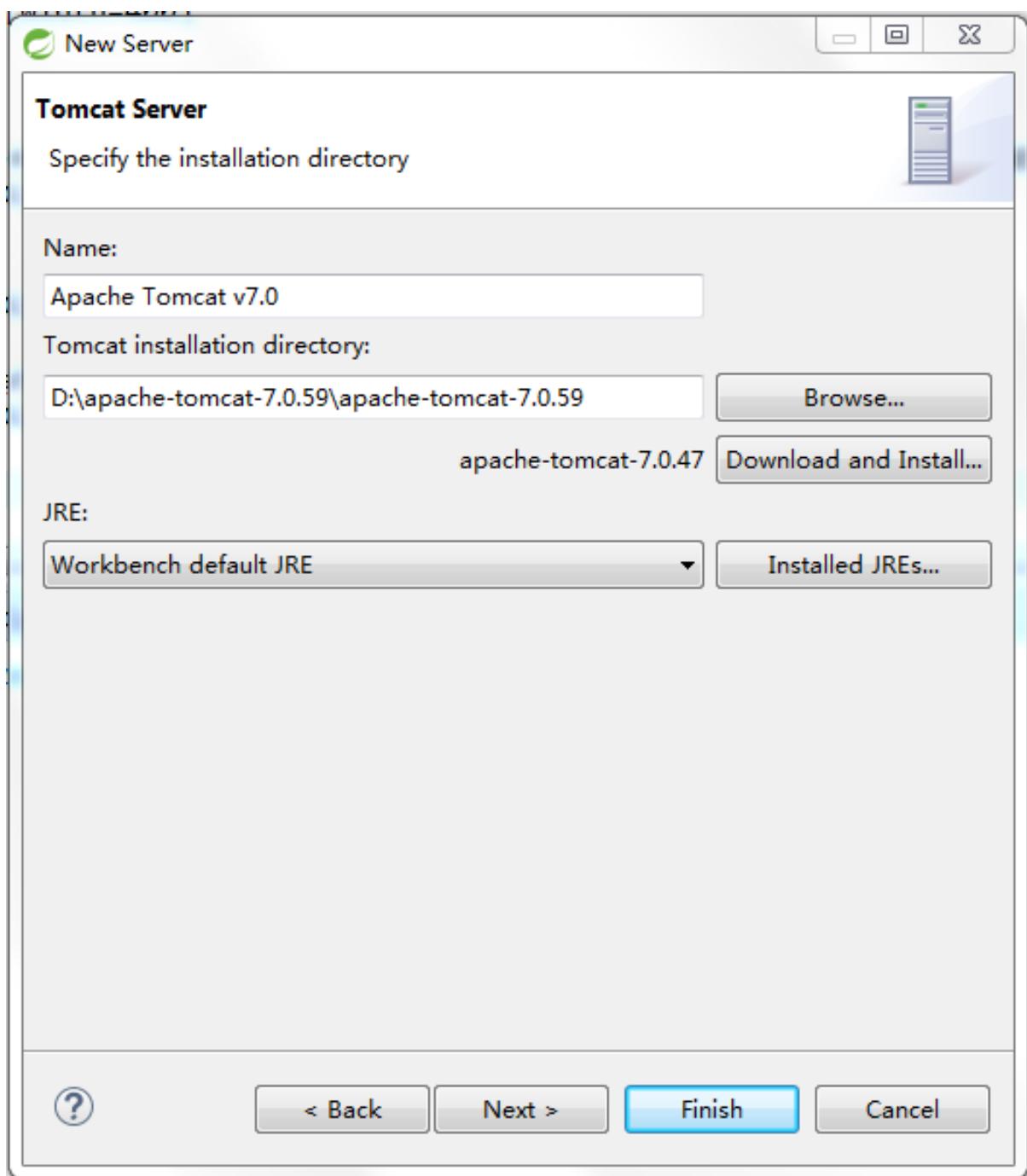
step2. 向tomcat中添加应用，点击链接，如下截图：



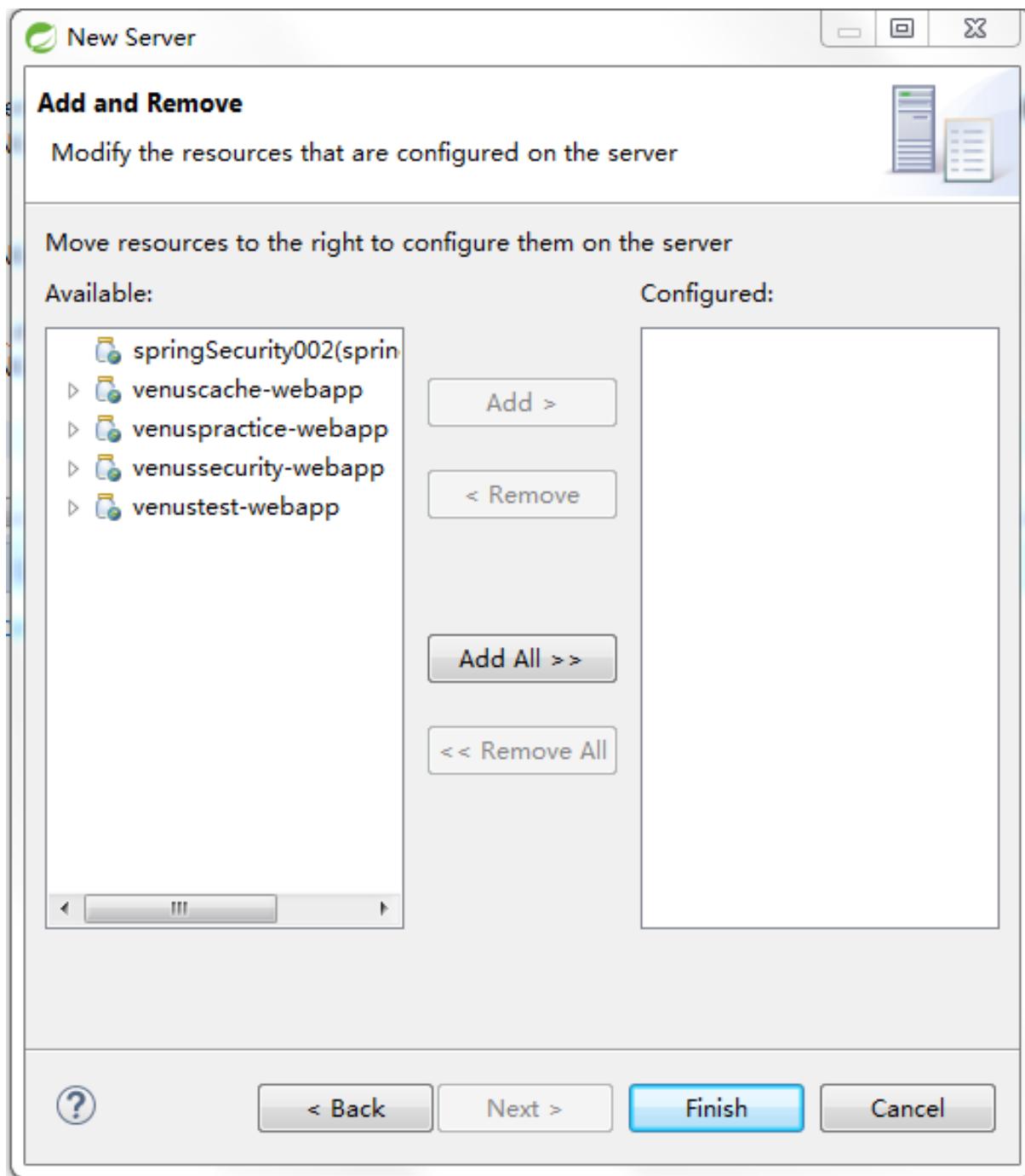
step3. 链接点开后, 选择Apache, 然后选择本地安装的tomcat版本, 如下截图:



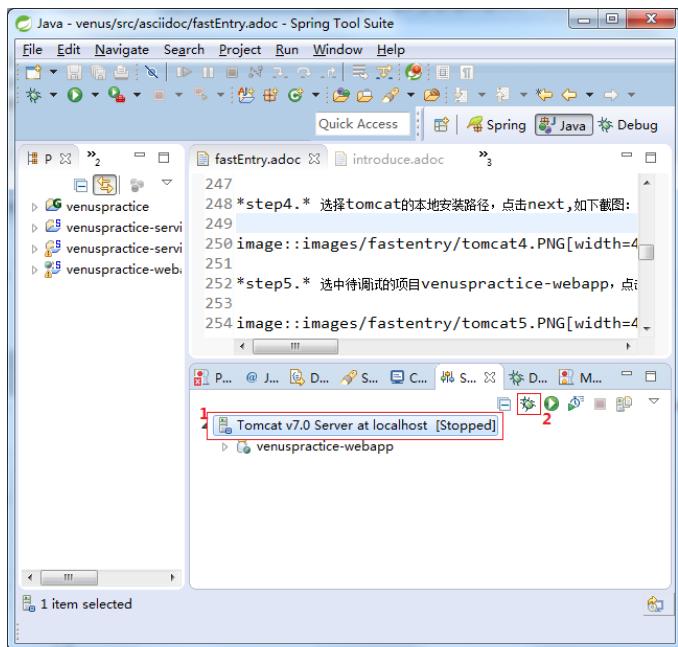
step4. 选择tomcat的本地安装路径，点击next, 如下截图：



step5. 选中待调试的项目venuspractice-webapp，点击add, 然后点击finish就可以了，如下截图：



step6. 右键下图中红框1→Debug 或者是点击下图中的红框2内小图标，设置好断点，就进入debug模式了



8. #OSP项目示例演示

Note

本文档针对OSP 2.x 编写，OSP 1.x 系列请参考 [Venus Framework Reference Document 1.3.0](#)

我们将以一个Student服务项目做例子，演示如何基于Venus平台及其codegen工具快速开发OSP项目。

8. 1#Venus开发环境

Venus项目要求JDK 7.0+，建议使用JDK 7.0，因为Venus框架是在JDK7.0上进行测试的。

OSP服务运行时需要连接配置中心的ZooKeeper，地址详见 [配置中心集成环境 & QA环境](#)

配置开发环境的环境变量：

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:2181
```

8. 2#Venus codegen工具的使用

Step1. [下载codegen-1.3.8](#)，解压缩至本地磁盘任意位置。

Step2. 准备示例数据库

codegen工具会读取数据库的表结构来生成代码，因此需要先在MySQL中设计好示例用的student表。

```
CREATE TABLE `student` (
  `id` bigint(20) NOT NULL,
  `name` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Note

用户可在本地MySQL上自行创建该表，也可以使用公共开发服务器上已创建好的表。

Step3. 配置并运行codegen

进入codegen/bin文件夹，打开config.properties配置文件，按下面的示例进行编辑。

```
#####
# project configuration
#####
project.exportPath=E:/venusosp
project.name=venusosp
project.packageName=com.vip.venus
project.moduleName=userservice
project.type=ospservice
project.buildTool=gradle

#####
# database configuration
#####

db.driver=com.mysql.jdbc.Driver
db.url=jdbc:mysql://10.101.18.72:3306/osp
db.userId=vipuser
```

```
db.pwd=xR54PUU8GWia
db.name=osp
db.tables=student:Student
db.shardingTables=
```

配置信息请参考webapp项目示例演示：Section#7.2，“Venus codegen工具的使用”

此处假设目录为E:/venusosp，下同。示例中的MySQL连接指向已创建好Student表的公共MySQL服务器。

运行codegen.bat文件，可以在E:\venusosp\venusosp下看到：

位置	名称	修改日期	类型	大小
	.gradle	2015/3/31 18:00	文件夹	
	gradle	2015/3/31 18:00	文件夹	
	venusosp-service	2015/3/31 18:00	文件夹	
	venusosp-service-api	2015/3/31 18:00	文件夹	
	build.gradle	2015/3/31 18:00	GRADLE 文件	4 KB
	gradle.properties	2015/3/31 18:00	PROPERTIES 文件	2 KB
	gradlew	2015/3/31 18:00	文件	6 KB
	gradlew.bat	2015/3/31 18:00	Windows 批处理...	3 KB
	libraries.gradle	2015/3/31 18:00	GRADLE 文件	6 KB
	README.md	2015/3/31 18:00	MD 文件	0 KB
	settings.gradle	2015/3/31 18:00	GRADLE 文件	1 KB

Note

当一个venus项目生成后，如果增加/修改数据库和数据表，还可以利用venus-codegen只生成关于修改的数据库/表的代码，

具体使用参考Chapter#44，代码生成器(Venus Codegen)

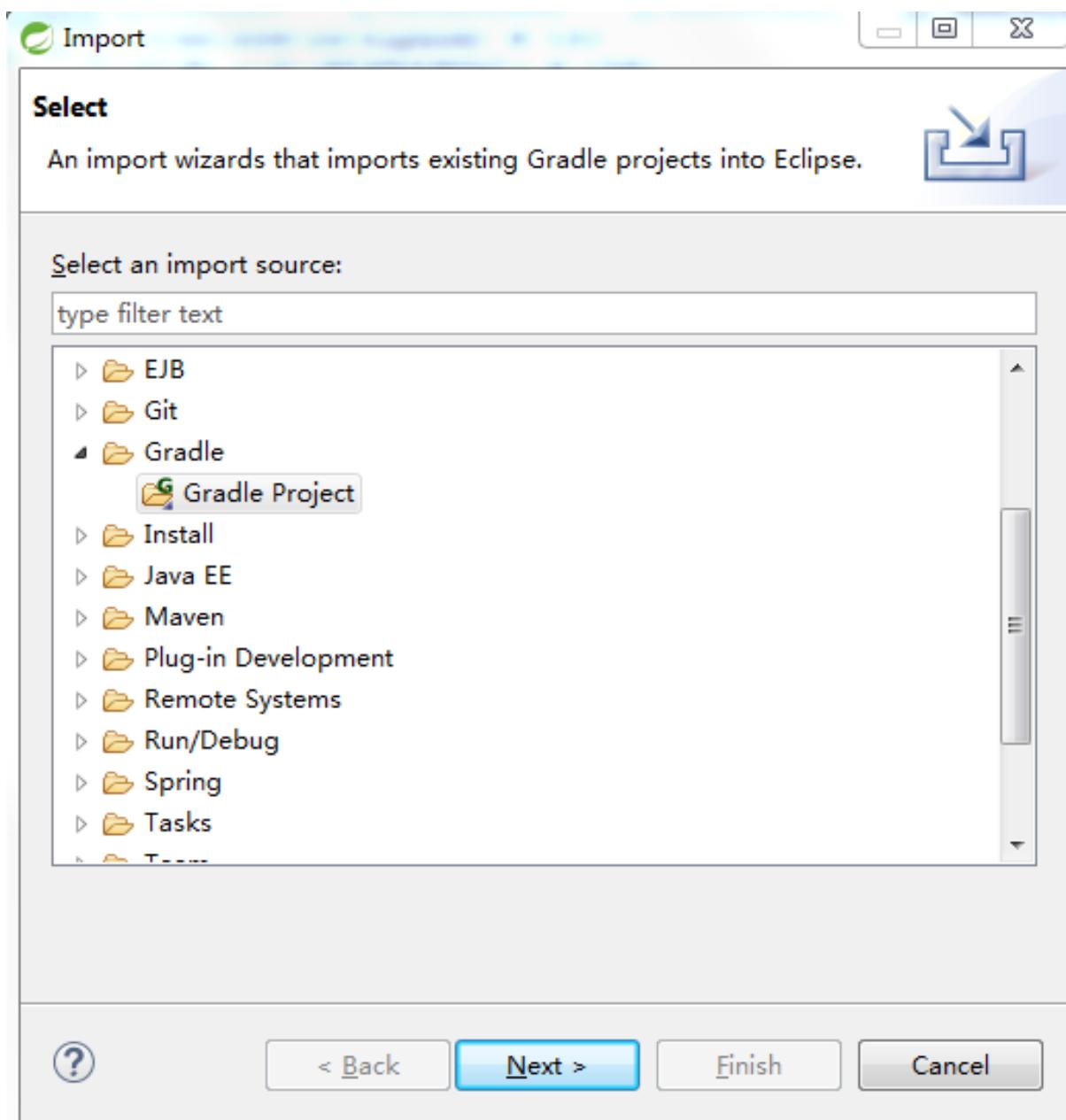
Step4. Eclipse安装Gradle插件

通过Eclipse Marketplace安装：Help→Eclipse Marketplace

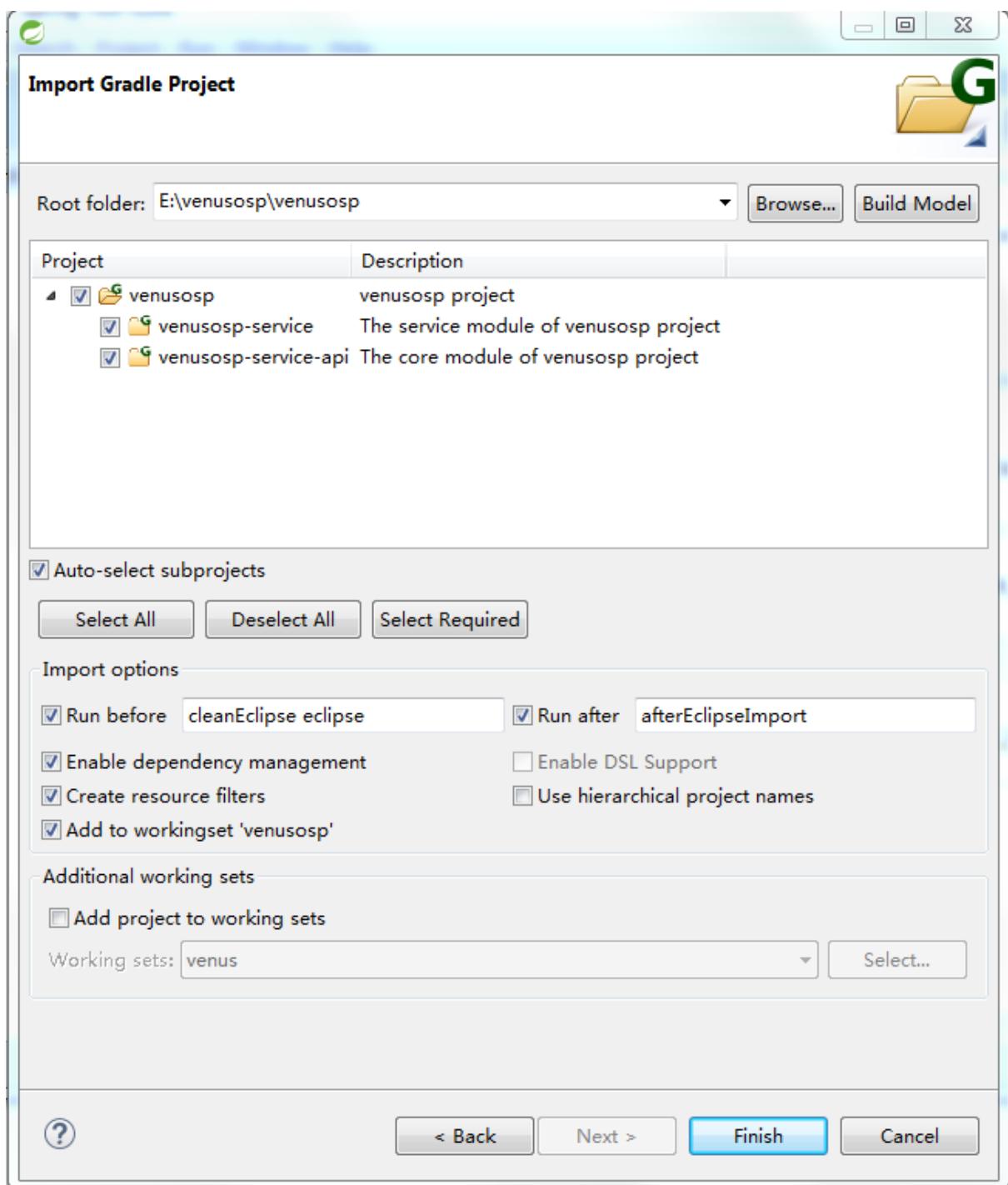
查询Gradle，选择“Gradle Integration for Eclipse”，安装后重启Eclipse。

Step5. 将生成的代码导入Eclipse

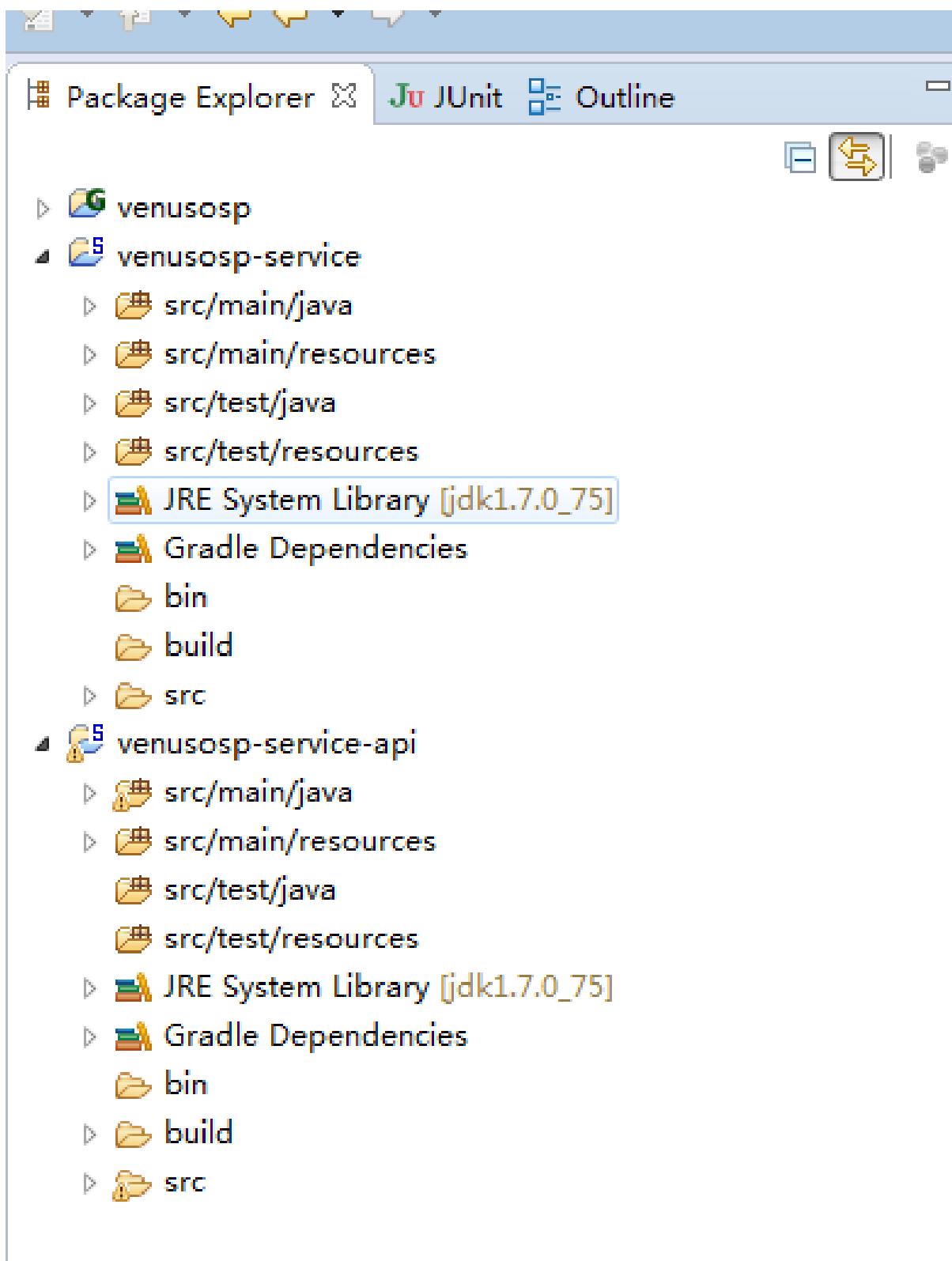
File→Import→Gradle→Gradle Project:



选择代码所在目录E:\venusosp\venusosp→Build Module→全选所有项目→Finish



完成后，在Eclipse中能看到如下三个项目：



Note

现实开发中如果项目A依赖项目B，然后B可能很频繁的出SNAPSHOT版本的jar给A依赖使用，这种情况请参考Chapter#46，项目依赖

8. 3#OSPI项目的文件说明

venusosp项目: gradle的相关配置文件

文件	描述
build.gradle	gradle标准build配置文件
gradle.properties	gradle相关的配置项，里面的配置项都可通过gradle命令行-P同名参数覆盖
gradlew	linux下的gradle wrapper执行文件
gradlew.bat	windows下的gradle wrapper执行文件
libraries.gradle	所有的依赖包的定义，用于在dependencies里引入依赖变量
setting.gradle	gradle项目的定义文件

venusosp-service-api项目: 服务API项目，提供接口和模型

文件	描述
StudentModel.java	service接口使用的数据结构
StudentModelHelper.java	数据结构的帮助类
StudentService.java	service接口
StudentServiceHelper.java	service接口的帮助类
StudentIDL.java	service定义文件

Note

StudentModel, StudentModelHelper, StudentService, StudentServiceHelper都是编译idl文件生成的，开发人员不要手工编写。

venusosp-service项目: 服务实现项目，对service-api接口的实现

文件	描述
Student.java	DAL层数据的映射类
StudentRepository.java	DAL层的接口类
StudentServiceImpl.java	service接口的实现类
META-INF/service.xml	OSPI使用的定义服务实现类的Spring Bean 配置文件
applicationContext.xml	服务实现类依赖的其他类的Spring Bean的配置文件

applicationCfgDef.xml	配置中心定义文件，具体请查看Chapter 18，配置定义文件
application.properties	应用配置文件，具体请查看Chapter 18，配置定义文件
mapper/StudentMapper.xml	myBatis的mapping文件
ospconfig/client_filter.properties	客户端自定义Filter定义文件
ospconfig/filter.properties	服务端自定义Filter定义文件
logback.groovy	Logback 日志配置文件
vip-mercury.properties	Mercury配置文件

8. 4#OSP项目的本地运行

Step1. 运行：

运行前再确认一次已完成 Section#8. 1，“Venus开发环境”的ZooKeeper地址配置。

打开CMD窗口通过cd命令进入你生成的项目目录下，运行 gradlew clean runOsp

运行成功后会弹出如下截图，在启动OSP服务的同时，还会启动一个仅在开发期嵌入的Jetty服务器，内置在线测试与在线文档的Web应用：

The screenshot shows the 'Online Testing Tool' section of the Vip.com Open Platform interface. The left sidebar has a green header '唯品会开放平台便捷工具' with two buttons: '文档展示' (Documentation Display) and '在线测试' (Online Testing). The main area has a title '在线测试工具'. It contains several input fields and buttons:

- Return type: A dropdown menu set to 'JSON'.
- Service name/version: A dropdown menu set to '---请选择---'.
- Method name: A dropdown menu set to '---请选择---'.
- Save data: A button labeled '保存' (Save) next to a text input field '描述一下这个数据吧' (Describe this data). There is also a link '保存最近一次提交的测试' (Save the last submitted test).
- Load data: A button labeled '载入' (Load) next to a dropdown menu '载入数据' (Import Data) set to '---请选择---'. There is also a link '载入历史数据' (Import historical data).
- Request parameters: A large text input area for entering request parameters.
- Test submission: Two buttons at the bottom of the request parameters area: '提交测试' (Submit Test) and '生成示例代码' (Generate Example Code).
- Request data: A text input area for entering request data, with a button '请求数据' (Request Data) and a link '全部展开/收起' (Expand/Collapse All).
- Response data: A text input area for displaying response data, with a button '返回数据' (Return Data) and a link '全部展开/收起' (Expand/Collapse All).

Step2. 测试:

点击“在线测试”，进入测试页面。

返回类型：选择“JSON”

服务名-版本号：选择运行的服务

方法名：选择你要操作的方法，这里以create为例

请求参数：先点击后面的小方框，然后点击展开符号，在里面输入请求参数，点击“提交测试”

请求参数：

studentModel(com.vip.venus_codegen.service.StudentModel)

<input type="checkbox"/> id	12	<input checked="" type="checkbox"/>
<input type="checkbox"/> name	yuqian	<input checked="" type="checkbox"/>

在请求数据中可以看到你的请求参数，如果操作成功，在返回数据中看到如下截图：

请求数据：全部展开/收起

```
{
  "studentModel": {
    "id": 123,
    "name": "yuqianqian"
  }
}
```

返回数据：全部展开/收起

```
{
  "returnCode": "0",
  "result": 1
}
```

8. 5#OSP项目的调试

Step1. 运行时配置-PospService.debug参数，完整命令如下：

```
gradlew clean runOsp -PospService.debug=true ①
gradlew clean runOsp -PospService.debug=true -PospService.debug.port=*** ②
```

① 此时默认的debug端口是5005，必须与Eclipse中配置的端口一致

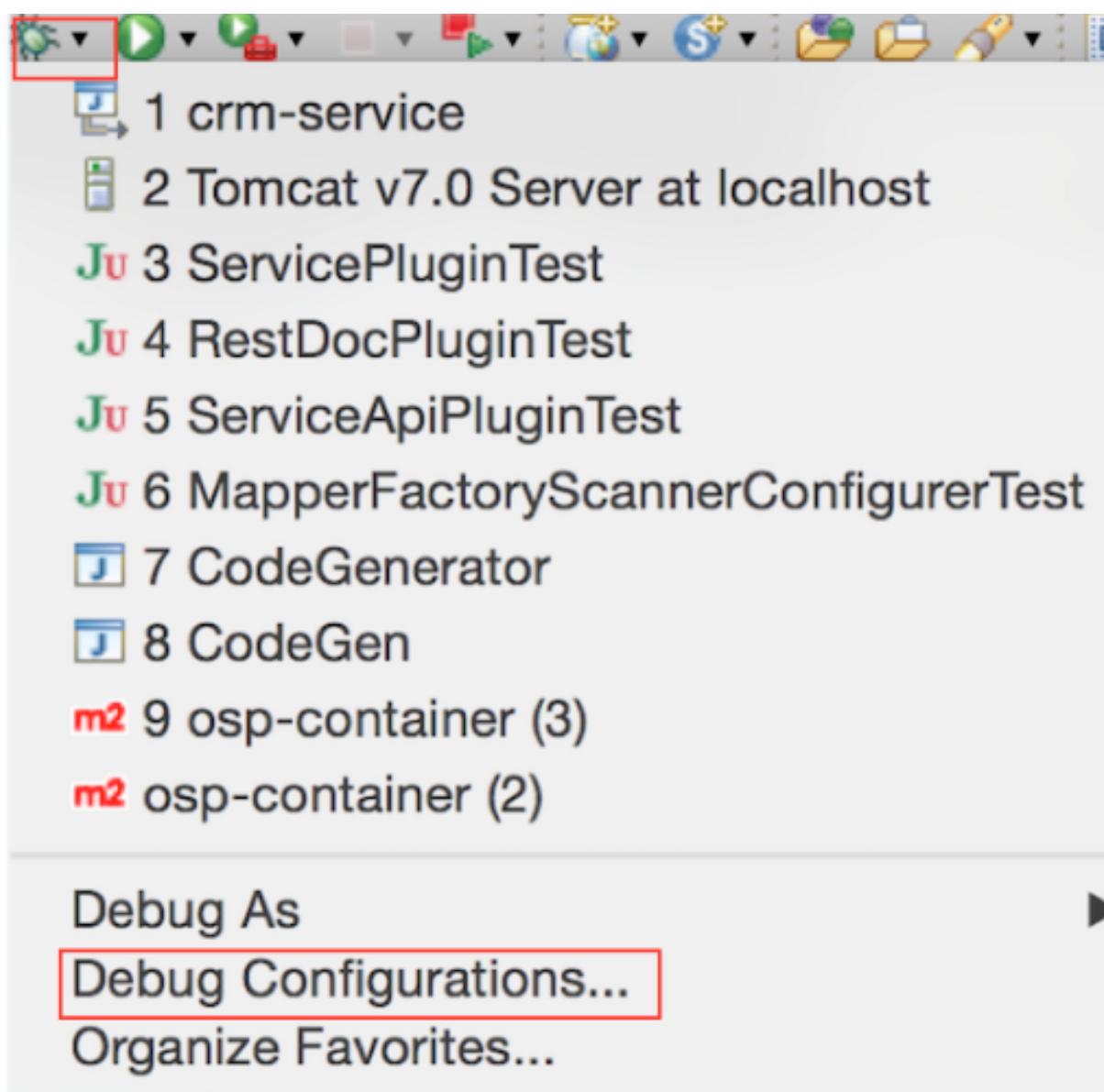
- ② 可通过`-PospService.debug.port`参数任意设置端口，必须与Eclipse中配置的端口一致

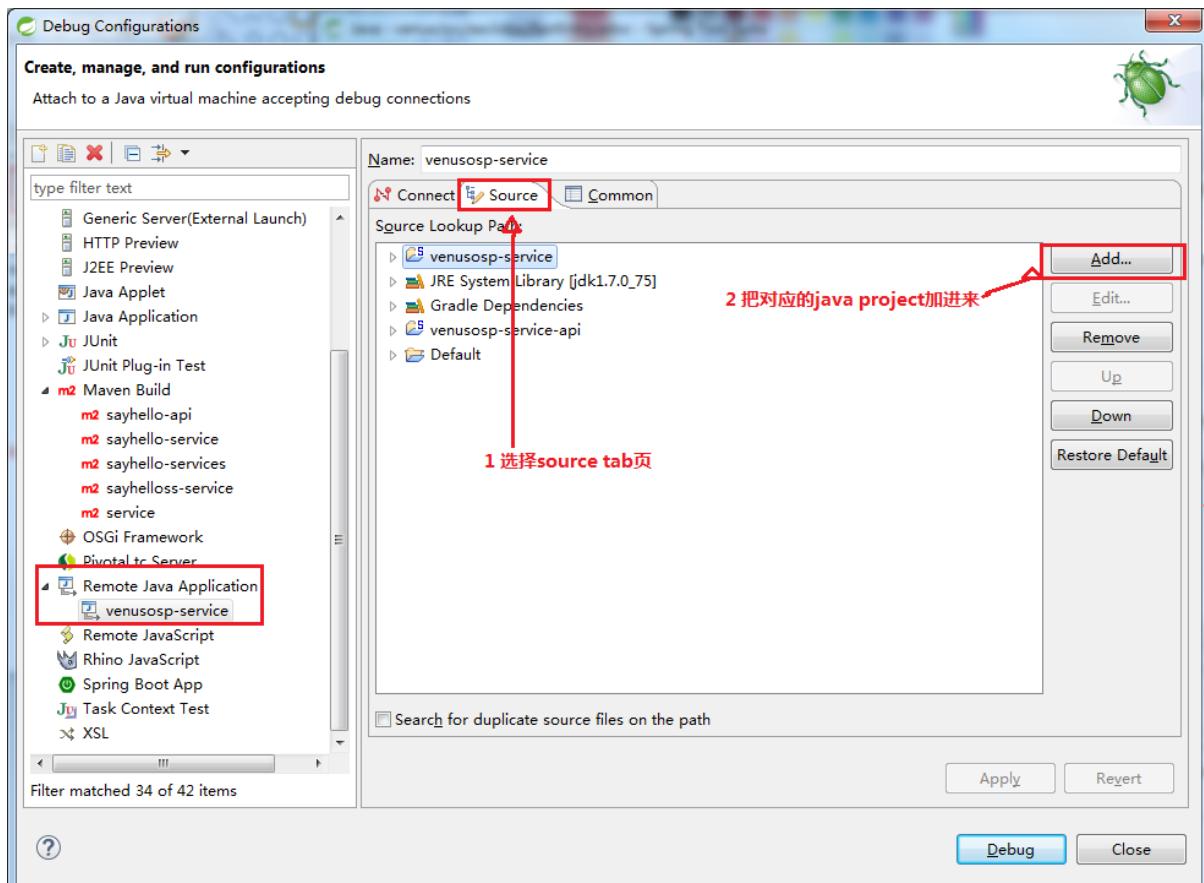
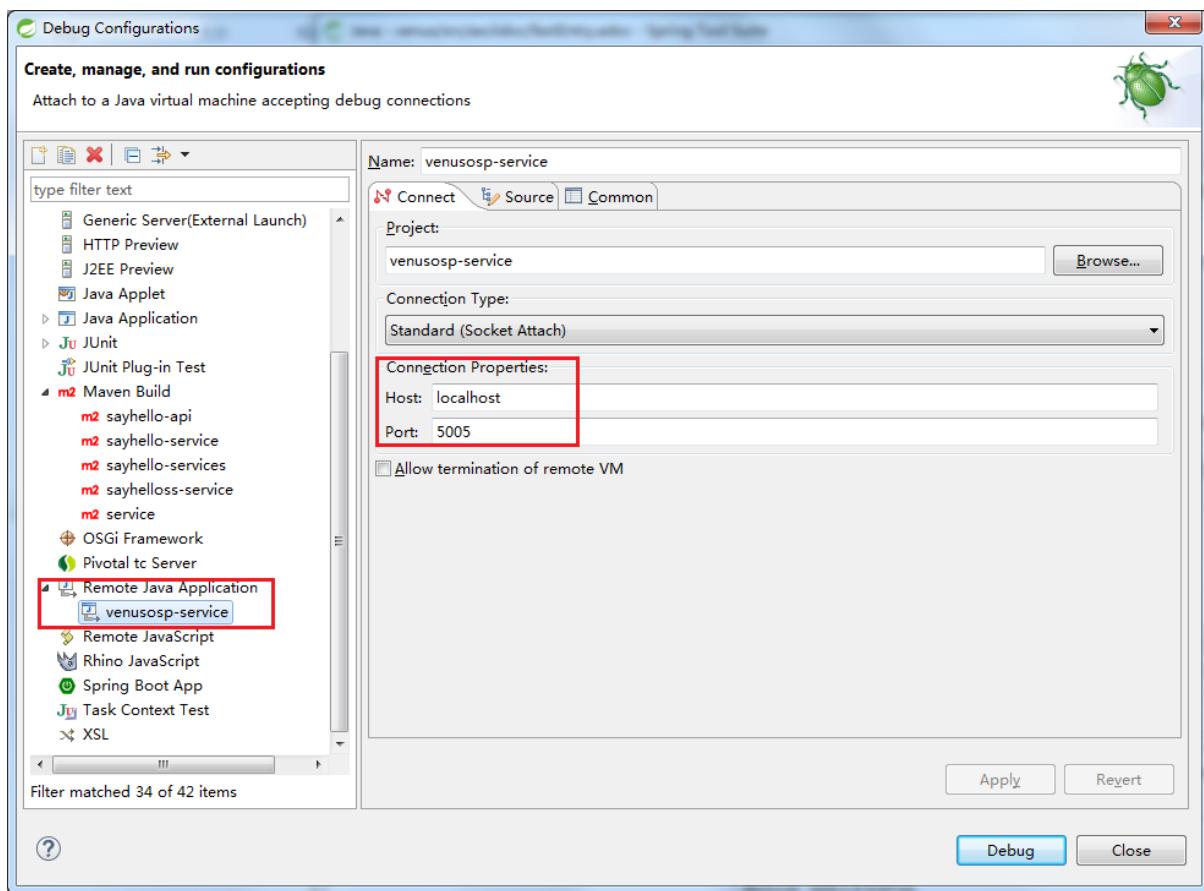
Note

可以在`gradle.properties`里直接增加`-PospService.debug=true`，使得项目默认开启debug

Step2. Eclipse中新建远程debug

如下图配置：配置Host和port，并引进Java工程





8. 6#进一步阅读

OSP项目的详细用户手册，请参阅Part#VII，“OSP”相关章节。

Part#III. #Venus

framework下的开发实践

在venus快速入门中我们介绍了如何利用venus进行简单开发，本章节包含了所有基于venus的实践类型。

本章节主要介绍如何利用venus进行数据库分库/读写分离/分表/缓存，认证授权以及测试，发布，代码管理和持续集成等操作。

9. 配置文件的变量定义	47
9.1. properties下的变量定义	47
9.2. applicationContext.xml下的变量定义	48
10. 使用venus-context读取配置文件	50
10.1. 属性配置	50
11. 使用venus-data进行分库/读写分离/分表	54
11.1. 添加依赖	54
11.2. spring引入venus-datasource	54
11.3. spring添加数据库操作相关bean	55
11.4. properties文件配置	55
11.5. 分库操作	56
spring配置	57
properties配置	57
分库规则	57
服务层配置	58
嵌套事务处理遵循规则	58
11.6. 读写分离操作	59
properties配置	59
spring配置	59
服务层配置	59
11.7. 分表操作	60
spring配置	60
properties配置	60
分表规则	61
MyBatis Mapper配置	61
数据操作层配置	61
12. 使用venus-jdbc进行分库/读写分离/分表	63
12.1. 添加依赖	63
12.2. spring引入venus-jdbc	63
12.3. spring添加数据库操作相关bean	63
12.4. spring配置	64
12.5. properties文件配置	64
13. 使用venus-cache进行缓存操作	66
13.1. spring引入venus-cache	66
13.2. venus-cache配置和使用	66
13.3. cache cluster配置	67
14. 使用配置中心配置channel和queue	69
14.1. 配置要使用的配置中心地址	69
14.2. 然后在配置中心创建channel, 选择相应的集群	69
14.3. 创建queue, 选择相应的集群	70
14.4. 如果需要设置routingkey, 在下面绑定routingkey	70
15. 使用venus-test进行单元测试	71
15.1. 添加依赖	71
15.2. 配置文件说明	71
15.3. 测试类说明	72
15.4. 注解说明	72
16. 使用venus-security进行认证和授权	74
16.1. 添加依赖	74
16.2. 使用venus-security接入OA流程	74
17. 使用配置中心读取配置信息和监控配置变更	78

17.1. 配置中心的zk集群	78
17.2. venus-context配置	78
17.3. 从配置中心读取cache信息	79
17.4. 从配置中心监控配置变更	80
18. 使用Mercury	88
18.1. Mercury trace接入步骤（开发篇）	88
添加依赖	88
server配置	88
添加trace配置文件	89
添加log4j/logback配置	90
验证	92
18.2. Mercury trace接入步骤（部署篇）	93
配置AspectJ拦截	93
安装flume	93
验证	94
19. 使用Hermes	96
19.1. 添加依赖	96
19.2. XML配置	96
Spring应用配置applicationContext.xml	96
19.3. 添加log4j/logback配置	96
19.4. 定义Item	98
HermesStatic	98
HermesGauge	98
HermesCounter	98
HermesMeter	98
HermesHistogram	98
HermesTimer	99
19.5. 注册Item	99
register	99
registerInternal	99
20. 使用中央文档系统	100
20.1. 系统简介	100
20.2. 主要功能	100
功能点	101
20.3. 使用步骤	101
21. 使用git管理源码	102
21.1. git配置	102
21.2. git分支模型	102
主分支	104
辅助分支	104
21.3. 使用git上传项目流程	104
22. 使用jenkins持续集成	106

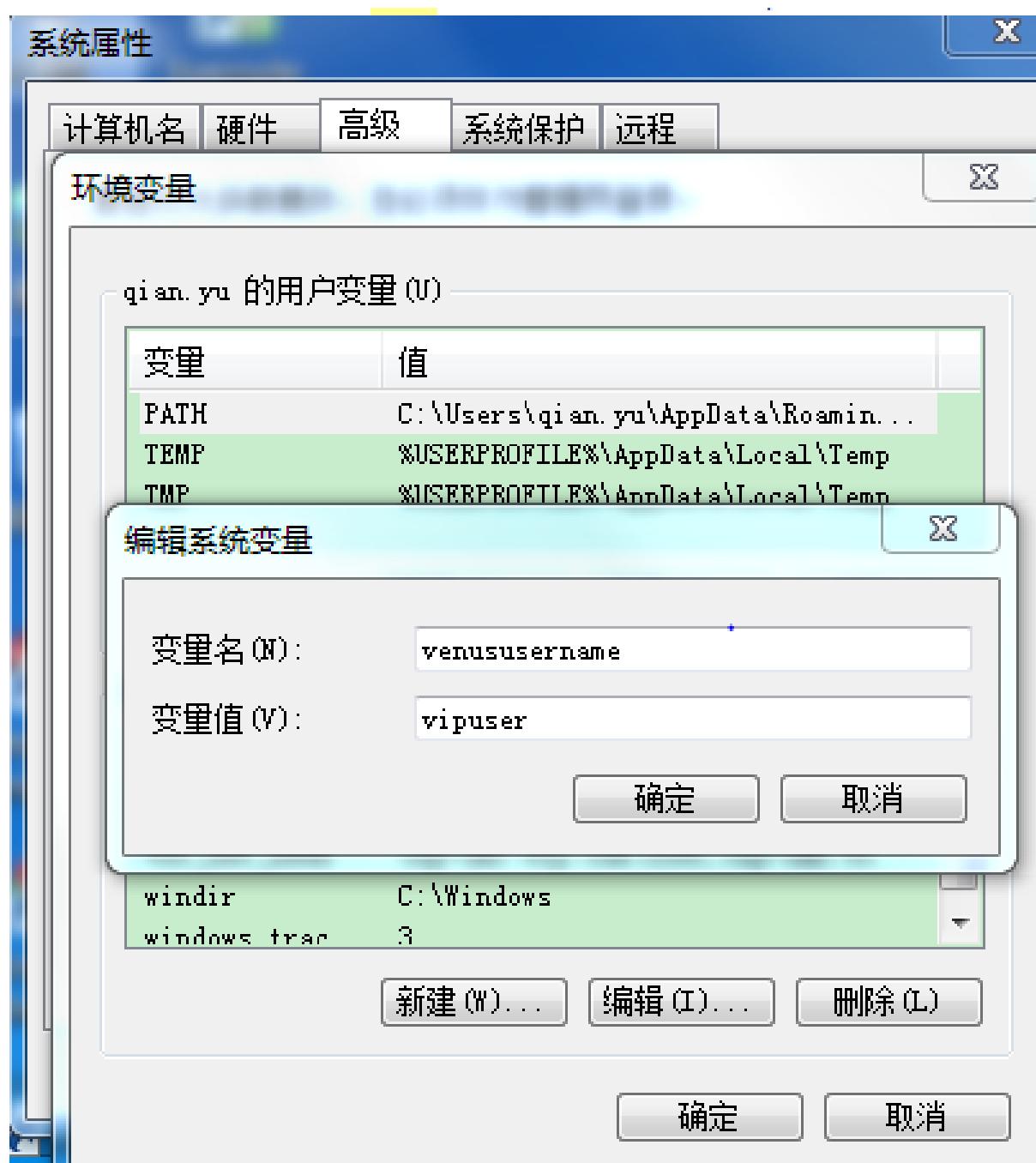
9. #配置文件的变量定义

9. 1 #properties下的变量定义

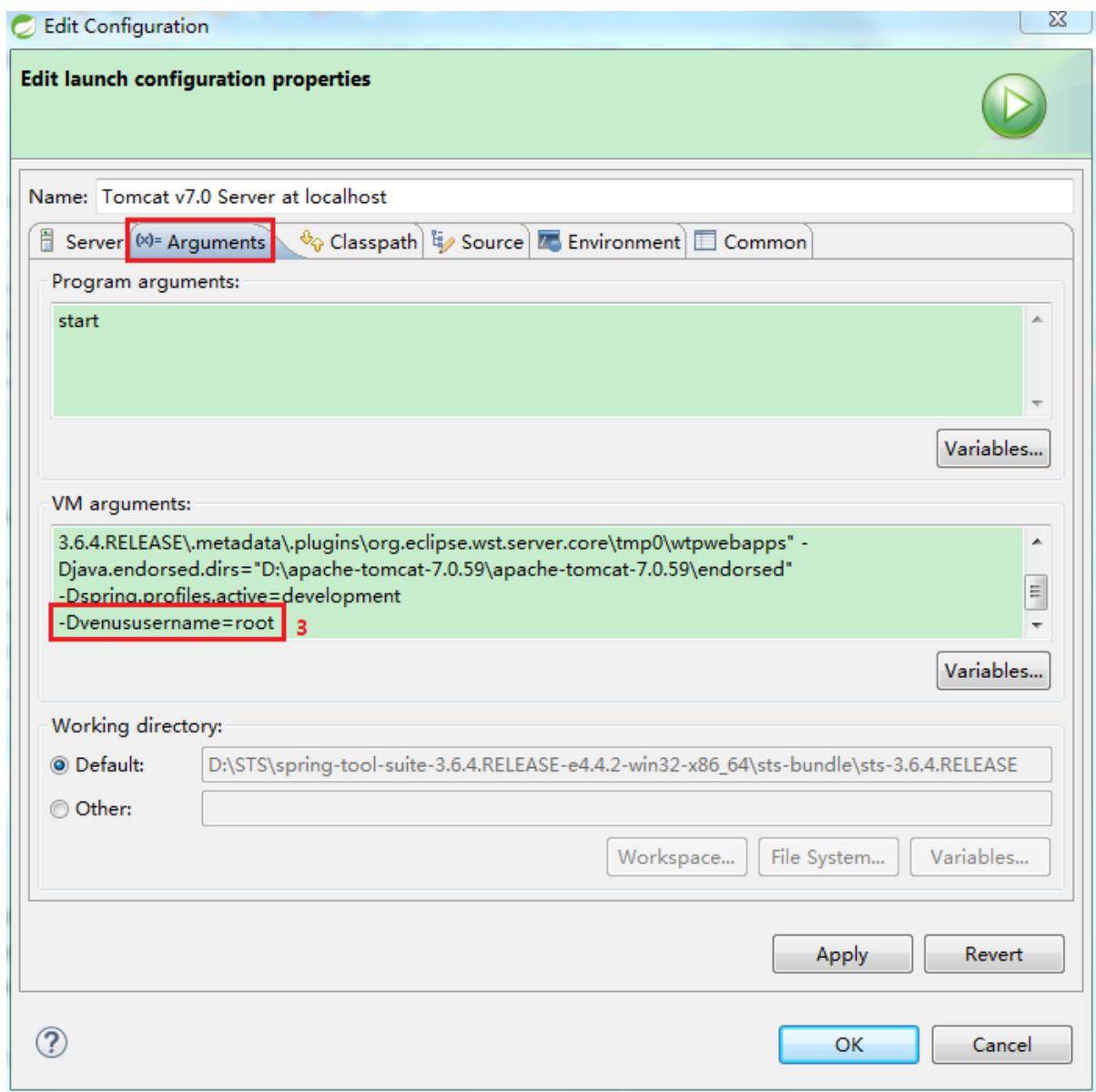
目前properties文件中的所有配置都支持变量定义：假如properties中配置`username=\${venususername}`，如果username为root

可以使用环境变量或者系统变量直接定义替换变量

window下设置环境变量如下图，linux设置环境变量：export venususername=root



windows下系统变量如下图设置，linux下直接在运行脚本中-Dvenususername=root



9. 2#applicationContext.xml下的变量定义

applicationContext.xml支持变量定义，可以直接在代码中已有的properties文件中定义，也可以使用环境变量指定appPropertiesFile=/xxx/config/xxx.properties，然后直接在 xxx.properties文件里直接配置变量值。

比如applicationContext.xml定义如下

```
<bean id="singleConnectionFactory" class="com.vipshop.rabbitmq.SingleConnectionFactory">
<property name="host" value="${rabbit.host}" />
<property name="port" value="${rabbit.port}" />
<property name="username" value="${rabbit.userName}" />
<property name="password" value="${rabbit.password}" />
</bean>
```

那么我们可以直接在运行环境下对应的properties文件中定义如下：

```
rabbit.host=10.199.219.218
```

```
rabbit.port=5672  
rabbit.userName=test  
rabbit.password=test
```

也可以另外写个app.properties如上定义，但是需要使用环境变量指定appPropertiesFile=/xxx/config/app.properties(app.propertiesd的目录)

10. #使用venus-context读取配置文件

venus-context是对Spring:context的扩展，详细属性定义请参考Section#24.4，“Venus Context”

venus-context:property-placeholder主要预定义了相关配置，支持了应用程序对配置中心的使用。

venus-context的所有属性都有默认值，开发者即使不配置venus-context任何属性也能使用配置中心。

10.1#属性配置

use-cfgcenter:

false表示不使用配置中心，本地读取配置文件；true表示使用配置中心，默认值是true；

```
<venus-context:property-placeholder use-cfgcenter="false" />
```

Note

当值为true即使用配置中心时，默认在location属性末尾增加以下字符串：

```
zk:/artifact/${cfgdef.group}/${cfgdef.name}/${cfgdef.version}/config
```

{cfgdef.group}, {cfgdef.name}, {cfgdef.version}都是定义在applicationCfgDef.xml中，

在环境变量或者系统变量中配置VIP_CFGCENTER_PARTITION参数来指定zk的partition

如果使用配置中心，需要在环境变量或者系统变量配置配置中心的zk以及相关属性

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:2181 ①
VIP_CFGCENTER_PARTITION=default ②
VIP_CFGCENTER_ZK_SLEEPMSBETWEENRETRY=1000 ③
VIP_CFGCENTER_ZK_MAXRETRIES=3 ④
VIP_CFGCENTER_ZK_CONNECTION_IMEOUT_MS=50000 ⑤
VIP_CFGCENTER_ZK_SESSION_IMEOUT_MS=10000 ⑥
```

① 必须配置：配置中心配置在zk的client的IP和PORT

② 配置中心上传项目的分区，默认值是default <3>~<6>如果不配置，则采用默认值

Note

配置中心是部署在zookpeer上的，所以需要配置上述代码即zkclient的IP和port才能连接配置中心

location: 需要读取的配置文件的路径。

当use-cfgcenter="true"时，location的默认配置如下：

```
<venus-context:property-placeholder location="classpath:/properties/application.properties,
                                         classpath:/properties/${spring.profiles.default}/application-
                                         ${spring.profiles.default}.properties,
                                         classpath:/properties/${spring.profiles.active}/application-
                                         ${spring.profiles.active}.properties,
                                         file:${appPropertiesFile},
                                         zk:/artifact/{cfgdef.group}/{cfgdef.name}/{cfgdef.version}/config"/>
```

当 use-cfgcenter="false" 时, location的默认配置如下:

```
<venus-context:property-placeholder location="classpath:/properties/application.properties,
    classpath:/properties/${spring.profiles.default}/application-
    ${spring.profiles.default}.properties,
    classpath:/properties/${spring.profiles.active}/application-
    ${spring.profiles.active}.properties,
    file:${appPropertiesFile}" />
```

Note

因为location有默认值, 所以即使开发者不用对location做任何配置也能读取到类路径下的 properties文件, 当前运行环境相关的properties文件和配置中心上的配置文件。

用户也可以自定义location配置, 配置如下:

```
<venus-context:property-placeholder location="classpath:zk.properties,
    classpath:app.properties,
    zk:/resources/app/config"/>
```

这段代码的配置定义是读取根路径下的app.properties/zk.properties文件和zookeeper中节点 resources/app/config下的配置文件。

Note

如果在location中自定义zk, 这个zk的IP/port的默认值是配置中心即 VIP_CFGCENTER_ZK_CONNECTION配置的IP/port,

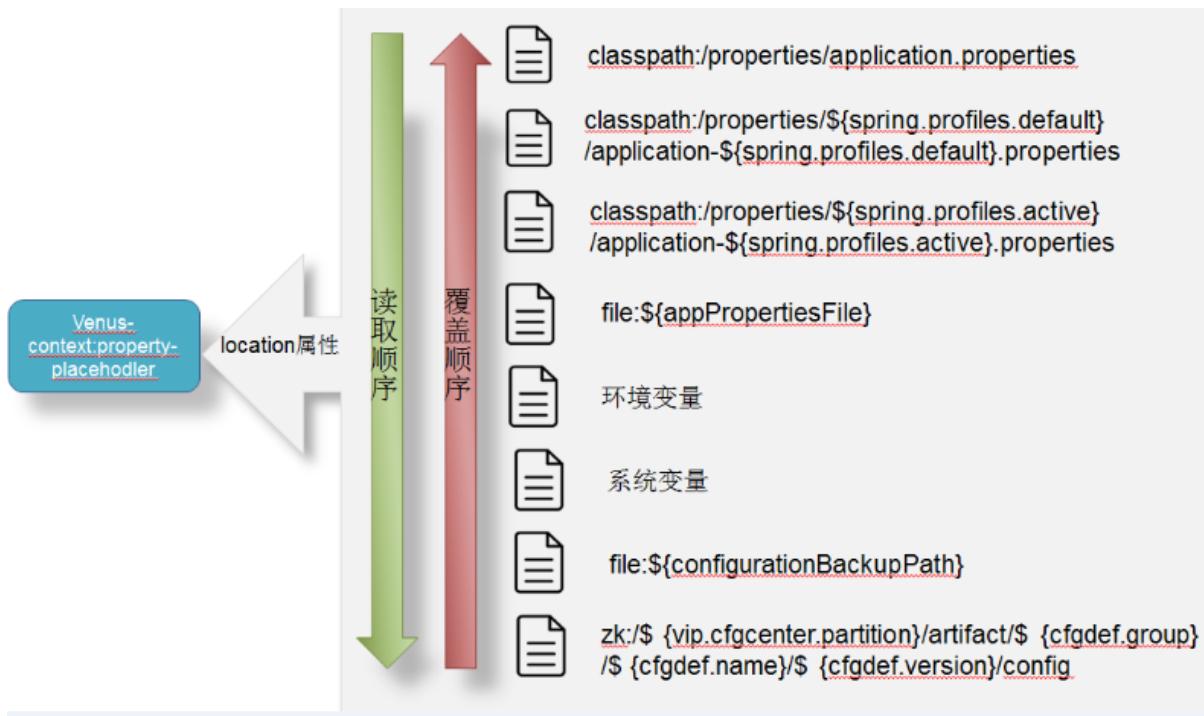
如果想连别的zk, 用户需要在配置中心的partition/bootstrap中配置要连接zk的IP/port, 配置如下:

bootstrap中增加如下代码:

```
/resources/app/config = 10.101.18.151:2181,10.101.18.152:2181
```

无论用户自定义的zk是否是连接配置中心的zk, 我们都要配置配置中心的zk即 VIP_CFGCENTER_ZK_CONNECTION

location默认配置逻辑(use-cfgcenter=true, properties文件中配置configurationBackupPath的情况下)



Note

环境变量是全局的，比如`export`这种方式

系统变量是针对某个应用的，通过`java`的`-D`参数设置

handlers:

```
#####
#handlers#bean#
```

所有从配置中心节点取到的数据都会被监听，一旦数据发生改变则实时通知应用程序，应用程序自定义响应事件。

配置示例：

```
<venus-context:property-placeholder ignore-resource-not-found="true">
    <venus-context:property-handlers>
        <bean class="xxx.xxx.xxx.xxx" name="TestHanlder"/>
    </venus-context:property-handlers>
</venus-context:property-placeholder>
```

自定义Handler类代码示例：

```
@PropertyHandler
public class TestHandler implements PropertyChangedHandler {

    Logger logger = LoggerFactory.getLogger(TestHandler.class);

    @Override
    public void execute(Properties oldProperties, Properties newProperties) {
        logger.info("property changed handler");
    }
}
```

manager-name:

用来定义PropertyManager Bean. PropertyManager Bean里的Property数据是会在配置中心变更数据后自动更新。

如果不配置，则使用venus框架下定义的PropertyManager

使用PropertyManager代码示例:

```
@Service
public class TestServiceImpl implements TestService {

    Logger logger = LoggerFactory.getLogger(TestServiceImpl.class);

    @Autowired
    private PropertyManager propertyManager;

    @Override
    public void doSomeThing() {
        if(propertyManager.getProperty("somekey").equals("somevalue")){
            ...
        }
    }
}
```

11. #使用venus-data进行分库/读写分离/分表

针对关系数据库，venus提供两种分库/分表/读写分离的实现方式：一种是基于拦截器方式即venus-data，一种是在jdbc底层实现的方式即venus-jdbc。

venus-data只支持Mybatis作为数据操作层框架，本章节详细讲解如何基于拦截器方式即venus-data进行分库/分表/读写分离操作

无论是分表还是分库，必须要事先在数据库中建好database和table。

Venus data demo的相关代码存放在公司gitlab中，如果需要看代码，请执行以下命令：

```
git@gitlab.tools.vipshop.com:venus-samples/venus-demo.git
git checkout venusdata_1.3.1
```

Note

本章节中提到的spring配置一般是指applicationContext.xml配置文件。

如果另写.xml配置文件，需要修改web.xml的classpath的值才能读取到。

11. 1#添加依赖

在gradle添加依赖：

在build.gradle文件中添加：

```
compile('com.vip.venus:venus-data:1.3.8')
```

Note

利用venus-codegen生成的gradle项目依赖已添加

在maven添加依赖：

在pom.xml文件中添加：

```
<dependency>
  <groupId>com.vip.venus</groupId>
  <artifactId>venus-data</artifactId>
  <version>1.3.8</version>
  <scope>compile</scope>
</dependency>
```

11. 2#spring引入venus-datasource

利用venus codegen生成的项目中的spring配置文件已引入venus-datasource，不需要手动引入

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  ...
  xmlns:venus-datasource="http://venus.vip.vip.com/schema/datasource"
  ...
  xsi:schemaLocation="
  ...
  http://venus.vip.vip.com/schema/datasource
  http://venus.vip.vip.com/schema/datasource/venus-datasource-1.3.0.xsd
  ...
  ">
```

11. 3#spring添加数据库操作相关bean

目前只支持MyBatis作为数据操作层框架，因此需要在Spring配置文件中配置MyBatis会话工厂和Mapper自动扫描器。

默认情况下目标数据库是MySQL；数据库连接池是C3P0；依赖的Spring事务管理器是transactionManager；

依赖的MyBatis会话工厂是myBatisSqlSessionFactory；

venus codegen自动生成的项目中已自动添加这些bean，无需手动添加

```
<!-- ##### -->
<bean id="transactionManager" class="com.vip.venus.data.datasource.SmartDataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<!-- MyBatis### -->
<bean id="myBatisSqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mapperLocations" value="classpath*:mapper/*Mapper.xml" /> ①
</bean>

<!-- Mapper### -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.vip.venus.data.mybatis.repository" /> ②
    <property name="annotationClass" value="org.springframework.stereotype.Repository" />
</bean>

<!-- <matrix-datasource>#####MyBatis#####Mapper##### -->
<venus-datasource:matrix-datasource matrix-name="dataSource">
...
</venus-datasource:matrix-datasource>
```

① Mapper所在的包路径

② repository所在的包路径

Warning

<matrix-datasource>必须配置在MyBatis会话工厂和Mapper自动扫描器之后；

事务管理器类：请使用com.vip.venus.data.datasource.SmartDataSourceTransactionManager，

不要使用org.springframework.jdbc.datasource.DataSourceTransactionManager.

Note

如果MyBatis会话工厂的id为myBatisSqlSessionFactory，则无需在<matrix-name>中手工指定，反之就需要手工指定。

如果事务管理器的id为transactionManager，则无需在<matrix-name>中手工指定，反之就需要手工指定

11. 4#properties文件配置

有关数据库的所有信息：数据库类型，数据库定义（包括host, port, username, password等信息），分库/读写分离/分表等信息都是定义在properties文件中的

根据当前的运行环境选择配置不同的properties文件，比如：

如果当前的环境变量是development，那么就配置webapp项目中的src/main/resources/properties/development/application-development.properties

配置示例：

```
#json#####json##### http://www.bejson.com/jsoneditoronline/ #

#####
/resource/RDBMS/matrix.dataSource={"matrixName": "dataSource","state": "online","type": "MySQL",\
"groups": [\n    {"groupName": "rwds1","state": "online","loadBalance": "roundRobin",\n        "atoms": [{"atomName": "write01","host": "10.101.18.191","port": "3306","username":\n            "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": true,"state":\n            "online"},{\n            "atomName": "read101","host": "10.101.18.209","port": "3306","username":\n            "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": false,"weight":\n            1,"state": "online"},{\n            "atomName": "read102","host": "10.101.18.163","port": "3306","username":\n            "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": false,"weight":\n            1,"state": "online"}],\n    {"groupName": "rwds2","state": "online","loadBalance": "roundRobin",\n        "atoms": [{"atomName": "write02","host": "10.101.18.211","port": "3306","username":\n            "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": true,"state":\n            "online"},{\n            "atomName": "read201","host": "10.101.18.208","port": "3306","username":\n            "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": false,"weight":\n            1,"state": "online"},{\n            "atomName": "read202","host": "10.101.18.177","port": "3306","username":\n            "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": false,"weight":\n            1,"state": "online"}]\n}]\n}
```

本配置是配置了个两组数据库rwds1和rwds2，每组数据库中又配置了一个写数据库和两个读数据库

以下是对这个配置的解释：

```
matrixName# #spring##matrix-name##\n\nstate#online#####offline#####\n\ntype#####\n\n\ngroups#####rwds1#rwds2#rwds1#####rwds2#####\n\ngroupName#####\n\natomName#####host#port#username#password###\n\nisMaster:true#####false#####\n\nparam##### & ####\n##"param": "characterEncoding=UTF-8&zeroDateTimeBehavior=convertToNull&defaultBatchValue=1000"
```

11. 5#分库操作

应用背景：

分库操作有两种应用场景： 1 不同的业务数据写入不同的数据库，不同的数据库中表结构可能不同：这种是有两套service, service-api和controller

2 数据量过大时，为了缓解数据库压力，把相同的业务数据写入不同的数据库，不同的数据库中表结构必须相同：这种是共用一套service, service-api和controller

这里讲的分库操作是指第二种应用

Note

进行分库操作前确保spring已经引入venus-datasource，已经添加依赖和数据库操作相关bean。

利用venus codegen生成的项目代码都已经自动添加，无需手动添加。

具体请参考Section# 11.1，“添加依赖”，Section# 11.2，“spring引入venus-datasource”，Section#11.3，“spring添加数据库操作相关bean”。

spring配置

分库配置示例：

```
<venus-datasource:matrix-datasource matrix-name="dataSource" > ①
    <venus-datasource:pool-configs pool-type="dbcp">
        <venus-datasource:pool-config atom-names="write01,read*" > ②
            <venus-datasource:property name="maxIdle" value="30"/>
        </venus-datasource:pool-config>
    </venus-datasource:pool-configs>
    <venus-datasource:repository-sharding strategies-
package="com.vip.venus.userservice.strategy.repository"/> ③
    .....
</venus-datasource:matrix-datasource>
```

- ① matrix-name必须与properties文件中matrixName一致
- ② 配置数据库的连接池属性，数据库支持通配，即这个连接池的属性支持所有以read开头的数据库和write01数据库
- ③ strategies-package:分库规则类的包名, repository-sharding其他属性的详细配置请参考the section called “venus-datasource配置详细说明”。注意：分表规则类和分库规则类必须放在不同的包下

properties配置

properties配置请参考Section#11.4，“properties文件配置”

分库规则

提供抽象类RepositoryShardingStrategy供用户自定义分库规则（如下代码），框架通过拦截器的方式实现分库路由，

因此需要在spring配置文件的venus-datasource: repository-sharding指定分库规则的包名，框架自动扫描该包下的分库规则

代码示例：

```
// #####ServiceImpl#####
@Strategy( "venus_departdatabase" ) ①
public class DepartRepositoryShardingStrategy extends RepositoryShardingStrategy {

    @Override
    public String getReadWriteDataSource(Object obj) {
        // #####SPEL#####Service#####
        Integer value = (Integer) obj; ②
        // #####value#(10000,+-#)##rwds2##<matrix-datasource>#####
        if (value != null && value > 10000) {
            return "rwds2"; ③
        }
        // ##(-#,10000]##rwds1
        return "rwds1";
    }
}
```

- ① 策略名，供***ServiceImpl类中的方法使用
- ② 分库参数，需要转换成参数具体的数据类型即****Model.java中的类型
- ③ 返回值和properties文件中读写数据库名对应

服务层配置

服务层即***ServiceImpl类根据实际场景配置分库策略以及事务控制。

代码示例：

```
@Service
public class StudentServiceImpl implements StudentService {

    .....

    @Transactional
    @Override
    // #####"venus_departdatabase"#####studentModel.id####SPEL#####depart#####
    @RepositorySharding(strategy = "venus_departdatabase", key = "#studentModel.id") ①
    public int create(StudentModel studentModel) {
        return studentRepo.insert(beanMapper.map(studentModel, Student.class));
    }

    @Transactional
    @Override
    @RepositorySharding(strategy = "venus_departdatabase", key = "#id") ②
    public int deleteByPrimaryKey(Integer id) {
        return studentRepo.deleteByPrimaryKey(id);
    }

    .....
}
```

- ① 分库策略名即strategy必须与the section called “分库规则”类中的策略名即@Strategy的值一致
- ② 代码中只是以creat/deleteByPrimaryKey方法为例，应该在所有操作数据库的方法上都加注解

嵌套事务处理遵循规则

transaction behavior	处理方式
REQUIRES_NEW	允许与外层分库策略不一致
NESTED	与外层分库策略不一致，异常；否则，采用当前策略
REQUIRED	与外层分库策略不一致，异常；否则，采用当前策略
NOT_SUPPORTED	与外层分库策略不一致，异常；否则，采用当前策略
MANDATORY	与外层分库策略不一致，异常；否则，采用当前策略
SUPPORTS	与外层分库策略不一致，异常；否则，采用当前策略
NEVER	允许与外层分库策略不一致

11.6#读写分离操作

应用背景：

在实际的数据库操作中，读操作一般都远远多于写操作，读写分离能大大提高数据库的效率

Note

进行读写分离操作前确保spring已经引入venus-datasource，已经添加依赖和数据库操作相关bean。

利用venus codegen生成的项目代码都已经自动添加，无需手动添加。

具体请参考Section# 11.1，“添加依赖”，Section# 11.2，“spring引入venus-datasource”，Section#11.3，“spring添加数据库操作相关bean”。

properties配置

properties配置请参考Section#11.4，“properties文件配置”

spring配置

在applicationContext.xml中要添加如下代码：

```
<venus-datasource:matrix-datasource matrix-name="dataSource">
    .....
    <venus-datasource:repository-sharding>
        <venus-datasource:beanName value="*Service" /> ①
        <venus-datasource:beanName value="*ServiceImpl" />
    </venus-datasource:repository-sharding>
    <venus-datasource:repository-sharding/> ②
    .....
</venus-datasource:matrix-datasource>
```

① 待拦截的bean

② 必须配置

服务层配置

服务层即***ServiceImpl根据实际场景配置分库策略以及事务控制。

代码示例：

```
@Service
public class StudentServiceImpl implements StudentService {

    .....

    @Transactional
    @Override
    // #####
    @ReadWrite(type = ReadWriteType.WRITE) ①
    public int create(StudentModel studentModel) {
        return studentRepo.insert(beanMapper.map(studentModel, Student.class));
    }

    @Transactional(readOnly = true)
    @Override
    // #####
    @ReadWrite(type = ReadWriteType.READ) ②
}
```

```

public StudentModel findByPrimaryKey(Integer id) {
    Student student = studentRepo.selectByPrimaryKey(id);
    return beanMapper.map(student, StudentModel.class);
}

.....
}

```

- ① 涉及到数据库的增/删/改操作的方法的type是ReadWriteType.WRITE
- ② 涉及到数据库的查操作的方法的type是ReadWriteType.READ

11.7#分表操作

应用背景:

当数据量过大时，可以把数据写入不同的数据表，从而提高数据库的读写效率.

Note

进行分表操作前确保spring已经引入venus-datasource，已经添加依赖和数据库操作相关bean。

利用venus codegen生成的项目代码都已经自动添加，无需手动添加。

具体请参考Section# 11.2，“spring引入venus-datasource”，Section# 11.1，“添加依赖”，Section#11.3，“spring添加数据库操作相关bean”。

spring配置

配置示例:

```

<venus-datasource:matrix-datasource matrix-name="dataSource"> ①
    <venus-datasource:pool-configs pool-type="c3p0">
        <venus-datasource:pool-config atom-names="qiyu*"> ②
            <venus-datasource:property name="initialPoolSize" value="5"/>
        </venus-datasource:pool-config>
    </venus-datasource:pool-configs>

    <venus-datasource:table-sharding strategies-package="com.vip.venus.userservice.strategy.table"> ③
        .....
    </venus-datasource:table-sharding>
</venus-datasource:matrix-datasource>

```

- ① matrix-name必须与properties文件中matrixName一致
- ② 配置数据库的连接池属性，数据库支持通配，即这个连接池的属性支持所有以read开头的数据库和write01数据库
- ③ strategies-package: 分表规则类的包名, table-sharding其他属性的详细配置请参考the section called “venus-datasource配置详细说明”. 注意：分表规则类和分库规则类必须放在不同的包下

Note

venus-datasource详细配置请参考the section called “venus-datasource配置详细说明”

properties配置

properties配置请参考Section#11.4，“properties文件配置”

分表规则

提供抽象类TableShardingStrategy供用户自定义分表规则（如下代码），框架通过拦截器的方式实现分表路由，

因此需要在spring配置文件的venus-datasource: table-sharding指定分表规则的包名，框架自动扫描该包下的分表规则

代码示例：

```
@Strategy("venus_departtable") ①
public class DepartTableShardingStrategy extends TableShardingStrategy {

    @Override
    public String getTargetSql(String sql) {
        // #####SPEL#####Repository#####
        Integer value = (Integer) getShardingParameterValue(); ②
        // #####2#####value##2##student0#####value##3##student1#####
        String realTableName = "student" + (value % 2); ③
        // ####MyBatis Mapper##SQL#####
        return Utils.getShardingTableName("venus_departtable", realTableName, sql); ④
    }
}
```

- ① 策略名，供***Repository类中的方法使用
- ② 分表参数，需要转换成分表参数具体的数据类型，即***Model.java中的类型
- ③ 表前缀很重要（即代码中的“student”）：根据分表映射规则得到的具体数据表名，此处的分表映射规则是按照2取模，如果value的值为2，则操作student0这个表，如果value的值为3，则操作student1这个表
- ④ “venus_departtable”虚拟表名，虚拟表名必须要和策略名一致

MyBatis Mapper配置

分表场景下，需要把虚拟表名\$[venus_departtable]\$替换成具体表名。框架会自动扫描虚拟表名并自动映射为实际分表名

配置示例：

```
.....
<insert id="insert" parameterType="com.vip.venus.userservice.entity.Student" >
    insert into ${venus_departtable}$ (id, name) ①
    values (#{student.id,jdbcType=INTEGER}, #{student.name,jdbcType=VARCHAR}) ②
</insert>
<delete id="deleteByPrimaryKey" parameterType="java.lang.Integer" >
    delete from ${venus_departtable}$
    where id = #{id,jdbcType=INTEGER}
</delete>
.....
```

- ① \${venus_departtable}\$：虚拟表名，必须要和策略名一致
- ② 如果parameterType是个实体类，参数需要根据StudentRepository对应方法中的参数名写成参数名.属性，比如代码中的student.id和student.name

数据操作层配置

数据操作层需要根据实际场景配置分表策略。

对于venus codegen生成的项目代码，是在***Repository类中配置分表策略

代码示例:

```
@Repository
public interface StudentRepository {
    @TableSharding(strategy = "venus_departtable", key = "#student.id")
    int insertSelective(@Param("student") Student student);

    @TableSharding(strategy = "venus_departtable", key = "#id")
    Student selectByPrimaryKey(@Param("id") Integer id);
    .....
}
```

Warning

关于@Param, 类中一定要import import org.apache.ibatis.annotations.Param

不能import org.springframework.data.repository.query.Param

12. #使用venus-jdbc进行分库/读写分离/分表

针对关系数据库，venus提供两种分库/分表/读写分离的实现方式：一种是基于拦截器方式即venus-data，一种是在jdbc底层实现的方式，即venus-jdbc。

本章节详细讲解如何基于venus-jdbc进行分库/分表/读写分离操作.

无论是分表还是分库，必须要事先在数据库中建好database和table。

Venus jdbc demo的相关代码存放在公司gitlab中，如果需要看代码，请执行以下命令：

```
git@gitlab.tools.vipshop.com:venus-samples/venus-demo.git
git checkout venusjdbc_1.3.1
```

12. 1#添加依赖

在gradle添加依赖:

在build.gradle文件中添加:

```
compile('com.vip.venus:venus-jdbc:1.3.8')
```

在maven添加依赖:

在pom.xml文件中添加:

```
<dependency>
  <groupId>com.vip.venus</groupId>
  <artifactId>venus-jdbc</artifactId>
  <version>1.3.8</version>
  <scope>compile</scope>
</dependency>
```

12. 2#spring引入venus-jdbc

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  ...
  xmlns:venus-jdbc="http://venus.vip.vip.com/schema/jdbc"
  ...
  xsi:schemaLocation="
    ...
    http://venus.vip.vip.com/schema/jdbc
    http://venus.vip.vip.com/schema/jdbc/venus-jdbc-1.3.0.xsd
  ...
  ">
```

12. 3#spring添加数据库操作相关bean

因为venus-jdbc是在jdbc底层操作数据库.

默认情况下数据操作层框架是Mybatis，目标数据库是MySQL；数据库连接池是C3P0；依赖的Spring事务管理器是transactionManager；

```
<!-- ##### -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource" />
</bean>
```

```

<tx:annotation-driven transaction-manager="transactionManager"/>

<!-- Mapper##### -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.vip.venus.jdbc.mybatis.repository" /> ①
    <property name="sqlSessionFactoryBeanName" value="mysqlSessionFactory" />
</bean>

<bean id="mysqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="typeAliasesPackage" value="com.vip.venus.jdbc.mybatis.entity" /> ②
    <property name="mapperLocations" value="classpath*:mapper/*Mapper.xml" /> ③
</bean>

<venus-jdbc:allinone-datasource matrix-name="dataSource">
    ...
</venus-jdbc:allinone-datasource>

```

- ① repository所在的包路径
- ② entity所在的包路径
- ③ Mapper文件所在的包路径

12. 4#spring配置

```

#####
<venus-jdbc:allinone-datasource matrix-name="dataSource">
    <venus-jdbc:pool-configs pool-type="c3p0"> ①
        <venus-jdbc:pool-config atom-names="qiyu*"/> ②
    </venus-jdbc:pool-configs>
</venus-jdbc:allinone-datasource>

```

- ① 数据连接池类型，只能是druid/dbcp/c3p0中的一种
- ② 配置连接池属性，支持数据库通配，以qiyu开头的包括qiyu数据库用c3p0数据连接池的配置，也可自定义数据连接池的属性，具体配置请参考the section called “venus-jdbc配置详细说明”

12. 5#properties文件配置

有关数据库的所有信息：数据库类型，数据库定义(包括host, port, username, password等信息)，分库/读写分离/分表等信息以及分库规则/分表规则都是定义在properties文件中的

根据当前的运行环境选择配置不同的properties文件，比如：

如果当前的环境变量是development，那么就配置webapp项目中的src/main/resources/properties/development/application-development.properties

配置示例：

```

#####json#####json#####json#####json##### http://www.bejson.com/jsoneditoronline/ #

#####
/resource/RDBMS/matrix/matrixDataSource={"matrixName": "dataSource","state": "online","type": "MySQL","groups": [\n{"groupName": "rwds1","state": "online","loadBalance": "roundRobin",\n"atoms": [{"atomName": "write01","host": "10.101.18.191","port": "3306","username": "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": true,"state": "online"},\n{"atomName": "read101","host": "10.101.18.209","port": "3306","username": "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": \"useDynamicCharsetInfo=false\",\"isMaster": false,"weight": 1,\"state": "online"}],\n"sharding": {"shardingType": "range",\n"shardingKey": "id",\n"shardingValue": "1000",\n"shardingCount": 2},\n"shardingRules": [{"ruleName": "rule1", "shardingKey": "id", "shardingValue": "1000", "shardingCount": 2}],\n"shardingTable": "t_order",\n"shardingStrategy": "roundRobin",\n"shardingColumn": "id",\n"shardingValue": "1000",\n"shardingCount": 2},\n"state": "online"}]

```

```
{
  "atomName": "read102", "host": "10.101.18.163", "port": "3306", "username": "vipuser", "password": "xR54PUU8GWia", "dbName": "db_sharding", "param": "useDynamicCharsetInfo=false", "isMaster": false, "weight": 1, "state": "online"}], \
  {"groupName": "rwds2", "state": "online", "loadBalance": "roundRobin", \
  "atoms": [{ "atomName": "write02", "host": "10.101.18.211", "port": "3306", "username": "vipuser", "password": "xR54PUU8GWia", "dbName": "db_sharding", "param": "", "isMaster": true, "state": "online"}, \
    {"atomName": "read201", "host": "10.101.18.208", "port": "3306", "username": "vipuser", "password": "xR54PUU8GWia", "dbName": "db_sharding", "param": "useDynamicCharsetInfo=false", "isMaster": false, "weight": 1, "state": "online"}, \
    {"atomName": "read202", "host": "10.101.18.177", "port": "3306", "username": "vipuser", "password": "xR54PUU8GWia", "dbName": "db_sharding", "param": "useDynamicCharsetInfo=false", "isMaster": false, "weight": 1, "state": "online"}], \
  "rules": [{"groupShardRule": "((#id# as int).intValue() % 4).intdiv(2)", "groupIndex": "rwds1,rwds2", "tableNames": "*", "tableSuffix": "_0,_1", "tableShardRule": "(#id# as int).intValue() % 2"}]}]
```

本配置是配置了个两组数据库rwds1和rwds2，每组数据库中又配置了一个写数据库和两个读数据库

以下是对这个配置的解释：

- **matrixName:** 必须与spring中allinone-datasource中的matrix-name一致
- **state:** online表示上线服务，offline表示下线服务
- **type:** 数据库类型
- **groups:** 里面可以配置多组数据库。本配置是配置了两组数据库rwds1和rwds2。rwds1有一个写数据库和两个读数据库，rwds2有一个写数据库和两个读数据库
- **groupName:** 定义每组数据库。每组数据库中必须有一个也只能有一个写数据库，可以配置多个或者不配置读数据库。
- **atomName:** 定义每个数据库，包括host, port, username, password等信息
- **param:** 配置数据库的一些相关参数，多个参数用 & 隔开。useDynamicCharsetInfo=false: 关闭数据库的动态连接，因为我们汇集不同数据库/表的查询结果时要使用CachedRowSet，

如果表的返回结果集中有string时，CachedRowSet会动态连接数据库获取字符集类型，为了提高效率，我们通过设置参数useDynamicCharsetInfo=false关闭这个动态连接

- **isMaster:** true表示是写数据库，false表示是读数据库
- **rules:**

```
groupShardRule:#####groovy####id#####id#
groupIndex#####groupShardRule#####0##
#####
##id=15->15%4=3->3.intdiv(2)=1##groupShardRule=1,groupIndex#####"rwds2",#####"rwds2"##
##

tableNames: #####
tableSuffix#####tableShardRule,tableNames#####0##
tableShardRule#####groovy####id#####id#
#####
##id=15->15%2=1 #tableShardRule=1, tableSuffix##_##_1,#####$tableNames_1
```

13. #使用venus-cache进行缓存操作

在数据库的读操作中，很多读操作都是读取的相同的数据，如果把这部分相同的数据放入到缓存时，每次从缓存读取，这样即提高了读的效率又缓解了数据库的压力。

Venus Cache 不仅是对Spring Cache的封装和集成，而且还在Spring Cache的基础上进行增强，提供了一整套Venus自己的注解，同时无缝支持redis, memcached，统一上层API，使得缓存服务简单易用。另外通过配置中心，让开发者方便的配置缓存集群。

目前Venus Cache支持多种cache, 包括Redis, Memcached。

本章节所讲的Spring配置均是在applicationContext.xml文件中配置。

如果另写.xml配置文件，需要修改web.xml的classpath的值才能读取到。

Venus cache demo的相关代码存放在公司gitlab中，如果需要看代码，请执行以下命令：

```
git@gitlab.tools.vipshop.com:venus-samples/venus-demo.git
git checkout venuscache_1.3.1
```

13. 1#spring引入venus-cache

```
<beans xmlns:venus-cache="http://venus.vip.vip.com/schema/cache"
       xsi:schemaLocation="http://venus.vip.vip.com/schema/cache http://venus.vip.vip.com/schema/cache/
venus-cache-1.4.0.xsd">
```

Note

关于venus-codegen生成的项目，Spring配置已引入venus-cache，无需手动添加。

13. 2#venus-cache配置和使用

venus-cache配置示例：venus-cache的详细配置请参考Chapter#26, Venus Cache

```
<venus-cache:cache-manager name="cacheManager">
    <venus-cache:caches>
        <venus-cache:cache-cluster name="accountCache" template-name="accountCacheTemplate" cluster-
name="account-redis-cluster" type="redis" /> ①
    </venus-cache:caches>
    <venus-cache:caches>
        <venus-cache:cache-cluster name="userCache" template-name="userCacheTemplate" cluster-name="user-
memcached-cluster" type="memcached" /> ②
    </venus-cache:caches>
</venus-cache:cache-manager>
```

①② 定义一个cache cluster标签，name属性将会被用于 @Cacheable 的 value 来映射对应的cache，type决定使用哪种类型的缓存（支持memcached和redis两种类型，必配），cluster-name用于映射配置中心缓存的配置名称（必配），template-name用于通过该名字注入缓存的模板（必配）；

Warning

如果从本地读取配置文件，那么properties文件中的名称必须与cluster-name保持一致

venus-cache使用：

在***serviceimpl.java中可以使用venus的全套注解 @Caching @Cacheable @CacheEvict @CachePut 注解，注解的value的值必须和缓存配置的名称相对应，并支持在注解上配置过期时间，时间单位为秒。（推荐） 同时也可直接使用spring cache的全套注解 @Caching @Cacheable @CacheEvict @CachePut 注解，注解的value的值必须和缓存配置的名称相对应。（不建议使用）

如果希望使用Cache注解的使用方式，要确保方法的返回值是实现JDK的Serializable接口。

Note

codegen生成的simpleService/webapp项目中的serviceimpl中的方法返回值model都已经序列化了，但是osp项目中的model（api工程下）需要手动实现序列化

@Caching：包括 @Cacheable @CacheEvict @CachePut 的组合操作形式

@Cacheable：首次操作会调用真实方法对其结果进行缓存，待下次操作时直接从缓存中查询，不会再调用真实方法

@CachePut：每次操作都会调用真实方法，同时更新缓存结果

@CacheEvict：每次操作都会清空缓存

具体spring cache的相关知识，可参考 [Spring Cache](#)

```
@Cacheable(value="accountCache") ❶
public TeacherModel findByPrimaryKey(Long id) {
    Teacher teacher = teacherRepo.selectByPrimaryKey(id);
    return beanMapper.map(teacher, TeacherModel.class);
}

@CachePut(value="accountCache",key="#teacherModel.id")
public int updateByPrimaryKey(TeacherModel teacherModel) {
    return teacherRepo.updateByPrimaryKey(beanMapper.map(teacherModel, Teacher.class));
}
```

① 注解中value的值必须与spring中cache配置中name保持一致

venus cache的使用方式

```
@Cacheable(value="accountCache", expiration="100") ❶
public TeacherModel findByPrimaryKey(Long id) {
    Teacher teacher = teacherRepo.selectByPrimaryKey(id);
    return beanMapper.map(teacher, TeacherModel.class);
}

@CachePut(value="accountCache",key="#teacherModel.id", expiration="100")
public int updateByPrimaryKey(TeacherModel teacherModel) {
    return teacherRepo.updateByPrimaryKey(beanMapper.map(teacherModel, Teacher.class));
}
```

① 注解中value的值必须与spring中cache配置中name保持一致；过期时间expiration设置为XX秒

13. 3#cache cluster配置

cache cluster有两种配置方式：1 本地配置，读取properties文件；2 配置中心配置。

应用程序在数据库的读操作时会根据cache cluster配置中的信息去缓存读取数据

本地配置：

根据当前运行环境，在配置webapp项目下的application-development.properties文件添加如下代码：

```
# venus cache dummy zk data

/resource/Cache/Redis/order-redis-cluster={"loadBalance":"RoundRobin","name":"order-
redis-cluster","nodes":[{"host":"127.0.0.1","logicName":"redis-
node-01","port":"6379","weight":"1","state":"online"}],"proxy":true,"sharding":"","type":"Redis","support-
type":""}
```

Warning

/resource/Cache/Redis/order-redis-cluster中的“order-redis-cluster”是与spring配置中的cluster-name中的值一样的

配置中心配置：

首先在配置中心配置好redis，然后在环境变量中或者是系统变量中配置zkclient，配置如下：

```
VIP_CFGCENTER_ZK_CONNECTION=127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183 ①
```

- ① 配置中心部署在zookeeper上，这里配置的是zookeeper客户端的ip

14. #使用配置中心配置channel和queue

14. 1#配置要使用的配置中心地址

在环境变量中或者是系统变量中配置VIP_CFGCENTER_ZK_CONNECTION， 配置如下：

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.94:2181,10.101.18.94:2182,10.101.18.94:2183 ⓘ
```

14. 2#然后在配置中心创建channel， 选择相应的集群

The screenshot shows a configuration interface for creating a channel. At the top, there is a title bar with the text "创建频道". Below it, the form fields are as follows:

- 名称 :** A text input field containing "channel" with a red asterisk indicating it is required.
- 所属域 :** A dropdown menu showing "物流-VIP/WPH56" and a text input field showing "live.vip.vip.com" with a dropdown arrow.
- 描述 :** A large text area for description, currently empty.
- 集群 :** A dropdown menu showing "venus.mq.rabbitmqtest (rabbitmq)" and a "高级" (Advanced) button with the value "OFF".
- 操作按钮 :** Two buttons at the bottom left: "创建" (Create) and "取消" (Cancel).

14. 3#创建queue，选择相应的集群

The screenshot shows the 'Create Queue' configuration page. Key fields include:

- 名称:** queue (必填)
- 所属域:** 物流-VIP/WPH 56 (下拉) | live.vip.vip.com (下拉) | **添加**
- 集群:** venus.mq.rabbitmqtest (下拉)
- 参数:** [empty] = [empty]
 - Vhost | Username | Password | Message TTL
 - Auto expire | Max length | Max length bytes
 - Dead letter exchange | Dead letter routing key
 - Maximum priority
- 描述:** [empty]
- 绑定:** [empty] | **添加**
 - 并
 - 频道 ▾
 - 路由 ▾

绑定列表为空

Buttons at the bottom: **创建** (blue) | **取消**

14. 4#如果需要设置routingkey，在下面绑定routingkey

添加绑定

The screenshot shows the 'Add Binding' configuration page. Key fields include:

- 频道:** channel.venus.mq.rabbitmqtest.01 (下拉)
- 路由:** * (必填)

Buttons at the bottom: **添加** (blue) | **取消**

15. #使用venus-test进行单元测试

venus框架下，将单元测试变得更类似于集成测试的方式，通过对service层相关接口的直接调用，同时测试了service层 代码和repository层代码。

venus-test完全兼容spring-test单元测试方式；支持完全隔离的DB Testing；支持class级别和method级别的initDb和cleanDb操作。

venus codegen生成的项目代码中已经包含了一些Junits用例。

在1.3.0版本中，新增支持h2数据库的功能，并且是codegen工具生成项目中junit的默认形式。Junit中的H2数据库只支持分库分表操作，不支持读写分离（因为不支持主从同步更新）

在1.3.0版本中，新增了.sql文件的注解形式：---!begin atom:db1_master;group:group-01，并且以---!end结束，在该注解内的所有sql只在注解所列出的库上执行；否则在全库执行

Venus test demo的相关代码存放在公司gitlab中，如果需要看代码，请执行以下命令：

```
git@gitlab.tools.vipshop.com:venus-samples/venus-demo.git
git checkout venustest_1.3.1
```

15. 1#添加依赖

gradle中添加依赖：

```
testCompile('com.vip.venus:venus-test:1.3.8')
```

Note

使用venus codegen生成的gradle项目依赖已添加，无需手动添加

maven中添加依赖：

```
<dependency>
<groupId>com.vip.venus</groupId>
<artifactId>venus-test</artifactId>
<version>1.3.8</version>
<scope>compile</scope>
</dependency>
```

15. 2#配置文件说明

- src/test/resources/applicationContext-test.xml：初始化spring相关bean
- src/test/resources/db.properties：数据库信息

h2数据库：codegen工具生成项目中的默认配置

```
db.url=jdbc:h2:mem:
db.password=xR54PUU8GWia
db.username=vipuser
db.driver=org.h2.Driver
db.param=DB_CLOSE_DELAY=10;MODE=MySQL;
```

mysql数据库：其配置都以json格式定义在application-test.properties文件中了

- src/test/resources/db.sql：定义DB初始化需要的SQL

- `src/test/resources/properties/test/application-test.properties` : 数据库相关信息，包括 ip, port, name, password 等

15. 3# 测试类说明

venus test 框架有两个 test 基类： `BaseDbTest` 和 `BaseTest`

- 测试类继承 `BaseDbTest`

支持完全隔离的DB Testing

h2数据库： Junit 运行时会先根据 `applicationContext-test.xml`, `application-test.properties`, `db.properties` 在内存中创建 h2 数据库, 然后执行 `db.sql` 文件创建表进行表操作

Mysql数据库： Junit 运行时会先根据 `applicationContext-test.xml` 和 `application-test.properties` 创建临时数据库（数据库名字是： `dbname+随机生成的数字`），然后执行 `db.sql` 文件创建表进行表操作

- 测试类继承 `BaseTest`:

不会动态创建 h2 数据库和临时的 mysql 数据库， Junit 运行时操作的数据库即 `applicationContext-test.xml` 和 `application-test.properties` 中配置的数据库

Warning

如果继承 `BaseTest`，那么 sql 语句中用到的数据库必须要提前建好，代码不会在运行时自动创建

Note

如果进行分库分表读写分离操作的 Junit，需要配置 `applicationContext-test.xml` 和 `application-test.properties`，具体请参考 Chapter#11，使用 `venus-data` 进行分库/读写分离/分表，还要修改 `db.sql` 文件，在里面增加建表的代码

15. 4# 注解说明

- java 代码中的注解

venus test 下有丰富的注解，开发人员可利用这些注解灵活开发测试用例

```
@PrePostSql(preSqlFile="class_pre.sql",postSqlFile="class_post.sql") ❶
public class SomeTest extends BaseDbTest{

    @Autowired
    private TestService testService; ❷

    @Test
    @PrePostSql(preSqlFile="method_pre.sql",postSqlFile="method_post.sql") ❸
    public void test1(){
        testService.test1();
    }
}
```

❶ 定义测试类初始和结束需要执行的sql文件(可选配置)

❷ 定义需要注入的服务

❸ 定义测试方法执行前后需要执行的sql文件(可选配置)

- sql 脚本中的注解

```
---!begin atom:qiyu;group:rwdsl ❶

CREATE TABLE student (
    id int(11) NOT NULL DEFAULT '0',
    name varchar(255) DEFAULT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

---!end

CREATE TABLE teacher (
    id int(11) NOT NULL DEFAULT '0',
    name varchar(255) DEFAULT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- ❶ group和atom对应properties文件中定义的数据库信息，示例中是注解内的脚本只在rwdsl上的qiyu这个数据库执行。不加注解的sql脚本时全库执行

16. #使用venus-security进行认证和授权

Venus Security 是对Spring security的封装和集成，主要提供了易用的认证和授权服务，引入realm来支持安全资源配置的多途径多方式加载。

另外Venus security采用了插件机制，易于集成外部系统，比如venus-plugin-security-authsys.

目前venus security只支持CAS认证

Venus security demo的相关代码存放在公司gitlab中，如果需要看代码，请执行以下命令：

```
git@gitlab.tools.vipshop.com:venus-samples/venus-demo.git
git checkout venussecurity_1.3.1
```

16. 1#添加依赖

gradle中添加依赖：

```
compile('com.vip.venus:venus-security:1.3.8')
//####OA, #####
compile('com.vip.venus.plugin:oa-plugin:0.0.2')
//#####, #####
compile('com.vip.venus.plugin:authsys-plugin:0.0.2')
```

Note

venus codegen生成的gradle项目没有自动添加依赖，需要开发人员手动在spring配置文件中添加依赖

maven中添加依赖：

```
<dependency>
    <groupId>com.vip.venus</groupId>
    <artifactId>venus-security</artifactId>
    <version>1.3.8</version>
    <scope>compile</scope>
</dependency>
<!--####OA, #####-->
<dependency>
    <groupId>com.vip.venus.plugin</groupId>
    <artifactId>oa-plugin</artifactId>
    <version>0.0.2</version>
</dependency>
<!--#####, #####-->
<dependency>
    <groupId>com.vip.venus.plugin</groupId>
    <artifactId>authsys-plugin</artifactId>
    <version>0.0.2</version>
</dependency>
```

Note

本章节所讲的配置均是在applicationContext.xml文件中配置。

如果另写.xml配置文件，需要修改web.xml的classpath的值才能读取到。

16. 2#使用venus-security接入OA流程

step1. 在web.xml中添加cas拦截器和监听器

```

<!-- cas start -->
<filter>
    <filter-name>CAS Single Sign Out Filter</filter-name>
    <filter-class>org.jasig.cas.client.session.SingleSignOutFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>CAS Single Sign Out Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<listener>
    <listener-class>org.jasig.cas.client.session.SingleSignOutHttpSessionListener</listener-class>
</listener>

<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

Note

主要配置了cas single sign out filter，使用DelegatingFilterProxy代理所有请求

step2. 修改spring配置文件

venus-security的引入:

```

<beans xmlns:venus-security="http://venus.vip.vip.com/schema/security"
       xsi:schemaLocation="http://venus.vip.vip.com/schema/security http://venus.vip.vip.com/schema/security/
venus-security-1.0.0.xsd">

```

venus-security的配置:

```

<venus-security:config>
    <venus-security:auth-type>
        <venus-security:cas>
            <property name="casServerUrl" value="https://cas.test.vipshop.com:8443" /> ①
            <property name="clientServerUrl" value="http://localhost:8080/venussecurity-webapp" /> ②
        </venus-security:cas>
    </venus-security:auth-type>

    <venus-security:metadata-source>
        <venus-security:realms>
            <venus-security:simple-realm>
                <venus-security:intercept-url pattern="/**" access="cat.operation" /> ③
            </venus-security:simple-realm>
        </venus-security:realms>
    </venus-security:metadata-source>

    <venus-security:access-decision-
manager class="org.springframework.security.access.vote.AffirmativeBased">
        <venus-security:voters>
            <bean class="org.springframework.security.access.vote.AuthenticatedVoter" /> ④
            <ref bean="authSysVoter" />
        </venus-security:voters>
    </venus-security:access-decision-manager>
</venus-security:config>

<bean id="authSysVoter" class="com.vip.venus.plugin.security.authsys.AuthSysRoleVoter">
    <property name="useAuthSys" value="false" /> ⑤

```

```

<property name="superAdmins">
<set>
    <value>sheldon.zhang</value>
    <value>tony.wang</value>
</set>
</property>
</bean>

```

- ① 配置oa的地址: oa Url为 <https://cas.test.vipshop.com:8443>
- ② 具体app的访问地址
- ③ 配置拦截的url pattern, 上例是拦截所有请求。
- ④ 配置voter。默认加载AuthenticatedVoter和授权系统Voter, 可以通过实现org.springframework.security.access.AccessDecisionVoter<S>接口来配置个性化voter.
- ⑤ 添加访问和授权配置。若useAuthSys为false, 默认用户具有所有url的访问权限。若useAuthSys为true, 则需要在授权系统中配置相应权限, 然后联调

Note

使用默认voter(授权系统voter)的时候, 项目需要主动申明依赖com.vip.venus.plugin:authsys-plugin的jar

使用默认UserDetails(OA系统的OAUserDetails)的时候, 项目需要主动申明依赖com.vip.venus.plugin:oa-plugin的jar

venus-security的典型配置和详细配置请参考Chapter#29, Venus Security

step3. 修改build.gradle文件, 在webapp工程下添加oa-plugin和authsys-plugin的依赖

在webapp工程下添加代码如下:

```

project("venussecurity-webapp") {
    .....
    dependencies {
        compile('com.vip.venus.plugin:oa-plugin:0.0.2')
        compile('com.vip.venus.plugin:authsys-plugin:0.0.2')
        .....
    }
}

```

step4. 在当前的jdk中导入oa cas证书

oa证书下载: [测试环境证书下载](#) [正式环境证书下载](#)

```

<!--#####-->
keytool -import -alias tomcatsso -file "#####" -keystore "#####" -storepass changeit

<!--#-->
keytool -import -alias tomcatsso -file "C:\Users\wei18.zhang\Desktop\cas.test.vipshop.com.cer"
-keystore "C:\Program Files\Java\jdk1.7.0_71\jre\lib\security\cacerts" -storepass changeit

```

Note

证书路径: 下载oa证书的存放路径

```
#####jdk#####\jre\lib\security\cacerts
```

step5. 在代码中使用cas证书得到的用户相关信息

- java获取用户信息如下

以***RestApiController.java为例说明：

```
import com.vip.venus.plugin.security.oa.OAUserDetails;
import com.vip.venus.security.core.context.VenusSecurityContextHolder;
import com.vip.venus.security.core.context.VenusSecurityContext;
@RestController
@RequestMapping("/userservice")
public class CatRestApiController {
    @RequestMapping(value = "/cat", method = RequestMethod.POST)
    public ResponseEntity<ResponseEnvelope<Integer>> createCat(@RequestBody CatVO catVO) {
        VenusSecurityContext vsc = VenusSecurityContextHolder.getContext(); ①
        OAUserDetails user = (OAUserDetails)vsc.getAuthentication().getPrincipal(); ②
        String uid = user.getUid(); ③
        String username = user.getName(); ④
        catVO.setUid(uid);
        catVO.setUsername(username);
        CatModel catModel = beanMapper.map(catVO, CatModel.class);
        Integer result = catService.create(catModel);
        ResponseEnvelope<Integer> responseEnv = new ResponseEnvelope<Integer>(result);
        return new ResponseEntity<>(responseEnv, HttpStatus.OK);
    }
}
```

<1>~<4> :获得用户信息

- jsp获得用户信息方法如

```
<%@page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@page import="com.vip.venus.plugin.security.oa.OAUserDetails"%>
<%@page import="com.vip.venus.security.core.context.VenusSecurityContextHolder"%>
<%@page import="com.vip.venus.security.core.context.VenusSecurityContext"%>
<%
    VenusSecurityContext vsc = VenusSecurityContextHolder.getContext();
    OAUserDetails user = (OAUserDetails)vsc.getAuthentication().getPrincipal();
    String uid = user.getUid();
    String username = user.getName();
%>
```

Note

第一次接入 CAS，用 OA登陆时会报错：“IP is not authorized to use CAS”，请Email To：牛江飞 申请对 IP登记授权。

17. #使用配置中心读取配置信息和监控配置变更

我们可以从配置中心读取cache和db的配置信息，也可以从配置中心监控属性变更。

使用配置中心需要在<venus-context>, applicationCfgDef.xml以及properties文件中定义配置中心的相关信息。

配置中心使用请参考 [配置中心使用手册](#)

OSP路由配置请参考 [OSP路由配置说明](#)

17. 1#配置中心的zk集群

如果想要使用配置中心，必须先要环境变量或者系统变量或者properties配置配置中心的zk集群。

properties配置方式对gradlew tomcatRun/gradlew run0sp方式有效，在gradle.properties中配置如下：

```
#zk config
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:2181
```

配置中心环境分为开发环境，测试环境，回归测试环境和生产环境，不同的环境下使用不同的zk。

开发环境是供开发人员开发使用的，测试环境是供QA功能测试用的。

回归测试环境是供集成测试联调用的，生产环境是供项目上线生产使用的。

开发环境zk集群：

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:2181
```

测试环境zk集群：

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.94:2181,10.101.18.94:2182,10.101.18.94:2183
```

回归测试环境zk集群：

```
VIP_CFGCENTER_ZK_CONNECTION=10.199.166.238:2181,10.199.166.239:2181,10.199.166.240:2181
```

生产环境zk集群： VIP_CFGCENTER_ZK_CONNECTION=gd6-cfgcenter-zk-001.idc.vip.com:2181, gd6-cfgcenter-zk-002.idc.vip.com:2181, gd6-cfgcenter-zk-003.idc.vip.com:2181

如果venus项目是OSP项目，同时还要配置VIP_0SP_ZOOKEEPER，具体请参考OSP服务的???

不同的环境要用不同的配置中心，配置中心如下：

[开发环境的配置中心](#), [测试环境的配置中心](#), [回归测试环境的配置中心](#), [生产环境的配置中心](#)

17. 2#venus-context配置

配置示例：

```
<beans xmlns:venus-context="http://venus.vip.vip.com/schema/context"
       xsi:schemaLocation="http://venus.vip.vip.com/schema/context http://venus.vip.vip.com/schema/
context/venus-context-1.0.0.xsd">
    <venus-context:property-placeholder ignore-resource-not-found="true"/> ❶

```

❶ 如果指定位置没有找到对应属性文件，true是忽略该文件，false是报错。

Note

venus codegen生成的项目代码中已包含上述示例.

venus-context中未配置属性均使用其默认值,

其具体配置请参考 Chapter#10, 使用venus-context读取配置文件

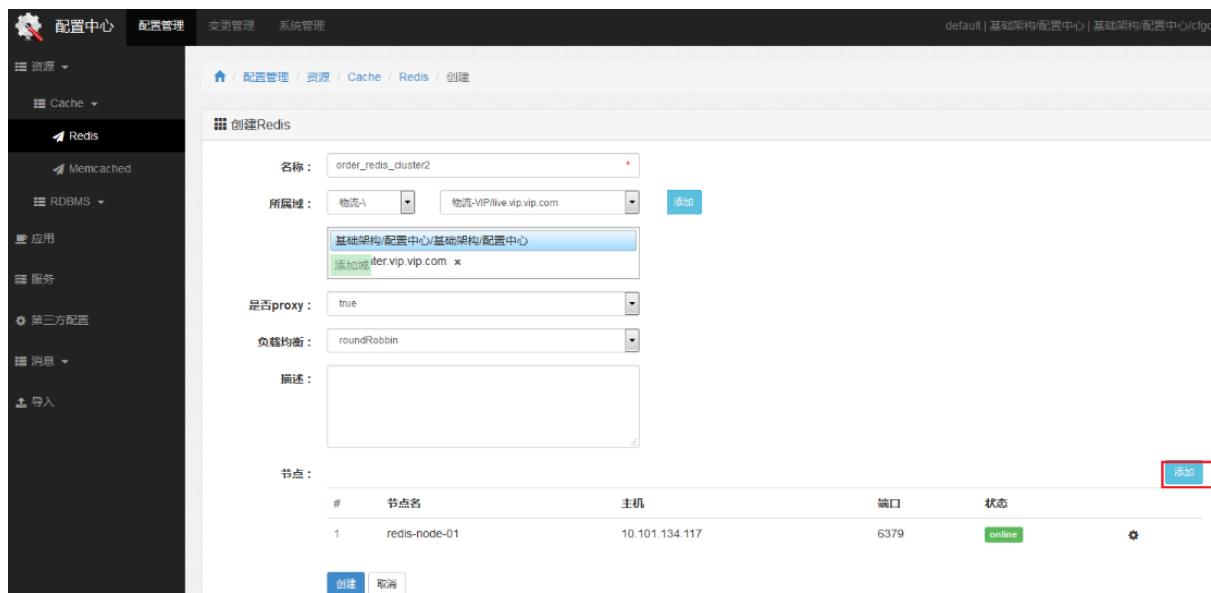
17. 3#从配置中心读取cache信息

我们可以在配置中心创建cache的应用信息，应用服务从配置中心读取cache信息。配置中心是部署在zk上的，一旦缓存的配置信息修改变化，会主动推送给应用服务。

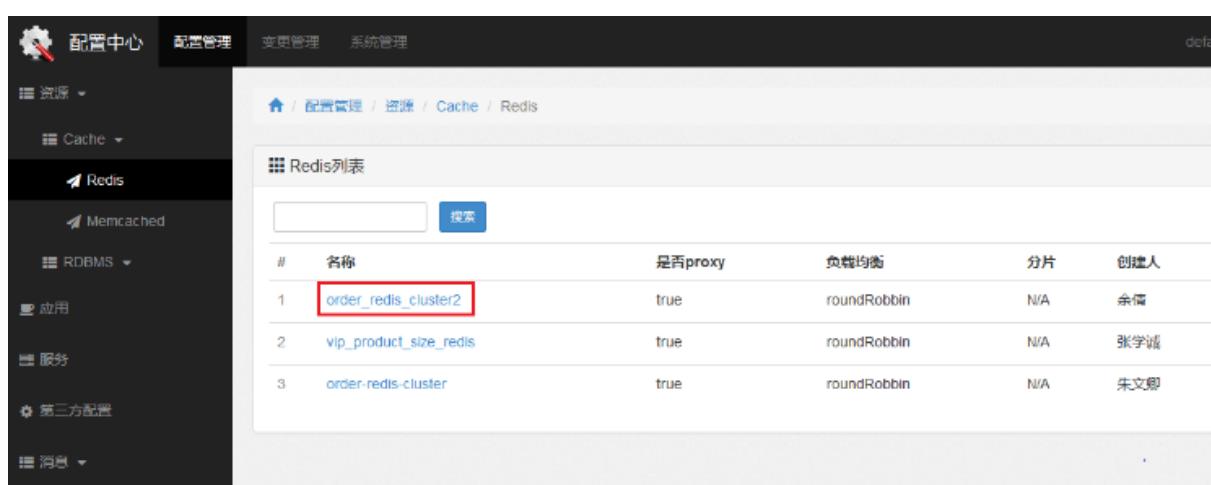
step1：在配置中心上创建cache的应用信息：配置好redis的IP和port

在配置中心的 资源/Cache/Redis下创建了一个Redis, name为order_redis_cluster2, name必须与applicationContext.xml中的cluster-name一致。

创建如下图：



创建成功后，能在配置中心查询到你创建的redis结点，然后提交审核→下发



step2: 在applicationContext.xml中配置venus-cache时，需要定义cluster-name为order_redis_cluster2（必须与step1中配置中心定义的redis name一致），type为redis。

配置示例：

```
<venus-cache:cache-manager name="cacheManager">
    <venus-cache:caches>
        <venus-cache:cache-cluster name="accountCache" cluster-name="order_redis_cluster2" type="redis" /> ①
    </venus-cache:caches>
</venus-cache:cache-manager>
```

step3：配置配置中心的zk集群：

在环境变量或者系统变量中按照实际情况配置配置中心的zk，具体配置请参考Section#17.1，“配置中心的zk集群”：

配置示例(示例是配置中心开发环境的zk)：

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:2181
```

step4：启动webapp,首次查询是从数据库查询并且把结果存入redis，首次查询后我们能在redis查询到结果。

首次在数据库中查询id=334的数据：

The screenshot shows the RESTClient interface. In the Request tab, the method is set to GET, the URL is http://localhost:8080/venuscache-webapp/userservice/teacher/334, and the Body section is empty. In the Response tab, the Response Headers tab is selected, showing the following headers:

Content-Type	application/json
Content-Length	111
Date	Mon, 10 Dec 2018 10:44:27 GMT
Server	Apache-Coyote/1.1

The Response Body tab is also selected, displaying the following JSON data:

```
{"data": {"id": 334, "name": "d78c2e75-1e95-4546-b51d-b822afb5e28c"}}
```

在redis中查询key=334的结果就是查询db的返回结果，说明使用缓存成功。

```
redis 127.0.0.1:6379> GET 334
"\"\\xac\\xed\\x00\\x05sr\\x00,com.vip.venus.userservice.model.TeacherModel\\xc9\\xb7\\xe4\\xb5\\xa7`\\x02\\x00\\x02L\\x00\\x02idt\\x00\\x10Ljava\\lang\\Long;L\\x00\\x04name\\x00\\x12Ljava\\lang\\String;xpsr\\x00\\x0e.java.lang.Long;\\x8b\\xe4\\x90\\xcc\\x8f#\\xdf\\x02\\x00\\x01J\\x00\\x05value\\x00\\x10java.lang.Number\\x86\\xac\\x95\\x1d\\x0b\\x94\\xe0\\x8b\\x02\\x00\\x00xp\\x00\\x00\\x00\\x00\\x00\\x00\\x01Nt\\x00\\$d78c2e75-1e95-4546-b51d-b822afb5e28c"
```

17.4#从配置中心监控配置变更

当配置中心的配置发生更改后，若我们希望能得到通知对此做一些逻辑上的业务处理，该怎么做呢？

我们需要自定义handler，在handler中做一些逻辑上的业务处理，并且把handler配置到applicationContext.xml.

当项目被上传到配置中心后，venus代码把handler注册为zookeeper的watcher，当配置中心的配置文件发生更改后，zookeeper会触发handler，主动推送更新的配置信息。

操作流程是在配置中心导入项目，提交审核下发，就可以在配置中心上修改属性配置。修改属性配置后，要提交审核下发，才能把配置主动推送给handler。

我这里以一个osp项目为例具体讲解下操作流程：

step1. 在项目中配置applicationCfgDef.xml：

webapp项目中，此文件一般位于webapp项目下src/main/resources

osp项目中，此文件一般位于service项目下src/main/resources

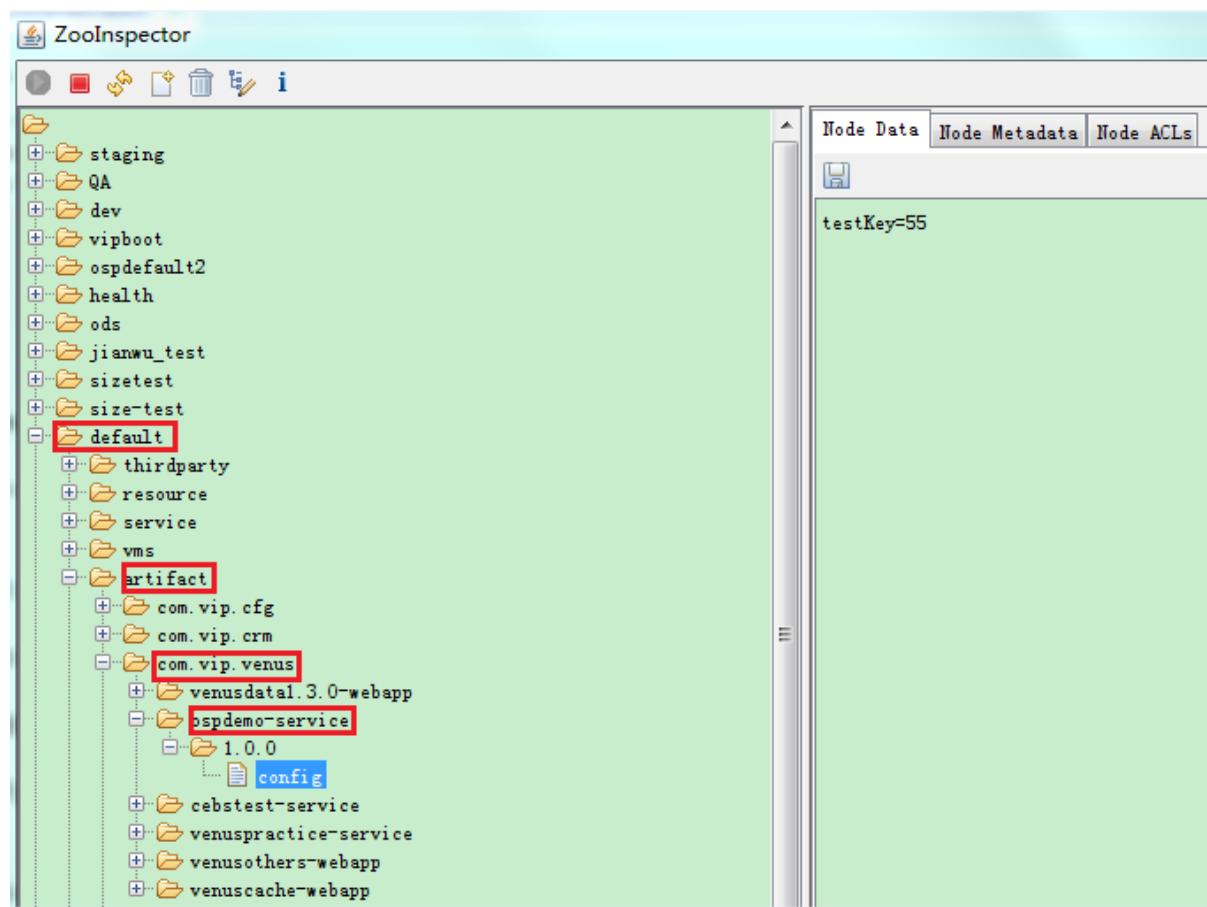
配置示例（item中配置待监控的property，本配置中要监控的property就是“testKey”）：

```
<cfgdef group="com.vip.venus" name="ospdemo-service" version="1.0.0">
  <itemgroup name="all">
    <item name="testName" key="testKey" defaultValue="0" dataType="integer" hotReload="true"/>
  </itemgroup>
</cfgdef>
```

Note

配置中心会根据group和name在zk的artifact下创建一个节点存储“testKey”，我们通过ZooInspector能看到，如下图示

cfgdef的详细配置请参考Chapter#39，配置定义文件



step2: 配置配置中心的zk集群

venus项目需要在环境变量或者系统变量中按照实际情况配置配置中心的zk，具体配置请参考Section#17.1，“配置中心的zk集群”：

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:2181
```

osp项目还需要在环境变量或者系统变量中配置VIP_OSP_ZOOKEEPER:

```
VIP_OSP_ZOOKEEPER=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:2181
```

step3. 自定义handler

代码示例:

```
###handler###PropertyChangedHandler###
@PropertyHandler
public class TestHandler implements PropertyChangedHandler {

    @Override
    public void execute(Properties oldProps, Properties newProps) {
        System.out.println(oldProps.get("testKey").toString() + "----" + newProps.get("testKey").toString());
    }
}
```

step4. 在spring中的applicationContext.xml注册handlers

配置示例:

```
<venus-context:property-placeholder ignore-resource-not-found="true">
    <venus-context:property-handlers>
        <bean class="com.vip.venus.userservice.handlers.TestHandler" name="testHandler"/>
    </venus-context:property-handlers>
</venus-context:property-placeholder>
```

Note

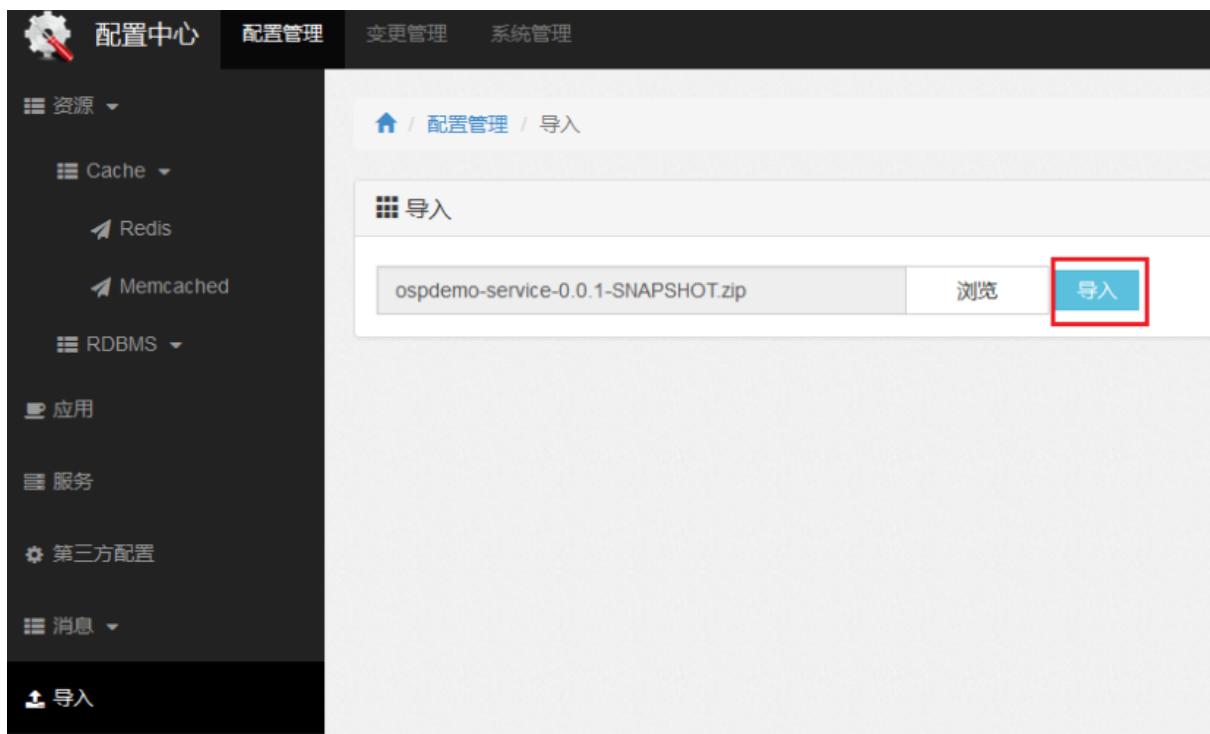
如不需要监听配置变化，请在spring中不要配置handler

step5. 应用项目打包上传和导入配置中心

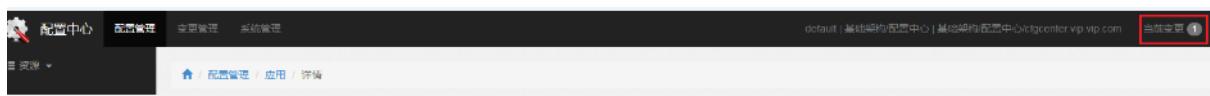
osp项目是上传zip包，venus项目是上传war包或者applicationCfgDef.xml文件。

这里以ospdemo项目为例，先用命令打包：‘gradlew clean ospServiceZip’，然后上传ospdemo-service\build\distributions下的zip包，上传后提交审核→下发

导入:

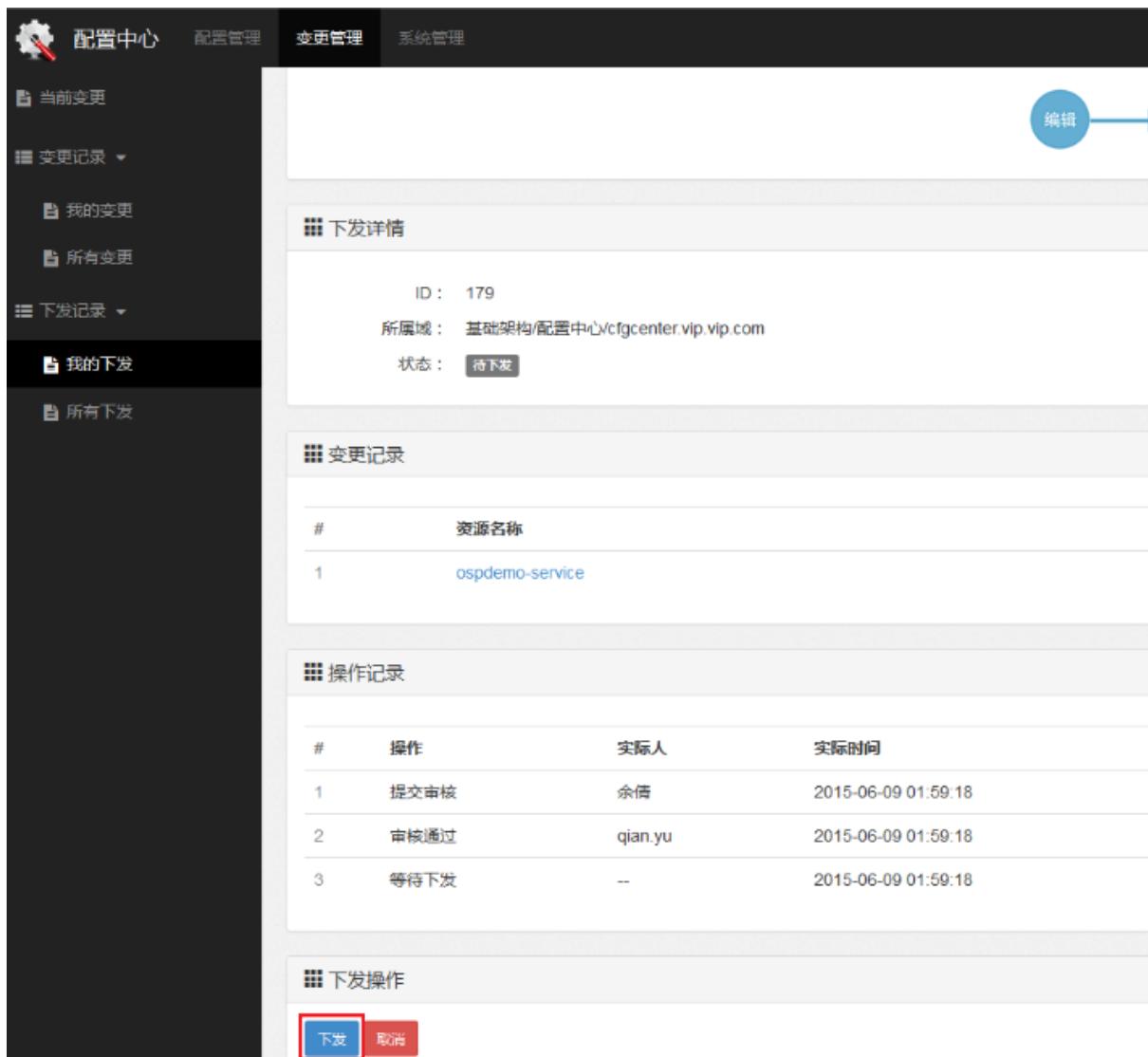


提交审核:



下发:





Note

默认上传项目的分区是default，我可以通过环境变量设置上传项目到别的分区，设置如下：'VIP_CFGCENTER_PARTITION=**'

不同的环境要用不同的配置中心：

[开发环境的配置中心](#), [测试环境的配置中心](#), [回归测试环境的配置中心](#), [生产环境的配置中心](#)

配置中心使用请参考：[配置中心wiki](#)

step6. 运行项目，以runOsp方式运行应用项目：'gradlew clean runOsp'

step7. 配置中心修改属性testkey, 然后提交审核下发，就会触发handler

编辑：

The screenshot shows the 'Configuration Center' application management interface. On the left, there is a sidebar with categories: Cache, Application (highlighted with a red box), Service, Third-party Configuration, Message, and Import. The main area displays a table of services with columns: #, Name, Namespace, and Version. A row for 'ospdemo-service' is selected and highlighted with a red box. A red annotation above the table says '双击项目进入编辑页面' (Double-click the project to enter the editing page).

#	名称	命名空间	版本
1	venusdata1.3.0-webapp	com.vip.venus	1.0.0
2	ospdemo-service	com.vip.venus	1.0.0
3	venuspractice-service	com.vip.venus	1.0.0
4	demo101-service	com.vip.venus demo101	1.0.0
5	venusothers-webapp	com.vip.venus	1.0.0
6	size-table-schedule-service	com.vip.sizeable	1.0.0
7	osptest-service	com.vip.cls	1.0.0
8	penguin	com.vip.penguin	1.0.0
9	demo01-service	com.vip.venus demo01	1.0.0
10	hello-service	com.vip.cls hello demo	1.0.0

The screenshot shows the 'Configuration Center' application configuration interface. The sidebar is identical to the previous screenshot. The main area shows the 'Edit Application' form for 'ospdemo-service'. It includes fields for Name (ospdemo-service), Namespace (com.vip.venus), Version (1.0.0), and a Domain field containing '物流-VIP/live.vip.vip.com'. Below this is a 'Config' table with one entry: # 1, Name testName, Key testKey, Value 66. At the bottom are 'Save' and 'Cancel' buttons.

提交审核:

The screenshot shows the 'Configuration Center' review interface. The top bar shows 'default | 基础架构/配置中心 | 基础架构配置中心/fgcenter.vip.vip.com'. A red box highlights the '当前变更' (Current Change) button. The main area shows the 'Review' tab selected, with the URL 'http://配管管理 / 应用 / 审核'.

下发:

#	ID	发起人	预期下发人	预期下发时间
1	179	余倩	qian.yu	2015-06-08 17:59:37

ID : 179
所属域 : 基础架构/配置中心/cfgcenter.vip.vip.com
状态 : 待下发

#	资源名称
1	ospdemo-service

#	操作	实际人	实际时间
1	提交审核	余倩	2015-06-09 01:59:18
2	审核通过	qian.yu	2015-06-09 01:59:18
3	等待下发	--	2015-06-09 01:59:18

handler被触发，我们可以在osp服务端看到handler的方法中的打印：

```
17:38:01.739 [Thread-0-SendThread<10.101.18.111:2181>] INFO org.apache.zookeeper.ClientCnxn - Session establishment complete on server 10.101.18.111/10.101.18.111:2181, sessionid = 0x24cb427ef350a04, negotiated timeout = 40000
17:38:01.739 [Thread-0-EventThread] INFO o.a.c.f.state.ConnectionStateManager - State change: CONNECTED
17:38:01.760 [Thread-0] INFO c.v.v.c.c.p.impl.PropertyManagerImpl - cfgcenter [/default/artifact/com.vip.venus/ospdemo-service/1.0.0/config:testKey=55
]
55----66
66----88
> Building 93% > :ospdemo-service:runOsp
```

18. #使用Mercury

18. 1#Mercury trace接入步骤（开发篇）

如果是codegen生成的项目，只需要配置vip-mercury.properties中的name（具体请参考the section called “添加trace配置文件”），其他配置codegen都已经设置好了，无需手动配置

添加依赖

maven工程：pom.xml中配置如下：

```
<dependency>
<groupId>com.vip.mercury</groupId>
<artifactId>mercury-client-all</artifactId>
<version>${mercury.release.version}</version>
<type>pom</type>
</dependency>
```

gradle工程：build.gradle中配置如下：

```
dependencies {
    .....
    compile("com.vip.mercury:mercury-client-all:${mercuryVersion}")
    .....
}
```

Note

关于codegen生成的venus项目，文件中已自动配置，无需再手动添加依赖

server配置

Tomcat应用配置Web.xml

添加filter：

```
<!-- mercury trace begin-->
<filter>
<filter-name>traceFilter</filter-name>
<filter-class>com.vipshop.mercury.client.filter.MercuryTraceFilter</filter-class>
</filter>

<filter-mapping>
<filter-name>traceFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
<!-- mercury trace end-->
```

Note

关于codegen生成的venus项目，文件中已自动配置，无需再手动添加filter

OSP应用配置Filter

OSP应使用1.1.5以上的版本

OSP server端

在项目classpath下（一般放在resources文件夹下面）添加ospconfig文件夹，并新建filter.properties文件，内容如下：

```
filters=com.vipshop.mercury.client.filter.MercuryOspTraceFilter
```

Note

关于codegen生成的venus项目，此文件已自动生成，无需再手动添加

OSP client端

在项目classpath下（一般放在resources文件夹下面）添加ospconfig文件夹，并新建client_filter.properties文件，内容如下：

```
filters=com.vipshop.mercury.client.filter.MercuryOspStubFilter
```

For version_2.0之后的版本：

```
beforeSendFilters=com.vipshop.mercury.client.filter.MercuryOspStubBeforeFilter
afterReceiveFilters=com.vipshop.mercury.client.filter.MercuryOspStubAfterFilter
beforeReceiveFilters=com.vipshop.mercury.client.filter.MercuryOspPreReceiveFilter
```

Note

关于codegen生成的venus项目，此文件已自动生成，无需再手动添加

OSP既是client又是Server

filter.properties和client_filter.properties都要添加

Note

关于codegen生成的venus项目，这两个文件都已自动生成，无需再手动添加

添加trace配置文件

在项目classpath下添加vip-mercury.properties文件，一般放在resources文件夹下面，文件内容如下：

```
mercury.application.name=captcha_service#####home#####
mercury.application.sample=false#####
mercury.application.frequency=0.8#####
mercury.url.pattern=/{{code1}}/urlPattern/{{code2}} ①
```

① mercury.url.pattern主要是针对restful应用，如果有restful url，需要配置url pattern，变量部分使用{}代替，中括号里是该变量的含义，如{user_id}, {user_name}

Note

关于codegen生成的venus项目，这个文件已自动生成，无需再手动添加。

但是只有name与CMDB的域名保持一致时mercury会自动将域加入到对应的业务组，否则，目标域会放在“未知域”下，

| 可codegen自动生成的name名就是服务名，请开发者根据实际情况配置name。

添加log4j/logback配置

这是接入trace最重要的一步，依应用情况配置log4j或者logback配置文件，下面以passport应用配置的log4j.xml为例：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

<appender name="consoleAppender" class="org.apache.log4j.ConsoleAppender">
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value="%d [%p][%t]:%l %m%n" />
</layout>
</appender>

<!-- mercury trace#logger#### -->
<appender name="MercuryAppender" class="com.vipshop.mercury.log.log4j.MercuryLog4jAppender">
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value "[%d{MMdd HH:mm:ss SSS\} %-5p] [%t] %c{3} - %m%n" />

</layout>
</appender>
<logger name="com.vip"><!-- ##### -->
<level value="info" /><!-- #####-->
<appender-ref ref="MercuryAppender" />
</logger>

<appender name="TraceFileAppender" class="org.apache.log4j.RollingFileAppender">
<param name="File" value="/apps/logs/trace/logs/trace-log.out" />
<param name="BufferSize" value="512" />
<!-- ##### -->
<param name="Append" value="true" />
<param name="MaxFileSize" value="1GB"/></param>
<param name="MaxBackupIndex" value="10" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value "%m%n" />
</layout>
</appender>
<logger name="mercury-trace-logger" additivity="false">
<level value="info" />
<appender-ref ref="TraceFileAppender" />
</logger>

<appender name="LogFileAppender" class="org.apache.log4j.RollingFileAppender">
<param name="File" value="/apps/logs/trace/logs/log-log.out" />
<param name="BufferSize" value="512" />
<!-- ##### -->
<param name="Append" value="true" />
<param name="MaxFileSize" value="1GB"/></param>
<param name="MaxBackupIndex" value="10" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value "%m%n" />
</layout>
</appender>
<logger name="mercury-log-logger" additivity="false">
<level value="info" />
<appender-ref ref="LogFileAppender" />
</logger>

<appender name="MetricFileAppender" class="org.apache.log4j.RollingFileAppender">
<param name="File" value="/apps/logs/trace/logs/metric-log.out" />
<param name="BufferSize" value="512" />
```

```

<!-- ##### -->
<param name="Append" value="true" />
<param name="MaxFileSize" value="1GB"></param>
<param name="MaxBackupIndex" value="10" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value="%m%n" />
</layout>
</appender>
<logger name="mercury-metric-logger" additivity="false">
<level value="info" />
<appender-ref ref="MetricFileAppender" />
</logger>
<!-- mercury trace#logger### -->

<!-- #logger###-->
<root>
<priority value ="info"/>
<appender-ref ref="consoleAppender"/>
<appender-ref ref="R"/>
</root>

</log4j:configuration>

```

若应用使用的logback组件，则参照以下配置文件：

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<!-- mercury trace#logger### -->
<appender name="MERCURY" class="com.vipshop.mercury.log.logback.MercuryLogbackAppender">
<encoder>
<pattern>%date [%thread] %-5level %logger{80} - %msg%n</pattern>
</encoder>
</appender>

<logger name="com.vip">
<level value="info" />
<appender-ref ref="MERCURY" />
</logger>

<!-- trace log -->
<appender name="MercuryTraceFileAppender" class="ch.qos.logback.core.rolling.RollingFileAppender">
<file>/apps/logs/trace/logs/trace-log.out</file>
<rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
<!-- daily rollover -->
<fileNamePattern>/apps/logs/trace/logs/trace-log-%i.out</fileNamePattern>
<!-- keep 10 G' worth of history -->
<minIndex>1</minIndex>
<maxIndex>10</maxIndex>
</rollingPolicy>
<append>true</append>
<triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
<maxFileSize>1GB</maxFileSize>
</triggeringPolicy>
<encoder>
<pattern>%msg%n</pattern>
</encoder>
</appender>

<logger name="mercury-trace-logger" additivity="false">
<level value="info" />
<appender-ref ref="MercuryTraceFileAppender" />
</logger>

<appender name="MercuryLogFileAppender" class="ch.qos.logback.core.rolling.RollingFileAppender">
<file>/apps/logs/trace/logs/log-log.out</file>
<rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
<!-- daily rollover -->
<fileNamePattern>/apps/logs/trace/logs/log-log-%i.out</fileNamePattern>

```

```

<!-- keep 10G' worth of history -->
<minIndex>1</minIndex>
<maxIndex>10</maxIndex>
</rollingPolicy>
<append>true</append>
<triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
<maxFileSize>1GB</maxFileSize>
</triggeringPolicy>
<encoder>
<pattern>%msg%n</pattern>
</encoder>
</appender>
<logger name="mercury-log-logger" additivity="false">
<level value="info" />
<appender-ref ref="MercuryLogFileAppender" />
</logger>

<appender name="MercuryMetricFileAppender" class="ch.qos.logback.core.rolling.RollingFileAppender">
<file>/apps/logs/trace/logs/metric-log.out</file>
<rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
<!-- daily rollover -->
<fileNamePattern>/apps/logs/trace/logs/metric-log-%i.out</fileNamePattern>
<!-- keep 10G' worth of history -->
<minIndex>1</minIndex>
<maxIndex>10</maxIndex>
</rollingPolicy>
<append>true</append>
<triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
<maxFileSize>1GB</maxFileSize>
</triggeringPolicy>
<encoder>
<pattern>%msg%n</pattern>
</encoder>
</appender>
<logger name="mercury-metric-logger" additivity="false">
<level value="info" />
<appender-ref ref="MercuryMetricFileAppender" />
</logger>
<!-- trace log end-->
<!-- mercury trace#logger#### -->

</configuration>

```

Note

关于codegen生成的venus项目，自动生成了logback.groovy文件，不要再手动配置log4j.xml或logback.xml。

验证

请求应用，在/apps/logs/trace/logs路径下出现下面三个文件：log-log.out, metric-log.out和trace-log.out。

log-log.out和trace-log.out有日志生成。每种log的作用如下：

log-log.out：应用程序使用log4j、logback组件打的日志，包括日志的堆栈信息等，框架抛出的程序未捕获的异常日志

Note

关于codegen生成的venus项目，根据logback.groovy的配置，只有运行环境为production/integreatetest(同时要配置对应的properties文件)，日志才会输出到/apps/logs目录下。

若当前的运行环境为development，则日志输出到控制台上。osp项目可以在`ospServiceProjects.gradle`中配置环境变量，配置如下：

```
ospService {
    ospVersion = project.ospVersion
    profile = 'production'
}
```

18. 2#Mercury trace接入步骤（部署篇）

配置AspectJ拦截

WEB应用配置Tomcat

首先下载 [aspectjweaver-1.7.3.jar](#) 到本地，假如存放目录是`/home/test/tomcat/`

如果是windows，在`catalina.bat`首行添加：`set JAVA_OPTS="-javaagent:/home/test/tomcat/aspectjweaver-1.7.3.jar"`

如果是linux，在`catalina.sh`首行添加：`export JAVA_OPTS="-javaagent:/home/test/tomcat/aspectjweaver-1.7.3.jar"`

OSP应用配置启动参数（`passport`等无osp服务的应用跳过此步骤）

启动脚本如下：

```
nohup java -javaagent:${ASPECTJWEAVER_PATH}\aspectjweaver-1.7.3.jar -jar ${OSP_CONTAINER_PATH}/osp-engine.jar -servicesdir ${OSP_APP_PATH} -useivy false > osp.log 2>&1 &
```

正常情况下，唯品会的线上环境都应基于JDK1.7。但如应用是基于JDK1.8，则需要依赖`aspectjweaver-1.8.6.jar`，下载 [aspectjweaver-1.8.6.jar](#)。

Note

OSP应用启动前不要忘记在环境变量中设置`VIP_CFGCENTER_ZK_CONNECTION`。

关于codegen生成的项目，根据`logback.groovy`的配置，只有设置运行环境为`production/integreatetest`时，才能输出日志到mercury指定的文件

设置运行环境时要同时配置当前运行环境下相关的`properties`文件，否则报错无法正常启动。

还可以在osp的启动脚本中加上`-restport 8989`，启动后就可以访问`http://${serverIP}:8989`中的在线测试进行服务测试

安装flume

QA环境flume

```
ssh root@10.199.168.133 密码vipshop path:/apps/svr/flume1.5
```

QA环境需要配置hosts，在`etc/hosts`增加如下代码：

```
10.199.164.195 Cloud195
10.199.167.197 Cloud197
```

```
10.199.167.135 Cloud135
```

与Mercury服务器同步下时间，执行下面的命令：

```
/usr/sbin/ntpdate 10.101.18.66
```

启动flume：

通过python启动（需使用python2.6及以上版本，并且更改<mercury-flume>/external-lib/flume_init.py脚本中的CWD为当前路径（默认在/apps/svr/mercury-flume下）

```
cd <mercury-flume>/bin
chmod +x flume-ng
cd <mercury-flume>/external-lib/
python flume_init.py start
```

停止flume：

```
python flume_init.py stop
```

线上的flume

[线上版本flume svn下载](#), commitid: 0f32fef4b4e

验证

在原始日志滚动的前提下，查看flume的日志是否在滚动，具体方法：

```
cd /apps/logs/flume/
tail -f file2kafka8.log
```

Merucry的接入是自发现的。只要有数据上报，就可以在http://10.101.18.61/index#/ [Mercury Home (QA)] 上看对应域的监控数据

The screenshot shows the Mercury Home (QA) monitoring interface. At the top, it displays the time as 18:29:08 and has a feedback link, a user profile icon, and an exit button. Below the header, there's a search bar with placeholder text "请输入域名....." and a "搜索" button. A navigation bar includes tabs for "业务组" (selected), "网站/用户和会员中心", and "全局". On the right side of the header, it shows "总接入域: 81", "今天接入域: 1", and "一周内接入域: 9". The main content area is titled "网站/用户和会员中心". It lists domain statistics in a table:

域名	告警数	平均响应时间(ms)	每秒请求数	总请求数	失败率	失败个数	异常率	异常个数
schedule-uris-tag.api.vip.com	0	0	0	0	0%	0	0%	0
user_info	3	4.36	1,146.43	4,127,151	1.51%	62,249	0.34%	14,224
vmark-biz.vip.vip.com	0	0.65	1.35	4,866	0%	0	0%	0

At the bottom of the table, there are navigation links: 首页, 上一页, 1, 下一页, 尾页. A note at the bottom left says "此列表为当前1小时内的数据".



最后，在trace-home上能看到应用的运行状态，表示部署成功。

19. #使用Hermes

19. 1#添加依赖

maven工程: pom.xml 中配置如下:

```
<dependency>
<groupId>com.vip.hermes</groupId>
<artifactId>hermes-core</artifactId>
<version>${hermes.release.version}</version>
<type>pom</type>
</dependency>
```

gradle工程: build.gradle中配置如下:

```
dependencies {
    .....
    compile("com.vip.hermes:hermes-core:${hermesVersion}")
    .....
}
```

19. 2#XML配置

Spring应用配置applicationContext.xml

添加bean定义:

```
<bean id="reporter" class="com.vip.hermes.core.reporter.HermesSlf4jReporter" init-method="start" lazy-init="false">
    <property name="period" value="5"/>
</bean>
```

Note

其他框架请参照spring的配置

19. 3#添加log4j/logback配置

依应用情况配置log4j或者logback配置文件, 下面以passport应用配置的log4j.xml为例:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

    <appender name="consoleAppender" class="org.apache.log4j.ConsoleAppender">
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
value="%d [%p][%t]:%l %m%n" />
        </layout>
    </appender>

    <appender name="FILE" class="org.apache.log4j.FileAppender">
        <param name="File" value="/apps/logs/hermes/logs/hermes-log.out"/>
    </appender>

    <logger name="com.vip.hermes.core.reporter" additivity="false">
        <level value="INFO" />
        <appender-ref ref="FILE" />
    </logger>
```

```

<!-- #logger###-->
<root>
<priority value ="info"/>
<appender-ref ref="consoleAppender"/>
<appender-ref ref="R"/>
</root>

</log4j:configuration>

```

若应用使用的logback组件，则参照以下配置文件：

```

import ch.qos.logback.classic.encoder.PatternLayoutEncoder
import ch.qos.logback.classic.net.*
import ch.qos.logback.core.*
import ch.qos.logback.core.encoder.*
import ch.qos.logback.core.read.*
import ch.qos.logback.core.rolling.*
import ch.qos.logback.core.status.*
import com.vipshop.mercury.log.logback.MercuryLogbackAppender

statusListener(OnConsoleStatusListener)

def props = new Properties()
props.load(this.getClass().getClassLoader().getResourceAsStream("properties/application.properties"))

def config = new ConfigSlurper().parse(props)

def env = System.properties['spring.profiles.active'] ?: 'production'
def appenderList = []
def level = ERROR
def LOG_DIR = '/apps/logs/log_receiver/'+config.appname+'/logs'

jmxCConfigurator()

if (env == 'production') {
    appenderList.add("ROLLING")
} else if(env == 'integratetest') {
    appenderList.add("ROLLING")
    level = WARN
} else if(env == 'development') {
    appenderList.add("CONSOLE")
    level = INFO
}
if(env=='development') {
    appender("CONSOLE", ConsoleAppender) {
        encoder(PatternLayoutEncoder) { pattern = "%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n" }
    }
}

if(env=='production' || env=='integratetest') {
    appender("ROLLING", RollingFileAppender) {
        encoder(PatternLayoutEncoder) { Pattern = "%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n" }
        rollingPolicy(TimeBasedRollingPolicy) {
            fileNamePattern = "${LOG_DIR}/translator-%d{yyyy-MM-dd-HH}.zip"
            triggeringPolicy(timeBasedFileNamingAndTriggeringPolicy) { maxFileSize = '10M' }
        }
    }
}

appender("event", RollingFileAppender) {
    file = "/apps/logs/hermes/logs/hermes-log.out"
    rollingPolicy(FixedWindowRollingPolicy) {
        fileNamePattern = "/apps/logs/hermes/logs/hermes-log-%i.out"
        minIndex = 1
        maxIndex = 10
    }
    append = true
    triggeringPolicy(SizeBasedTriggeringPolicy) {
        maxFileSize = "10GB"
    }
}

```

```

encoder(PatternLayoutEncoder) {
    pattern = "%msg%n"
}
}
logger("com.vip.hermes.core.reporter", INFO, ["event"], false)
root(level, appenderList)

```

Note

== 关于codegen生成的venus项目，自动生成了logback.groovy文件，加入相关配置即可。 ==

19. 4#定义Item

Hermes提供六种类型的API方便埋点

HermesStatic

非统计类型的数据可通过此种类型埋点，例如配置信息等。

```

HermesStatic hs = new HermesStatic() {
    @Override
    public Map<Object, Object> getValue() {
        Map<Object, Object> map = new HashMap<Object, Object>();
        map.put("staticKey", "staticValue");
        return map;
    }
};

```

HermesGauge

一般用来统计瞬时状态的数据信息，例如数据库连接池当前空闲连接数

```

HermesGauge<Integer> gauge = new HermesGauge<Integer>() {
    @Override
    public Integer getValue() {
        return 1;
    }
};

```

HermesCounter

维护一个计数器，可以通过inc()和dec()方法对计数器做修改，是一种特殊类型的HermesGauge。

```

HermesCounter counter = new HermesCounter();
counter.inc();
counter.dec();

```

HermesMeter

用来度量某个时间段的平均处理次数，统计结果有总的请求数，平均每秒的请求数，以及最近的1、5、15分钟的平均QPS。

```

HermesMeter meter = new HermesMeter();
meter.mark();

```

HermesHistogram

一般使用来统计数据的分布情况，最大值、最小值、平均值、中位数，百分比，例如，需要统计某个页面的请求响应时间分布情况。

```
HermesHistogram histogram = new HermesHistogram();
histogram.update(1);
```

HermesTimer

是对HermesHistogram和HermesMeter的一种汇总。

```
HermesTimer timer = new HermesTimer();
Context context = timer.time();
try {
    //some operator
    Thread.sleep(20);
} catch (InterruptedException e) {
    logger.error(e.getMessage());
} finally {
    context.stop();
}
```

19. 5#注册Item

Item注册分为两种方式

register

注册Item到registry，并且定时写日志上报

```
public <T extends Metric> T register(String name, T metric);
public <T extends Metric> T register(String name, T metric, Map<String, String> tag);
public <T extends Metric> T register(String name, T metric, BaseModule module);
public <T extends Metric> T register(String name, T metric, Map<String, String> tag, BaseModule module);
```

registerInternal

注册Item到registry，但不写入日志文件上报

```
public <T extends Metric> T registerInternal(String name, T metric);
public <T extends Metric> T registerInternal(String name, T metric, Map<String, String> tag);
public <T extends Metric> T registerInternal(String name, T metric, BaseModule module);
public <T extends Metric> T registerInternal(String name, T metric, Map<String, String> tag, BaseModule module);
```

20. #使用中央文档系统

20. 1#系统简介

[中央文档系统](#), 是为基于Venus框架的应用服务(osp或RESTful)提供中心化的API文档支持的系统。

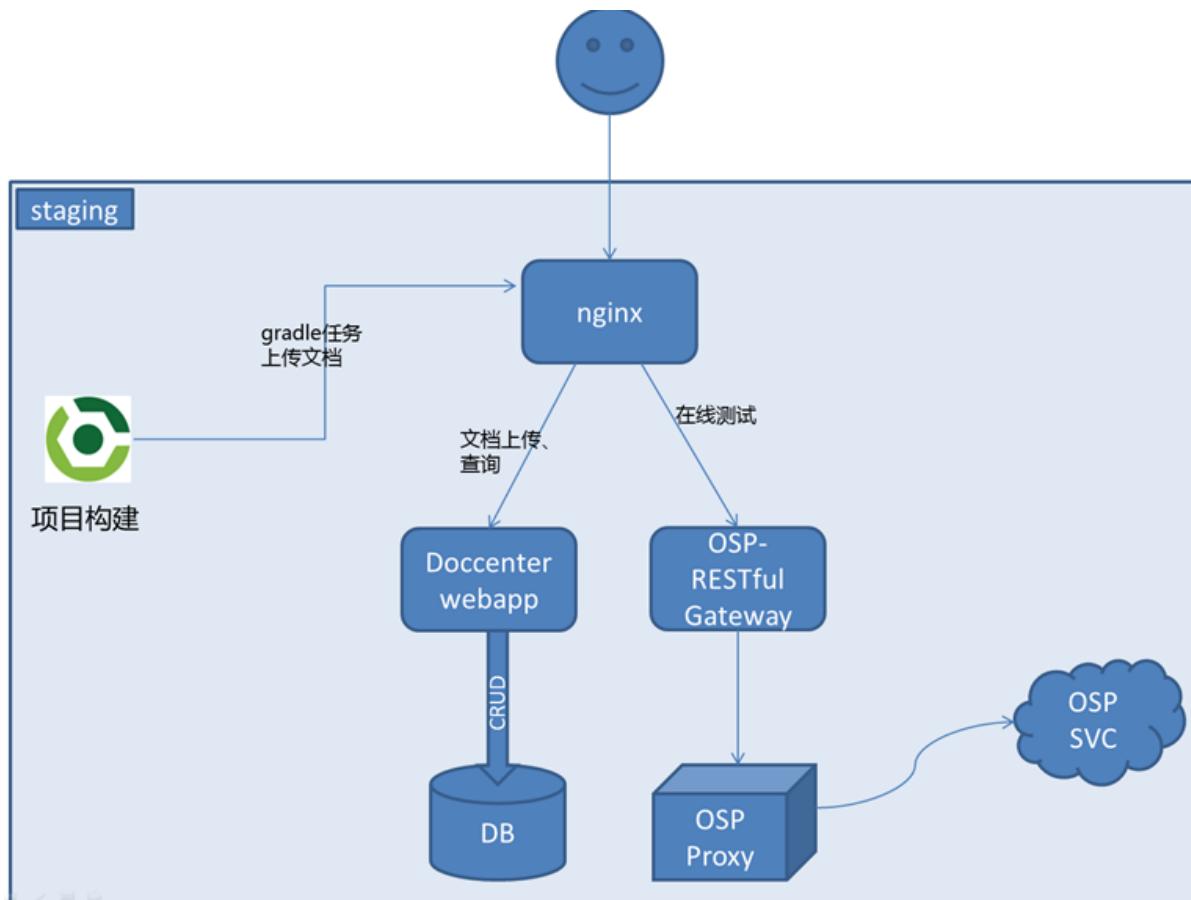
在传统的项目开发中开发人员的系统联调对接, 有以下形式: SVN维护office文档、手工维护wiki, 甚至是文档以非正式方式通过邮件或IM共享。

这有几个弊端, 一是很容易出现代码与文档不同步; 二是维护成本高, 操作复杂; 三是沟通成本高。

中央文档系统提供的文档服务, 使开发者可以无须干预文档的更新过程, 自动化构建过程会收集最新的与代码配套的文档并上传至文档库以供查看;

用户界面为开发提供了友好的API文档展示方式, 并且提供了在线测试功能, 如遇到疑问可以联系服务所关联的接口人解决。

中央文档系统由三大模块构成: 文档收集模块, 文档库模块以及在线测试模块, 架构示意图如下:



目前的版本仅支持OSP服务的文档功能, RESTful服务将在下个版本支持

使用要求: 基于Venus框架的OSP项目接入中央文档系统必须满足osp-plugin 1.2.2+版本

20. 2#主要功能

中央文档系统主要功能分为两部分:

其一为文档收集与上传，由gradle的任务完成

其二为面向用户的文档解析查看功能，用户可登陆 [中央文档系统](#) 使用
功能点

当前版本包含以下功能：

优化文档上传和使用的流程，工程构建自动完成文档收集和上传；

支持OSP文档的上传、查看、在线测试、收藏；

支持对OSP服务设置接口人，方便项目成员沟通；

文档系统与CMDB对接，且允许手工设置项目所属的域。

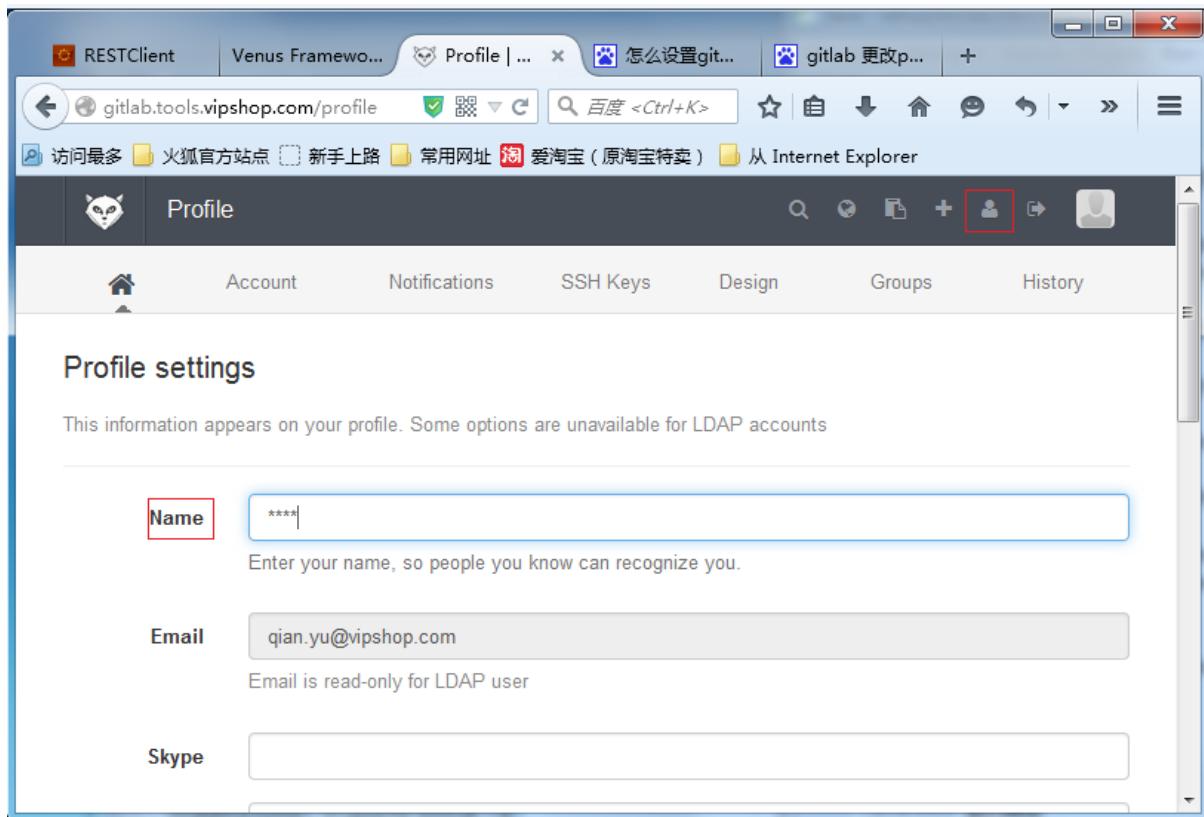
20. 3#使用步骤

具体使用请参考 [中央文档系统wiki](#)

21. #使用git管理源码

21. 1#git配置

step1. 在git lab中更改自己的profile settings, 把name项改为自己的中文名, 以利于别人查找, 如图所示:



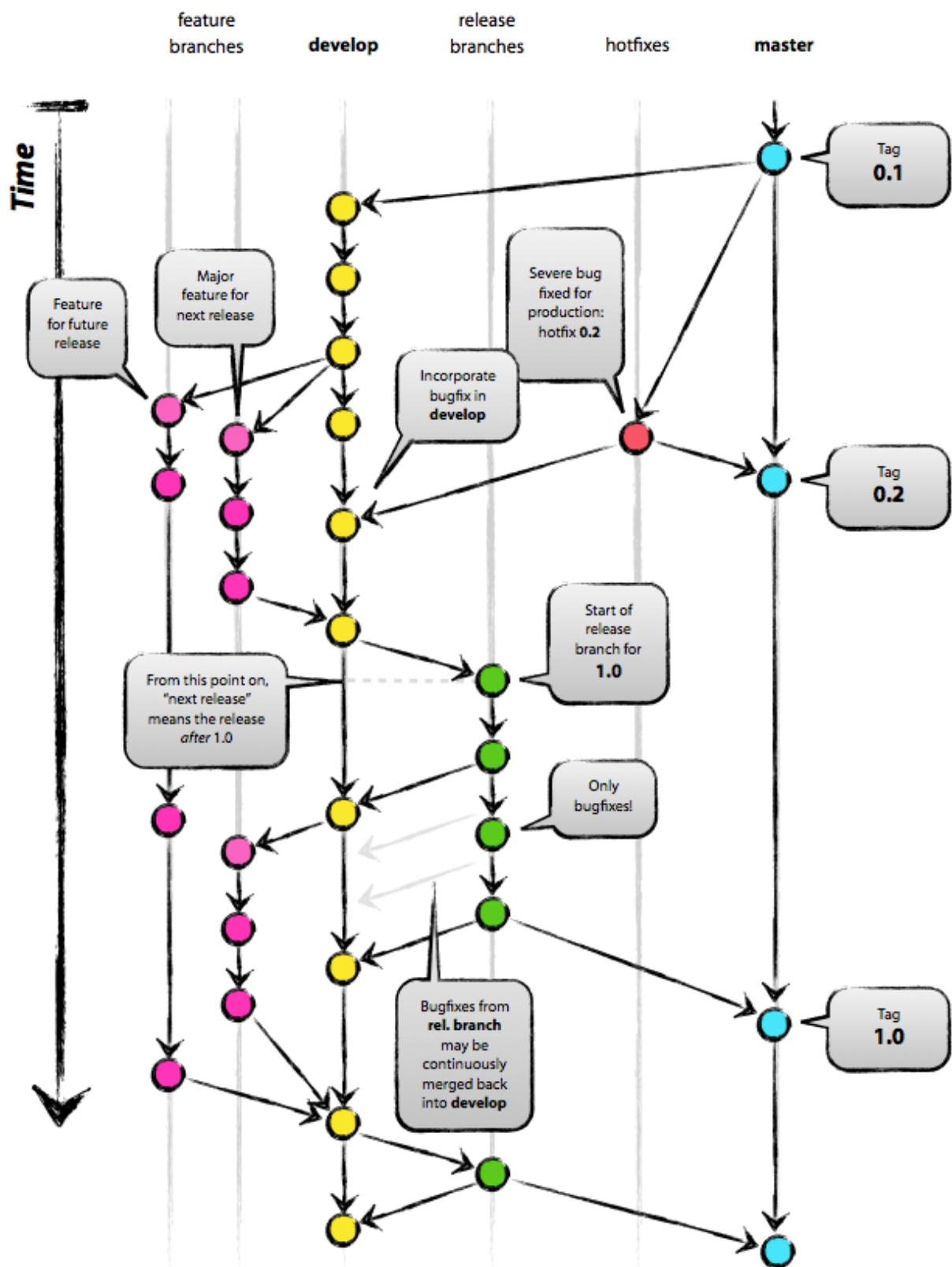
step2. 在开发机器上设置用户名和邮箱: cmd进入到git的安装目录, 依次输入以下命令

```
git config --global user.name "##"
git config --global user.email "san.zhang@vipshop.com"
```

21. 2#git分支模型

Git Flow是一套使用Git进行源代码管理时的一套行为规范和简化部分Git操作的工具, 是在Git之上构建的一项软件开发最佳实践。

Git Flow重点解决了源代码在开发过程中的各种冲突导致开发活动混乱的问题, 基本思想看下图:



git flow模型中定义了主分支和辅助分支：主分支用于组织与软件开发、部署相关的活动；辅助分支组织为了解决特定的问题而进行的各种开发活动。

主分支

主分支是所有开发活动的核心分支。所有的开发活动产生的输出物最终都会反映到主分支的代码中。主分支分为master分支和development分支。

- **master分支** master分支上存放的应该是随时可供在生产环境中部署的代码（Production Ready state）。

当开发活动告一段落，产生了一份新的可供部署的代码时，master分支上的代码会被更新。每一次更新，最好添加对应的版本号标签（TAG）。

- **develop分支** 保存当前最新开发成果的分支。通常这个分支上的代码也是可进行每日夜间发布的代码（Nightly build）。

当develop分支上的代码已实现了软件需求说明书中所有的功能，通过了所有的测试后，代码足够稳定时，就会将所有的开发成果合并回master分支了。

辅助分支

辅助分支是用于组织解决特定问题的各种软件开发活动的分支。辅助分支都是从develop分支派生的，最后必须合并到develop分支。

辅助分支包括feature分支，release分支和hotfix分支。这些分支其实没有什么区别，只是通过命名区別使用目的。

- **feature分支：** 用于开发新功能
- **release分支：** 辅助版本发布
- **hotfix分支：** 修正生产代码中的bug

关于gitflow的详细介绍：[git flow介绍](#)

21. 3#使用git上传项目流程

假如projectstart项目是使用codegen生成的项目，以projectstart为例演示：

step1. 下载 [gitignore.txt](#)，改为.gitignore文件并放入你的项目根目录

Note

window下文件命名为“.gitignore”会错误，提示“必须键入文件名”，需要输入文件名“.gitignore.”，这样就可以命名成功了

这个文件的作用是忽略掉一些eclipse的文件不上传

step2. 在 <http://gitlab.tools.vipshop.com> 上创建项目，如图所示，如图所示

New Project

Project name: startproject

Customize repository name?

Import existing repository?

Description (optional): Awesome project

Visibility Level (?):

- Private**
Project access must be granted explicitly for each user.
- Internal**
The project can be cloned by any logged in user.
- Public**
The project can be cloned without any authentication.

Create project

step3. 设置Deploy keys，使jenkins有访问权限，选上enable，如图所示：

Deploy keys allow read-only access to the repository

+ New Deploy Key

Deploy keys can be used for CI, staging or production servers. You can create a deploy key or add an existing one

Enabled deploy keys for this project

Create a new deploy key or add an existing one

jenkins_deploy

+ Enable

Deploy keys from projects available to you

usersys / user_info usersys / user_vmark
usersys / user_center usersys / user_bench
platform / venus user_center_en / user_ui
WMS / VIP_WMS_Galaxy
walter_zhang / test_group_submodule

step4. 运行GIT BASH，进入projectstart项目，依次执行如下命令，上传项目：

```
cd projectstart
git init
touch README
git add -A
git commit -m "first commit"
git remote add origin git@gitlab.tools.vipshop.com:owen.lu/projectstart.git ①
git push -u origin master
```

① 你自己所建项目的git地址

22. #使用jenkins持续集成

Jenkins的主要功能是监视重复工作的执行：软件的持续构建和测试和监视外部运行的job的执行。

step1. 申请云主机用于创建jenkins上的节点

如果没有云主机，需要按照流程申请云主机，申请地址：<http://hh.yun.vipshop.com>

step2. 访问jenkins的控制台 <http://jenkins.tools.vipshop.com/computer/>，如果是新用户先要注册用户，然后向管理员申请新建job/node的权限。

The screenshot shows the Jenkins 'Sign up' page. On the left, there's a sidebar with various Jenkins-related links: 查看用户, 任务历史, 项目的关系, 检查文件指纹, Selenium Grid, Job Import Plugin, FLOW Dashboard, and Bulk Builder. The main area is titled 'Sign up' and contains a form for creating a new user account. The form fields are: 用户名 (username) - san.zhang, 密码 (password) - ***** (redacted), 确认密码 (confirm password) - ***** (redacted), 全名 (full name) - 张三 (Zhang San), and 电子邮件地址 (email address) - san.zhang@vipshop.com. At the bottom of the form is a 'Sign up' button.

step3. 创建新的节点，配置节点，具体请参考 <http://wiki.corp.vipshop.com/pages/viewpage.action?pageId=27626790>

step4. 配置节点完成后，先在node列表中找到这个节点，点击进入，再点击launch slave agent，开始节点的配置，需要一段时间，如图所示：



step5. 设置云主机环境，这个是根据具体环境自己来设置

venus-framework项目是登录到云主机上执行如下命令下载脚本来设置环境，主要是设置hosts和下载osp engine包：

```
wget http://10.101.18.78/tools/ci/setenv.sh
chmod +x setenv.sh
./setenv.sh
```

step6. 创建新的job：

访问jenkins的控制台(比如: <http://jenkins.tools.vipshop.com/>), 点击新job链接, 进入创建新的job流程, 可以选择复制一个任务:

任务名称

- 构建一个自由风格的软件项目**
这是Jenkins的主要功能. Jenkins将会结合任何SCM和使用任何构建系统来构建你的项目, 甚至可以使用软件构建以外的系统.
- MultiJob Project**
MultiJob Project, suitable for running other jobs
- 构建一个maven2/3项目**
构建一个maven2/3项目.Jenkins利用你的POM文件,这样可以大大减轻构建配置.
- 构建一个多配置项目**
适用于多配置项目,例如多环境测试,平台指定构建,等等.
- 监控一个外部的任务**
这个类型的任务允许你记录执行在外部Jenkins的任务, 任务甚至运行在远程机器上.这可以让Jenkins作为你所有自动构建系统的控制面板.参阅 [这个文档查看详细内容](#).
- 拷贝已存在任务**
要复制的任务名称

step7. 配置job:

主要是配置git地址和分支

Project名称	10.199.210.185-projectstart
描述	projectstart in release branch
预览	
<input checked="" type="checkbox"/> 丢弃旧的构建	
保持构建的天数	10
保持构建的最大个数	如果非空，构建记录将保存此天数 如果非空，最多此数目的构建记录将被保存
<input type="checkbox"/> Use custom icon <input type="checkbox"/> 参数化构建过程 <input type="checkbox"/> Sidebar Links <input type="checkbox"/> Prepare an environment for the run <input type="checkbox"/> 停止构建（直到允许项目构建,否则不能进行新的构建） <input type="checkbox"/> 在必要的时候并发构建	
JDK	(Default)
JDK to be used for this project	
<input checked="" type="checkbox"/> Restrict where this project can be run	
Label Expression	10.199.210.185

高级项目选项

源码管理

<input type="radio"/> CVS	
<input type="radio"/> CVS Projectset	
<input checked="" type="radio"/> Git	
Repositories	<input type="text"/> Repository URL: git@gitlab.tools.vipshop.com:owen.lu/projectstart.git <input type="text"/> Credentials: gitlab_deploy

Branches to build

Branch Specifier (blank for 'any') */master

step8. 在Execute shell中添加shell脚本，如图所示：

构建环境

- Provide Configuration files
- Abort the build if it's stuck
- Copy data to workspace
- Inject environment variables to the build process
- Inject passwords to the build as environment variables
- Run buildstep before SCM runs
- SSH Agent

构建**Execute shell**

```
Command|export PATH=$PATH:$JAVA_HOME/bin
        export BUILD_ID="dontKillMe"

##kill tomcat
echo "kill tomcat"
for id in `jps -v|grep tomcat|awk '{print $1}'`
do
    kill -9 $id
done

#####start tomcat
cd /apps/svr/tomcat7/bin
./startup.sh

##go into the project folder
cd ${WORKSPACE}
##run permission
chmod +x ./gradlew
./gradlew --refresh-dependencies clean build -x test upload cargoRedeployRemote -Porg.gradle.jvmargs='-Xms512m -Xmx1024m -XX:PermSize=256m -X
.....
```

Execute shell脚本分为3种： web服务； osp服务； web&osp服务

- 启动webapp服务的shell：

```
export PATH=$PATH:$JAVA_HOME/bin
export BUILD_ID="dontKillMe"

##kill tomcat
echo "kill tomcat"
for id in `jps -v|grep tomcat|awk '{print $1}'`
do
    kill -9 $id
done

#####start tomcat
cd /apps/svr/tomcat7/bin
./startup.sh

##go into the project folder
cd ${WORKSPACE}
##run permission
chmod +x ./gradlew
./gradlew --refresh-dependencies clean build upload cargoRedeployRemote
-Porg.gradle.jvmargs='-Xms512m -Xmx1024m -XX:PermSize=256m -XX:MaxPermSize=512m'
```

- 启动osp服务的shell：

```
export PATH=$PATH:$JAVA_HOME/bin
export BUILD_ID="dontKillMe"

##make osp service directory
rm -rf /apps/osp/service/*
mkdir -p /apps/osp/service/

##go into the project folder
cd ${WORKSPACE}
##run permission
chmod +x ./gradlew

##make osp service zip
./gradlew clean ospServiceZip upload
unzip -o ./*-service/build/distributions/*zip -d /apps/osp/service/

##kill osp service
echo "kill osp-service"
for id in `jps -v|grep osp|awk '{print $1}'`
do
```

```

kill -9 $id
done

##start osp service
nohup java -jar -Xms512m -Xmx1024m -XX:PermSize=256m -XX:MaxPermSize=512m
-Dspring.profiles.active=integreatetest /apps/osp/osp-engine-1.1.2/osp-engine-1.1.2-zip/osp-engine.jar -
servicesdir /apps/osp/service -useivy false &

```

- 启动web&osp服务的shell:

```

export PATH=$PATH:$JAVA_HOME/bin
export BUILD_ID="dontKillMe"

##kill tomcat
echo "kill tomcat"
for id in `jps -v|grep tomcat|awk '{print $1}'` 
do
    kill -9 $id
done

##kill osp service
echo "kill osp-service"
for id in `jps -v|grep osp|awk '{print $1}'` 
do
    kill -9 $id
done

#####start tomcat
cd /apps/svr/tomcat7/bin
./startup.sh

##go into the project folder
cd ${WORKSPACE}
##run permission
chmod +x ./gradlew
./gradlew --refresh-dependencies clean build upload cargoRedeployRemote
-Porg.gradle.jvmargs='-Xms512m -Xmx1024m -XX:PermSize=256m -XX:MaxPermSize=512m'

##make osp service directory
rm -rf /apps/osp/service/*
mkdir -p /apps/osp/service/

##unzip osp service zip
unzip -o ./*-service/build/distributions/*zip -d /apps/osp/service/

##start osp service
nohup java -jar -Xms512m -Xmx1024m -XX:PermSize=256m -XX:MaxPermSize=512m
-Dspring.profiles.active=integreatetest /apps/osp/osp-engine-1.1.2/osp-engine-1.1.2-zip/osp-engine.jar -
servicesdir /apps/osp/service -useivy false &

```

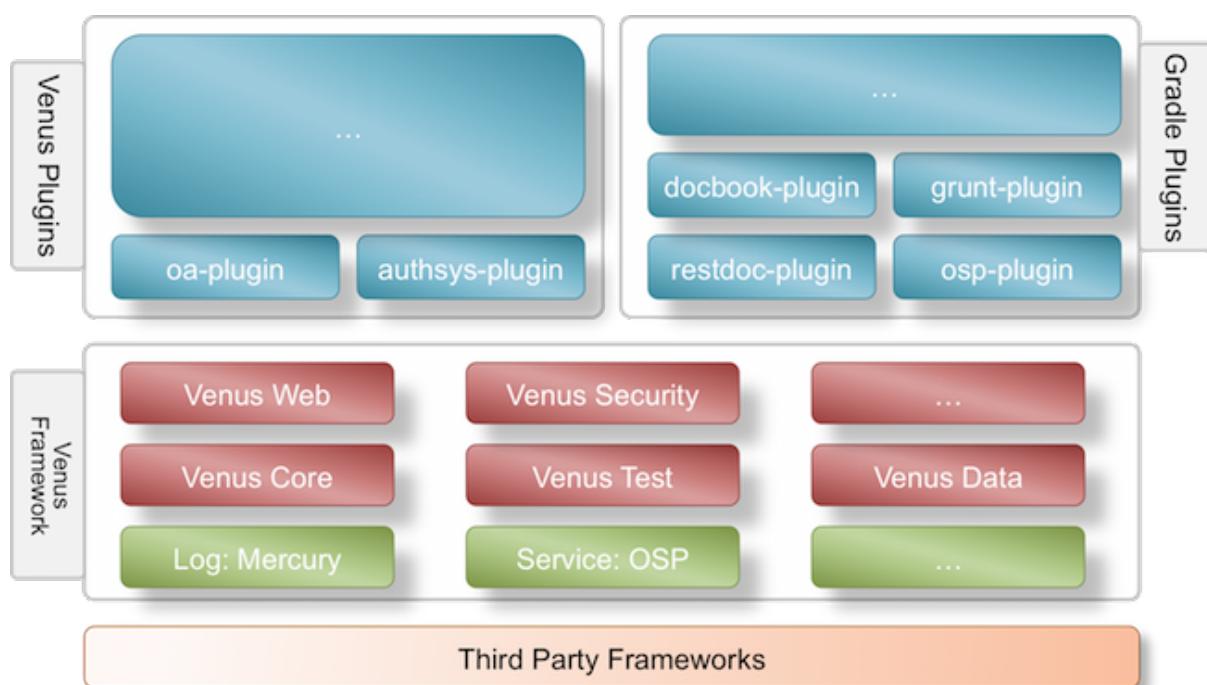
Note

如果需要在gradle运行脚本中增加相关task, 比如增加sonnar, 增加cargo, 增加JVM参数等等, 请参考 Section#36.1, “gradle与Jekins”

Part#IV. #Venus Framework模块

23. Venus概览	113
24. Venus Core	115
24.1. Lombok	115
24.2. Bean Mapping	115
24.3. Exception	116
24.4. Venus Context	117
25. Venus Data Access	119
25.1. 关系数据库	119
venus-data方式	119
venus-jdbc方式	122
25.2. HBase	128
25.3. MongoDB	128
26. Venus Cache	129
26.1. venus-cache典型配置	129
<cache-manager>	129
<cache-cluster>	130
缓存配置的策略参考:	130
<pool-config>	131
27. Venus MQ	133
27.1. venus-mq典型配置	133
27.2. venus-mq配置详细说明	133
<mq-client>	133
<mq-template>	133
<listener-container>	133
<listener>	133
消息发送	134
venus-mq使用到各属性的含义解释	134
28. Venus Web	135
28.1. venus web中的注解与类	135
28.2. controller代码示例	135
29. Venus Security	137
29.1. venus-security典型配置	137
29.2. venus-security配置详细说明	138
<none-security>	138
<custom-filters>	138
<auth-type>	138
<metadata-source>	139
<access-decision-manager>	140
30. Venus Test	141
30.1. venus test中的注解	141

23. #Venus概览



Figure#23.1. #modules of venus

- **Venus Framework:** venus框架核心模块
 - **core:** 提供整个venus framework的基础工具类和核心代码，比如beanmapping, exception定义等
 - **data:** 数据存储（仓库）模块
 - **service:** 服务层，目前主要是提供OSP的支持
 - **web:** 对Spring MVC做进一步封装处理，提供RestApiController和FrameworkController，异常处理等
 - **security:** 封装比较通用的安全框架
 - **test:** 提供测试相关工具，简化测试
 - **log:** 集成mercury
- **Venus Plugins:** 业务相关的框架plugin
 - **oa-plugin:** 与OA单点登入集成相关的plugin项目
 - **authsys-plugin:** 与授权系统集成相关的plugin项目
- **Grdle Plugins:** 提供gradle的plugin, 增强venus框架下构建的能力
 - **docbook-plugin:** 用于asciidoc生成文档
 - **restdoc-plugin:** 用于对webapp动态生成rest documentation
 - **osp-plugin:** 用于支持OSP

- **tools**

- **codegen**: 代码生成器，用于根据表生成项目代码

24. #Venus Core

24. 1#Lombok

Lombok是一个简化POJO对象的一个工具，作为可选工具在项目中使用，在venus framework中的部分示例代码会使用到lombok

Lombok官方地址：[Lombok](#)

Note

建议在项目中使用Lombok，会给POJO对象带来极大的精简和可读性。

24. 2#Bean Mapping

Bean Mapping用于快速，高性能的对两个POJO对象做数据双向映射。

目前底层使用了 [orika](#) 作为beanmapper

- 简单使用

spring中bean的配置：

```
<bean id="orikaBeanMapper" class="com.vip.venus.core.beans.mapping.orika.OrikaBeanMapper">
    <property name="basePackage" value="com.vip.venus.userservice.model.**"></property>
    .....
</bean>
```

代码示例：以ABean与BBean为例，两者做数据双向映射

```
@MapClass("com.vip.venus.core.beans.mapping.BBean") ❶
public class ABean {
    @Getter @Setter
    @MapField("name") ❷
    private String name;

    @Getter @Setter
    @MapField("desc")
    private String desc;

    @Getter @Setter
    @MapField(complexMap="listField[0]=forList0,listField[1]=forList1")
    private List<String> listField;

    @Getter @Setter
    @MapField(complexMap="mapField['first']=forMapFirst")
    private Map<String, String> mapField;

    @Getter @Setter
    @MapField(complexMap="innerBean.innerName=forInnerName")
    private InnerBean innerBean;

    @Getter @Setter
    @MapField("nestBBean")
    private NestABean nestABean;

    @Getter @Setter
    @MapField(complexMap="innerBeans{innerName}=innerBeansMap{key},innerBeans{}=innerBeansMap{value}")
    private List<InnerBean> innerBeans;
}
```

- ① @MapClass 用于指定需要映射的bean
- ② @MapField 可以不指定，默認為两个bean的同名属性映射，用于指定当前属性与@MapClass映射的bean的哪个属性映射，支持简单的名称指定，也支持复杂的List, Array, Map的映射

```
public class BBean {
    @Getter
    @Setter
    private String name;
    @Getter
    @Setter
    private String desc;
    @Getter
    @Setter
    private String forList0;
    @Getter
    @Setter
    private String forList1;

    @Getter
    @Setter
    private String forInnerName;

    @Getter @Setter
    private NestBBean nestBBean;

    @Getter @Setter
    private Map<String, InnerBean> innerBeansMap;
}
```

织入beanMapper的bean:

```
@Autowired
private BeanMapper beanMapper;
```

映射对象:

```
ABean abean = beanMapper.map(bBean, ABean.class);
```

- 进阶功能: 自定义Orika的Converter

实现 ma.glasnost.orika.Converter 接口

在spring bean配置文件中配置ma.glasnost.orika.Converter的实现类bean

系统会自动扫描注册ma.glasnost.orika.Converter的实现类bean（一般双向映射实现ma.glasnost.orika.converter.BidirectionalConverter<S, D> 抽象类）

具体请参考 [自定义converter](#)

24. 3#Exception

`BusinessException: runtime exception`, 用于作为业务异常的基类, 便于统一处理业务异常

`BeanValidationException: 继承自BusinessException, 针对spring mvc的bean validation error的exception`

`LocalizableBusinessException: 继承自BusinessException, 提供国际化支持`

`ValidationException: 继承自LocalizableBusinessException, 用于独立区分validation的异常`

24. 4#Venus Context

`venus-context`是对Spring:`context`的扩展。`venus-context:property-placeholder`主要预定义了相关配置，支持了应用程序对配置中心的使用。

`venus-context 1.3.1`版本变更：

`local-override`默认值改为`true`, 表示环境变量、系统变量覆盖从`file`读取的配置; `false`表示环境变量，系统变量不可用。

`backup-file`备份路径配置改为在`properties`文件中配置或者环境变量配置(例如:`configurationBackupPath=c:/development`)。

具体开发实践请参考 Chapter#10, 使用`venus-context`读取配置文件

属性定义：

`Spring:context`标签包括`property-placeholder`, `property-override`, `annotation-config`, `component-scan`,

`load-time-weaver`, `spring-configured`, `mbean-export`, `mbean-server`等元素。

`venus-context 1.0`版本主要实现对`property-placeholder`的扩展。

`venus-context: property-placeholder`包含以下属性：

Table#24.1.#Attributes of the <`venus-context`> element

Attribute	Description
<code>manager-name</code>	定义 <code>PropertyManager bean</code> 的名称
<code>use-cfgcenter</code>	是否使用配置中心，默认为 <code>true</code>
<code>cfgdef</code>	默认值为 <code>classpath:applicationCfgDef.xml</code> (只有默认值，此属性暂不支持更改)。 <code>applicationCfgDef.xml</code> 定义应用程序需要的配置。应用程序使用到的所有属性必须在该文件中定义(部分预定义属性除外)
<code>location</code>	配置属性文件的位置，Spring通过这些属性文件解析bean配置中的占位符。可以使用 <code>classpath</code> , <code>file</code> 等前缀指定文件位置，也可以使用 <code>zk</code> 作为前缀，指定所需要的配置文件在zookeeper集群中的节点路径。多个配置文件间以逗号分隔，按顺序加载。
<code>properties-ref</code>	指定 <code>Properties</code> 对象的 <code>beanname</code> ，通过该对象解析占位符
<code>file-encoding</code>	<code>file-encoding</code> : 指定用于解析属性文件的编码，默认使用 <code>java.util.properties</code> 默认编码。此属性不适用于解析 <code>xml</code> 文件
<code>order</code>	如果使用了多个 <code>property-placeholder</code> 标签，则使用 <code>order</code> 指定加载顺序
<code>ignore-resource-not-found</code>	如果指定位置没有找到对应属性文件，是否忽略。默认为 <code>false</code> ，当在指定位置没有找到对应属性文件时，程序抛出异常

Attribute	Description
<code>ignore-unresolvable</code>	如果有未能解析的占位符，是否忽略。默认为 <code>false</code> ，如果有key不能解析，程序抛出异常
<code>local-override</code>	本地配置是否覆盖从file读取的配置。默认为 <code>true</code> ，则覆盖

25. #Venus Data Access

Venus Data Access Layer提供各个方面（RDBMS, NOSQL, CACHE）的数据访问的抽象和简化。

25. 1#关系数据库

针对关系数据库，venus data access主要提供了分库/读写分离/分表等相关功能。

有两种实现方式：一种是是基于spring拦截器实现的即venus-data；一种是基于jdbc底层实现的即venus-jdbc

venus-data方式

venus-data方式是基于spring拦截器实现的分库/分表/读写分离，具体实践开发请参考Chapter#11，使用venus-data进行分库/读写分离/分表

venus-datasource配置详细说明

在1.3.0版本中，增加了新的配置项<matrix-datasource>代替<jdbc-matrix>

<matrix-datasource>

配置示例：

```
<venus-datasource:matrix-datasource matrix-
name="dataSource" transactionManager="transactionManager" myBatisSqlSessionFactory="myBatisSqlSessionFactory">
<venus-datasource:pool-configs pool-type="dbcp">
<venus-datasource:pool-config atom-names="write01,read101,read102"> ①
<venus-datasource:property name="maxIdle" value="30"/>
</venus-datasource:pool-config>
<venus-datasource:pool-config atom-names="write02">
<venus-datasource:property name="maxIdle" value="20"/> ②
</venus-datasource:pool-config>
</venus-datasource:pool-configs>

<venus-datasource:repository-sharding strategies-
package="com.vip.venus.data.common.strategy.repository" />
<venus-datasource:table-sharding strategies-package="com.vip.venus.data.common.strategy.table" />
</venus-datasource:matrix-datasource>
```

① 数据库write01, read101, read102采用这个数据连接池的配置

② 数据库write02采用这个数据连接池的配置

属性	是否必填	默认值	可选值	描述
matrix-name	是	#	#	自动在Spring容器中生成的DataSource Bean的id, 事务管理器、MyBatis会话工厂等容器Bean通过该id引用自动生成的DataSource Bean
transactionManager	否	transactionManager#		Spring容器中注册的事务管理器(TransactionManager) id

myBatisSqlSessionFactory

myBatisSqlSessionFactory

Spring容器中
注册的MyBatis
会话工厂
(SqlSessionFactoryBean) id

<pool-config>

用户可以自定义连接池，matrix-datasource的选填元素。如果不定义，采用默认配置

连接池属性有3种配置方式：

方式1：在venus-datasource中直接定义值

```
<venus-datasource:matrix-datasource matrix-name="dataSource">
    <venus-datasource:pool-configs pool-type="dbcp"> ①
        <venus-datasource:pool-config atom-names="write01,read101,read102"> ②
            <venus-datasource:property name="minIdle" value="5" /> ③
            <venus-datasource:property name="maxActive" value="15" />
            <venus-datasource:property name="removeAbandoned" value="true" />
            <venus-datasource:property name="removeAbandonedTimeout" value="300" />
        </venus-datasource:pool-config>

        <venus-datasource:pool-config atom-names="write02">
            <venus-datasource:property name="maxIdle" value="30"/>
        </venus-datasource:pool-config>
    </venus-datasource:pool-configs>
</venus-datasource:matrix-datasource>
```

- ① pool-type：连接池类型，默认配置是c3p0。只能是druid/dbcp/c3p0
- ② atom-names：数据库名称，配置连接池属性。支持数据库通配，write01数据库和名字以read开头的数据库都采用这个数据连接池的配置
- ③ property：配置连接池属性

方式2：venus-datasource中以变量定义，可以直接在代码中已有的properties文件中定义，也可以使用环境变量指定appPropertiesFile=/xxx/config/xxx.properties，然后直接在 xxx.properties文件里直接配置变量值。

applicationContext.xml中：

```
<venus-datasource:matrix-datasource matrix-name="dataSource">
    <venus-datasource:pool-configs pool-type="dbcp">
        ...
        <venus-datasource:pool-config atom-names="write02">
            <venus-datasource:property name="initialPoolSize" value="${datasource.initialpoolsize}"/>
        </venus-datasource:pool-config>
        ...
    </venus-datasource:pool-configs>
</venus-datasource:matrix-datasource>
```

那么我们可以直接在运行环境下对应的properties文件中定义如下：

```
datasource.initialpoolsize=5
```

也可以另外写个app.properties如上定义，但是需要使用环境变量指定appPropertiesFile=/xxx/config/app.properties(app.propertiesd的目录)

```
##3# venus-datasource#####
```

applicationContext.xml中：

```
<venus-datasource:matrix-datasource matrix-name="dataSource">
    <venus-datasource:pool-configs pool-type="dbcp">
        .....
        <venus-datasource:pool-config atom-names="write02">
            <venus-datasource:property name="initialPoolSize" value="${datasource.initialpoolsize}"/>
        </venus-datasource:pool-config>
        .....
    </venus-datasource:pool-configs>
</venus-datasource:matrix-datasource>
```

applicationCfgDef.xml中：

```
<cfgdef group="com.vip.venus" name="venustest-webapp" version="1.0.0">
    <itemgroup name="all">
        <item name="#####"
#" key="datasource.initialpoolsize" defaultValue="5" dataType="integer" desc="##### hotReload="false"/>
    </itemgroup>
</cfgdef>
```

把applicationCfgDef.xml文件导入到配置中心，就可以在配置中心按需要修改值。

<repository-sharding>

配置示例：

```
<venus-datasource:matrix-datasource matrix-
name="dataSource" transactionManager="transactionManager" myBatisSqlSessionFactory="myBatisSqlSessionFactory">
    .....
    <venus-datasource:repository-sharding strategies-
package="com.vip.venus.data.common.strategy.repository" order="1"/>
    .....
</venus-datasource:matrix-datasource>
```

属性/元素	描述
strategies-package	指定分库策略所在包，框架会自动进行扫描
order	拦截器的执行顺序，默认值100.

<table-sharding>

配置示例：

```
<venus-datasource:matrix-datasource matrix-
name="dataSource" transactionManager="transactionManager" myBatisSqlSessionFactory="myBatisSqlSessionFactory">
    .....
    <venus-datasource:table-sharding strategies-
package="com.vip.venus.data.common.strategy.table" order="1"/>
    .....
</venus-datasource:matrix-datasource>
```

属性/元素	描述
strategies-package	指定分表策略所在包，框架会自动进行扫描注册
order	拦截器的执行顺序，默认值100.

<连接池默认值>

- C3P0: 属性名(默认值)

```
driverClass(com.mysql.jdbc.Driver), acquireIncrement(2), idleConnectionTestPeriod(10),
initialPoolSize(5), maxIdleTime(60), maxPoolSize(10), minPoolSize(3),
checkoutTimeout(30000)
```

- DBCP: 属性名 (默认值)

```
driverClassName(com.mysql.jdbc.Driver), initialSize(5), maxTotal(10), maxIdle(5),
minIdle(3), maxWaitMillis(60000), validationQuery(SELECT 1 FROM DUAL)
```

- Druid: 属性名 (默认值)

```
driverClassName(com.mysql.jdbc.Driver), initialSize(5), maxActive(10), minIdle(3),
validationQuery(SELECT 1 FROM DUAL), testWhileIdle(true),
timeBetweenEvictionRunsMillis(60000)
```

venus-jdbc方式

venus-jdbc方式是基于jdbc底层实现的分库/分表/读写分离，具体开发实践请参考Chapter# 12，使用venus-jdbc进行分库/读写分离/分表

venus-jdbc的功能列表

分库

Venus分片组件分析SQL语句，并根据分库规则，确定SQL语句最终在一个或多个数据库中执行。

如果在多个数据库中执行，查询语句的结果会被聚合后返回给用户，删除和修改语句的结果会被相加后返回给用户。

Note

INSERT语句目前只能在一个数据库中执行，不支持批量新增

分表

Venus分片组件分析SQL语句，并根据分表规则，确定SQL语句最终在一个或多个分表中执行。

如果在多个分表中执行，查询语句的结果会被聚合后返回给用户，删除和修改语句的结果会被相加后返回给用户。

Note

INSERT语句目前只能在一个分表中执行，不支持批量新增

读写分离

Venus分片组件分析SQL语句，根据SQL的类型（查询或增、删、改），确定SQL是在主库上执行，还是在从库上执行。

如果从库有多个，分片组件提供Random和RoundRobin两种负载均衡策略。

持久层支持

目前支持MyBatis、Hibernate3、Hibernate4、JDBC。

但由于codegen配套的功能还没有实现，目前在开发实践中只用MyBatis。

数据库支持

目前只支持mysql数据库

事务支持

支持单库事务；分布式事务不能保持强一致性。目前分布式事务采用两阶段执行方式，即分为执行阶段和提交阶段。

执行阶段出错，整个分布式事务可以回滚；提交阶段出错，整个分布式事务不可以回滚。

强制读

读写分离的场景下，写事务在主库上进行，读事务在从库上进行。

如果写事务中包含耗时的读操作，且该操作不依赖当前事务中的写操作，则读操作完全可以迁移到从库上进行，减少主库的压力，提高系统整体性能。

分片组件提供Hint注解实现该功能，MyBatis和JDBC场景下，只需要在SQL语句前添加“`/* hint force_read */`”这个注解即可；

Hibernate场景下需要开启“`hibernate.use_sql_comments=true`”，同时“`Query.setComment("hint force_read");`”即可。

Note

如果用户标记某个Service方法的事务为`ReadOnly=true`，但在实际开发过程中这个Service中出现了写方法，

分片组件并不会提示用户，写方法之后的操作都会在主库上进行。

规则引擎

分库和分表路由时，需Venus分片组件提供规则引擎便于用户自定义分库和分表规则。

规则需符合groovy语法且在`properties`中按照JSON格式进行配置。具体请参考开发实践Section# 12.5，“`properties`文件配置”

Note

分表场景下，需要给分表设定一个虚拟表名，例如分表`user_0`、`user_1`、`user_2`，则虚拟表名可以为`user`。

使用Venus分片组件进行分表操作时，SQL语句和配置规则都需要使用这个虚拟表名。

查询聚合

使用分库或分表功能时，用户提交的查询语句可能会触发数据聚合操作，单条查询语句可能会转化为多条查询语句到多个分库或者分表中执行，

由此产生的多个查询结果最终会聚合成一个查询结果返回给用户。

- 默认聚合

默认情况下，多个分库或分表的查询结果会直接聚合成一个查询结果返回给用户。

例如：SELECT age FROM user WHERE id IN (1, 2, 3, 4) ORDER BY age LIMIT 10, 5,

假设有两个分库（分库1和分库2），每个分库又有两张分表（分表1和分表2），一共要查询四张表（分库1中分表1和分表2，分库2中分表1和分表2）。

理论上应该从四张表中筛选出id为1、2、3和4的数据，然后将筛选出的数据汇总在一起并进行排序，最后从中提取次序为11到15的数据。

但实现该功能非常复杂，因此目前只是在四张分表中分别执行该条语句，然后将四个查询结果直接汇总在一起返回给用户。

Note

使用Venus分片组件，上面语句返回的数据从全局来看可能不是有序的，且返回的条数可能超过5条，具体条数取决于四张分表的执行结果。

假设如上语句在四张表中分别执行，分库1中分表1返回{10, 13, 17, 19, 21}，分库1中分表2返回{11, 12, 13, 14, 15}，

分库2中分表1返回{8, 10, 11, 15, 17}，分库2中分表2返回{9, 11, 13, 17, 29}，

最终返回的结果并不是{8, 9, 10, 10, 11}，而是{10, 13, 17, 19, 21, 11, 12, 13, 14, 15, 8, 10, 11, 15, 17, 9, 11, 13, 17, 29}，按照数据库和表的字典顺序进行排列。

- 函数聚合

目前版本支持MAX、MIN、SUM和COUNT 4个统计函数的数据聚合。

例如：SELECT COUNT(*) FROM user WHERE id IN (...),

假设有两个分库（分库1和分库2），每个分库又有两张分表（分表1和分表2），一共要查询四张分表（分库1中分表1和分表2，分库2中分表1和分表2），

得到四张分表的查询结果后，分片组件会根据函数类型对查询结果进行合并统计并返回结果给用户。

Note

假设如上语句在四张表中分别执行，分库1中分表1返回10，分库1中分表2返回20，分库2中分表1返回5，分库2中分表2返回15，经过分片组件聚合后的结果是50。

暂不支持

- 分片字段只支持单字段，不支持多字段
- 不支持子查询
- 不支持>、>=、<、#、NOT IN、LIKE、NOT LIKE、BETWEEN等关系运算
- 不支持GROUP BY、HAVING等操作
- 不支持批量新增操作

- 不支持CLOB、BLOB等字段类型
- 不支持MAX、MIN、SUM、COUNT以外的MySQL函数
- 不支持存储过程
- 不支持强一致性的分布式事务
- 不支持SavePoint，自然不支持Spring事务的Nested传播行为
- 没有对DataSource进行深度封装，只是对用户定义的DataSource进行了浅包装，因此需要用户根据实际场景自定义DataSource相关配置

venus-jdbc的SQL支持

- SELECT语句支持

```
SELECT
  { * | select_expr } [ , select_expr ... ]
  FROM table_references
  WHERE where_condition
  [ORDER BY col_name [ASC | DESC], ...]
  [LIMIT {[offset], row_count}]
```

以下是对这条语句的解释说明：

```
select_expr:
  col_name [[AS] alias] |
  MAX(col_name) [[AS] alias] |
  MIN(col_name) [[AS] alias] |
  AVG(col_name) [[AS] alias] |
  SUM(col_name) [[AS] alias] |
  COUNT(* | col_name) [[AS] alias]

table_references:
  table_expr [, {table_expr | join_table} ...]

table_expr:
  tbl_name [[AS] alias]

join_table:
  table_expr [INNER] JOIN table_expr ON col_name = col_name |
  table_expr {LEFT | RIGHT} [OUTER] JOIN table_expr ON col_name = col_name

where_condition#
  expr [{OR | AND} expr ...]

expr:
  {col_name = col_name} |
  {col_name = ?} |
  {col_name = String} |
  {col_name = Number} |
  {col_name IN {?} ...} |
  {col_name IN {String ...}} |
  {col_name IN {Number ...}} |
```

- INSERT语句支持

```
INSERT
  [INTO] tbl_name
  (col_name [, col_name ...])
  {VALUES | VALUE}
  (insert_expr [, insert_expr ...])
```

以下是对这条语句的解释说明：

```
insert_expr:
? |
String |
Number
```

Note

INSERT语句目前只能在一个分表/分库中执行，不支持批量新增

- UPDATE语句支持

```
UPDATE
table_reference
SET set_expr [, set_expr ...]
WHERE where_condition
```

以下是对这条语句的解释说明：

```
set_expr:
{col_name = col_name} |
{col_name = ?} |
{col_name = String} |
{col_name = Number} |
```

- DELETE语句支持

```
DELETE
FROM tbl_name
WHERE where_condition
```

venus-jdbc配置详细说明

<allinone-datasource>

配置示例：

```
<venus-jdbc:allinone-datasource matrix-
name="dataSource" transactionManager="transactionManager" myBatisSqlSessionFactory="myBatisSqlSessionFactory">
<venus-jdbc:pool-configs pool-type="dbcp">
<venus-jdbc:pool-config atom-names="write01,read*">
<venus-jdbc:property name="maxIdle" value="30"/> ①
</venus-jdbc:pool-config>
<venus-jdbc:pool-config atom-names="write02">
<venus-jdbc:property name="maxIdle" value="20"/> ②
</venus-jdbc:pool-config>
</venus-jdbc:pool-configs>
</venus-jdbc:allinone-datasource>
```

- ① atom-names：数据库名称，配置连接池属性。支持数据库通配，write01数据库和名字以read开头的数据库都采用这个数据连接池的配置
- ② 数据库write02采用这个数据连接池的配置

属性	是否必填	默认值	可选值	描述
matrix-name	是	#	#	自动在Spring容器中生成的DataSource Bean的id，事务管理器、MyBatis会话工厂等容器Bean通

			通过该id引用自动生成的DataSource Bean
transactionManager	否	transactionManager#	Spring容器中注册的事务管理器(TransactionalManager) id
myBatisSqlSessionFactory	否	myBatisSqlSessionFactory	Spring容器中注册的MyBatis会话工厂(SqlSessionFactoryBean) id

<pool-config>

用户可以自定义连接池，allinone-datasource的选填元素。如果不定义，采用默认配置

连接池属性有三种配置方式：

方式1：直接在venus-datasource中配置值

代码如下：

```
<venus-jdbc:allinone-datasource matrix-name="dataSource">
    <venus-jdbc:pool-configs pool-type="dbcp"> ①
        <venus-jdbc:pool-config atom-names="write01,read01,read102"> ②
            <venus-jdbc:property name="minIdle" value="5" /> ③
            <venus-jdbc:property name="maxActive" value="15" />
            <venus-jdbc:property name="removeAbandoned" value="true" />
            <venus-jdbc:property name="removeAbandonedTimeout" value="300" />
        </venus-jdbc:pool-config>
        <venus-jdbc:pool-config atom-names="write02">
            <venus-jdbc:property name="maxIdle" value="30"/>
        </venus-jdbc:pool-config>
    </venus-jdbc:pool-configs>
</venus-jdbc:allinone-datasource>
```

① pool-type：连接池类型，默认配置是c3p0。只能是druid/dbcp/c3p0

② atom-names：数据库名称。write01, read01, read102数据库用下面配置的连接池属性

③ property：配置连接池属性

方式2：venus-datasource中以变量定义，在properties文件定义变量的值

applicationContext.xml中：

```
<venus-jdbc:allinone-datasource matrix-name="dataSource">
    <venus-jdbc:pool-configs pool-type="dbcp">
        .....
        <venus-jdbc:pool-config atom-names="write02">
            <venus-jdbc:property name="initialPoolSize" value="${datasource.initialpoolsize}"/>
        </venus-jdbc:pool-config>
        .....
    </venus-jdbc:pool-configs>
</venus-jdbc:allinone-datasource>
```

properties文件中：

```
datasource.initialpoolsize=5
```

方式3：venus-datasource中以变量定义，通过配置中心定义变量的值

applicationContext.xml中：

```
<venus-jdbc:allinone-datasource matrix-name="dataSource">
    <venus-jdbc:pool-configs pool-type="dbcp">
        .....
        <venus-jdbc:pool-config atom-names="write02">
            <venus-jdbc:property name="initialPoolSize" value="${datasource.initialpoolsize}"/>
        </venus-jdbc:pool-config>
        .....
    </venus-jdbc:pool-configs>
</venus-jdbc:allinone-datasource>
```

applicationCfgDef.xml中：

```
<cfgdef group="com.vip.venus" name="venustest-webapp" version="1.0.0">
    <itemgroup name="all">
        <item name="#####">
            #<!-- key="datasource.initialpoolsize" defaultValue="5" dataType="integer" desc="#####" hotReload="false"-->
        </itemgroup>
    </cfgdef>
```

把applicationCfgDef.xml文件导入到配置中心，就可以在配置中心按需要修改值。

连接池默认值>

- C3P0: 属性名(默认值)

```
driverClass(com.mysql.jdbc.Driver), acquireIncrement(2), idleConnectionTestPeriod(10),
initialPoolSize(5), maxIdleTime(60), maxPoolSize(10), minPoolSize(3),
checkoutTimeout(30000)
```

- DBCP: 属性名(默认值)

```
driverClassName(com.mysql.jdbc.Driver), initialSize(5), maxTotal(10), maxIdle(5),
minIdle(3), maxWaitMillis(60000), validationQuery(SELECT 1 FROM DUAL)
```

- Druid: 属性名(默认值)

```
driverClassName(com.mysql.jdbc.Driver), initialSize(5), maxActive(10), minIdle(3),
validationQuery(SELECT 1 FROM DUAL), testWhileIdle(true),
timeBetweenEvictionRunsMillis(60000)
```

25. 2#HBase

25. 3#MongoDB

26. #Venus Cache

Venus Cache 不仅是对Spring Cache的封装和集成，而且还在Spring Cache的基础上进行增强，提供了一整套Venus自己的注解，同时无缝支持redis, memcached，统一上层API，使得缓存服务简单易用。另外通过配置中心，让开发者方便的配置缓存集群。

具体使用配置中心配置queue和channel请参考Chapter#13，使用venus-cache进行缓存操作

26. 1#venus-cache典型配置

```
<venus-cache:cache-manager name="redisCacheManager">
    <venus-cache:caches>
        <venus-cache:cache-cluster name="orders" cluster-name="redis-cluster-order" type="redis" key-
prefix="childkeyprefix" expiration="1000" template-name="redis01wRedisTemplate"> ①
            <venus-cache:pool-config> ②
                <property name="maxTotal" value="200" />
                <property name="maxIdle" value="50" />
                <property name="timeBetweenEvictionRunsMillis" value="30000" />
                <property name="minEvictableIdleTimeMillis" value="60000" />
                <property name="maxWaitMillis" value="30000" />
            </venus-cache:pool-config>

            <venus-cache:serializer-config> ③
                <venus-cache:key-serializer class="com.vip.venus.data.cache.redis.serializer.VenusJdkSerializer"/>
                <venus-cache:value-serializer type="jdk"/>
                <venus-cache:hashkey-serializer type="jdk"/>
                <venus-cache:hashvalue-serializer type="jdk"/>
            </venus-cache:serializer-config>
        </venus-cache:cache-cluster>

        <venus-cache:cache-cluster name="users" cluster-name="redis-cluster-user" type="redis" key-
prefix="childkeyprefix" expiration="1000" template-name="redis02wRedisTemplate"> ④
            <venus-cache:pool-config> ⑤
                <property name="maxTotal" value="200" />
                <property name="maxIdle" value="50" />
                <property name="timeBetweenEvictionRunsMillis" value="30000" />
                <property name="minEvictableIdleTimeMillis" value="60000" />
                <property name="maxWaitMillis" value="30000" />
            </venus-cache:pool-config>

            <venus-cache:serializer-config> ⑥
                <venus-cache:key-
                    serializer class="com.vip.venus.data.memcached.transcoder.VenusMemcachedSerializingTranscoder"/>
                <venus-cache:value-serializer type="jdk"/>
            </venus-cache:serializer-config>
        </venus-cache:cache-cluster>
    </venus-cache:caches>
</venus-cache:cache-manager>
```

- ①④ template-name配置了RedisTemplate bean的名字，可以在程序中根据此template-name注入对应的RedisTemplate Bean，然后直接进行一些高级redis命令操作。key-prefix: 存放到缓存中的key前缀。 expiration: 缓存失效时间，时间单位为秒；
- ②⑤ pool-config配置了缓存连接池的参数，包括最大连接数等。
- ③⑥ 序列化方式：Redis目前支持jdk和string方式，还可以自己自定义序列化类，自定义的序列化类必须实现VenusRedisSerializer接口；Memcached目前支持jdk方式，还可以自己自定义序列化类，自定义的序列化类必须继承AbstractSerializingTranscoder抽象类， === venus-cache配置详细说明

<cache-manager>

cache-manager 是对多个不同缓存的聚合。

当cache manager name别定义为“cacheManager”时，则被spring cache 默认使用

<cache-cluster>

cluster-name: 由cluster name来生成properties的key, 用来寻找cache cluster的json配置,
cache-cluster有两种配置方式: 1 本地配置 2 配置中心配置

本地配置:

一般配置在当前运行环境的src/main/resources下的application-development.properties文件中
配置示例:

如果是redis:

```
/resource/Cache/Redis/order-redis-cluster={"loadBalance":"RoundRobin","name":"order-redis-cluster","nodes":[{"host":"127.0.0.1","logicName":"redis-node-01","port":6379,"weight":1,"state":"online"}],"proxy":true,"sharding":"","type":"Redis","support-type":"CLUSTER"}
```

如果是Memcached:

```
/resource/Cache/Memcached/order-redis-cluster={"loadBalance":"RoundRobin","name":"order-redis-cluster",\ "nodes":[{"host":"127.0.0.1","logicName":"redis-node-01","port":6379,"state":"online"}],"proxy":true,"sharding":"","type":"Memcached"}
```

参数说明:

1. proxy: 服务端启用代理时配置为“proxy”:“true”
2. weight: Memcached在客户端进行分片时, 该配置决定在某个物理节点上分布的虚拟节点的数量, 该物理节点配置的值越高, 分布的虚拟节点数量越多
3. sharding: 指定Memcached缓存在客户端采用的分片方式, 配置为“sharding”:“KETAMA”
4. support-type: 该配置仅对Redis集群提供支持, 当使用Redis集群时, 配置为“support-type”:“CLUSTER”
5. type: 该配置指定支持缓存的类型, 如需要对redis, 或memcached提供支持时, 配置为“type”:“Redis”或者“type”:“Memcached”

配置中心配置:

1. 如果是redis, 则生成的properties的key是’/resource/Cache/Redis/{cluster-name}’
2. 如果是Memcached, 则生成的properties的key是’/resource/Cache/Memcached/{cluster-name}’

缓存配置的策略参考:

针对Redis缓存(type=“Redis”):

- 同时配置集群、代理、分片; 例: “proxy”:“true”, “sharding”:“KETAMA”, type=“Redis”, “support-type”:“CLUSTER”
 1. 优先走集群;
 2. 其次走代理;

3. 暂不支持分片（报错提示）；

4. 最后单点直连；

- 仅配置集群，走集群模式；例：“proxy”=”false”，“sharding”=””，“type”=”Redis”，“support-type”=”CLUSTER”
- 仅配置代理，走代理；例：“proxy”=”true”，“sharding”=””，“type”=”Redis”，“support-type”=””
- 仅配置分片，报错提示不支持分片；例：“proxy”=”false”，“sharding”=”KETAMA”，“type”=”Redis”，“support-type”=””
- 无显式配置，默认是单点直连，保证只有一个节点，如有多个节点，会报错提示配置一个策略，比如集群，代理，分片；例：“proxy”=”false”，“sharding”=””，“type”=”Redis”，“support-type”=””

针对Memcached缓存(type=”Memcached”):

- 同时配置集群、代理、分片；例：“proxy”=”true”，“sharding”=”KETAMA”，“type”=”Memcached”，“support-type”=”CLUSTER”
- 1. 优先走代理；
- 2. 其次走分片；
- 3. 最后单点直连；
- 仅配置集群，报错提示不支持集群；例：“proxy”=”false”，“sharding”=””，“type”=”Memcached”，“support-type”=”CLUSTER”
- 仅配置代理，走代理；例：“proxy”=”true”，“sharding”=””，“type”=”Memcached”，“support-type”=””
- 仅配置分片，走分片；例：“proxy”=”false”，“sharding”=”KETAMA”，“type”=”Memcached”，“support-type”=””
- 无显式配置，默认是单点直连，保证只有一个节点，如有多个节点，会报错提示配置一个策略，比如集群，代理，分片；例：“proxy”=”false”，“sharding”=””，“type”=”Memcached”，“support-type”=””

Note

关于cache-cluster会先从本地properties文件中找对应的配置，然后从配置中心zookeeper中找对应的配置。如果都存在，以配置中心的为准

<pool-config>

pool-config配置了缓存连接池的参数，包括最大连接数等。如果不配置，则采用程序的默认值 .redis 连接池的配置属性

redis连接池的配置属性

属性	默认值	可选值	描述
maxTotal	#		最大连接数

属性	默认值	可选值	描述
maxIdle	#		最大空闲连接数
timeBetweenEvictionRunsMillis	30000		剔除运行间隔时间
minEvictableIdleTimeMillis	60000		最短剔除空闲时间
maxWaitMillis	#		最大等待时间

27. #Venus MQ

Venus MQ 是对Vms的封装和集成，主要提供了易用的消息服务。并在配置中心的基础上，让开发者方便的配置消息服务。

具体实践开发请参考Chapter#14，使用配置中心配置channel和queue

27. 1#venus-mq典型配置

```
<venus-mq:mq-client id="mqclient" failFastEnable="true" autoCommit="true"> ①
    <venus-mq:rabbitmq confirmable="true" confirmTimeout="2000" priority="1" durable="true" />
</venus-mq:mq-client>

<venus-mq:mq-template id="mqtemplate" mq-client="mqclient" />②

<bean id="messageProcessor1" class="com.vip.venus.mq.bean.MessageProcessor1" /> ③
<bean id="messageConverter" class="com.vip.venus.mq.bean.SimpleVenusMessageConverter2" /> ④
<bean id="errorHandler" class="com.vip.venus.mq.bean.SimpleErrorHandler" /> ⑤

<venus-mq:listener-container id="mqcontainer1" mq-client="mqclient" error-handler="errorHandler"> ⑥
    <venus-mq:listener id="listener1" ref="messageProcessor1" method="listen" message-
converter="messageConverter" queue-names="queue.venusmq.lusong" /> ⑦
</venus-mq:listener-container>
```

- ① mq-client配置了mq使用的vms client
- ② mq-template配置了VenusMqTemplate bean的名字，可以在程序中根据此id获得对应的 VenusMqTemplate Bean，然后进行消息发送。
- ③ 自定义消息处理bean，需要定义listener中指定的方法，必须要定义方法。如public void listen(Object message) {}
- ④ 自定义消息转换器，需要实现接口VenusMessageConverter。默认可以不配置。
- ⑤ 自定义消息接收错误处理器，需要实现接口ConsumeErrorHandler。默认可以不配置。
- ⑥ listener-container配置了订阅消息的处理。
- ⑦ 配置了订阅消息的处理方法和订阅的消息queue，可以配置多个queue

27. 2#venus-mq配置详细说明

<mq-client>

mq-client配置了mq使用的vms client，同时可以配置后端mq集群的特性，如

```
<venus-mq:rabbitmq confirmable="true" confirmTimeout="2000" priority="1" durable="true" />
```

<mq-template>

mq-template 指定消息发送的mq-client，及指定发送消息的channel及routing-key属性。

<listener-container>

listener-container 指定消息订阅的客户端，定义多个listener及消息的处理bean。

<listener>

listener 指定消息订阅的queue名字，定义消息处理bean及处理方法名，可以自定义消息转换器，及指定prefetchCount属性。

消息发送

```

@Autowired
private VenusMqTemplate mqtemplate;

Message message = new Message(data.getBytes("utf-8"));
mqtemplate.send(channelName3, "", message);

```

venus-mq使用到各属性的含义解释

属性	默认值	集群支持	描述
failFastEnable	RabbitMq, kafka都支持	RabbitMq, kafka	快速失败机制
autoCommit	RabbitMq, kafka都支持	RabbitMq, kafka	默认是会自动commit消费进度到zk的，设为false的话每消费消息一条，会自动提交消费进度
prefetchCount	RabbitMq支持，kafka不支持	RabbitMq支持	订阅消息时，RabbitMq预取消息数
confirmable	RabbitMq支持，kafka不支持	RabbitMq支持	发送消息时，如果confirmTimeout内没有返回confirm ack，则进行重试。如果confirmTimeout设置为负数，则一直阻塞直到返回结果
confirmTimeout	RabbitMq支持，kafka不支持	RabbitMq支持	发送消息时，如果confirmTimeout内没有返回confirm ack，则进行重试。如果confirmTimeout设置为负数，则一直阻塞直到返回结果
routing-key	RabbitMq, kafka都支持	RabbitMq, kafka	消息路由的key
channel	RabbitMq, kafka都支持	RabbitMq, kafka	发送消息时指定的channel，可以不配，在template调用时传进去
priority	RabbitMq支持，kafka不支持	RabbitMq支持	发送消息的优先级，数字大的表示优先级高。在同一个topic中，优先级高的消息先于优先级低的消息被消费。
durablie	RabbitMq支持，kafka不支持	RabbitMq支持	消息的持久化
ttl #	RabbitMq支持，kafka不支持	RabbitMq支持	消息的生存时间

28. #Venus Web

28. 1#venus web中的注解与类

`@RestController`: 与普通Spring MVC的controller的用法无差异；区分Spring MVC的Controller (view与REST区分开)；

便于异常处理和整体控制rest接口的response

`@FrameworkController`: 提供框架级别的controller, URL可被普通controller覆盖；

venus web框架默认提供的/_health_check的rest接口, 即使用`@FrameworkController`来声明

`ResponseEnvelope`: REST接口返回数据的规范化

28. 2#controller代码示例

```

@RestController ❶
@RequestMapping("/demo") ❷
public class UserRestApiController {
    private final Logger logger = LoggerFactory.getLogger(UserRestApiController.class);

    @Autowired
    private BeanMapper beanMapper;

    @Autowired
    private UserService userService;

    @RequestMapping(value = "/sysmgmt/user/{id}", method = RequestMethod.GET) ❸
    public ResponseEntity<ResponseEnvelope<UserVO>> getUserById(@PathVariable Long id){
        UserModel userModel = userService.findByPrimaryKey(id); ❹
        UserVO userVO = beanMapper.map(userModel, UserVO.class); ❺
        ResponseEnvelope<UserVO> responseEnv = new ResponseEnvelope<UserVO>(userVO); ❻
        return new ResponseEntity<>(responseEnv, HttpStatus.OK); ❼
    }

    @RequestMapping(value = "/sysmgmt/user", method = RequestMethod.POST)
    public ResponseEntity<ResponseEnvelope<Integer>> createUser(@RequestBody UserVO userVO){
        UserModel userModel = beanMapper.map(userVO, UserModel.class);
        Integer result = userService.create(userModel);
        ResponseEnvelope<Integer> responseEnv = new ResponseEnvelope<Integer>(result);
        return new ResponseEntity<>(responseEnv, HttpStatus.OK);
    }

    @RequestMapping(value = "/sysmgmt/user/{id}", method = RequestMethod.DELETE)
    public ResponseEntity<ResponseEnvelope<Integer>> deleteUserByPrimaryKey(@PathVariable Long id){
        Integer result = userService.deleteByPrimaryKey(id);
        ResponseEnvelope<Integer> responseEnv = new ResponseEnvelope<Integer>(result);
        return new ResponseEntity<>(responseEnv, HttpStatus.OK);
    }

    @RequestMapping(value = "/sysmgmt/user", method = RequestMethod.PUT)
    public ResponseEntity<ResponseEnvelope<Integer>> updateUserByPrimaryKeySelective(@RequestBody UserVO userVO){
        UserModel userModel = beanMapper.map(userVO, UserModel.class);
        Integer result = userService.updateByPrimaryKeySelective(userModel);
        ResponseEnvelope<Integer> responseEnv = new ResponseEnvelope<Integer>(result);
        return new ResponseEntity<>(responseEnv, HttpStatus.OK);
    }
}

```

❶ 使用`@RestController`来标明当前controller类型

- ② 根url路径
- ③ 当前method的相关配置
- ④ 调用服务的实现，查询user，获得模型(POJO)对象
- ⑤ 通过bean mapper把model转换为VO
- ⑥ 创建response的信封，将VO作为数据
- ⑦ 返回数据

29. #Venus Security

Venus Security 是对Spring security的封装和集成，主要提供了易用的认证和授权服务，引入realm来支持安全资源配置的多途径多方式加载。

Venus security采用了插件机制，易于集成外部系统，比如venus-plugin-security-authsys

具体实践开发请参考Chapter#16，使用venus-security进行认证和授权

29. 1#venus-security典型配置

```

<venus-security:config>
    <venus-security:none-security> ①
        <venus-security:pattern value="/_health_check" />
        <venus-security:pattern value="/nonsecure/**" />
    </venus-security:none-security>

    <venus-security:custom-filters> ②
        <venus-security:custom-filter ref="xxxyFilter" after="CAS_FILTER" />
        <venus-security:custom-filter ref="aabbFilter" before="FILTER_SECURITY_INTERCEPTOR" />
    </venus-security:custom-filters>

    <venus-security:auth-type> ③
        <venus-security:cas>
            <property name="casServerUrl" value="${cas.server.url}" />
            <property name="casLoginUrl" value="${cas.login.url}" />

            <property name="clientServerUrl" value="${client.server.url}" />
            <property name="clientServiceUrl" value="${client.service.url}" />

            <property name="defaultTargetUrl" value="${authentication.target.url}" />
            <property name="defaultFailureUrl" value="${authentication.failure.url}" />

            <property name="accessDeniedUrl" value="${access.denied.url}" />

            <property name="logoutSuccessUrl" value="${logout.success.url}" />

            <property name="proxyReceptorUrl" value="${proxy.receptor.url}" />
            <property name="proxyCallbackUrl" value="${proxy.callback.url}" />

            <property name="casAuthProviderKey" value="${cas.auth.provider.key}" />
            <property name="casSendRenew" value="${cas.send.renew}" />
        </venus-security:cas>
    </venus-security:auth-type>

    <venus-security:metadata-source>
        <venus-security:realms> ④
            <venus-security:simple-realm>
                <venus-security:intercept-url pattern="/userapi/**" method="get" access="user_read" />
                <venus-security:intercept-url pattern="/userapi/**" method="post" access="user_write" />
            </venus-security:simple-realm>
        </venus-security:realms>
    </venus-security:metadata-source>

    <venus-security:access-decision-
manager class="org.springframework.security.access.vote.AffirmativeBased">
        <venus-security:voters>
            <bean class="org.springframework.security.access.vote.AuthenticatedVoter" />
            <ref bean="authSysVoter" /> ⑤
        </venus-security:voters>
    </venus-security:access-decision-manager>

    <venus-security:user-details-
manager class="com.vip.venus.security.core.userdetails.VenusUserDetailsByNameServiceWrapper"> ⑥
        <venus-security:user-details class="com.vip.venus.plugin.security.oa.OAUserDetailsManager" />
    </venus-security:user-details-
manager>

```

```

</venus-security:user-details-manager>
</venus-security:config>

<bean id="authSysVoter" class="com.vip.venus.plugin.security.authsys.AuthSysRoleVoter">
    <property name="useAuthSys" value="false" />
    <property name="superAdmins">
        <set>
            <value>sheldon.zhang</value>
            <value>tony.wang</value>
        </set>
    </property>
</bean>

```

- ① none-security 配置了不需要认证和授权的url pattern
- ② custom-filters 可以让开发者把自定义的filter放到 spring security filter chain的对应位置中
- ③ auth-type 可以配置认证类型。现在仅支持cas类型，以后会支持更多的认证类型
- ④ realms 具体资源权限控制的元数据来源，配置可以通过实现com.vip.venus.security.access.intercept.RequestToAttributesMapRealm接口来自定义获取方式，比如从关系型数据库，从缓存，从文件等等。
- ⑤ 配置voter。默认加载AuthenticatedVoter和授权系统Voter，可以通过实现org.springframework.security.access.AccessDecisionVoter<S>接口来配置个性化voter。
- ⑥ user-details-manager 可以配置具体userdetails的实现类， 默认使用OAUserDetails，通过继承org.springframework.security.core.userdetails.UserDetailsService做个性化实现。

Note

使用默认voter(授权系统voter)的时候，项目需要主动申明依赖com.vip.venus.plugin:authsys-plugin的jar

使用默认UserDetails(OA系统的OAUserDetails)的时候，项目需要主动申明依赖com.vip.venus.plugin:oa-plugin的jar

29. 2#venus-security配置详细说明

<none-security>

用于配置不需要认证和授权的url;默认值包括“/_health_check”和“/restdoc/**”

<custom-filters>

custom-filters 用于开发者配置自定义filter；

custom-filter 属性有 ref(自定义filter的id), after, before, position；

after, before, position中是spring security filter chain中 filter的名字，具体可以参考[spring security custom filter](#)

<auth-type>

auth-type 用于配置认证类型，当前只支持cas，以后会支持更多的认证类型（比如 form, ldap）

cas认证类型的属性配置如下表：其中casServerUrl和clientServerUrl为必填

属性	默认值	描述	casServerUrl
# cas服务url, 比如: https://cas.test.vipshop.com:8443	casLoginUrl		`\${casServerUrl}/login
cas 服务登录url, 比如: https://cas.test.vipshop.com:8443/login	clientServerUrl #		接入端url, 比如: http://localhost:8080/cfgcenter-console-webapp
	clientServiceUrl j_spring_cas_security_check	接入端服务 如: http://localhost:8080/cfgcenter-console-webapp/j_spring_cas_security_check	defaultTargetUrl
"/"	认证成功跳转url		defaultFailureUrl"/casfailed.jsp"
认证失败跳转url	accessDeniedUrl	"/403.html"	没有权限跳转url
	logoutSuccessUrl logout.jsp"	成功登出跳转url	proxyReceptorUrl
"/"	proxyReceptorUrl	proxyCallbackUrl\${clientServerUrl}/secure/receptor	secure/receptor"
	proxyCasAuthenticationProviderKey	id_for_this_acs_provider	key

<metadata-source>

metadata-source用于配置权限配置的来源。 metadata-source底下可以配置多个 realms，包括继承自特定接口的bean和simple-realm

默认的metadata-source
`com.vip.venus.security.access.intercept.RequestToAttributesMapMetaSource`

`realms`标签下可以定义多个 `spring bean` 或者 `spring ref`。这些 Bean 都需要继承 `com.vip.venus.security.access.intercept.RequestToAttributesMapRealm` 接口。接口中的权限格式和下面 simple-realm 中提到的一样

simple-realm 可以定义对各个 url 的访问的权限。

intercept-url	method	access
Url, 比如"/userapi/**"	get, post, put, delete, 当不提供的时候, 则是对访问这个url的所有http method都拦截	具体定义的权限的标记, 可按照系统需求自己定义, 比如: order_read

<access-decision-manager>

```
access-decision-manager#####
#####
##### com.vip.venus.security.core.userdetails.VenusUserDetailsByNameServiceWrapper
#####
##### com.vip.venus.security.plugin.oa.OAUserDetailsManager
```

30. #Venus Test

venus框架下，将单元测试变得更类似于集成测试的方式，通过对service层相关接口的直接调用，同时测试了service层 代码和repository层代码。

venus-test完全兼容spring-test单元测试方式；支持完全隔离的DB Testing；支持class级别和method级别的initDb和cleanDb操作。

在1.3.0版本中，新增支持h2数据库的功能，并且是codegen工具生成项目中junit的默认形式。Junit中的H2数据库只支持分库分表操作，不支持读写分离（因为不支持主从同步更新）

在1.3.0版本中，新增了.sql文件的注解形式：---!begin atom:db1_master;group:group-01，并且以---!end结束，在该注解内的所有sql只在注解所列出的库上执行；否则在全库执行

具体实践开发请参考Chapter#15，使用venus-test进行单元测试

30. 1#venus test中的注解

- java中的注解

Table#30. 1. #Annotation of the <venus-test>

Annotation	Description
@RunWith	执行Unit Testing的runner
@ContextConfig	指明如何加载和配置Testing
@ActiveProfiles	声明profiles，激活test profile
@TestExecution	用于指定在测试类执行之前与之后，测试方法执行之前与之后，可以做的一些动作
@CreateDb	创建数据库所需配置，默认位置为classpath: db.properties；初始化数据库执行sql， 默认位置为classpath: db.sql。默认位置不允许修改
@PrePostSql	preSqlFile: 在测试类执行之前，测试方法执行之前，执行sql文件位置；postSqlFile: 在测试类执行之后，测试方法执行之后，执行sql文件位置。用户自行配置

- sql脚本中的注解

```
#---!begin atom:qiyu;group:rwdsl ①

CREATE TABLE `student` (
  `id` int(11) NOT NULL DEFAULT '0',
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

#---!end

CREATE TABLE `teacher` (
  `id` int(11) NOT NULL DEFAULT '0',
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- group和atom对应properties文件中定义的数据库信息，示例中是注解内的脚本只在rwdsl上的qiyu这个数据库执行。不加注解的sql脚本时全库执行

Part#V. #Venus framework下gradle的使用

31. 为什么选择gradle作为项目构建工具	144
32. gradle的安装与运行	145
33. eclipse安装gradle插件	146
34. gradle项目添加依赖	147
35. 常用的gradle命令	149
35. 1. gradle基本命令	149
35. 2. venus下的gradle命令	149
35. 3. 使用gradle部署文件到maven仓库	149
36. 新建gradle project	151
36. 1. gradle与Jekins	152

31. #为什么选择gradle作为项目构建工具

目前市面上比较流行的项目构建工具有很多，比如maven，gradle，但是gradle有如下特点：

- 简单的依赖配置：

maven工程配置依赖如下：

```
<dependency>
    <groupId>com.google.code.kaptcha</groupId>
    <artifactId>kaptcha</artifactId>
    <version>2.3</version>
    <classifier>jdk15</classifier>
</dependency>
```

gradle工程配置如下：

```
dependencies {
    compile('com.google.code.kaptcha:kaptcha:2.3:jdk15')
}
```

- 灵活的依赖管理

maven的依赖管理很死板，只能依赖于标准的maven artifact，不能依赖本地的某个jar文件或者其它的源码。

而gradle则可以混合地同时支持这些依赖方法，这样可以让旧项目的迁移容易得多。

- 灵活自定义task：

maven可以帮助管理依赖关系，但是要在maven里实现一个简单的自定义功能，就很麻烦，要得写maven插件，

而在gradle里，可以使用groovy轻易的添加task实现自己的功能。

- 使用Artifact仓库

Maven有自己的单一仓库格式。Gradle可以使用Ivy仓库和Maven仓库。并且gradle项目也支持上传到Maven仓库

综上所述：Venus使用Gradle作为构建工具，并做了深度定制和集成，相对于maven实现了更简单/简洁的配置（没有XML的冗余），更符合java思维的配置

（groovy语法），更强的全局控制能力（全局控制依赖包的版本，全局替换某些包的引入），更方便的自定义构建逻辑的能力（直接写groovy脚本即可插入自己的构建逻辑），

更灵活的仓库配置策略，更灵活的参数配置和覆盖方式。

32. #gradle的安装与运行

gradle的安装与运行有2种方式:

- 类似maven 下载gradle后设置gradle环境变量, 使用gradle命令, 具体参考http://www.gradle.org/docs/current/userguide/userguide_single.html#installation
- Gradle Wrapper 使用gradle的wrapper做自启动, 项目根目录下有gradlew文件, 直接运行gradlew即可等同于运行gradle命令, 具体参考http://gradle.org/docs/current/userguide/gradle_wrapper.html

Note

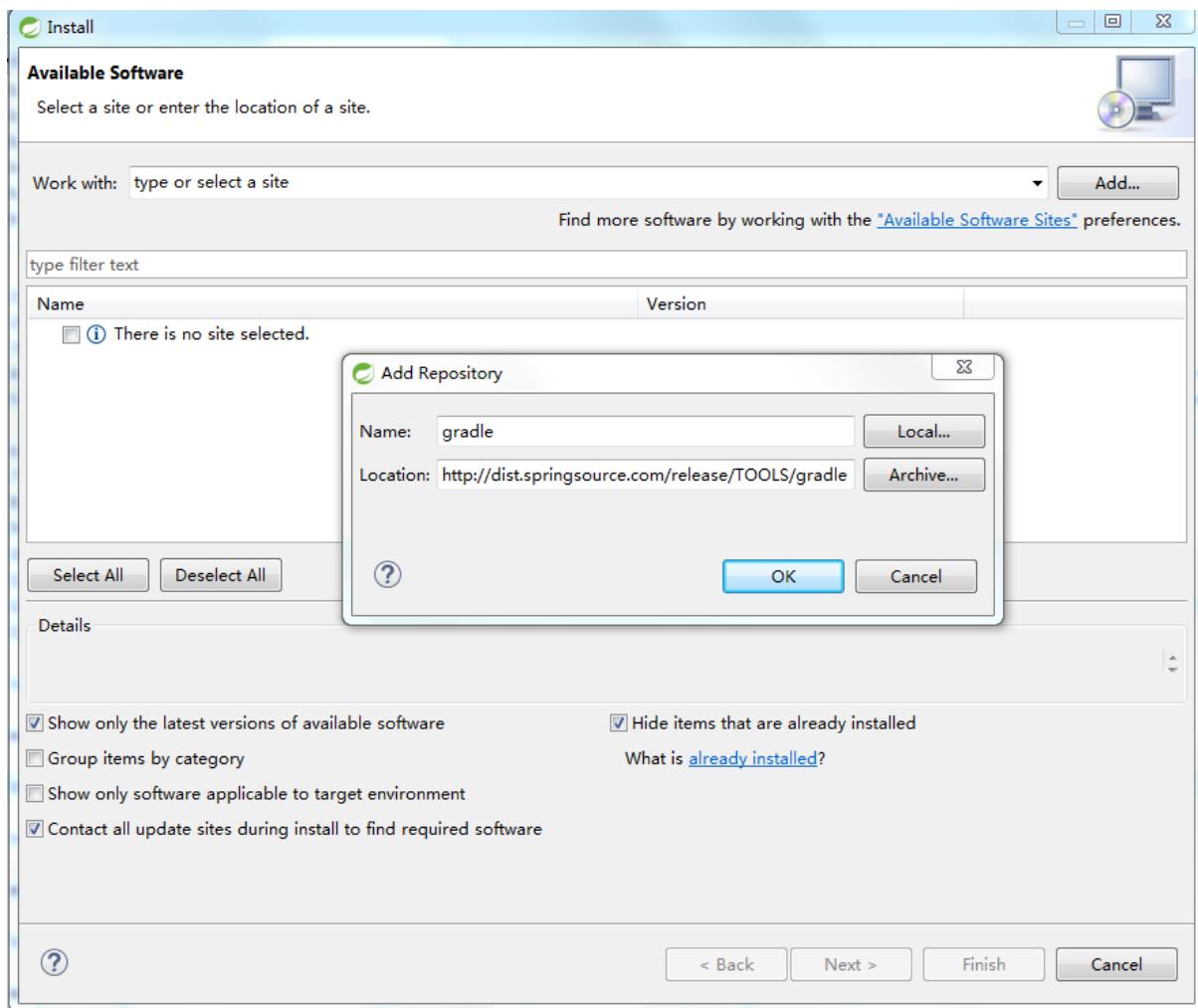
Wrapper可以使得项目组成员不必预先安装好gradle, 便于统一项目所使用的gradle版本

但在第一次运行的时候会自动去下载gradle的包(大概56M), 需要花费一段时间, 请耐心等待

33. #eclipse安装gradle插件

运用venus codegen生成venus项目时，eclipse中必须先已经安装好gradle插件。

可以通过Eclipse Marketplace安装：help→Eclipse Marketplace→find框中输入 gradle→go



Note

如果没有eclipse Marketplace可以直接在线安装：help→install New Software→add,

在Add Repository中输入Name和Location, 点击ok就可以了

gralde插件Location: 'http://dist.springsource.com/release/TOOLS/gradle/'

34. #gradle项目添加依赖

gradle项目是在build.gradle中添加依赖

对于codegen生成的venus项目：

如果是在构建时的依赖，则添加在buildscript中，示例如下：

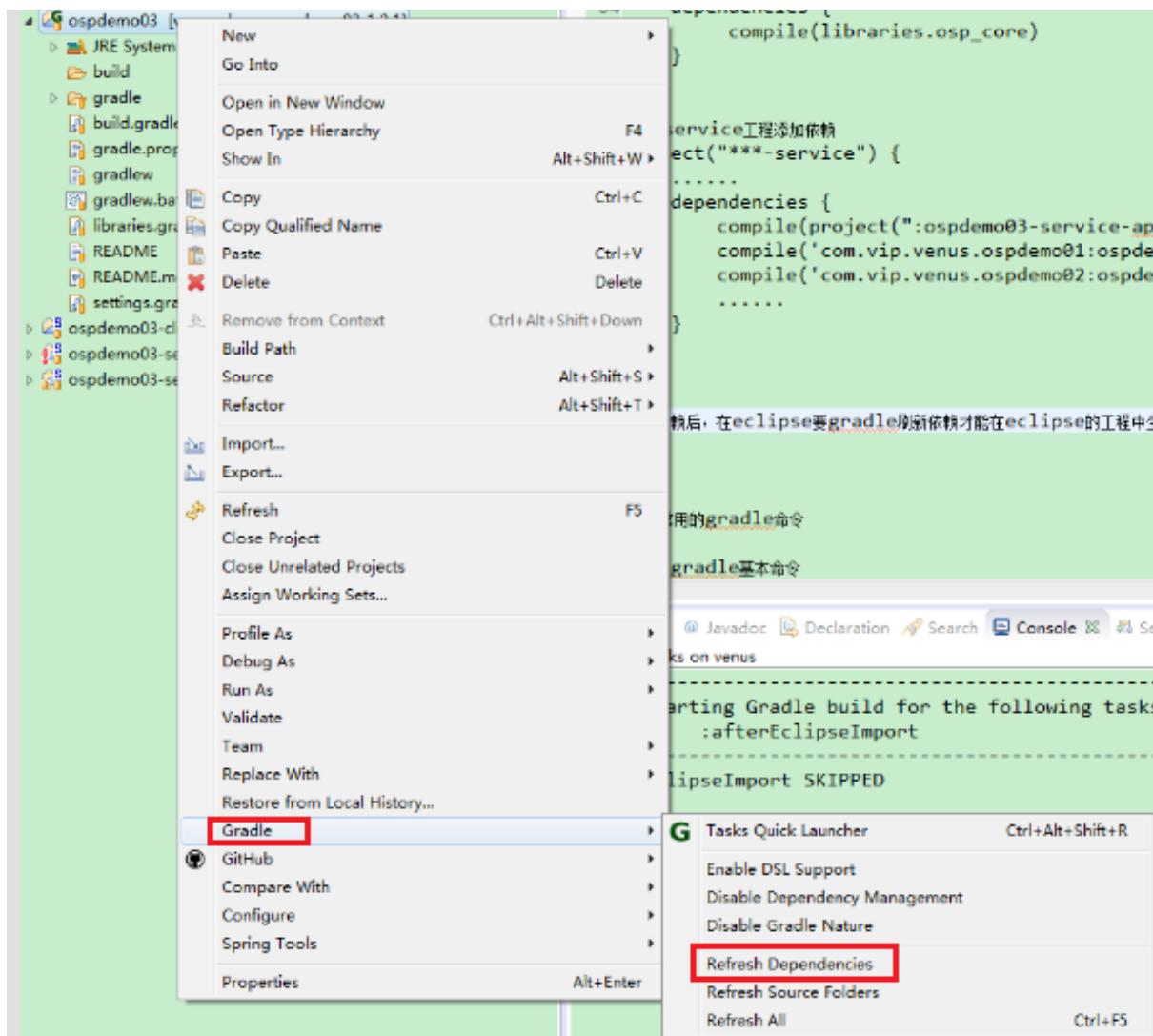
```
buildscript {
    .....
    dependencies {
        classpath ("org.springframework.build.gradle:propdeps-plugin:0.0.7")
        classpath ("com.vip.gradleplugin:restdoc-plugin:1.1.1")
        classpath ("com.vip.gradleplugin:osp-plugin:1.1.1")
        classpath ("com.bmuschko:gradle-cargo-plugin:2.0.3")
        .....
    }
}
```

如果是为某个工程添加依赖，则添加在指定工程下，示例如下：

```
// #service-api#####
project("****-service-api") {
    dependencies {
        compile(libraries.osp_core)
    }
}

//#service#####
project("****-service") {
    .....
    dependencies {
        compile(project(":ospdemo03-service-api"))
        compile('com.vip.venus.ospdemo01:ospdemo01-service-api:0.0.1-SNAPSHOT')
        compile('com.vip.venus.ospdemo02:ospdemo02-service-api:0.0.1-SNAPSHOT')
        .....
    }
}
```

添加依赖后，在eclipse要gradle刷新依赖才能在eclipse的工程中生效（建议每个工程下都刷新下）如下截图：



35. #常用的gradle命令

35. 1#gradle基本命令

`gradlew -h`: 列举gradle参数用法

`gradlew task`: 查看有哪些task

`gradlew build`: 构建项目

`gradlew clean`: 清除之前的构建(任务)

`gradlew upload`: 上传artifact到仓库

`gradlew -x test`: 不运行test task(即build时候不做单元测试), -x参数后面跟的task就是被忽略的

`gradlew htmlDependencyReport`: 生成项目依赖关系的报告, 用于分析项目的依赖关系

`gradlew htmlDependencyReport`: 生成项目依赖关系的报告, 用于分析项目的依赖关系

`gradlew build jacocoTestReport`: 获取单元测试覆盖率报告(报告目录: {子项目}/build/jacoco/jacocoHtml/index.html)

`gradlew --refresh-dependencies`: 强制更新依赖关系

35. 2#venus下的gradle命令

- OSP项目

- 运行OSP服务: `gradlew runOsp`

- 编译IDL文件: `gradlew runIdlc -PidlClasses=xxx.xxx.xxx.xxIDL,xxx.xxx.xxx.xxIDL`

- 打包OSP: `gradlew ospServiceZip`, zip包位于项目的build/distribution目录

- webapp项目

- 打包webapp: `gradlew war`, war包位于build/lib目录

- 打包restdoc: `gradlew restDocWar`, war包位于build/lib目录

Note

restdoc对原生的war进行了增强, 动态生成rest文档页面, 把war包放入tomcat的webapp下发布启动后,

可通过访问 [http://\[IP:端口\]/\[项目名称\]/restdoc](http://[IP:端口]/[项目名称]/restdoc) 来查看rest文档

比如: <http://localhost:8080/restdoc-webapp/restdoc/>

35. 3#使用gradle部署文件到maven仓库

在`gradle.properties`文件中, 按照实际情况配置下列参数, 然后执行命令然后执行命令 `gradlew uploadArchives`

```
# define the repository
vipShopMavenCentral=http://mvn1.tools.vipshop.com/nexus/content/groups/public
vipShopMavenSnapshotRepository=http://mvn1.tools.vipshop.com/nexus/content/repositories/snapshots
vipShopMavenReleaseRepository=http://mvn1.tools.vipshop.com/nexus/content/repositories/releases
deploymentUsername=vDeployer
deploymentPassword=deploy123
```

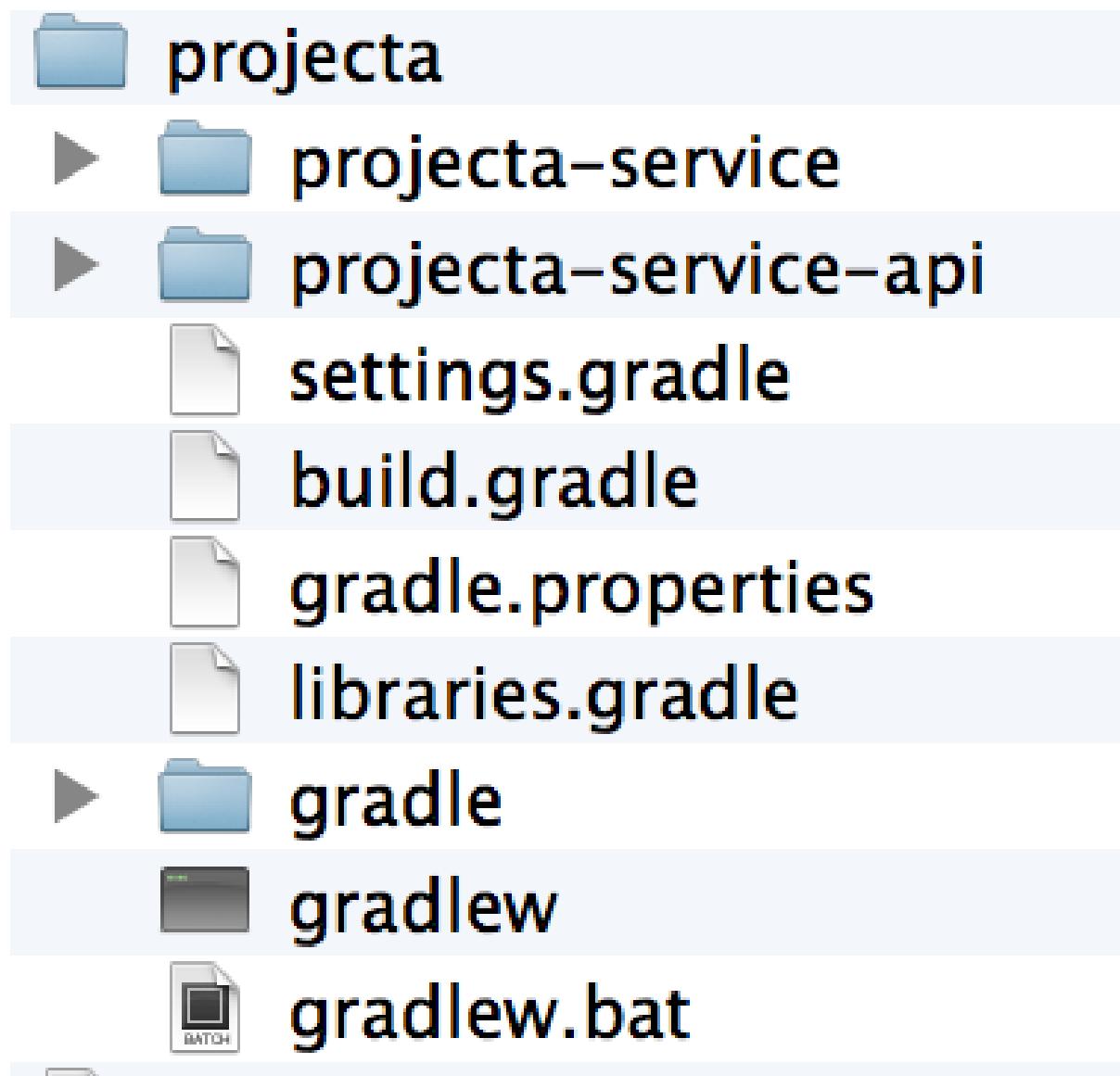
Note

上述代码中的`deploymentUsername`和`deploymentPassword`是直接可用的正式用户名密码，应用可以用修改直接执行`gradlew`命令的

36. #新建gradle project

venus codegen生成的项目就是一个gradle项目，如果想在已生成的项目下添加一个新的project，如下操作：

step1： 在root根目录下新建需要增加的project的文件夹，比如projecta-service



step2： 修改settings.gradle配置

添加代码如下：

```
rootProject.name = 'projecta'
include 'projecta-service-api'
include 'projecta-service'
```

step3： 配置project

在build.gradle文件里添加如下代码：

```
project("projecta-service") {
```

```

description = "The service module"
dependencies {
    compile(project(":projecta-service-api")) ❶
    optional(libraries.c3p0) ❷
    runtime(libraries.mysql)
    provided(libraries.lombok)
    testCompile(libraries.venus-test)
}
}

```

- ❶ 对同一个root下的其他project的依赖
- ❷ libraries.xxx是一个变量（比如libraries.c3p0），取libararies.gradle文件里设置c3p0的配置

Note

venus framework下gradle的依赖类型与maven的依赖的scope相对应，有compile, optional, runtime, provided, testCompile

36. 1#gradle与Jekins

- 最简单的build代码示例

```

#go into the project folder
cd ${WORKSPACE}
#run permission
chmod +x ./gradlew
./gradlew clean build -Porg.gradle.jvmargs='-Xms512m -Xmx2048m -XX:PermSize=512m -XX:MaxPermSize=1024m'

```

- 使用sonar build代码示例

```

#go into the project folder
cd ${WORKSPACE}
#run permission
chmod +x ./gradlew
./gradlew clean build sonarRunner -Psonar_host_url=http://10.100.90.200:9099
-Psonar_jdbc_url=jdbc:mysql://10.100.90.200:3306/sonar -Psonar_jdbc_username=sonar
-Psonar_jdbc_password=sonar -Psonar_projectName='Authsys Develop'
-Porg.gradle.jvmargs='-Xms512m -Xmx2048m -XX:PermSize=512m -XX:MaxPermSize=1024m'

```

Note

gradle2.1只支持4.0的sonar，如果要支持4.0+的sonar，需要升级gradle版本2.2+

- 使用cargo发布webapp

先在tomcat目录/conf/tomcat-users.xml文件里增加如下配置

```

<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<role rolename="manager-status"/>

<user username="managerbackend" password="managerbackend" roles="manager-script,manager-jmx"/>
<user username="managergui" password="managergui" roles="manager-gui"/>

```

然后依次运行如下命令发布：

```

#go into the project folder
cd ${WORKSPACE}

```

```
#run permission  
chmod +x ./gradlew  
.gradlew clean build cargoRedeployRemote -Pcargo.container.id=tomcat7x -Pcargo.port=8080  
-Pcargo.hostname=10.101.18.179 -Pcargo.username=managerbackend -Pcargo.password=managerbackend  
-Porg.gradle.jvmargs='-Xms512m -Xmx2048m -XX:PermSize=512m -XX:MaxPermSize=1024m'
```

Note

gradle与jenkins集成时，会有一些约定的配置文件：

```
javaProjects.gradle; webappProjects.gradle; appProjects.gradle;  
mavenDeployment.gradle;  
  
ospServiceApiProjects.gradle; ospServiceProjects.gradle; report.gradle;
```

Part#VI. #配置中心

配置中心的使用目前有两种：1 只是单纯的读取配置信息 2 不仅读取配置信息，还要监控配置变更

具体实践开发请参考Chapter#17，使用配置中心读取配置信息和监控配置变更

37. 配置中心设计	156
38. 配置中心介绍	157
39. 配置定义文件	158

37. #配置中心设计

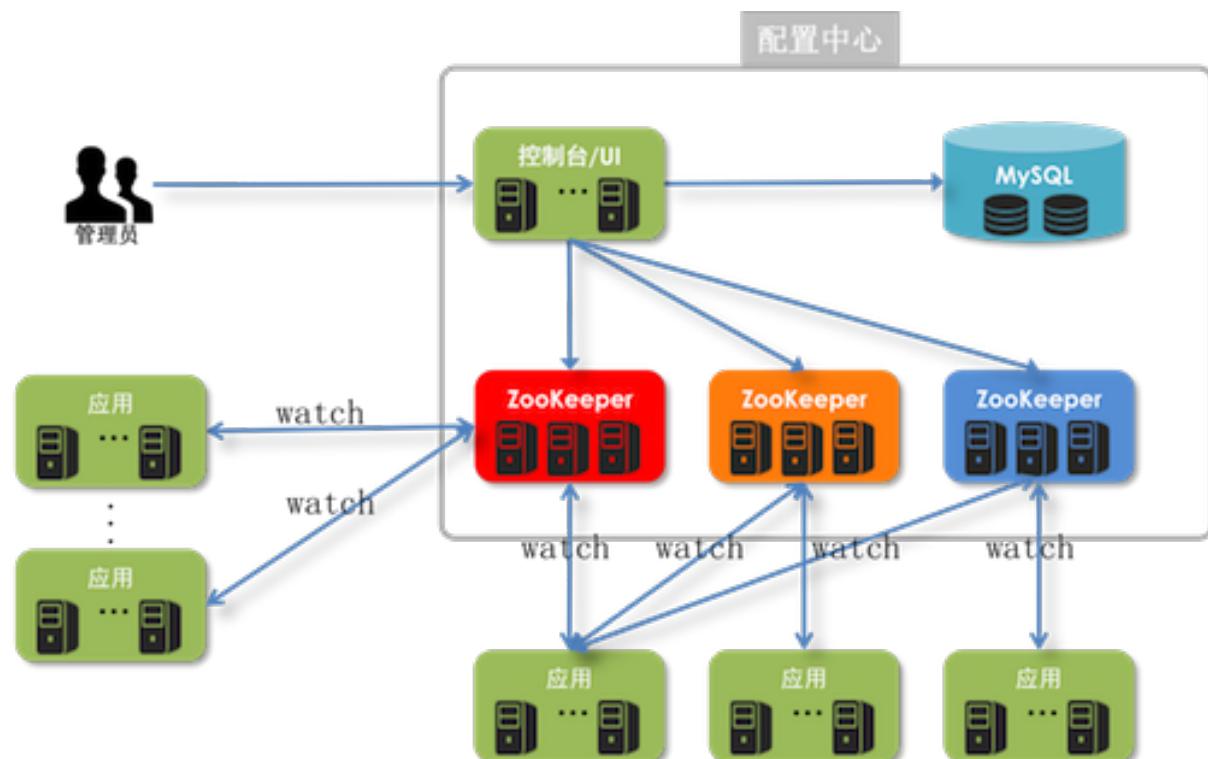
配置中心是部署在zookeeper上的，这样就可以做到：

集中配置：无需再对应用集群的各个服务器做配置，集中一次配置，应用集群的所有服务器生效

主动推送：配置一旦变更，zookeeper就会向所有注册为watcher的应用主动推送新的配置信息

规范配置：配置使用统一的定义方式

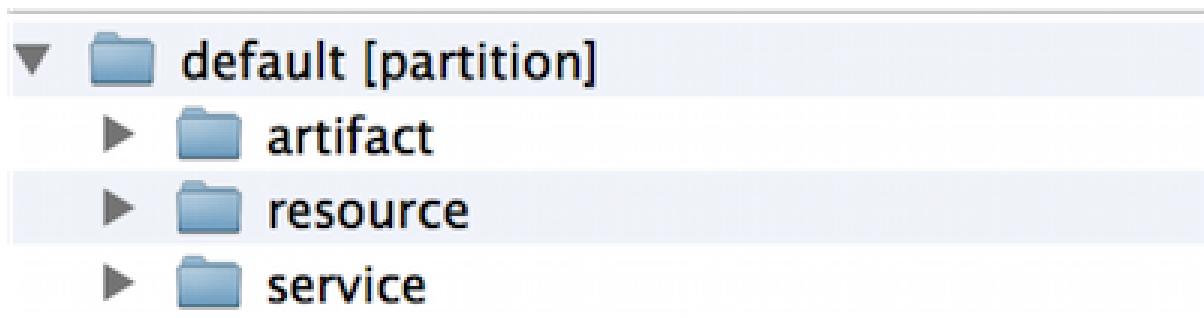
配置可读：每项配置可以增加足够的说明来增加可读性



38. #配置中心介绍

配置中心的根节点是partition， 默认值为default

Partition下的子节点会随着配置中心功能的增强而会增加， 目前为artifact, resource, service



Partition用于在部分情况下(集成测试环境)一个配置中心支持多套配置

Partition用于在开发/测试环境下隔离不同开发/测试人员对配置中心的配置

Client(应用)如何使用partition? jVM参数里增加vip.cfgcenter.partition参数来指定partition

39. #配置定义文件

配置定义文件，主要是为了约束和规范应用的properties配置项，结合<venus-context:property-placeholder>和配置中心来使用

配置定义文件的名称约定为：applicationCfgDef.xml，存放路径为src/main/resources下

典型配置示例：

```
<cfgdef group="com.vip.osp" name="osp-configuration" version="1.0.0"> ①
  <itemgroup name="provider">
    <item name="#####" key="threadPool" defaultValue="false" dataType="boolean" desc="#####"
    ## hotReload="true"/> ②
    <item name="###" key="threads" defaultValue="0" dataType="integer" desc="#####0#####"
    ## hotReload="true">
      <restriction>
        <minInclusive value="0"/>
        <maxInclusive value="20"/>
      </restriction>
    </item>
    <item name="##" key="loadBalance" defaultValue="roundRobin" dataType="string"
    hotReload="true" desc="#####">
      <restriction>
        <enumeration value="roundRobin" name="##"/>
        <enumeration value="randomRobin" name="##"/>
      </restriction>
    </item>
  </itemgroup>
  <itemgroup name="consumer">
    .....
  </itemgroup>
</cfgdef>
```

- ① 配置中心根据group和name在zk上创建节点用来存储item中的property
- ② item中配置待监控的property

Table#39. 1. #cfgDef元素的属性

属性	描述	类型	是否必须	默认值
group	配置定义的组	string	是	无
name	名称	string	是	无
version	配置定义的版本	string	是	无

Table#39. 2. #itemgroup元素的属性

属性	描述	类型	是否必须	默认值
name	名称	string	是	无

Table#39. 3. #item元素的属性

属性	描述	类型	是否必须配置	默认值
name	Property项的名称	string	是	无
key	Property项的key	string	是	无

属性	描述	类型	是否必须配置	默认值
defaultValue	Property项的默认值	string	否	无
defaultValue0	只能使用默认配置，不能更改配置，配置了此项为 true，defaultValue必须配置	boolean	否	false
dataType	property项的值的数据类型的定义	string	是	无
desc	Property项的描述	string	是	无
hotReload	Property项的值变更后是否能自动重新加载和生效新的配置而不需要重启（人工干预）	boolean	否	false

Part#VII. #OSP

Note

本文档针对OSP 2.x 编写，OSP 1.x 系列请参考 [Venus Framework Reference Document 1.3.0](#)

40. OSP入门	163
40.1. OSP概述	163
40.2. OSP详解	163
远程调用机制	163
服务协议(OSP Protocol)	163
服务契约(IDL)	164
服务提供方(服务端)	164
服务代理层(OSP-Proxy)	165
服务消费方(客户端)	165
服务治理	166
服务管理	168
部署方式	169
40.3. OSP的未来	169
41. OSP服务提供方用户手册	170
41.1. 快速开始	170
41.2. 服务定义	170
IDL文件的编写	170
API文档的编写	175
IDL文件的修改与编译	176
使用codegen新增服务和结构体	176
服务版本匹配原则	178
服务版本兼容性原则	178
服务粒度的控制	181
41.3. 服务实现	181
服务编写	181
服务本地运行	183
服务本地调试	185
服务本地测试	185
41.4. 服务测试与发布	185
服务发布	185
服务测试	186
41.5. 服务部署	187
环境变量设置	187
服务运行脚本	187
服务配置	188
服务监控	189
42. OSP服务接入方使用手册	192
42.1. Java客户端接入	192
Java客户端环境准备	192
Java客户端对服务的同步调用	193
Java客户端对服务的异步调用	193
Java客户端的InvocationContext设置	194
Java客户端的测试	195
Java客户端配置	195
Java客户端常见问题	196
42.2. PHP客户端接入	196
PHP客户端的代码架构	196
OSP-PHP C扩展安装部署方案	198
PHP客户端对服务的调用	199
PHP客户端的异常处理	199

PHP客户端的常见问题	200
42.3. OSP客户端配置	200
osp-proxy地址的环境变量	200
其他环境变量	201
Java客户端的JVM启动参数	201
43. OSP公共知识	202
43.1. ZooKeeper及配置中心配置	202
43.2. OSP-Proxy	203
公用Remote Proxy	203
OSP-Proxy的安装	203
本地环境运行	203
生产环境运行	204
OSP-Proxy的升级	205
OSP-Proxy的配置	205
OSP-Proxy的监控	206
43.3. OSP服务的配置	206
优先级	206
配置项	207
43.4. OSP的异常处理	207
返回信息机制	207
异常处理机制	208
43.5. OSP路由功能	208
客户端路由模型	208
路由处理方式	209
代码示例	210
配置中心	210
43.6. 静态路由表	210
背景	210
配置使用说明	210
其他注意事项	211
43.7. 自定义OspFilter	211
OspFilter的编写	211
OspFilter的配置	212
TransactionContext使用	212
43.8. 常见问题	212
集合元素为Null	212
如何设置超时	212
OSP 服务第一次访问超时	212

40. #OSP入门

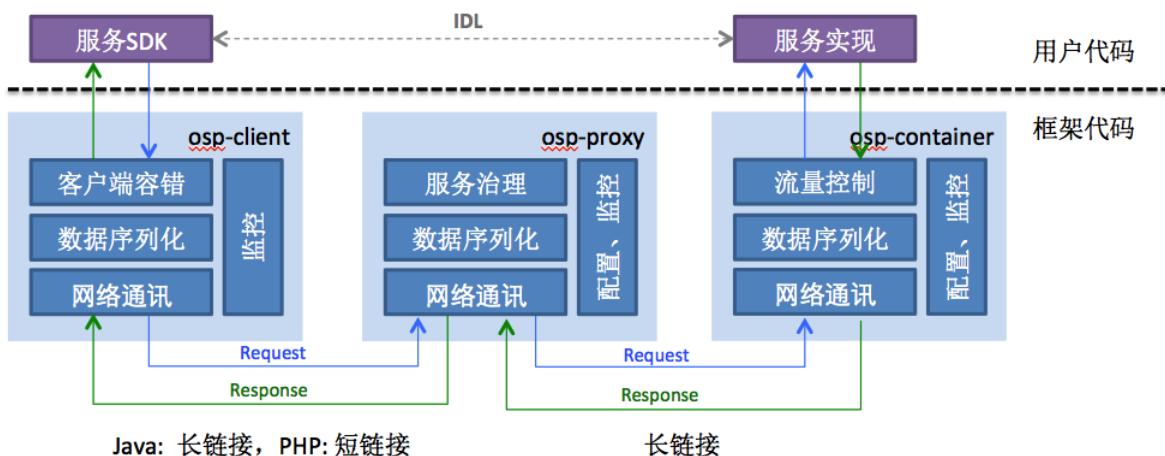
40. 1#OSP概述

开放服务平台(venus-osp)是Venus体系的核心组成部分之一，主要目标是提供服务化的核心远程调用机制以及基础服务治理功能。契约化的服务接口保证系统间的解耦清晰、干净；基于Thrift的通信和协议层确保系统的高性能；服务可以自动注册并被发现，易于部署；配合配置中心，服务配置可以动态更新；客户端与治理逻辑的分离使服务接入得到极大简化；除此之外，OSP提供了丰富的服务治理能力，如路由、负载均衡、服务保护和优雅降级等。

40. 2#OSP详解

远程调用机制

OSP提供的核心功能是一套高性能、高可扩展的远程调用机制(RPC)，在实现上采用了Apache Thrift作为基本框架，软件栈仍采用Thrift的工作模式，并在其基础上做了二次开发。



如上图所示，OSP从远程调用的角度主要分为服务调用方，代理层和服务提供方。整体设计和实现分为架构代码和用户代码，服务化功能如服务契约处理、远程调用机制、服务治理、中央及本地配置、监控埋点、并发处理、容错机制和可扩展性等由框架代码实现；用户代码主要实现业务逻辑，另外用户代码也可以插件的形式植入框架，以满足不同场景的需求。服务化体系的目标是降低开发复杂度并最大限度地提高开发效率。

服务协议(OSP Protocol)

OSP采用了基于二进制的通讯协议，以消息为基本通讯单元；每条消息包含消息头和消息体。协议设计上以无状态为原则，每条消息包含调用所需所有信息，一次调用通过一个消息交换(请求/响应)完成。

OSP协议追求以下特性：

- 统一性（编程、管理）
- 高性能
- 高可扩展性
- 灵活部署

目前业界很多公司采取基于HTTP/RESTful的通讯协议，但大型公司绝大部分采用二进制协议，以保证以上特性。

OSP的契约化接口定义通过IDL实现；为便于开发人员快速熟悉和掌握，我们采用了Java语法作为接口定义的语法，并自主开发了支持多语言（目前Java和PHP）的编译器（osp-idlc）。IDL编译器集成到Venus开发环境里，以实现环境的透明；服务开发者只需定义服务接口，开发工具自动生成服务提供方（stub）和消费方（SDK）的代码以及文档。

服务开发者只需使用Venus代码生成器生成开发项目，并定义服务接口（IDL）即可开始开发服务；开发者可以进行在线测试，服务文档自动推送至中央文档系统。如果要消费某个服务，服务消费者只需根据语言将服务SDK作为依赖引入，即可对服务进行调用。

服务契约（IDL）

OSP的契约化接口定义通过IDL实现；为便于开发人员快速熟悉和掌握，我们采用了Java语法作为接口定义的语法，并自主开发了支持多语言（目前Java和PHP）的编译器（osp-idlc）。IDL编译器集成到Venus开发环境里，以实现环境的透明；服务开发者只需定义服务接口，开发工具自动生成服务提供方（stub）和消费方（SDK）的代码以及文档。

服务开发者只需使用Venus代码生成器生成开发项目，并定义服务接口（IDL）即可开始开发服务；开发者可以进行在线测试，服务文档自动推送至中央文档系统。如果要消费某个服务，服务消费者只需根据语言将服务SDK作为依赖引入，即可对服务进行调用。

```

1 package com.vip.hello.world.idl;
2
3 import com.vip.osp.core.idl.annotation.*;
4
5 @idl
6 @namespace("com.vip.hello.world")
7 public class HelloWorldIDL {
8     @service(version="1.0.0")
9     interface HelloWorld {
10         String sayHello(@tag(1) String helloMsg);
11     }
12 }
```

上图是一个非常简单Hello World 服务的定义，对于有Java背景的使用者基本上没有什么门槛。

服务提供方（服务端）

服务端包含服务容器和服务本身。服务容器的目的是通过将共享功能分离出来以便于集中管理；而服务本身基本以业务逻辑为主，由业务团队实现。

为简便设计、开发和部署，OSP服务端容错基于无状态服务的理念，服务实例之间互不感知。在服务启动时，每个服务首先从配置中心获取配置，然后将自己的ID、地址和端口注册到服务注册中心；服务代理层从注册中心获取当前服务提供方的所有实例，通过设定的负载均衡策略对服务进行调用。这样在服务端部署成本低，而且可以做到无限扩展。

OSP的一大特色是高性能，在测试环境单机测试服务端QPS可达10万以上。我们在设计和实现上采取了各种措施

- 在整个调用链路（客户端至代理层、代理层至服务端）尽可能采用TCP长连接，以降低通讯层损耗
- 将原生Thrift协议层改为主动开发的基于二进制的通讯协议，对通讯协议进行优化，最大限度地降低协议开销
- 以基于Netty的异步传输层取代原生Thrift的传输层，整个传输内核基于流式模型并使用异步处理，大大提高了并发量和单机性能

OSP服务端容错基于无状态服务的理念，通过代理层的错误感知和负载均衡等功能自动摘出有问题的服务器；开发团队无需做特殊处理，也不必部署负载均衡器（LB），既简化了部署也同时降低了成本。

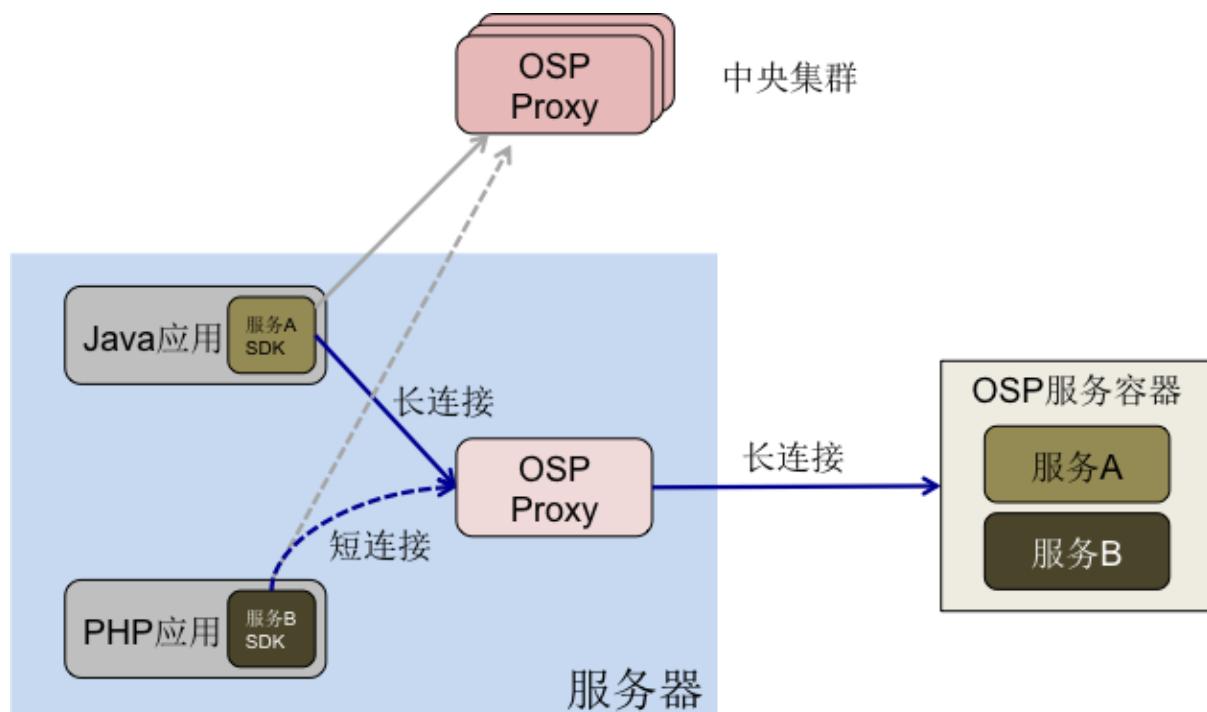
服务代理层 (OSP-Proxy)

服务代理层的使用是OSP框架的另一个特色，这个方案相对优雅地解决了几个系统层面的难题。

首先，绝大部分逻辑如客户端配置获取、服务实例的获取和选择、服务容错、服务治理等需要在服务调用方实现，在多语言环境里这些逻辑可以在代理层实现一次实现，降低开发、部署和维护成本；

其次，发布各种调用方新功能和对现有功能的改进只需对代理层进行升级，对业务团队实现透明，降低发布的复杂性和成本；

最后，代理层可以对异构环境进行统一，解决异构环境独特的问题，比如PHP环境里进程短生命周期造成的配置、监控相关的问题。



值得关注的是服务代理层集中了绝大部分的服务治理智能，可以说远程调用的大脑。关于服务治理的细节请参照后面章节。

服务代理层部署在每个服务器上，以进程方式运行。服务客户端将请求发送至代理进程，代理进程根据服务治理逻辑对请求进行处理（转发、降级或拒绝），这样使服务治理的实现保持对客户端和服务端完全透明，而且动态可控。

出于容错考量，每个IDC会部署一个服务代理集群作为备用。当本地服务代理进程出现故障时，自动切换到集中代理集群。

服务消费方 (客户端)

由于服务客户端是业务应用的一部分，为避免框架代码改动对业务方的影响，服务客户端遵循的首要设计原则是单纯简单。除了必须的功能如通信协议、序列化、代理层容错以及监控埋点外，其他高级功能全部在代理层实现。目标是客户端一旦发布后，很长时间无需更新（SDK除外）。

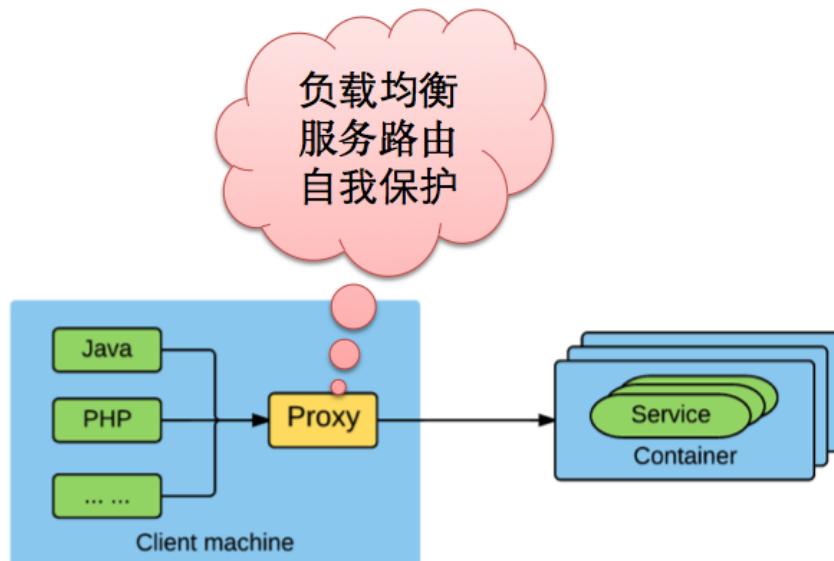
从服务开发者的角度，客户端SDK由osp-idlc自动生成，无额外工作；从服务使用者的角度，只需将服务客户端SDK作为依赖引入，即可对服务进行调用，极少额外的工作。

另一点值得关注的是使用客户端的动机和原因，目前业界不少公司不使用客户端，直接以RESTful/HTTP的形式完成调用。这样做的好处显而易见：就是简单；但在整体治理能力上付出了重大代价，如服务调用上下文的采集和传递几乎无法实现，使得服务化体系不可控；无法实现契约化服务，管理混乱；在协议的选择上没有余地；另外所有高级功能如授权、认证、签名、压缩等须由业务开发者承担。综上所述，引入客户端所带来的一点点不便，完全可以通过合理的设计和规划来消除。

服务治理

唯品会服务化体系的一个重要领域是较强的服务治理能力，这也是我们远胜于传统SOA架构的地方。这里的所谓服务治理主要集中在对负载均衡，路由选择以及自我保护等领域所提供的功能以及灵活性。

传统上像负载均衡大多通过集中式中间层的形式实现，这样使得每个服务都需要部署一套LB，不仅需要付出部署和维护成本，而且对性能挑战较大，不利于增加高级功能和对集群做大规模扩展。



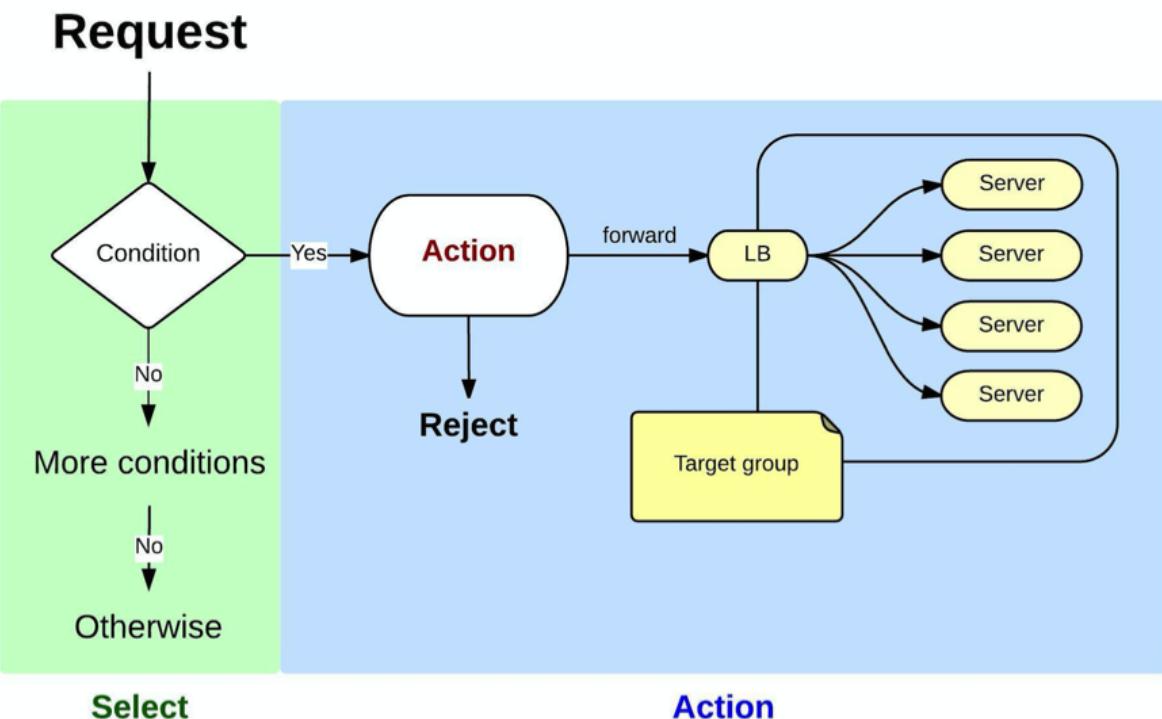
OSP选择了将复杂的服务治理逻辑放在服务代理层(osp-proxy)，同时将代理层推到服务调用端；原则上去除了中央代理集群，部署和维护上做的简单统一；通过本地代理层只对本机服务请求进行代理的方式，从根本上解决了代理层的性能问题，同时可以无限扩展。

负载均衡

OSP的负载均衡功能由osp-proxy提供，目前支持Least Active First, RR, Random等多种负载均衡策略；通过与配置中心集成，策略可以动态变更；另外，在支持基本服务端健康度检测（如ping）的基础上，利用对丰富的服务信息的统计（调用流量、延迟、错误率等），可以提供更加智能的健康检测机制。

路由选择

服务路由决定一个服务请求是否应该得到处理，以及由哪一个服务实例处理。这部分是各种服务治理政策的执行地，是一个极为重要的模块。大量的使用场景都可以结合服务路由来实现，如服务的上线/下线，在线测试，机房选择，A/B测试，灰度发布，流量控制，权限控制以及优雅降级等。



SP路由选择分二阶段执行：首先通过预先定义的规则对服务请求进行分类；然后对每一类请求进行操作。

服务请求分类规则的定义非常灵活，同时利用了服务请求本身信息以及服务调用的上下文。值得关注的是服务调用上下文不仅包含本次调用的信息，而且也包含从调用链上游传递过来的信息。例如，上下文包含登录用户信息，这样下游服务可以对不同用户的请求作出不同反应。

在对请求进行分类后，我们进一步定义对每一类的操作，如转发、降级或拒绝。

自我保护

自我保护的目标是实现“有弹性”服务，以解决以下几个问题 - 服务器失败影响服务质量 - 超负荷导致整个服务失败 - 服务失败造成的雪崩效应

下图概况了自我保护机制采用的手段



流量控制

基于服务端实现并根据环境及服务调用统计信息（如响应时间、错误率、CPU, memory等）自动触发。在此之上，可以考虑有选择地放弃流量，如根据主、辅流程，预先配置的预案等。

熔断器 (Circuit breaker)

熔断器部署在调用链路中，是实现快速失败，问题隔离，自动回复以及降低重试频率的主要部件。熔断器的状态（打开，关闭、半开）取决于由配置规则。例如，使用50%调用错误率作为规则，当错误率低于50%时，熔断器处于“关闭”状态，放行所有请求；当错误率高于50%时，进入“打开”状态，请求无法通过；这时会启动一个定时器，出发后进入“半开”状态，允许少量流量通过以测试是否恢复。

资源隔离 (Bulkhead)

其目的是在某个服务或功能出现故障时，避免服务器或进程内所有资源被完全占用，进而正常功能受到株连。常用的实现方式是通过服务器、进程以及线程池等为单元，分配部分资源，从而实现资源层面上的隔离。

Fallback

Fallback是服务降级的一种机制，在服务端由于某种原因（失败、繁忙、权限或政策等）无法提供服务时，仍向调用方返回合法响应，并作降级标记。最简单的做法是返回预定义的常数，也可以使用缓存策略，返回上一次成功调用的结果（Last good response）。

服务管理

注册与发现

在OSP中，一个或多个服务打包成为一个服务包，并通过服务容器加载启动。服务启动时，服务容器需要从配置中心获取服务的配置信息，服务容器根据获取的参数，对服务进行初始化操作。服务启动成功后，容器向服务注册中心（Service Registry）进行注册，系统记录该服务的一个实例已经运行。服务容器与服务注册中心保持长连接，当服务卸载或者服务容器故障退出时，服务注册中心可以自动的删除服务实例，维护最新有效的服务实例列表。

当代理层需要调用一个服务时，代理层查询服务注册中心，获取所需服务的全部服务实例，并结合服务路由策略（决定可以为当前服务请求提供服务的实例）及负载均衡策略（选择那一个服务实例进行服务），选择一个服务实例。

代理层维护到服务注册中心的长连接，这样如果有新的服务实例或者老的服务实例失效时，代理层可以及时获得最新的服务实例列表。与服务端不同，当代理层失去长连接后，仍然可以通过之前获得的实例列表进行服务调用。

配置管理

OSP的配置管理基于venus-config，详情参照 Part#VI，“配置中心”，在此不多描述。

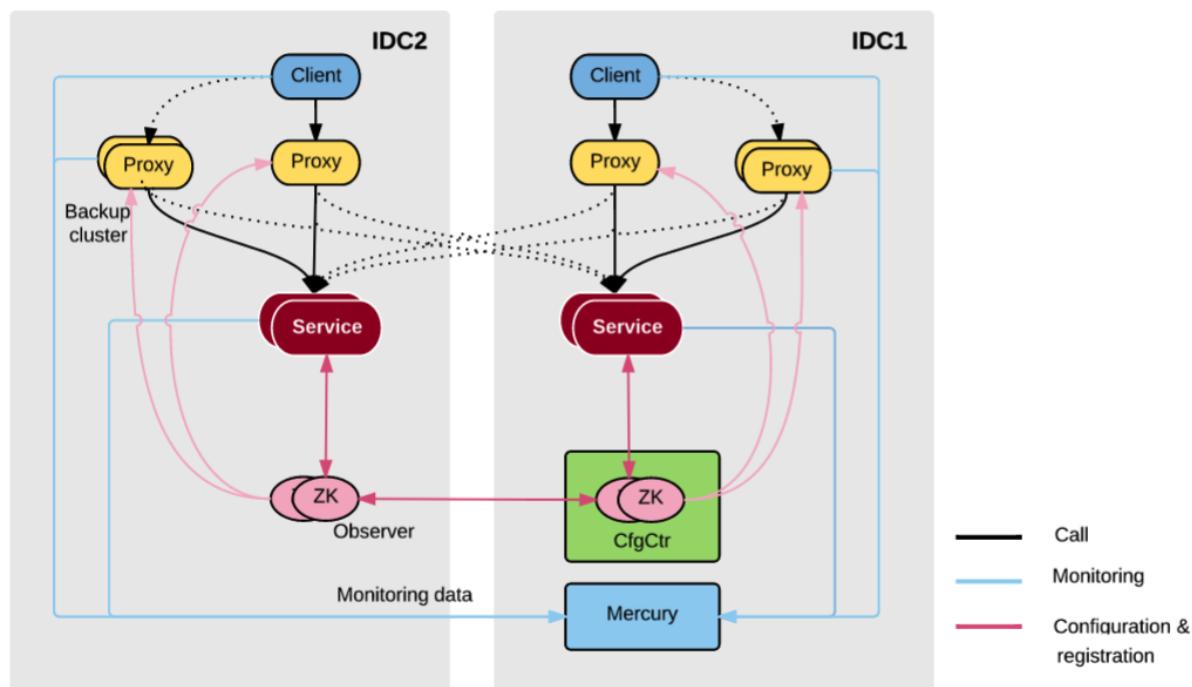
服务监控

集中式服务监控能力主要由Mercury提供，OSP主要提供监控数据的采集。在OSP客户端、代理层以及服务端的数据采集、上报以及查询能力主要依托于mercury-client, venus-hermes和venus-vi等项目。

Mercury-client主要采集调用链相关的数据，而venus-hermes则负责采集其他需要上报的数据。采集到的数据通过日志形式上报到Mercury做进一步处理。Venus-vi则提供在本机对采集的数据通过JMX以及HTTP的查询功能，同时也对无需上报的数据进行采集。Venus框架已经提供公共监控数据功能，在此基础上OSP进一步实现对其相关的数据的采集。

部署方式

下图为OSP的一个典型的多机房部署，整个结构分为三层：服务端、代理层和客户端。服务端以集群形式在单机房或多机房部署，服务器之间以及集群之间完全隔离，集群可以无限扩展。



代理层由本地和集中式备份集群组成，本地代理以独立进程方式部署在每一台调用者服务器上，只为本机调用者服务；集中式备份集群每个机房部署一个，由LVS提供负载均衡，为本地代理提供容错能力。在本地代理发生故障时，调用方切换到使用集中式备份集群完成服务调用。

代理层从注册中心获取各机房的服务实例信息，并优先选择使用本机房服务实例（可配置），在本机房无可用实例时选择使用其他机房实例。代理层从配置中心获取调用方的配置信息，并保持实时更新。

客户端内嵌在业务应用里，通过SDK向使用者提供调用接口和对象模型；从使用者的角度，只需进行调用。客户端不与配置中心产生联系，但会将监控信息上报到Mercury。

配置中心集中部署在一个机房内，在远程机房可用选择部署ZK观察集群，以提高性能和降低网络影响。Mercury集中部署在一个机房内。

40. 3#OSP的未来

OSP是唯品会服务化体系的核心组成部分，在公司软件体系的建设有深远影响。在团队齐心协力努力下，已基本完成远程调用机制，部分服务管理，部分服务治理；下阶段工作将在完善已有功能的基础上，进一步开发服务治理和服务管理领域的功能，同时进一步完善服务化生态环境，使之更为强大。

目前公司内已有十几个部门使用OSP，已经开发、上线了一百多个服务。在此感谢所有OSP使用团队信任，是你们的支持让OSP不断前进；同时我们也会给大家全力支持。相信在不远的将来，服务化体系可以公司帮助公司所有的团队。

41. #OSP服务提供方用户手册

41. 1#快速开始

服务提供方的快速开始请参考 Chapter#8, OSP项目示例演示，描述了开发环境变量定义，OSP服务的代码生成、项目文件说明、本地运行及调试过程。

Note

必须先阅读此文档，设置开发环境变量

VIP_CFGCENTER_ZK_CONNECTION=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:218

41. 2#服务定义

我们采用了Java文件，以Annotation的方式作为服务接口定义的语法。

IDL文件服务可对服务接口，方法，结构体，异常类，枚举类进行定义，示例如下，后面的章节将逐一进行讲解。

```
@idl
public class UserIDL {

    @struct
    class UserModel {

        @default_value("1234")
        @tag(1)
        Long id;
        .....
    }

    @service(version="1.0.0")
    interface UserService {

        @ErrorCodes({@ErrorCode(name="405",description="####")})
        int create(@tag(1) UserModel userModel);

        int query(@tag(1) UserModel userModel);
    }
}
```

IDL文件的编写

Note

OSP1.x中的过时Annotation将不在本文档中列出。

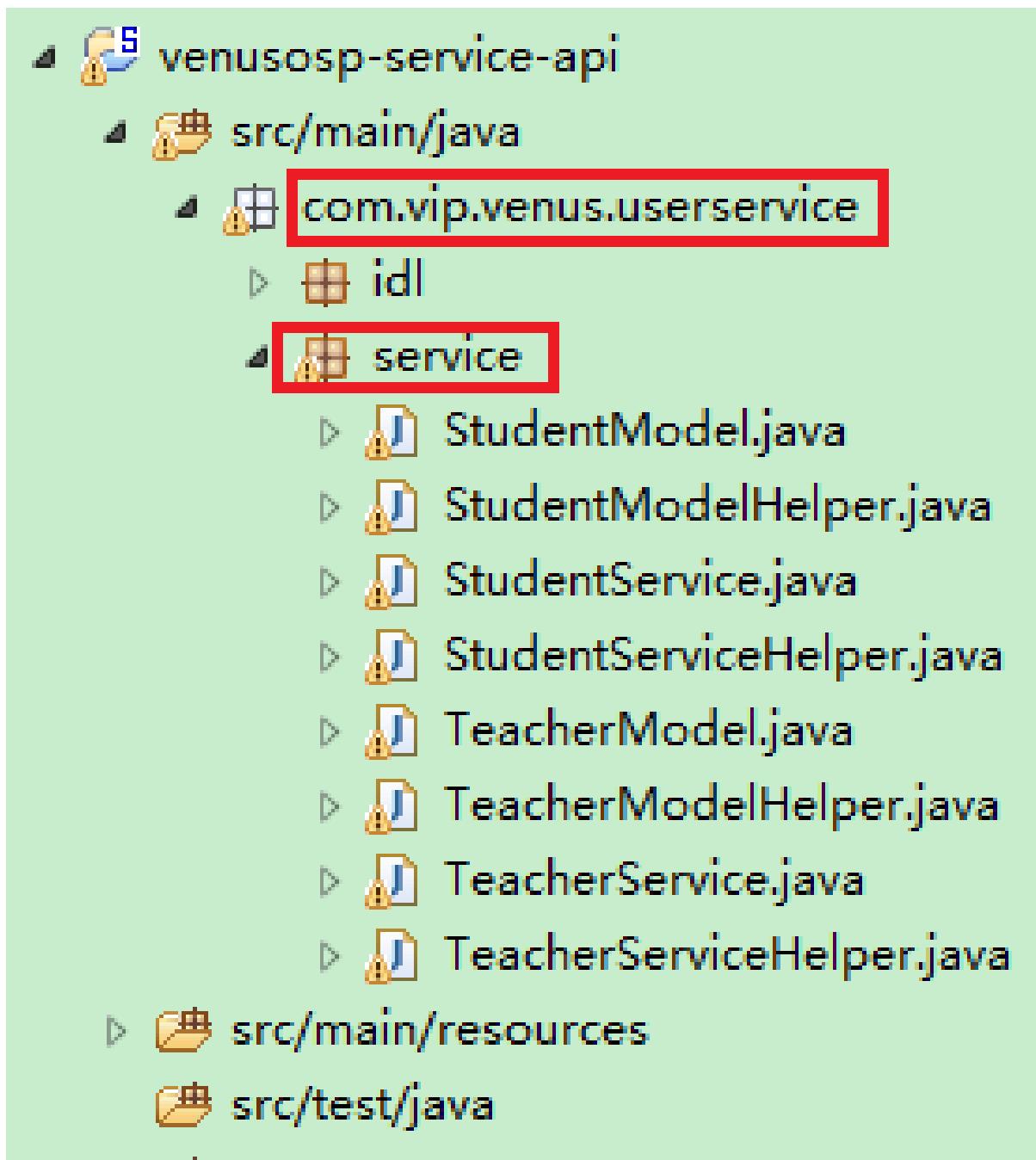
1. IDL文件定义

IDL文件必须使用注解@idl和@namespace，示例代码如下：

```
@idl ①
@namespace("com.vip.venus.userservice.service") ②
public class TeacherIDL {
    .....
}
```

① @idl：标识这个Java文件是IDL文件。

- ② @namespace: 标识服务接口的命名空间，即编译这个IDL文件所生成文件的Package名。



2. 服务接口定义

接口定义必须使用注解@service, 示例代码如下:

```

@idl
@namespace("com.vip.venus.userservice.service")
public class TeacherIDL {

    @service(version="1.0.0", timeout=1000)
    interface TeacherService {

        int createSelective(@tag(1) TeacherModel teacherModel);

        .....
    }
}

```

```
}
```

@service的属性:

version	接口的版本号，必填值。
loadbalance	负载均衡策略。 LoadBalanceStratage可选值包括RoundRobin(轮询)、 RandomRobin(随机)、 LeastActive(最少当前连接数)、 Undefined(默认为LeastActive) 和 Hash(暂不支持)。默认值是 LoadBalanceStratage.Undefined。
timeout	超时时间，单位为ms， 默认值为200ms。如果超时，服务端会返回一个超时异常。
failovertimes	超时后的自动重试次数。默认值为0，不重试。
threadPool	是否使用服务端工作线程池，还是直接使用底层IO线程池。可选值包括 True, False, Undefined (默认为True)。默认值是Undefined。
threads	该服务最多只占用服务端工作线程的个数，默认值是服务端线程池的总大小。而服务端线程池的总大小由启动参数:-Dosp.container.threadpool.size来设置，默认是CPU处理器数量*2。

服务接口interface不支持继承，不能extends其他接口。

Note

除version外的其他属性与配置中心配置值的关系，详见Section#43.3，“OSP服务的配置”

3. 方法定义

示例代码如下：

```

@idl
@namespace("com.vip.venus.userservice.service")
public class TeacherIDL {
    @service(version="1.0.0")
    interface TeacherService {
        @method(timeout=500)
        int create(@tag(1) @required TeacherModel teacherModel);

        int selectAll(@tag(1) @default_value("1") Integer limit);
        .....
    }
}

```

方法参数和返回值只支持java中的原始类型以及其对应的包装类型，以及基于@struct注释的结构体，可以使用其他IDL文件里的结构体。

注解@method是可选的，可用来标注方法级的属性，覆盖@service中的配置，如果不需要覆盖定义可省略。

- `loadbalance`: 同服务定义。
- `timeout`: 同服务定义。
- `failovertimes`: 同服务定义。
- `threadPool`: 同服务定义。
- `threads`: 同服务定义。

其他属性已过时，包括`hash`, `tags`, `authorization`。

Note

所有属性与配置中心配置值的关系，详见Section#43.3，“OSP服务的配置”

方法参数的可选annotation:

<code>@tag</code>	表明是第几个参数，从1开始，必填值，在序列化时将是参数的唯一标识。
<code>@required</code>	这意味着在序列化之前，它必须被设置值。本注解只用于对象类型(e.g. <code>Student</code> , <code>String</code> , <code>Integer</code>)。而Java原始类型(<code>byte</code> , <code>short</code> , <code>int</code> , <code>long</code> , <code>boolean</code> , <code>double</code>)默认且必须为 <code>required</code> ，可省略不写。
<code>@default_value</code>	当请求报文中参数未设值时的默认值。本注解只用于Java原始类型及其封装类(e.g. <code>int</code> , <code>String</code> , <code>Integer</code>)，值的类型为字符串，OSP会自动转换为实际类型。

Note

建议参数不要使用Java原始类型(`int`, `long`)，而使用其封装类(`Integer`, `Long`)，保持是否`optional`的灵活性。

4. 结构体定义

结构体的类定义必须使用注解`@struct`，示例代码如下:

```

@idl
@namespace("com.vip.venus.userservice.service")
public class TeacherIDL {
    ...
    @struct
    class TeacherModel {
        @tag(1)
        Long id;

        @tag(2)
        @default_value("Anonymous")
        String name;
    }
}

```

```

@tag(3)
@required
String phone;
}
.....
}

```

结构体类不能继承。

结构体类中的成员变量只支持java中的原始类型以及其对应的包装类型， 以及基于@struct注释的结构体， 基于@default_value的枚举和实现IErrorCode接口的自定义错误码类(见后)， 结构体， 枚举等类型可以使用其他IDL文件中的定义。

结构体类中的类成员变量只支持强类型， 比如定义一个list不能写成List list， 必须为List指明元素类型， 例如List<String> list。

结构体类中的类成员变量建议不要使用Java原始类型(int, long)， 而使用其封装类(Integer, Long)， 保持是否optional的灵活性。

类成员变量的可选annotation:

- @tag: 同方法参数定义。
- @required: 同方法参数定义。
- @default_value: 同参数定义， 编译IDL时将生成类成员变量的默认值。

5. 枚举定义

枚举定义中的枚举值必须使用@default_value注释， 作用类似于@tag， 值必须是字符串类型的数字， 表示第几个枚举值， 示例代码如下：

```

@idl
@namespace("com.vip.venus.userservice.service")
public class TeacherIDL {
    .....
    enum EnumDemo {
        @default_value("1")
        ADD,
        @default_value("2")
        SUB,
    }
    .....
}

```

6. 错误码定义

错误码枚举必须实现 IErrorCode 接口， 示例代码如下：

```

@idl
@namespace("com.vip.venus.userservice.service")
public class TeacherIDL {
    .....
    enum MyErrorCode implements IErrorCode {
        @ErrorCode(name="TIME_OUT", description="#####", detail="")
        TIME_OUT,
        @ErrorCode(name="CONNECT_ERROR", description="####", detail="")
        TIME_OUT2,
    }
    .....
}

```

@ErrorCode的属性：

- **name:** 必填项，错误码名字。
- **description:** API文档中的错误码描述。
- **detail:** API文档中的错误码细节及解决方案。

`@ErrorCode` 生成类的使用方式见 the section called “抛出异常”

API文档的编写

自动生成的API文档是OSP平台的重要功能，可以下列方式提供：

- 编译IDL文件后，会在xxx-service-api项目的src/main/html 目录下生成html文档。
- gradlew runOsp 运行OSP项目时，可在弹出的Web界面里观看在线文档。
- 参见中央文档相关文档，将文档发送到中央文档库。

通用的annotation：

- `@label`: 定义简单名字，在API文档的概述页面出现。可修饰服务、方法、结构体、枚举、错误码。
- `@doc`: 定义注释，在API文档的详细页出现。可修饰服务、方法、方法参数、结构体、结构体属性、枚举、错误码。
- `@sample_value`: 定义示例值，可修饰结构体属性与 方法参数。

方法上的错误定义annotation：

- `@ErrorSets`: 定义多个错误码枚举类。
- `@ErrorCode`: 定义单个错误码。
- `@ErrorCodes`: 定义多个错误码。

```

@label("#####")
@doc("#####IDL")
@idl
public class UserIDL {
    .....

    @label("#####")
    @doc("#####")
    @struct
    class UserModel {
        @doc("##id")
        @default_value("1234")
        @sample_value("8888")
        @tag(1)
        Long id;
        .....
    }

    @label("###")
    @doc("#####")
    @service(version="1.0.0")
    interface UserService {
        @label("#####")
        @doc("#####")
        @ErrorCodes({@ErrorCode(name="405",description="####",detail="#####")})
        int create(@tag(1) UserModel userModel);
    }
}

```

```

    @ErrorSets({MyErrorCode.class})
    int query(@tag(1) UserModel userModel);
}
}

```

IDL文件的修改与编译

编译IDL文件

用户对IDL文件修改后，打开命令行窗口进入项目目录下，运行：

```
gradlew runIdlc -PidlClasses=com.vip.venus.userservice.idl.StudentIDL
```

`-PidlClasses`的值可以逗号分割，一次编译多个IDL文件。

如果一个package下有多个IDL文件，也可以使用 -PidlPackages 编译目录下的所有IDL文件

```
gradlew runIdlc -PidlPackages=com.vip.venus.userservice.idl
```

上面指令一次过生成所有语言的代码，如果想只生成html、php或者java的内容

```
gradlew runIdlc -gen=html -PidlPackages=xcom.vip.venus.userservice.idl
```

手工新增服务接口或方法

用户以不依赖codegen，手工新增服务接口或方法：

1. 手工修改IDL文件并重新编译。
2. 在xxx-service项目里自行实现新接口或新方法。
3. 修改xxx-service项目中的文件 META-INF/service-conf/service.xml，添加新接口实现的相关Bean

```

<bean class="com.vip.venus.userservice.service.impl.TeacherServiceImpl" id="teacherService"/>
<osp:service id="teacherOsp" ref="teacherService"/> ①

```

① 使用了OSP的Spring Tag，简化了原来复杂的写法。

使用codegen新增服务和结构体

codegen工具生成项目代码后，如何利用codegen再添加新的结构体和服务呢？

step 1：

事先在数据库中建立好新表，比如示例中的Teacher表。

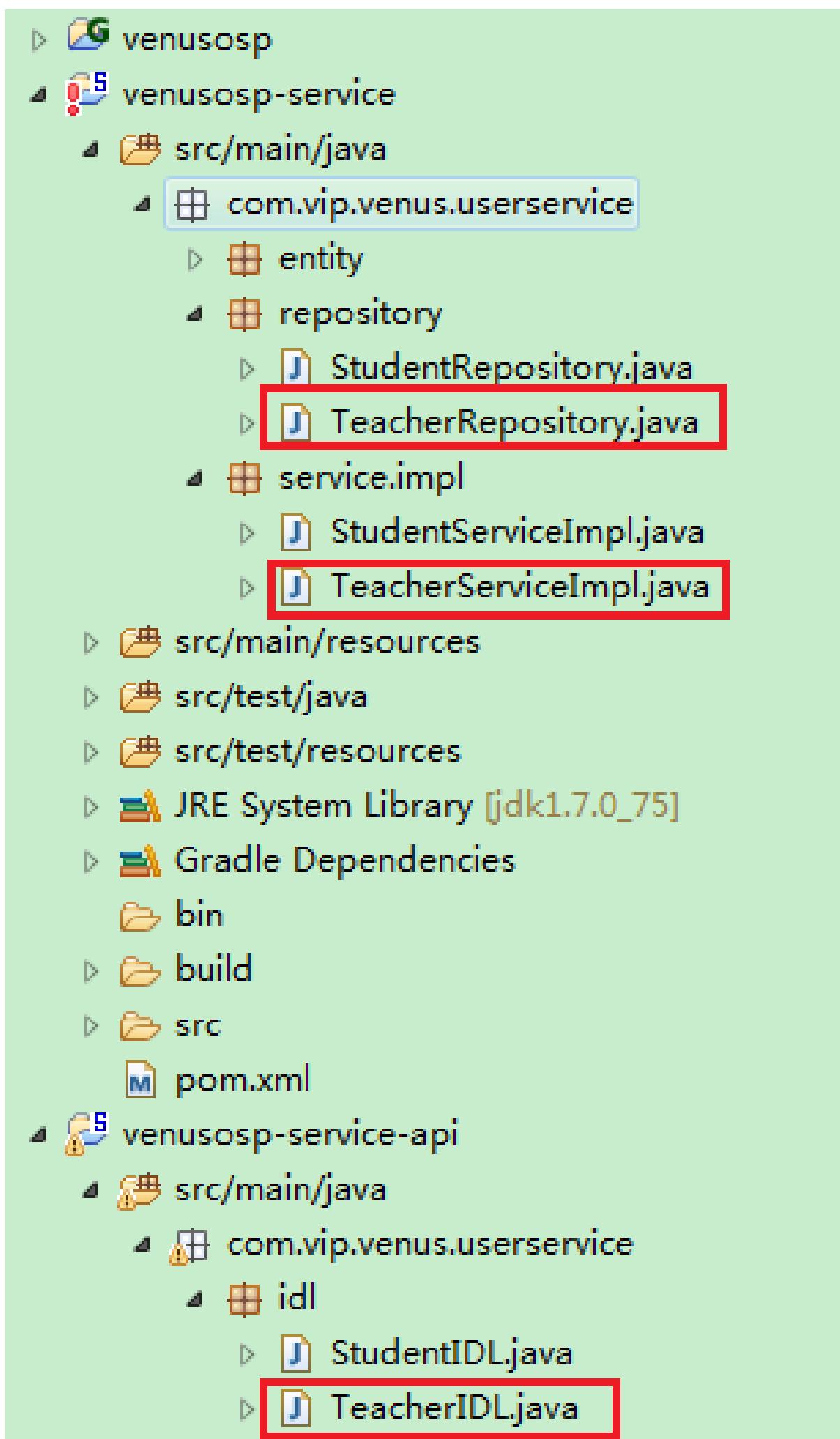
step2：

修改codegen的配置文件里的db.tables指向新表。

```
db.tables=teacher:Teacher
```

在命令行窗口进入codegen目录下，运行 codegen -codeonly，生成过程中会提示是否覆盖文件，这里选择 n

然后刷新Eclipse工程，能看到生成的对应java文件



step3:

修改xxx-service项目中的文件META-INF/service-conf/service.xml，添加teacher相关bean

```
<bean class="com.vip.venus.userservice.service.impl.TeacherServiceImpl" id="teacherService"/>
<osp:service id="teacherOsp" ref="teacherService"/> ①
```

① 使用了OSP的Spring Tag，简化了原来的Bean定义写法。

服务版本匹配原则

版本定义

OSP框架以及服务采用三个数字作为版本：大版本、小版本以及补丁版本。版本兼容是指调用方和提供方的兼容性，采用以下策略：

- 大版本之间（如1.x和2.x）之间无兼容性保证。
- 小版本之间（如1.1.x和1.2.x）之间保持向后兼容。
 - 原则上不允许删除已有接口或方法，只允许新增；
 - 原则上不允许修改或删除已有方法的参数，只允许新增；
 - 原则上不允许修改或删除数据结构的成员变量，只允许新增；
- 补丁版本之间同上。

请求版本匹配

当服务客户端请求一个版本的服务时，下列所有服务端都可以被选择作为候选：

1. 版本完全一致的服务端
2. 大版本，小版本相同，补丁版本大于当前版本的服务端
3. 大版本相同，小版本大于当前版本的客户端

例如，客户端请求版本1.2.1的服务时，如果当前有1.2.2, 1.2.3, 1.3, 1.4 版本的服务端注册了，则它们都会被选择调用。而 2.0 则不会被选择的。

配置版本匹配

OSP-Proxy与服务端从配置中心提取配置的版本匹配时：

1. 优先选择到版本完全一致的配置
2. 如果没有版本一致的配置，则选择大版本，小版本相同，补丁版本大于当前版本的配置
3. 如果仍然没有，则选择大版本相同，小版本大于当前版本的配置

例如，请求版本1.2.1的服务时，优先选1.2.1的配置，其次选1.2.2，其次选1.2.3，其次选1.3，其次选1.4，而2.0 则不可选。

服务版本兼容性原则

Thrift协议提供了很好的服务版本兼容性支持，总的来说，只要总是增加@Tag中的数字而不修改原有的@Tag，而且是可选类型，则服务可以任意增减方法的参数，任意增减结构体的成员变量。

Note

原始类型(如int, long, boolean)为必选类型。

结构体(如Student)或原始类型的对象类型(如Integer, String)为可选类型, 同时用户可以用@required注解将其设为必选。

为了比较好的兼容性, 尽量不要使用原始类型(如int), 而使用原始类型的对象类型(Integer)

1. 接口定义的要点

```
/**  
 * 服务例子  
 * @label 服务  
 */  
@service(version="1.0.0")  
interface ServiceDemo {  
    public String sayHello(@tag(1) String name);  
}  
  
64= /**  
65  * 服务例子  
66  * @label 服务  
67  */  
68= @service(version="1.0.1")  
69  interface ServiceDemo {  
70  public Long sayHello(@tag(1) @optional Integer age, @tag(2) String name);  
71 }
```

Figure#41. 1. #方法可在任意位置增加可选参数, 但不能更改原有参数的tag值。

```
/**  
 * 服务例子  
 * @label 服务  
 */  
@service(version="1.0.0")  
interface ServiceDemo {  
    public String sayHello(@tag(1) String name);  
}  
  
64= /**  
65  * 服务例子  
66  * @label 服务  
67  */  
68= @service(version="1.0.1")  
69  interface ServiceDemo {  
70  public String sayHello(@tag(2) String name);  
71 }
```

Figure#41. 2. #相同参数的tag数字不可变更。

```
/**  
 * 服务例子  
 * @label 服务  
 */  
@service(version="1.0.0")  
interface ServiceDemo {  
    public String sayHello(@tag(1) String name);  
}  
  
64= /**  
65  * 服务例子  
66  * @label 服务  
67  */  
68= @service(version="1.0.1")  
69  interface ServiceDemo {  
70  public Long sayHello(@tag(1) String name);  
71 }
```

Figure#41. 3. #返回类型不可更改。

```
/**  
 * 服务例子  
 * @label 服务  
 */  
@service(version="1.0.0")  
interface ServiceDemo {  
    public String sayHello(@tag(1) String name);  
}  
  
64= /**  
65  * 服务例子  
66  * @label 服务  
67  */  
68= @service(version="1.0.1")  
69  interface ServiceDemo {  
70  public String sayHello(@tag(1) String name, @tag(2) @required Long age);  
71 }
```

Figure#41. 4. #不可增加必选参数(@required)。

```
/**  
 * 服务例子  
 * @label 服务  
 */  
@service(version="1.0.0")  
interface ServiceDemo {  
    public String sayHello(@tag(1) String name);  
}  
  
64= /**  
65  * 服务例子  
66  * @label 服务  
67  */  
68= @service(version="1.0.1")  
69  interface ServiceDemo {  
70  public String sayHello(@tag(1) String name, @tag(2) long age);  
71 }
```

Figure#41. 5. #不可增加必选参数(原始类型)。

```
/**  
 * 服务例子  
 * @label 服务  
 */  
@service(version="1.0.0")  
interface ServiceDemo {  
    public String sayHello(@tag(1) String name, @tag(2) @required Long age);  
}  
  
64= /**  
65  * 服务例子  
66  * @label 服务  
67  */  
68= @service(version="1.0.1")  
69  interface ServiceDemo {  
70  public String sayHello(@tag(1) String name);  
71 }
```

Figure#41. 6. #原必选入参不可被移除(@required)。

```

64  /**
65   * 服务例子
66   * @label 服务
67   */
68  @service(version="1.0.0")
69  interface ServiceDemo {
70     public String sayHello(@tag(1) String name, @tag(2) long age);
71 }

```

Figure#41. 7. #原必选入参不可被移除(原始类型)。

2. 结构体定义的要点

```

@struct
class StructDemo {
    /**
     * 字节类型定义
     * @sampleValue 1
     */
    @tag(1) byte _byte;
    @tag(2) short _short;
    @tag(3) int _int;
    @tag(4) long _long;
    @tag(5) double _double;
    @tag(6) String _string;
    @tag(7) boolean _boolean;
    @tag(8) EnumDemo _enumDemo;
    @tag(9) StructDemo _structDemo;
    @tag(10) List<StructDemo> _list;
    @tag(11) Set<StructDemo> _set;
    @tag(12) Map<String, StructDemo> _map;
}

37 @struct
38 class StructDemo {
39 /**
40   * 字节类型定义
41   * @sampleValue 1
42   */
43 @tag(1) String name;
44 @tag(2) byte _byte;
45 @tag(3) short _short;
46 @tag(4) int _int;
47 @tag(5) long _long;
48 @tag(6) double _double;
49 @tag(7) String _string;
50 @tag(8) boolean _boolean;
51 @tag(9) EnumDemo _enumDemo;
52 @tag(10) StructDemo _structDemo;
53 @tag(11) List<StructDemo> _list;
54 @tag(12) Set<StructDemo> _set;
55 @tag(13) Map<String, StructDemo> _map;
56 }

```

Figure#41. 8. #结构体可随意增加可选类型字段，但不能更改原有参数的tag值。

```

@struct
class StructDemo {
    /**
     * 字节类型定义
     * @sampleValue 1
     */
    @tag(1) byte _byte;
    @tag(2) short _short;
    @tag(3) int _int;
    @tag(4) long _long;
    @tag(5) double _double;
    @tag(6) String _string;
    @tag(7) boolean _boolean;
    @tag(8) EnumDemo _enumDemo;
    @tag(9) StructDemo _structDemo;
    @tag(10) List<StructDemo> _list;
    @tag(11) Set<StructDemo> _set;
    @tag(12) Map<String, StructDemo> _map;
}

37 @struct
38 class StructDemo {
39 /**
40   * 字节类型定义
41   * @sampleValue 1
42   */
43 @tag(1) byte _byte;
44 @tag(2) short _short;
45 @tag(10) int _int;
46 @tag(4) long _long;
47 @tag(5) double _double;
48 @tag(6) String _string;
49 @tag(72) boolean _boolean;
50 @tag(8) EnumDemo _enumDemo;
51 @tag(9) StructDemo _structDemo;
52 @tag(22) List<StructDemo> _list;
53 @tag(30) Set<StructDemo> _set;
54 @tag(12) Map<String, StructDemo> _map;
55 }

```

Figure#41. 9. #相同字段的tag值不可变更。

```

@struct
class StructDemo {
    /**
     * 字节类型定义
     * @sampleValue 1
     */
    @tag(1) byte _byte;
    @tag(2) short _short;
    @tag(3) int _int;
    @tag(4) long _long;
    @tag(5) double _double;
    @tag(6) String _string;
    @tag(7) boolean _boolean;
    @tag(8) EnumDemo _enumDemo;
    @tag(9) StructDemo _structDemo;
    @tag(10) List<StructDemo> _list;
    @tag(11) Set<StructDemo> _set;
    @tag(12) Map<String, StructDemo> _map;
}

```

```

37 @struct
38 class StructDemo {
39 /**
40     * 字节类型定义
41     * @sampleValue 1
42     */
43     @tag(2) short _short;
44     @tag(10) int _int;
45     @tag(4) long _long;
46     @tag(5) double _double;
47     @tag(6) String _string;
48     @tag(72) boolean _boolean;
49     @tag(8) EnumDemo _enumDemo;
50     @tag(9) StructDemo _structDemo;
51     @tag(22) List<StructDemo> _list;
52     @tag(30) Set<StructDemo> _set;
53     @tag(12) Map<String, StructDemo> _map;
54 }

```

Figure#41. 10. #原必选字段不可被移除(原始类型, @required同理)。

```

/**
 * 枚举定义例子
 * @label 枚举
 */
enum EnumDemo {
    /**
     * 加操作
     */
    @default_value("1") ADD,
    @default_value("2") @doc("减操作") SUB,
    /**
     * 乘操作(javadoc)
     */
    @default_value("3") @doc("乘操作(doc)") MUL,
    @default_value("4") @doc("除操作") DIV
}

```

```

16 /**
17     * 枚举定义例子
18     * @label 枚举
19     */
20 enum EnumDemo {
21 /**
22     * 加操作
23     */
24     @default_value("2") @doc("减操作") SUB,
25 /**
26     * 乘操作(javadoc)
27     */
28     @default_value("3") @doc("乘操作(doc)") MUL,
29     @default_value("4") @doc("除操作") DIV
30 }

```

Figure#41. 11. #枚举字段可新增但不可被移除。

服务粒度的控制

每个IDL文件代表一个服务的定义，一个部署在OSP-Engine中的应用可包含多个服务。

使用CodeGen工具可能会生成多个IDL文件。用户可根据需要，将属于同一个服务的所有方法，合并到同一个IDL文件里。

41. 3#服务实现

服务编写

服务生命周期

OSP容器使用Spring Framework管理服务Bean。

Spring Context首先加载service项目中META-INF/service-conf/service.xml的Bean，同时service.xml以下面的语句加载了applicationContext.xml。

```
<import resource="classpath*:applicationContext.xml" />
```

在容器启动时，会在初始化Spring Context后，再绑定服务侦听端口。如果Spring Bean设置成lazy load，则会在服务被请求时才进行初始化。为了避免首次连接超时，应尽量将初始化比较耗时的Bean设为 lazy-init=false 模式。

在容器关闭时，在停止了服务侦听端口后，会关闭Spring Context，如果Spring Bean定义了close方法，就会被Spring Context所调用。因此如果服务有资源清理，服务启动线程优雅停止等需求，需要实现相应Bean的关闭方法并在Spring Context中进行定义。（since OSP 2.4.7）

抛出异常

OSP的异常返回信息机制见 the section called “返回信息机制”。

服务提供者抛出的异常时只需要定义ErrorCode与ErrorMessage，OSP平台会自动在ErrorMessage中添加Location与Level信息。

在构造OSPException时传入引起异常的底层异常时，OSP平台会在日志中打印其异常栈信息，但不会向调用者返回该异常栈。

1. 使用预定义的Error类

```
throw new OSPException(MyErrorCode.USER_NOT_FOUND, "User not found");
throw new OSPException(MyErrorCode.USER_NOT_FOUND, "User not found", ex);
```

其中MyErrorCode是在IDL里定义并生成的异常类，见the section called “IDL文件的编写”

此时ErrorCode为MyError.USER_NOT_FOUND上的@ErrorCode annotation的name属性，ErrorMessage是“User not found”。

2. 直接指定ErrorCode

```
throw new OSPException("USER_NOT_FOUND", "User not found");
throw new OSPException("USER_NOT_FOUND", "User not found", ex);
```

此时ErrorCode是“USER_NOT_FOUND”，ErrorMessage是“User not found”。

3. 直接将底层异常封装为OSPException

```
throw new OSPException(ex); ①
throw new OSPException("User not found"#ex); ②
```

- ① ErrorCode为“GENERAL_EXCEPTION”，ErrorMessage为ex.toString()，包含ex的ClassName与message。
- ② ErrorCode为“GENERAL_EXCEPTION”，ErrorMessage为“User not found”。

不建议使用此方式抛出异常，因为服务调用者及Mercury将无法针对ErrorCode进行分类。

4. 直接抛出非OSPException的RuntimeException及其子类

```
throw ex; ①
throw new RuntimeException("User not found"); ②
```

- ① ErrorCode为“GENERAL_EXCEPTION”，ErrorMessage为ex.toString()，包含ex的ClassName与message。
- ② ErrorCode为“GENERAL_EXCEPTION”，ErrorMessage为“User not found”。

不建议使用此方式抛出异常，因为服务调用者及Mercury将无法针对ErrorCode进行分类。

5. Error Code的定义

为方便Mercury对业务进行错误类型分类，建议对Error Code的最前面添加符合规则的数字定义（等于Http中的40x错误），比如可以规定1000以前的数字是调用方错误，1000以后的数字是服务方错误（等于Http中的50x错误）

因此ErrorCode可定义为 100_USER_NOTFOUND

实现Health Check接口

服务实现HealthCheck接口，向ValidateInternal与Monitor WebApp 报告服务自身的健康度。

服务的健康度包含它本身的健康，以及它所依赖的服务（比如Redis，数据库）的健康度。示例如下：

```
import com.vip.hermes.core.health.CheckResult;

@Override
public CheckResult healthCheck() throws OspException {
    return CheckResult.up().build();
}
```

参照Hermes文档，可返回其他状态，并对状态附加描述。

服务本地运行

容器基于Gradle运行

打开CMD窗口通过cd命令进入你生成的项目目录下，运行

```
gradlew clean runOsp
```

在启动OSP服务的同时，还会启动一个Jetty服务器，内置在线测试与在线文档的Web应用，向服务开发者提供服务文档展示和服务测试的功能。

如果我们在同一个机器上启动多个osp服务，注意要为服务设置不同的restport和port，防止端口冲突。

- 设置restport

对于codgen生成的venus项目，在ospServiceProjects.gradle中配置如下：

```
ospService {
    restport = 9091
}
```

- 设置port

对于codgen生成的venus项目，在ospServiceProjects.gradle中配置如下：

```
ospService {
    port = 1081
}
```

然后运行如下指令指定rpc的通信端口。

```
gradlew runOsp -PospService.port=8080
```

容器基于Java命令行运行

此方法更多用于搭建测试服务器。

step1. 打包服务：

打开CMD窗口通过cd命令进入你的项目目录下，运行

```
gradlew ospServiceEngineZip
```

在 `xxx-service/build/distributions` 目录中，会有一个 `xxx-service-[service version]-engine.zip` 的服务包

step2. 上传并解压

上传到任意服务，目录说明：

- `bin`: 自带启动脚本。
- `lib`, `lib/container` 与 `lib/platform/lib`: `osp-engine` 依赖包目录。
- `logs`: 自带启动脚本使用的日志目录，不在生产环境中使用。
- `servicesdir`: 服务实现包目录。
- `thirddir`: 服务的依赖包目录。
- `osp-engine.jar`: `osp-engine` 核心包。

step3. 运行

命令行进入 `osp-engine` 的解压目录

执行如下指令启动服务，默认端口为 1080，JMX 端口为 8060，容器日志将输出到 `./logs` 目录。

```
bin/osp-default.sh start
```

可以如下参数重新指定端口为 1081，JMX 端口为 8061，容器日志目录为 `/apps/logs/osp/`

```
bin/osp-default.sh start -p 1081 -j 8061 -l /apps/logs/osp/
```

还可以在命令行里给出更多的 JVM 参数，还可以覆盖在 `osp-default.sh` 中已配置的参数

下例指定端口为 8080，增加 `-Dosp.accesslog=false` 参数，并重新指定 JVM 内存大小为 2G。

```
bin/osp-default.sh start -p 8080 -Dosp.accesslog=false -Xmx2048m -Xms2048m
```

如果我们在同一个机器上启动多个 `osp` 服务，注意要为服务设置不同的 `port` 和 `jmx port`，防止端口冲突。

其他参数：

<code>-s</code>	<code>service api</code> ，如果设置了，在 OSP 平台日志目录下日志前缀为此参数，否则默认为 <code>osp</code> ，如果同一个服务器上起多个 OSP 服务，且使用相同日志目录，需设置此参数。
<code>-t</code>	<code>start timeout</code> ，OSP 服务启动的超时时间，默认为 30 秒。

执行如下指令停止服务

```
./bin/osp-default.sh stop
```

如果启动时指定了port与jmx port，在关闭时同样需要指定

```
bin/osp-default.sh stop -p 1081 -j 8061
```

在Windows上运行，可双击bin/Start.bat文件启动服务，默认端口同Linux版，直接关闭Console窗口停止服务。

step4. 进一步配置

在osp-default.sh中对端口，连接数，JVM内存等做了默认配置，进一步配置参见 the section called “服务配置”。

服务本地调试

见 Chapter#8, OSP项目示例演示。

服务本地测试

单元测试

服务实现本身不依赖OSP框架，因此可独立进行测试。

既可以使用JUnit与Mock工具类如 [Mockito](#) 编写单纯的单元测试，

也可以参考 Chapter#15，使用venus-test进行单元测试，编写集成测试。

功能测试

既可以运行 gradlew runOsp，在弹出的OSP在线测试页面进行测试。

也可以自行编写客户端代码进行测试，参考 the section called “容器基于Java命令行运行” 在测试服务器部署服务。

如果自行编写的测试客户端，想直接访问服务端，可配置Proxy的环境变量，直接连接服务端的地址和端口

```
VIP_OSP_LOCAL_PROXY=127.0.0.1:1080
```

41. 4#服务测试与发布

服务发布

1. 发布客户端SDK

Java版SDK

将Java的客户端上传到公司的Maven私服，服务接入方可通过Maven/Gradle下载客户端SDK进行开发。

对于codegen生成的venus项目，在项目根目录下的gradle.properties文件中，找到以下配置：

```
# define the repository
vipShopMavenCentral=http://mvn1.tools.vipshop.com/nexus/content/groups/public
vipShopMavenSnapshotRepository=http://mvn1.tools.vipshop.com/nexus/content/repositories/snapshots
vipShopMavenReleaseRepository=http://mvn1.tools.vipshop.com/nexus/content/repositories/releases
deploymentUsername=deployment
deploymentPassword=deployment123
```

这个配置的是公司的maven私服，可以根据实际情况修改配置。

然后在项目目录执行命令 gradlew uploadArchives 会进行编译并上传。

SDK包的group Id, version 在同样在项目根目录点gradle.properties中定义，而artifact Id是API项目的名称。

例如，在OSP快速入门示例中的Java SDK包会被上传到以下地址：

<http://mvn1.tools.vipshop.com/nexus/content/repositories/snapshots/com/vip/venus/venusosp-service-api/0.0.1-SNAPSHOT/>

PHP版SDK

在项目目录执行命令 gradlew uploadArchives 同样会进行PHP版SDK的编译与上传。

PHP版SDK将上传到与 Java版SDK同一目录，名字为[artifact Id]-[version]-php.zip。

也可以执行以下命令，单独打包PHP版SDK。

```
gradlew ospPhpZip
```

在 xxx-service-api/build/distributions 目录中，会有一个xxx-service-api-[service version]-php.zip 的SDK包。

2. 发布服务端包

打开CMD窗口通过cd命令进入你的项目目录下，运行

```
gradlew ospServiceEngineZip
```

在 xxx-service/build/distributions 目录中，会有一个xxx-service-[service version]-engine.zip 的服务包，将该包交给测试，运维。

3. 发布配置中心导入包

打开CMD窗口通过cd命令进入你的项目目录下，运行

```
gradlew ospServiceSimpleZip
```

在 xxx-service/build/distributions 目录中，会有一个xxx-service-[service version].zip 的服务包，可将该包交给运维，测试。

与以前的 gradlew ospServiceZip 相比，新打出来的包名字一样，但大小会少很多，不再包含无关的第三方依赖包。

4. 部署服务供客户端测试

参考 the section called “容器基于Java命令行运行” ，在测试服务器上部署服务。

服务测试

测试环境准备

统一ZK

建议尽量使用配置中心的QA环境ZK，通过分区来隔离不同的测试环境。

- 在 [QA环境配置中心UI](#) 上，系统管理→分区→创建， 创建新的分区

2. 配置环境变量

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.94:2181,10.101.18.94:2182,10.101.18.94:2183 ⓘ
VIP_CFGCENTER_PARTITION=###
```

① 是测试环境的公用配置中心ZK地址

部署Local Proxy

参考 the section called “OSP-Proxy的安装” 安装Local Proxy

设置环境变量

```
VIP_OSP_LOCAL_PROXY=127.0.0.1:2080
VIP_OSP_REMOTE_PROXY=192.168.44.59:2080 ⓘ
```

① 是测试环境的公用Remote Proxy地址。

在特殊情况下，可以不部署LOCAL_PROXY，将LOCAL_PROXY的地址也指向公用REMOTE_PROXY

41. 5#服务部署

环境变量设置

配置Zookeeper的信息，请参阅 Section#43. 1，“ZooKeeper及配置中心配置”，在运行前必须进行设置。

配置OSP-Proxy的信息

```
export VIP_OSP_LOCAL_PROXY=127.0.0.1:2080
export VIP_OSP_REMOTE_PROXY=osp-proxy-remote.vip.vip.com:2080
```

服务运行脚本

OSP服务在生产环境的运行脚本，将调用osp-engine自带的osp-default.sh，并进行定制端口，日志路径和额外的JVM参数。

```
/apps/dat/web/working/osp-proxy.vip.vip.com/bin/osp-default.sh start -p 2081 -j 8062 -l /apps/logs/osp/
xxx.api/osp -Xmx2048m -Xms2048m
```

Note

2.4.14版本以后增加了restful端口，如果指定了的话就会根据端口启动jetty。

示例：sh osp-default.sh start -r 8080 或者 osp-default.sh start --restful-port 8080， 默认为-1不启动jetty。

更多启动参数参考 the section called “容器基于Java命令行运行”

完整的脚本示例，可对JVM_ADD变量进行定制化，加入应用特定的参数：

```

#!/bin/sh
#created by root
#for start and stop OSP service
#date 20150805 1.0 version
. /apps/sh/app_env.sh
. /apps/sh/web_env.sh
. /apps/sh/java_env.sh
PATH=$PATH:/usr/sbin/
export PATH

if [ "$USER" = "root" ]; then
    echo "don't run by root";
    exit 0;
fi
ulimit -s 20480

##### changed for your own application #####
OSP_HOME='/apps/dat/web/working/osp-monitor.api.vip.com' #your application domain
JAVA_PORT=1081 #your application port
JMX_PORT=8061 #your application JMX port
LOGDIR=/apps/logs/osp #no need change
JVM_ADD="" #additional jvm ARGS

##### changed end #####
chmod +x $OSP_HOME/bin/osp-default.sh
CMD="$1"
shift
STOP()
{
    cd $OSP_HOME/bin/ && ./osp-default.sh stop -p $JAVA_PORT -j $JMX_PORT
}

START()
{
    cd $OSP_HOME/bin/ && ./osp-default.sh start -p $JAVA_PORT -j $JMX_PORT -l $LOGDIR $JVM_ADD
}

case "$CMD" in
    stop) STOP;;
    start) START;;
    restart) STOP;sleep 3;START;;
    *) echo "usage: $0 start|stop|restart [-p|--port port] [-j|--jmxport jmxport] [-l|--log-dir logdir]" ;;
esac

```

Note

不能使用kill -9 停止进程，否则服务不能优雅退出，无法从ZooKeeper中剔除自己。

服务配置

JVM启动参数

- 服务端工作线程池配置

使用-Dosp.container.threadpool.size=[size]#设置，默认为cpu处理器个数的两倍。

- 服务端连接处理线程池配置

配置Netty boss线程池大小，使用-Dosp.container.bossgroup.size=[size]设置，默认为cpu处理器个数的两倍。

- 服务端IO处理线程池配置

配置Netty worker线程池大小，使用-Dosp.container.workergroup.size=[size]设置，默认为cpu处理器个数的两倍。

- 闲置连接清理时长配置

使用-Dosp.channel.timeout=[timeoutValue]设置，单位为ms，默认为60秒。

- Osp平台级日志设置

使用-Dosp.logfile=[logpath]设置日志保存目录及前缀名，必须进行设置，没有默认值。示例如下：

/apps/logs/osp/foo，表明日志目录在/apps/logs/osp，文件的前缀为foo。

- 优雅退出超时时间

使用-Dosp.proxy.shutdownawait设置，单位为ms，默认为30秒。

Container启动参数

- port：服务端口号，不设置时默认值为从1080开始往上寻找空闲端口。
- restport：如果不指定restport，就不会启动Jetty，没有在线测试与在线文档功能，在生产环境一般不设置。
- servicesdir：服务包目录。

配置中心

服务在配置中心中的配置，参见 Section#43.3，“OSP服务的配置”

服务监控

1. 日志

参考 the section called “服务运行脚本” 中的设定

- OSP容器的启停日志在/apps/logs/osp/osp-osp.out
- OSP容器的GC日志在/apps/logs/osp/gc-osp.log
- OSP容器的平台级日志目录在/apps/logs/osp/

平台日志有三份，文件前缀为[SERVER_API]：

- xxx_access. yyyy-mm-dd.log：调用的简要信息。
- xxx_flow. yyyy-mm-dd.log：所有INFO以上级别的日志都会被打印。
- xxx_error. yyyy-mm-dd.log：异常记录。

不会按日志文件大小进行滚动，每天生成一个新文件，同时也不会进行自动滚动清理。

access log 内容

每次服务调用将产生一条日志，格式如下：

```
[##] - [OspEnd][/##### /Proxy## /##### ### ####### #### ###### ###### ###### ##### #####]
```

示例：

```
[2015-07-23 10:40:22] - [OspEnd][192.168.200.73 /192.168.200.74:34037 /192.168.200.75:1080 11
com.vip.infrastructure.qa.testosprouter.nfservice.nextversion.RouteNfTestService.getOspTestDemo()_2.0.0
0 228 244 2ms 1ms]
```

flow log 内容

可为任意的INFO及以上级别的日志。

error log 内容

每次异常产生一条日志，格式如下：

```
[##][###][####][##]-[##### #####  ### ##### ####]
```

示例

```
[2014-10-11 14:56:53.080] [OspThreadPool-6] [ERROR] [MethodDispatcher] -
[/192.168.97.206:60874 /192.168.97.206:1080 182
com.vip.osp.test.api.ServiceDemo.methodDemoError()_1.0.0 NullPointerExceptionnull]
java.lang.NullPointerException: null
at com.vip.osp.test.ServiceAImpl.methodDemoError(ServiceAImpl.java:53) ~[na:na]
at com.vip.osp.test.api.ServiceDemoHelper
$methodDemoError_Dispatcher.getResult(ServiceDemoHelper.java:1317) ~[na:na]
at com.vip.osp.test.api.ServiceDemoHelper$methodDemoError_Dispatcher.getResult(ServiceDemoHelper.java:1)
~[na:na]
at com.vip.osp.core.base.MethodDispatcher.process(MethodDispatcher.java:63) ~[osp-core-1.1.0-
SNAPSHOT.jar:na]
at com.vip.osp.container.filter.MethodProcessFilter.doFilter(MethodProcessFilter.java:28) [osp-
container-1.1.0-SNAPSHOT.jar:na]
at com.vip.osp.core.filter.DefaultOspFilterChain.doFilter(DefaultOspFilterChain.java:17) [osp-
core-1.1.0-SNAPSHOT.jar:na]
at com.vip.osp.container.filter.MonitorFilter.doFilter(MonitorFilter.java:31) [osp-container-1.1.0-
SNAPSHOT.jar:na]
at com.vip.osp.core.filter.DefaultOspFilterChain.doFilter(DefaultOspFilterChain.java:17) [osp-
core-1.1.0-SNAPSHOT.jar:na]
at com.vip.osp.container.filter.ServiceDegradeFilter.doFilter(ServiceDegradeFilter.java:35) [osp-
container-1.1.0-SNAPSHOT.jar:na]
at com.vip.osp.core.filter.DefaultOspFilterChain.doFilter(DefaultOspFilterChain.java:17) [osp-
core-1.1.0-SNAPSHOT.jar:na]
at com.vip.osp.container.filter.TraceFilter.doFilter(TraceFilter.java:90) [osp-container-1.1.0-
SNAPSHOT.jar:na]
at com.vip.osp.core.filter.DefaultOspFilterChain.doFilter(DefaultOspFilterChain.java:17) [osp-
core-1.1.0-SNAPSHOT.jar:na]
at com.vip.osp.core.processor.OspProcessor.process(OspProcessor.java:72) [osp-core-1.1.0-
SNAPSHOT.jar:na]
at com.vip.osp.core.processor.OspUniteProcessor.process(OspUniteProcessor.java:72) [osp-core-1.1.0-
SNAPSHOT.jar:na]
at
com.vip.osp.container.connection.ServerConnectionHandler.callService(ServerConnectionHandler.java:219)
[osp-container-1.1.0-SNAPSHOT.jar:na]
at com.vip.osp.container.connection.ServerConnectionHandler.access$000(ServerConnectionHandler.java:56)
[osp-container-1.1.0-SNAPSHOT.jar:na]
at com.vip.osp.container.connection.ServerConnectionHandler
$PoolCallableTask.run(ServerConnectionHandler.java:113) [osp-container-1.1.0-SNAPSHOT.jar:na]
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142) [na:1.8.0]
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617) [na:1.8.0]
at java.lang.Thread.run(Thread.java:744) [na:1.8.0]
```

2. HealthCheck接口

所有服务需要自行实现HealthCheck接口，向外汇报服务的健康情况。

HealthCheck接口日后可被Venus-VI 调用

3. Mercury

Mercury 负责提供集中式服务监控能力。 按 Chapter#18, 使用Mercury 的说明接入Mercury, 修改osp filter定义, 加入依赖包, 修改Logback配置。

如果基于codegen 1.3.4以上版本生成的项目, 已默认进行了配置。

4. Venus-VI

按照文档进行Venus-VI接入, 在Mercury界面将提供展示。

如果基于codegen 1.3.4以上版本生成的项目, 已默认进行了配置。

5. Zabbix

Zabbix负责监控进程存活, 并通过JMX作进一步的监控, 详见运维文档。

42. #OSP服务接入方使用手册

42. 1#Java客户端接入

Java客户端环境准备

依赖的 group, name, version可以在maven私服上查找到, Maven私服Web版地址: <http://mvn1.tools.vipshop.com/nexus/index.html>

必须置公司maven的私服仓库地址才可以取得到服务SDK包

Gradle工程:

添加依赖: 修改build.gradle文件

```
dependencies {
    compile group: 'com.vip.venus', name: 'venusosp-service-api', version:'0.0.1-SNAPSHOT'
}
```

配置仓库: 修改build.gradle文件

```
repositories {
    mavenLocal() ①
    mavenCentral() ②
    mavenRepo urls: "http://mvn1.tools.vipshop.com/nexus/content/groups/public" ③
}
```

① maven本地仓库

② maven中央仓库

③ 公司的maven私服

Maven工程:

添加依赖: 修改pom.xml文件

```
<dependencies>
    .....
    <dependency>
        <groupId>com.vip.venus</groupId>
        <artifactId>venusosp-service-api</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
    .....
</dependencies>
```

配置仓库: 修改settings.xml文件, 配置对所有的工程都生效

```
<?xml version="1.0" encoding="UTF-8"?>
<profiles>
    <profile>
        <id>mvn1</id>
        <repositories>
            <repository>
                <id>vipshop</id>
                <url>http://mvn1.tools.vipshop.com/nexus/content/groups/public </url>
            </repository>
        <releases>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
            <checksumPolicy>warn</checksumPolicy>
        </releases>
        <snapshots>
```

```

<enabled>true</enabled>
<updatePolicy>always</updatePolicy>
<checksumPolicy>warn</checksumPolicy>
</snapshots>
</repository>
</repositories>
<pluginRepositories>
<pluginRepository>
<id>vipshop</id>
<url>http://mvn1.tools.vipshop.com/nexus/content/groups/public </url>
<releases>
<enabled>true</enabled>
<updatePolicy>always</updatePolicy>
<checksumPolicy>warn</checksumPolicy>
</releases>
<snapshots>
<enabled>true</enabled>
<updatePolicy>always</updatePolicy>
<checksumPolicy>warn</checksumPolicy>
</snapshots>
</pluginRepository>
</pluginRepositories>
</profile>
</profiles>

```

Java客户端对服务的同步调用

示例代码如下：

```

public class MyClient {
    public static void main(String[] args) throws OspException {
        StudentServiceClient client = new StudentServiceClient(); ①
        StudentModel model = new StudentModel(); ②
        int create = client.create(model);
        // ####.....
    }
}

```

- ① 构造服务的客户端
- ② 服务同步调用

Java客户端对服务的异步调用

异步调用要自己写回调处理类，回调处理类必须实现OspAsyncCallback接口

示例代码：

```

public class MyClient {
    public static void main(String[] args) throws OspException {
        StudentServiceClient client = new StudentServiceClient(); ①
        client.asyncCreate(model, new OspAsyncCallback<Integer>() { ②

            public void onTimeout() { ③

            }

            public void onSuccess(Integer arg0) { ④
                // #########
            }

            public void onError(OspException arg0) { ⑤
                ...
            };
        });
        .....
    }
}

```

```

    }
}
```

- ① 获取服务的客户端
- ② 服务异步调用：内部类实现回调处理。client下以async开头的方法都是异步调用方法
- ③ create操作超时后的处理方法
- ④ create操作成功后的处理方法，操作成功后要进行的业务逻辑都写在这个方法中。
- ⑤ create操作失败后的处理方法

Warning

为不对平台回调线程池造成阻塞，影响系统性能，建议事件回调中逻辑应是一些快速处理的。

若回调代码含有复杂逻辑，建议用客户自建线程池进行处理的转交，尽快释放osp回调线程池。

示例代码如下：

```

public class MyClient {
    public static volatile Executor executor = Executors.newFixedThreadPool(10);

    class DemoTask implements Runnable {
        Integer result;

        public DemoTask(Integer result) {
            this.result = result;
        }

        @Override
        public void run() {
            // ##result#####
        }
    }

    public static void main(String[] args) {
        StudentServiceClient client = new StudentServiceClient();
        client.asyncCreate(model, new OspAsyncCallback<Integer>() {

            public void onTimeout() {

            }

            public void onSuccess(Integer arg0) {
                Runnable task = new DemoTask(arg0);
                executor.execute(task);
            }

            public void onError(OspException arg0) {
                ...
            }
        });
    }
}
```

Java客户端的InvocationContext设置

在调用服务方法前，可以使用InvocationContext API对调用进行设置。

```

InvocationContext invocationContext = InvocationContext.Factory.getInstance();
invocationContext.setCookie("foo", "123"); ❶
invocationContext.setCookie("x-bar", "abc"); ❷
```

Cookie设置可用于路由规则条件，不过此处的Cookie与Http的Cookie不同，仅用于一次性的信息传递，不会进行持久化，更接近于Http Header。

cookie的名字中如果带有 x- 前缀，在 A Service Client → B Service Provider → C Service Provider 的调用链中，将一直传递下去。

Warning

InvocationContext所提供的其他函数，在OSP进一步整理之前先不要使用。

Java客户端的测试

1. 在应用的单元测试中：

如果要Mock OSP的Java客户端，直接Mock Client类即可，比如在 Chapter#8, OSP项目示例演示 中生成的 StudentServiceClient 。

```
// mock OSP client
StudentServiceClient studentServiceClient = Mockito.mock(studentServiceClient.class);

// inject mock client to your service.
myService.setStudentClient(studentServiceClient);

// set mock return
Mockito.when(studentServiceClient.getStudent(1)).thenReturn(student);

// run your service's method
assertEquals("abc",myService.myMethod());
```

1. 如果要编写OSP Java客户端的Stub类，则需要自行去实现服务接口。每个服务的接口有两个：

- 仅包含同步接口的Service接口，比如在 Chapter# 8, OSP项目示例演示 中生成的 StudentService
- 包含同步和异步接口的Stub接口，比如在 Chapter# 8, OSP项目示例演示 中生成的 StudentServiceStub

如果客户端只需要使用同步接口，则在应用中定义类型为Service接口的变量，然后在测试时注入自行编写的Service接口的实现类。

```
public static class StudentServiceStub implements StudentService {

    @Override
    public int create(StudentModel studentModel) throws OspException {
        return 0;
    }

    @Override
    public int deleteByPrimaryKey(Long id) throws OspException {
        return 0;
    }
}
```

如果客户端需要异步接口，则在应用中定义类型为Stub接口的变量，然后在测试时注入自行编写的Stub接口的实现类，如果不需要同步接口可留空不实现同步的函数。

如果需要联调测试，见联调测试章节。

Java客户端配置

参见 Section#42.3，“OSP客户端配置”

Java客户端常见问题

1. 设置日志输出

OSP用Logback输出日志，logger name为 com.vip.osp，请在logback配置文件中自行配置。

42. 2#PHP客户端接入

PHP客户端的代码架构

客户端代码由三部分组成：

- runtime

```
##### http://wiki.corp.vipshop.com/pages/viewpage.action?pageId=33659431[OSP-PHP####] #####
## "osp-php-runtime"####.
```

Note

TODO：下个版本将放入Maven公司私服

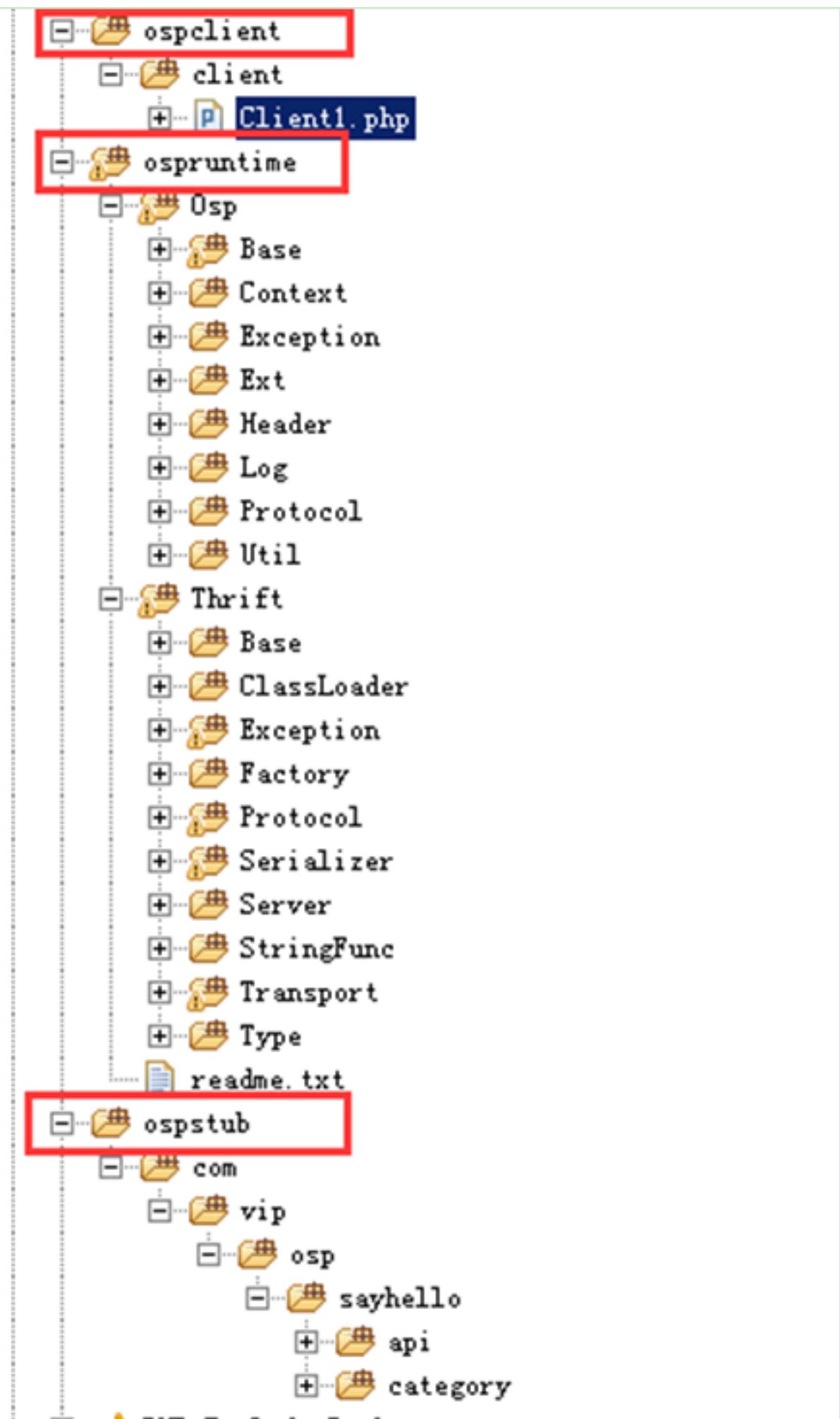
- osp服务的api stub

与服务相关的客户端存根，即服务端编译IDL文件时生成的stub代码，从服务提供方获取。

- PHP应用代码

PHP客户端只需要编写这一块。

整体代码架构如下图：



`ospclient`工程即接入方应用代码, 在这个工程中PHP客户端编写代码调用osp服务实现应用业务。

`ospruntime`工程即runtime代码。

`ospstub`工程即osp服务的api stub。

OSP-PHP C扩展安装部署方案

1. 概述

OSP-PHP支持C语言扩展（`osp_protocol`），在`osp_protocol`的环境下，经实际测算，C扩展比原生PHP序列化与反序列化性能能提升4-10倍，在高QPS与快速响应的api要求下，建议在PHP运行环境安装`osp_protocol`以提升OSP-PHP的代码性能。

2、`osp_protocol`安装

`osp_protocol`为标准的PHP C扩展程序，安装方法与业界标准的扩展安装方法基本一致。

安装方法参考以下步骤，具体操作以运维规范为准：

1. 下载 [扩展](#)
2. 在php源代码的ext目录下，新建目录`osp_protocol`
3. 把下载扩展中的4个文件`config.*`, `osp_protocol.*`, 拷贝到`osp_protocol`目录
4. cd进入`osp_protocol`目录，依次执行以下命令：

```
/usr/local/php5/bin/phpize ./configure --enable-osp_protocol
make
make install
```

Note

TODO: 下个版本将放入Maven私服

3、`osp_protocol`安装验证

1、使用`php -m`

`/apps/lib/php5/bin/php -m|grep osp`, 返回打印出'osp_protocol'则证明安装成功

2、使用SDK验证

`/apps/lib/php-5.3.17/bin/php ../../0sp/Check/extension.php` (`extension.php`路径以实际代码路径为准)，返回1则证明安装成功

3、使用nginx验证

<http://xxx/…/0sp/Check/extension.php> (`extension.php`路径以实际代码路径为准)，返回1则证明安装成功

4、`osp_protocol`使用

runtime自动识别`osp_protocol`，如果存在`osp_protocol`会优先使用扩展处理，扩展使用Binary协议进行序列化处理

PHP客户端对服务的调用

PHP客户端业务对osp服务的调用，示例代码如下：

```

<?php
//1####runtime#####
global $OSP_RUNTIME_PATH;
$OSP_RUNTIME_PATH = ".../..../ospruntime/";

//2#####php#####namespace+serviceName+"Client"#####
require_once '/ospstub/com/vip/osp/sayhello/api/ServiceDemoClient.php';
require_once '/ospstub/com/vip/osp/sayhello/category/CategoryClient.php';

//3###Stub#####
$serviceDemoService = \com\vip\osp\sayhello\api\ServiceDemoClient::getService();
$categoryService = \com\vip\osp\sayhello\category\CategoryClient::getService();

//4###InvocationContext#####
$ctx = \Osp\Context\InvocationContextFactory::getInstance();

$ctx->setLogPath ( "/app/logs/php/log/" ); //##log#####
$ctx->setLogLevel ( "info" ); //##debug,info,error####error##

$cookie = array();
$cookie["foo"] = "abc";
$cookie["x-bar"] = "123";
$ctx->setCookie($cookie); //##cookie#####Java#####

try {
    $_structDemo = new \com\vip\osp\sayhello\api\StructDemo();
    $_structDemo->_int = 465633;

    //5#####
    $serviceDemoRnt = $serviceDemoService->methodDemo($_structDemo);
    var_dump($serviceDemoRnt);

    $categoryRnt = $categoryService->getCategoryById("12334");
    var_dump($categoryRnt);
} catch (\Osp\Exception\OspException $e) {
    var_dump($e);
    echo $e->getReturnCode();
    echo $e->getReturnMessage();
} catch (\Exception $e) {
    //7#####
    var_dump($e);
}
}

```

Note

代码的顺序请按照上面例子来，尤其是`getService()`这一步骤，一定要在`new xxx()`实体对像之前，原因在于`getService()`这方法除了实例化Stub外，还会执行classloader机制逻辑，通过classloader，程序把相关依赖的php文件与php class作自动扫描及依赖绑定。如果在`getService()`之前就调其他逻辑如`new xxx()`等，这样就会报错。

Warning

`ctx`所提供的其他函数，在OSP进一步整理之前先不要使用。

PHP客户端的异常处理

通用的异常处理代码如下：

```

try {

```

```
//do something
} catch (\Osp\Exception\OspException $e) { ❶
    var_dump($e);
    echo $e->getReturnCode();
    echo $e->getReturnMessage();
} catch (\Exception $e) {
    var_dump($e);
    echo $e->getCode();
    echo $e->getMessage();
}
```

❶ 业务异常已经封装成\Osp\Exception\OspException了，调用方法时，需要先捕获OspException

Note

SOCKET异常为\Thrift\Exception\TException，客户端没有作封装处理，调用时可以获取TException，也可以直接捕获Exception

PHP客户端的常见问题

1. 调用api时报错: TSocket: Could not connect to osp-proxy.vip.vip.com:2080

解决方式：

- 检查osp-proxy.vip.vip.com的域名解析，目标域名是否指向proxy主机
- 检查 telnet osp-proxy.vip.vip.com 2080，看看能不能连接得上
- 通过配置环境变量VIP_OSP_SEND_TIMEOUT，调大连接超时时间

2. 调用api时报错: TSocket: timed out reading 4 bytes from osp-proxy.vip.vip.com:2080

解决方式：

- 查看proxy处理日志，跟踪交易trace
- 查看osp服务处理日志，跟踪交易trace
- 通过配置环境变量VIP_OSP_SEND_TIMEOUT，调大连接超时时间

42. 3#OSP客户端配置

Java客户端与PHP客户端共用的配置，将尽量配置在环境变量里。

另外，新版的Java与PHP的OSP客户端将完全不使用配置中心。

如果需要连接真正的服务端，请访问由服务端团队部署的Proxy和服务

请参考 [测试环境公共资源](#) 配置Proxy及在对应的配置中心进行服务配置。

osp-proxy地址的环境变量

OSP客户端需要连OSP-Proxy，再由OSP-Proxy发起对osp服务的请求。

新版的OSP-Proxy使用本地Proxy的模式，总是先尝试连接本地Proxy；当本地Proxy失效时，连接作为HA备份的中央Proxy集群。

所以我们需要配置环境变量，对本地和中央Proxy进行指定(osp-proxy的域名和端口)。

非生产环境（开发/测试/回归测试）：

在开发环境，如果不准备部署LOCAL_PROXY，可以将它的值设为与REMOTE_PROXY一样，否则可参考the section called “OSP-Proxy的安装”，自行部署LOCAL_PROXY，并设为127.0.0.1:28080

测试环境

```
VIP_OSP_LOCAL_PROXY=192.168.44.59:2080
VIP_OSP_REMOTE_PROXY=192.168.44.59:2080
```

回归测试环境

```
VIP_OSP_LOCAL_PROXY=10.199.172.91:2080
VIP_OSP_REMOTE_PROXY=10.199.172.91:2080
```

开发环境

```
VIP_OSP_LOCAL_PROXY=192.168.44.61:2080
VIP_OSP_REMOTE_PROXY=192.168.44.61:2080
```

生产环境：

直接配置环境变量：

```
export VIP_OSP_LOCAL_PROXY=127.0.0.1:2080
export VIP_OSP_REMOTE_PROXY=osp-proxy-remote.vip.vip.com:2080
```

- **VIP_OSP_LOCAL_PROXY**

为了保持兼容性，当此环境变量未定义时，OSP客户端完全不会尝试连接本地客户端。

- **VIP_OSP_REMOTE_PROXY**

有两个选项：osp-proxy-remote.vip.vip.com（东莞） gd9-osp-proxy-remote.vip.vip.com

因为生产环境提供集中的OSP-Proxy环境，会对osp-proxy.vip.vip.com这域名进行DNS解析及LVS服务，所以不需要配置hosts。

当此环境变量未定义时，OSP客户端会默认使用osp-proxy.vip.vip.com（东莞）

Note

当VIP_OSP_LOCAL_PROXY 与 VIP_REMOTE_OSP_PROXY 都没配置时，默认使用东莞机房的Proxy集群。

其他环境变量

- **VIP_OSP_SEND_TIMEOUT**: 即SO_CONNECTION_TIMEOUT，连接超时，单位为ms，默认值为5秒。
- **VIP_OSP_RECV_TIMEOUT**: 即SO_SOCKET_TIMEOUT，消息发送超时，单位为ms，默认值为10秒。

Java客户端的JVM启动参数

JVM启动参数可配置在客户端服务器的脚本中。

- 客户端连接池配置

使用-Dosp.stub.max-connections-per-server=[value] 配置，默认值为512.

43. #OSP公共知识

Note

OSP服务提供者和接入者都需要了解的知识。

43. 1#ZooKeeper及配置中心配置

OSP服务端需要使用注册中心来注册服务供客户端使用，因此OSP服务端需要配置注册中心的zk。

我们可以通过环境变量或者系统变量或者properties来配置注册中心zk集群。

properties配置方式如下：在gradle.properties中配置：

```
#zk config
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:2181
```

properties配置方式仅对gradlew runOsp启动方式有效，否则还是需要在环境变量/系统变量配置VIP_CFGCENTER_ZK_CONNECTION

OSP服务的环境分为开发环境，测试环境，回归测试环境和生产环境，分别用于OSP服务开发的不同阶段。

- 开发环境主要是开发人员开发的环境。
- 测试环境主要是QA进行一些功能性测试。开发环境和测试环境共用一套环境。
- 回归测试环境主要是集成测试，用于联调。
- 生产环境就是服务正式上线部署的环境。

开发环境：

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.110:2181,10.101.18.111:2181,10.101.18.112:2181
```

测试环境：

```
VIP_CFGCENTER_ZK_CONNECTION=10.101.18.94:2181,10.101.18.94:2182,10.101.18.94:2183
```

回归测试环境：

```
VIP_CFGCENTER_ZK_CONNECTION=10.199.166.238:2181,10.199.166.239:2181,10.199.166.240:2181
```

生产环境：

```
VIP_CFGCENTER_ZK_CONNECTION=gd6-cfgcenter-zk-001.idc.vip.com:2181, gd6-cfgcenter-zk-002.idc.vip.com:2181, gd6-cfgcenter-zk-003.idc.vip.com:2181
```

原则上，建议测试环境尽量使用配置中心的QA环境ZK，通过分区来隔离不同的测试环境。

1. 在 [QA环境配置中心UI](#)上，系统管理→分区→创建， 创建新的分区
2. 配置环境变量

```
VIP_CFGCENTER_PARTITION=###
```

43. 2#OSP-Proxy

公用Remote Proxy

- 开发环境: *

```
VIP_OSP_REMOTE_PROXY=192.168.44.61:2080
```

- 测试环境: *

```
VIP_OSP_REMOTE_PROXY=192.168.44.59:2080
```

- 回归测试环境: *

```
VIP_OSP_REMOTE_PROXY=10.199.172.91:2080
```

生产环境:

```
VIP_OSP_REMOTE_PROXY=osp-proxy-remote.vip.vip.com
```

Remote Proxy 会连入该环境相应的ZK的default分区（ZK地址见上），如果服务端使用了ZK的分区，需要另外部署Proxy。

OSP-Proxy的安装

在 <http://mvn1.tools.vipshop.com/nexus/content/repositories/releases/com/vip/osp/osp-proxy/>

下载最新版本的osp-proxy-[version]-zip.zip，解压到本地任意目录。

目录说明:

- bin: 自带启动脚本，在生产环境中不会直接使用。
- lib: osp-proxy依赖包目录。
- logs: 自带启动脚本使用的日志目录，在生产环境中不会使用。
- osp-proxy.jar: osp-proxy核心包。

本地环境运行

可解压到任意目录，进入目录中运行标准脚本。

执行如下指令启动Proxy。监听端口2080，JMX端口8061，日志输出到logs目录。

```
bin/osp-proxy-default.sh start
```

可以如下参数重新指定端口，设定日志目录，增加额外的JVM 启动参数

```
bin/osp-proxy-default.sh start -p 2081 -j 8062 -l /apps/logs/osp/osp-proxy/osp-proxy -Dosp.flowlog=true  
-Dosp.accesslog=true
```

执行如下指令停止Proxy，如果启动时指定了端口，关闭时同样需要指定相同端口

```
bin/osp-proxy-default.sh stop -p 2081 -j 8062
```

在Windows上运行，可双击./bin/osp-proxy.bat 文件启动Proxy，默认端口同Linux，直接关闭Console 窗口停止Proxy。

生产环境运行

服务部署请参见 [运维的部署文档](#)

- local-proxy 主机目录: /apps/dat/web/working/osp-proxy/
- 中央proxy 主机目录 /apps/dat/web/working/osp-proxy-remote.vip.vip.com
- 启动脚本: /apps/sh/osp/osp-proxy.sh
- 监听端口: 2080 / JMX 端口 2061, 注意不要与其他应用的JMX端口重复

生产环境的osp-proxy.sh脚本, 将调用 Osp Proxy 自带的osp-proxy-default.sh脚本, 并传入新的参数, 或覆盖原有的参数(如内存大小)。

```
bin/osp-proxy-default.sh start -p 2081 -j 8062 -l /apps/logs/osp/osp-proxy/osp-proxy -Dosp.flowlog=true
-Xmx2048m -Xms2048m
```

完整的脚本示例

```
#!/bin/sh
#created by root
#for start and stop OSP service
#date 20150805 1.0 version

. /apps/sh/app_env.sh
. /apps/sh/web_env.sh
. /apps/sh/java_env.sh

if [ "$USER" = "root" ]; then
    echo "don't run by root";
    exit 0;
fi

ulimit -s 20480

#changed for your own appliation
OSP_HOME='/apps/dat/web/working/osp-proxy.vip.vip.com'
JAVA_PORT=3080
JMX_PORT=3060
LOGDIR=/apps/logs/osp/osp-proxy
JVM_ADD=""
#changed end

PATH=$PATH:/usr/sbin/
export PATH

CMD="$1"
shift
STOP()
{
    cd $OSP_HOME/bin/ && ./osp-proxy-default.sh stop -p $JAVA_PORT -j $JMX_PORT
}

START()
{
    cd $OSP_HOME/bin/ && ./osp-proxy-default.sh start -p $JAVA_PORT -j $JMX_PORT -l $LOGDIR $JVM_ADD
}

case "$CMD" in
    stop) STOP;;
    start) START;;
    restart) STOP;sleep 3;START;;
    *) echo "usage: $0 start|stop|restart [-p|--port port] [-j|--jmxport jmxport] [-l|--log-dir logdir]
[additional jvm args]";;
esac
```

OSP-Proxy的升级

OSP-Proxy 支持平滑升级，可以在不摘除流量的情况下进行升级，在升级过程中客户端自动切换到

Local Proxy升级

1. 停止Proxy，运行 `/apps/sh/osp/osp-proxy.sh stop`
2. 等Proxy彻底停止后，将新版Proxy的解压目录 Link 到 `/apps/dat/web/working/osp-proxy`
3. 启动Proxy，运行 `/apps/sh/osp/osp-proxy.sh start`

Note

可同时批量更新多台Local Proxy，只要集中Proxy集群的容量足够支撑Local Proxy升级时的流量。

集中Proxy集群升级

1. 停止Proxy，运行 `/apps/sh/osp/osp-proxy.sh stop`
2. 等Proxy彻底停止，将新版Proxy的解压目录 Link 到 `/apps/dat/web/working/osp-proxy2.vip.vip.com` 或 `/apps/dat/web/working/gd9.osp-proxy2.vip.vip.com`
3. 启动Proxy，运行 `/apps/sh/osp/osp-proxy.sh start`

集中Proxy仅作为备份存在，在正常情况下没有流量。所以即使Proxy停止过程中，在LVS判断到Proxy失效前的流量丢失不会造成实际影响。

OSP-Proxy的配置

环境变量配置

Zookeeper信息的相关配置，请参阅 Section#43.1，“ZooKeeper及配置中心配置”，在运行前必须进行设置。

JVM启动参数

- 客户端连接池配置

使用`-Dosp.stub.max-connections-per-server=[value]` 配置，默认值为1.

- 服务端连接处理线程池配置

配置Netty boss线程池大小，使用`-Dosp.proxy.bossgroup.size=[size]`设置，默认为cpu处理器个数的两倍。

- 服务端I/O处理线程池配置

配置netty worker线程池大小，使用`-Dosp.proxy.workergroup.size=[size]`设置，默认为cpu处理器个数的两倍。

- OSP平台级日志设置

使用`-Dosp-proxy.logfile=[logpath]`设置目录，必须进行设置，没有默认值。示例如下：

/apps/logs/osp/osp-proxy/foo，表明日志目录在/apps/logs/osp/osp-proxy，文件的前缀为foo。

使用-Dosp.accesslog=false 屏蔽 accesslog，默认打开，详见日志部分。

使用-Dosp.flowlog=true 打开flowlog，默认屏蔽，详见日志部分。

Container启动参数

- port: 服务端口号，不设置时默认值为从1080 开始往上寻找空闲端口。

OSP-Proxy的监控

OSP-Proxy同样支持日志，Mercury，Hermes等方式进行监控，参见 the section called “服务监控”在生产环境

- OSP Proxy的启停日志在/apps/logs/osp/osp-proxy/osp-proxy.out
- OSP Proxy的GC日志在/apps/logs/osp/osp-proxy/gc.log
- OSP Proxy的平台级日志目录在/apps/logs/osp/osp-proxy/

平台日志有三份，文件前缀为[SERVER_API]：

- xxx_access.yyyy-mm-dd_i.log: 调用的简要信息。
- xxx_flow.yyyy-mm-dd_i.log: 所有INFO以上级别的日志都会被打印，默认被屏蔽。
- xxx_error.yyyy-mm-dd_i.log: 异常记录。

会按文件个数进行滚动清理，access log/flow log 最多10个文件，每个文件1G; error log最多10个文件，每个文件100mb。

access log示例

每次服务调用将产生两条日志，格式如下：

```
[##] - [ProxyStart][CSeq:##### /##### /Proxy## ##### ###### #### ####]
[##] - [ProxyEnd][CSeq:##### PSeq:Proxy### /##### /Proxy## /##### ##### #### #### ####]
```

示例：

```
[2015-07-23 12:18:59.293] - [ProxyStart][CSeq:17 /192.168.200.73:54817 /192.168.200.74:2080
com.vip.infrastructure.qa.testosprouter.nfservice.nextversion.RouteNfTestService.getOspTestDemo()_2.0.0
193]
[2015-07-23 12:18:59.293] - [ProxyEnd][CSeq:17
PSeq:17 /192.168.200.73:54817 /192.168.200.74:34061 /192.168.200.75:1080
com.vip.infrastructure.qa.testosprouter.nfservice.nextversion.RouteNfTestService.getOspTestDemo()_2.0.0
0 193 244 0ms]
```

43. 3#OSP服务的配置

OSP服务支持在配置中心的统一配置，也可使用基于IDL annotation的设置。

优先级

配置中心的配置优于IDL annotation的配置，细粒度的配置（方法级别）优于 粗粒度的配置（服务级别）。

因为配置中心暂时不支持方法级别的配置，所以 annotation的方法级别配置 > 配置中心的服务级别配置 > annotation的服务级别配置。

- **开发期**

未导入配置中心前，直接使用IDL中的配置。

- **新服务上线**

导入配置中心，配置中心的服务级别的配置会直接使用OSP中的默认值，客户必须重新进行配置。

Note

TODO. 配置中心下一个版本将支持导入配置中心时，导入annotation的服务级配置。

- **服务升级**

导入配置中心，配置中心会导入上一个服务版本的配置，忽略annotation的服务级配置。

配置项

与服务Annotation相同的配置：

- 是否使用线程池
- 并发数
- 负载均衡策略
- 超时重试次数
- 超时时间

其他配置：

- 服务是否被降级：可在服务端动态设置服务降级。
- 路由控制配置：详见 Section#43.5，“OSP路由功能”
- 是否使用路由：是否使用路由的开关量。
- 是否优先本地机房：是否使用本地机房(ip前两位相同)的匹配规则。

43.4#OSP的异常处理

返回信息机制

OSP异常包含返回码与详细信息。其中详细信息由三段的值组成：Location_Level_Message，比如：

```
CALLER_SYSTEM_NullPointerException
PROXY_SYSTEM_NullPointerException
CALLEE_USER_NullPointerException
```

字段解释：

- Location

标识异常发生的源头位置，可能值包括CALLER, PROXY, CALLEE。 CALLER表示异常发生在客户端，比如调用还没开始发送到服务端即已发生异常，或收到服务端的返回后反序列化出现异常。 同理， PROXY表示异常发生的OSP-Proxy， CALLEE标识异常发生在服务端。

- Level

标识异常是由OSP平台抛出的，还是服务提供者的服务实现代码抛出的。由OSP平台抛出的异常，值为SYSTEM，反之为USER。

- Message

异常返回的详细信息。

异常处理机制

1. 连接不通

如果一台服务端容器已在Zookeeper注册，但由于某种特殊原因不能进行初始连接，当客户端访问它时会抛出错误码为OSP_SERVICE_CONNECT_FAILED的异常。

2. 超时重试

如果一台服务端容器已在Zookeeper注册，但是服务器资源繁忙，或由于某种特殊原因不能再提供服务，OSP-Proxy将会根据配置中心里该服务的“超时重试次数”项，在其他服务端容器上进行重试。

3. ZK连接失效

如果一台服务端容器已失效，但未在Zookeeper进行取消注册(如kill -9 直接停止进程)，ZooKeeper将在秒级的时间内发现临时文件失效并通知客户端。

43. 5#OSP路由功能

路由是OSP 服务治理能力一个重要部分，可以通过路由支持服务上线 / 下线，在线测试，机房选择，A/B测试，灰度发布，流量控制和优雅降级等操作。

用户可以通过配置中心自定义规则，推送到OSP，实现流量分配和负载均衡策略。

客户端路由模型

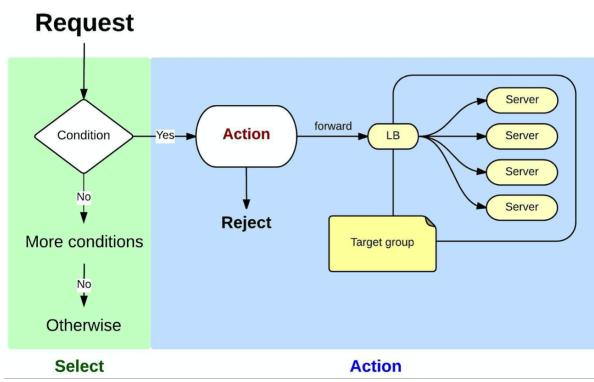
客户端发起请求，OSP根据请求的Header和Cookie信息，作为路由规则的选择，当匹配到某个路由规则后，会执行对应的Action。Action有Reject，拒绝这个请求，或Forward，Forward到某个Target Group，该Target Group包括多台server，server可以有自己的权重信息，作为灰度发布和流量控制操作。

同时，如果Action的类型是Forward，OSP提供了多种类型的LoadBalance策略。比如有：RandomRobin, RoundRobin 和LeastActive。 系统默认是LeastActive，服务开发方可以到配置中心配置服务的LoadBalance策略。

LeastActive 策略表示服务请求优先选择最少操作的服务器进行请求处理。

RoundRobin 策略表示服务请求通过轮询的方式找服务器进行请求处理。

RandomRobin 策略表示服务请求通过随机的方式找服务器进行请求处理。



Figure#43. 1. #客户端路由模型

路由处理方式

Select Phase

OSP检查请求Header或Cookie (K/V)，按路由规则定义的优先顺序执行，路由规则被选中即终止。

Header 支持的字段如下，注：大小写必须匹配。

Cookie是key/value，由调用方自己设置，注：大小写必须匹配。

支持的数据类型：字符串，数字，IP(支持CDIR格式)，正则表达式。

支持的操作类型： 完成匹配，范围匹配(逗号分割的字符串)，布尔操作(and, or, not)，IP Range匹配(192.168.200.12-24)，表示IP从12到24的IP地址。

支持的Otherwise分支：当所有路由规则都没有匹配到，路由可以设置Otherwise分支，类似IF Else语法。

Table#43. 1. #Header Name 定义

Header Name	描述	提供方
method	调用方法名	OSP 提供
service	调用服务名称，包括服务的package	OSP 提供
version	调用服务版本号	OSP 提供
callerip	服务调用方IP	OSP 提供

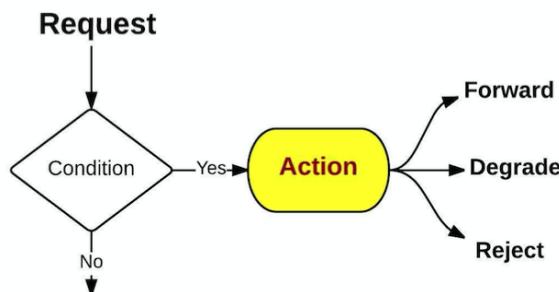
Action Phase

当路由规则匹配后，则执行对应的Action，Action支持多目标集+权重，目标集的负载均衡，路由决定(转发，拒绝)

在客户端路由模型里面提到过LoadBalance策略和权重的概念，它们可以互相结合起来使用。比如LoadBalance策略是RoundRobin，某条规则的Target Server设置了三台Server，Server A设置权重为1，Server B设置权重为2，Server C设置权重为3，当某个路由规则匹配，则服务请求都会路由到这个Target Server的三台机器上。处理请求的Server顺序为A, B, C, B, C, C。这个顺序是因为权重起了作用。

当LoadBalance策略设置为LeastActive，TargetServer三台机器的权重比例是1:2:3，当同时有4个并发，各服务器都有一个Active的请求，这是第四个请求会选择权重为3的服务器C。因为服务器C的权重最高。

注：LoadBalance策略和权重的互相结合的结果是当请求量达到一定的数量的时候，处理请求数越符合权重的比例。



Figure#43. 2. #请求操作机制

代码示例

```

##Cookie#####Cookie#####
@Override
public <I> void doBusiness() {
    InvocationContextImpl ctx = new InvocationContextImpl();
    try {
        ctx.setCookie("x-foo", "foo");
        //##OSP Client API
    } catch (Exception e) {
    } finally{
        ctx.getCookie().clear();
    }
}
  
```

配置中心

配置中心使用请参考配置中心使用文档。

43. 6#静态路由表

背景

在OSP服务端和客户端应用开发过程中，为了免去不必要的、繁琐的操作步骤，达到快速测试服务的目的，客户端希望不连Proxy，不走路由，不连ZK，直连服务端。尽管客户端可通过设置InvocationContext来指定调用的服务端，但是这种方法修改了客户端代码，并不推荐使用。因此，客户端希望能通过配置的方式来指定调用的服务端。

配置使用说明

- 使用静态路由表，无需修改任何代码。
- 配置环境变量，变量名：VIP_OSP_STATIC_ROUTE，变量值格式：
serviceName:serviceVersion:methodName=ip:port:timeout;serviceName2:serviceVersion2:methodName2=ip2:port
其中，:timeout是可选的，serviceName、serviceVersion、methodName可接收正则表达式。规则可

以理解为，一个或多个服务的某个或某些方法，映射到某台服务器，附带设置timeout，多条规则使用英文分号隔开。

- 特别的，对于java客户端，还可以通过jvm的-Dosp.static.route参数来配置它，参数值格式和环境变量的配置相同。
- 对于java客户端，通过jvm参数配置的静态路由表的优先级大于通过环境变量配置的静态路由表。

其他注意事项

- 静态路由表可配置多条匹配规则，使用英文分号隔开。一旦匹配，即返回其配置的服务端，不会尝试匹配其后的规则。
- 如果通过InvocationContext指定了服务端，那么静态路由表将不起作用。
- 如果通过静态路由表没有匹配到服务端，如果配置了Proxy，那么将尝试连接Proxy。
- 如果配置静态路由表有误，那么程序将继续执行，只是有关静态路由表的错误将会在日志中体现。

43. 7#自定义OspFilter

OSP引入了Filter Chain的设计模式，让服务开发者通过增加Filter到Chain里面，满足客户埋点的需求。客户端和Proxy端都是需要支持同步调用和异步调用，因此客户端和Proxy端的Filter Chain拆分为Send Filter Chain和Receive Filter Chain，顾名思义，一条是发送的Filter Chain，主要处理是client端发送消息到proxy端，和Proxy端发送消息到服务端的调用链。

服务端是同步调用，所以Proxy端的调用方法会经过Proxy端的Send Filter Chain，然后发送请求，到达服务端，服务端拿到请求，经过服务端的OspFilterChain里所有的filter，调用服务方法。

OspFilter的编写

OSP的Filter必须extends AbstractOspFilter抽象类。AbstractOspFilter接口中有3个方法：init, destroy, doFilter和onException方法。其中doFilter和onException方法都需要传参FilterContext，这是Filter间进行数据传输的Container，如果在Filter执行过程中需要把处理的临时结果或者上下文信息进行临时存放，供后面的filter使用，可以把数据放到FilterContext中。

AbstractOspFilter的所有工作都集中在doFilter方法中。如果在Filter的调用过程中出现错误，系统会执行已经执行过的Filter的onException方法，所以Filter开发者想对Filter在处理过程中报Exception的时候做些业务处理，可以在这个方法里面做。

示例代码如下，也可参考MercuryOspTraceFilter.java

```
public class MyFilter extends AbstractOspFilter {
    @Override
    public <I> void doFilter(FilterContext filterContext) throws TException {
        //#####
        .....
    }

    @Override
    public void onException(FilterContext filterContext, Throwable e){
        logger.error("##");
    }
}
```

OspFilter的配置

服务开发者可以在客户端增加自己的Filter，把新增的Filter注册到客户端的/ospconfig/client_filter.properties里面。

服务开发者可以在服务端增加自己的Filter，把新增的Filter注册到客户端的/ospconfig/filter.properties 里面。

同样Proxy端也可以增加Filter到Proxy端的/ospconfig/client_filter.properties，主要是满足Mercury可以在Proxy端加入自己的Filter，一般不提供给服务开发者使用。

Note

client_filter.properties分为Before Send Filter和After Receive Filter，服务开发者需要考虑自己开发的Filter是放到哪条Chain里面。

client_filter.properties示例代码如下：多个filter用,隔开

```
beforeSendFilters=com.vipshop.mercury.client.filter.MercuryOspStubFilter
afterReceiveFilters=com.vipshop.mercury.client.filter.MercuryOspStubFilter2
```

filter.properties示例代码如下：多个filter用,隔开

```
filters=com.vipshop.mercury.client.filter.MercuryOspTraceFilter
```

TransactionContext使用

服务端上下文，获取TransactionContext实例后，这个实例中很多方法，可以利用这些方法得到一些服务的相关信息

示例代码如下：

```
TransactionContextImpl transactionContext =
    (TransactionContextImpl) TransactionContext.Factory.getInstance();
String method = transactionContext.getMethod();
```

43. 8#常见问题

集合元素为Null

因为Thrift网络传输的集合元素不可为Null，无论是客户端调用osp服务，还是osp服务响应客户端时，用到的集合内的元素均不能为Null。

如何设置超时

注意有两种超时设定：

一种是客户端到Proxy的超时，默认是5秒，可在代码里设置InvocationContext中的timeout属性，单位为ms。

一种是Proxy到服务端的超时，可在配置中心配置。

OSP 服务第一次访问超时

OSP从客户端经过代理再到服务端的首次调用链初始化大约需要几十毫秒的时间。

如果出现首次方法超时，可先检查服务的Timeout时间是否正确设置 然后检查服务的实现有无造成首次调用的可能，下面是一些可能的原因：

- 请检查Spring 的初始化是否使用了Lazy Load的方式。

服务开发者在使用Spring进行Bean的生命周期管理时，可以使用lazy-load的方式初始化Bean，e.g. default-lazy-init="true"，这样会导致在第一次调用服务的时候，才花费时间来创建Bean，从而导致服务超时。建议需要耗时创建的Bean尽量不要设为lazy-load。

- 请检查数据库的Data Source是否主动调用了init方法初始化连接池。

要避免首次访问数据库时才被动初始化连接池。比如用Spring定义DataSource时，需设置 init-method。

```
<bean id="dataSourceS0" class="com.vip.venus_codegen.repository.DataSource"
      destroy-method="close" init-method="init">
```

除数据库之外其他资源的初始化，也可能存在类似的问题。

Part#VIII. #Tools

Venus framework提供各种工具来提高开发的效率

44. 代码生成器 (Venus Codegen)	216
44.1. Codegen功能介绍	216
44.2. 命令行版本使用说明	216
生成venus项目	216
只生成项目结构	219
只生成代码	219
44.3. UI版本版本使用说明	220
生成venus项目功能	220
只生成代码功能	222
44.4. codegen使用注意事项	223

44. #代码生成器 (Venus Codegen)

Codegen 是一款用来生成venus framework代码风格和项目结构的代码生成器工具，可以利用它去生成指定类型的项目。

目前codegen只支持生成gradle项目，不支持生成maven项目。

codegen可以生成simple service, simple service+ webapp项目, OSP service项目, webapp+OSP service项目。

目前venus codegen有命令行和UI两种版本，不喜欢命令行的同学可以使用UI版本。

44. 1#Codegen功能介绍

Codegen可以根据数据库中的database和table, 自动生成一套venus风格的gradle项目。目前只能支持只支持一个数据库多个数据表，还不支持多数据库。

目前命令行版本的codegen支持三种功能：

执行命令codegen，生成项目结构和相关代码；

执行命令codegen -projectstructure，只生成项目结构不生成代码；

执行命令codegen -codeonly, 这种方式是用于需要使用多个数据库或者生成项目代码后又有新的表增加/表变更，只需要生成新的表的代码到已有的项目中去，不生成项目关联的gradle配置等文件

目前gui版本只支持参数codeonly。

Note

目前使用venus codegen生成项目代码时，必须连接数据库，从表结构起步生成对应的数据库访问代码和项目代码，

因此在使用venus codegen前，我们必须在数据库中创建database和table

Warning

如果codegen工具中涉及的table有外键，那么在生成的junit代码中需要用户手动修改代码以确保这个外键的值已在对应的数据表中存在

如果在服务启动的时候报错“cannot run program “bash””，需要在环境变量里配置GIT_HOME。

44. 2#命令行版本使用说明

生成venus项目

step1： 在数据库中建立数据库database和数据表table

step2： 下载codegen.zip

在 [下载codegen-1.3.8](#)，解压缩至本地磁盘指定位置。

step3： 修改配置文件

进入codegen/bin文件夹，打开config.properties配置文件，可以看到一些配置项包括项目配置和数据库配置，如下所示：

```

#####
#   project configuration
#####
project.exportPath=E:\\codegen-demo ①
project.name=demo ②
project.packageName=com.vip.venus ③
project.moduleName=crm ④
project.type=ospService,webapp ⑤
project.buildTool=gradle ⑥

#####
#   database configuration
#####
db.driver=com.mysql.jdbc.Driver ⑦
db.url=jdbc:mysql://10.101.18.72:3306/db1_master ⑧
db.userId=vipuser ⑨
db.pwd=xR54PUU8GWia ⑩
db.name=db1_master ⑪
db.tables=tb_1:User,t_vlash_permission:VlashPermission,test_datatype:DataType ⑫
db.shardingTables= ⑬

#####
# ##project##database#####

```

- ① `project.exportPath` 表示生成项目的路径, windows操作系统, 可以指定路径为D:/XX文件夹, 生成的项目就位于该路径文件夹下面;
- ② `project.name` 表示生成的项目名称;
- ③ `project.packageName` 表示生成项目的包名前缀部分(按照Java的文件包命名规范填写), 例如: com.vip.venus
- ④ `project.moduleName` 表示生成项目的模块名称(按照Java的模块命名规范填写, 仅为字母, 不可含有特殊字符), 生成的包名为: <包名前缀部分>+<模块名称>
- ⑤ `project.type` 表示生成项目的类型, 可以不指定(如果为空则默认生成OSPServices项目), 指定为 simpleService, 生成独立的SimpleService项目; 指定为ospService, 生成独立的OSPServices项目, 不可同时指定为simpleService和ospService; 指定simpleService和webapp组合(用逗号分隔), 生成Simple Service和Webapp项目; 指定ospService和webapp组合(用逗号分隔), 生成OSPServices和Webapp项目, 不可单独指定生成Webapp项目;
- ⑥ `project.buildTool` 表示项目使用的构建工具, 目前只支持gradle;
- ⑦ `db.driver` 数据库驱动类名称, 根据不同的数据库配置对应的驱动类名称;
- ⑧ `db.url` 数据库的JDBC链接路径;
- ⑨ `db.userId` 连接数据库的用户账号, 即用户名;
- ⑩ `db.pwd` 连接数据库的密码;
- ⑪ `db.name` 连接数据库的schema, 即指定连接数据库的名称;
- ⑫ `db.tables` 指定表名称和生成的对象名称, 表名和对象名之间用冒号分隔, 多个表名对象名组之间用逗号分隔(table1:Object1, table2:Object2), 如上所示;
- ⑬ `db.shardingTables` 待升级!

Note

如果生成simple service项目, <5>就配置成' simple service'

如果生成simple service+ webapp项目, <5>就配置成' simpleService, webapp'

如果生成OSP service项目, <5>就配置成' ospService'

如果生成ospService + webapp, <5>就配置成' ospService, webapp'

`db.userId`指定的数据库的用户需要拥有新增database的权限, 单元测试需要动态创建和销毁
database

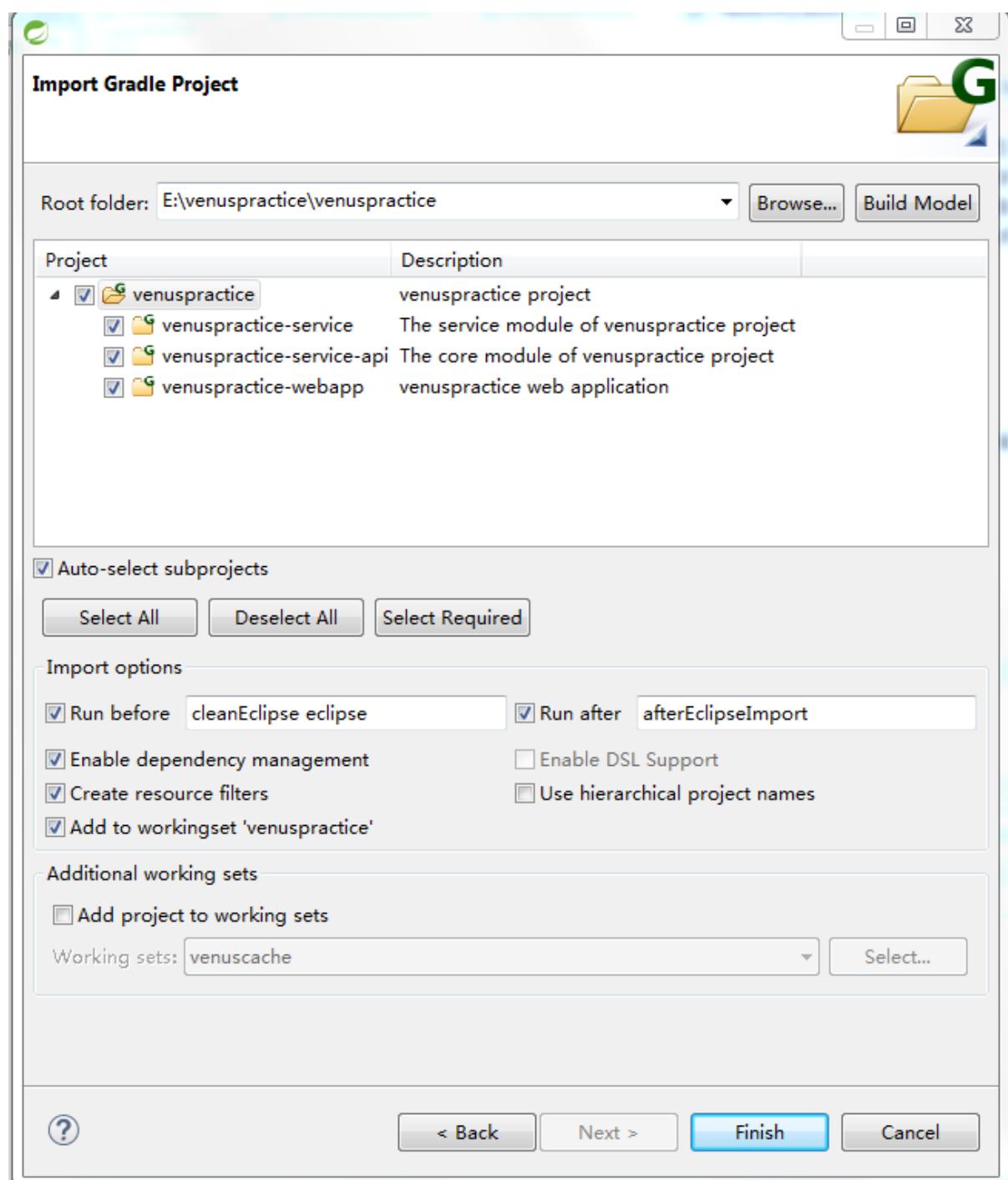
Warning

如果生成的项目是webapp+ospService, 必须先启动osp服务, 才能访问webapp服务.

访问webapp服务的过程: 发请求给webapp→webapp再调用osp服务→操作数据库

step4: cmd进入codegen/bin, 运行codegen.bat即生成项目代码

step5: 代码导入eclipse: 右键import→Gradle Project→选择生成项目的根目录→Build Module→项目全选→finish



只生成项目结构

如果开发者只想生成项目结构不想生成项目代码，这时可以利用参数`-projectstructure`

step1, 2, 3参考the section called “生成venus项目” step1, 2, 3

step4: cmd进入codegen/bin, 运行codegen -projectstructure, 只生成项目结构。

Note

运行codegen -projectstructure时, config.properties中关于db的配置, 只要配置不空着就行, 其实在codegen的过程中并没有连数据库。

只生成代码

如果开发过程中需要使用多个数据库或者生成项目代码后又有新的表增加/表变更, 需要生成新的代码到已有的项目中去,

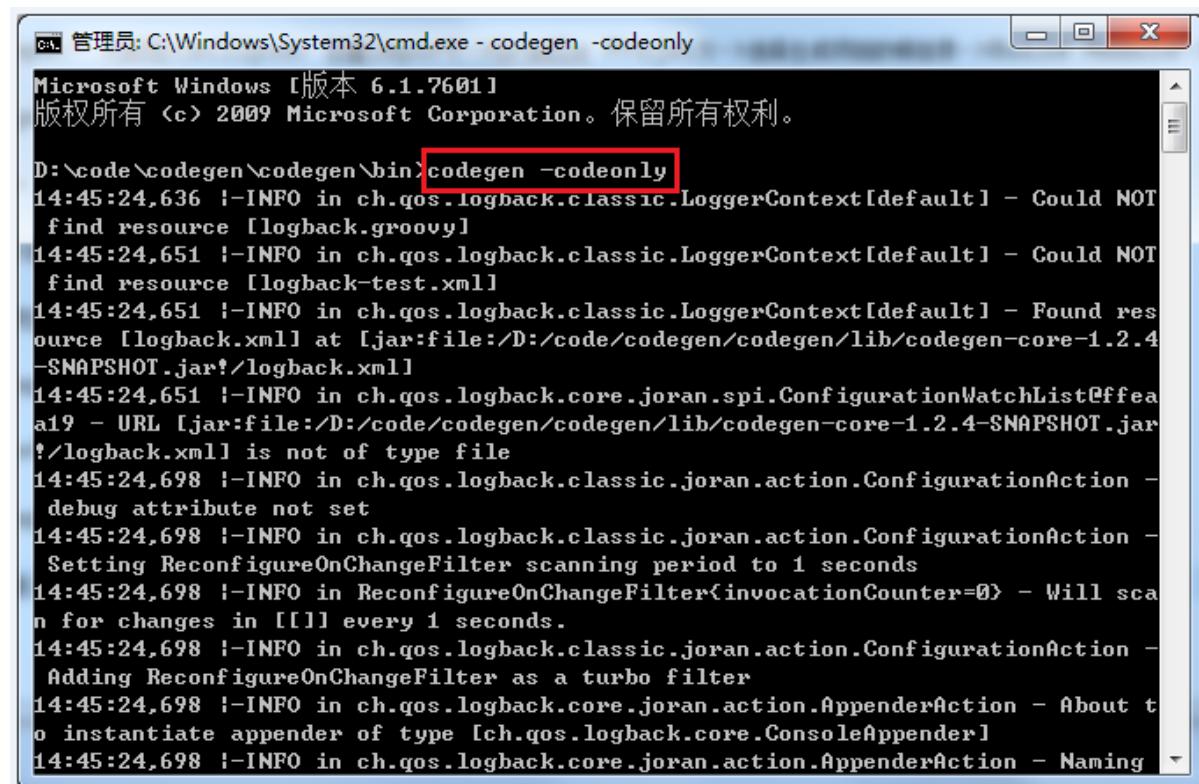
只需要生成这个表对应的代码, 不生成项目关联的gradle配置等文件, 这时就可以使用codegen工具的只生成代码功能。

step1: 修改配置文件

在config.properties配置文件更改为相关的数据库/表

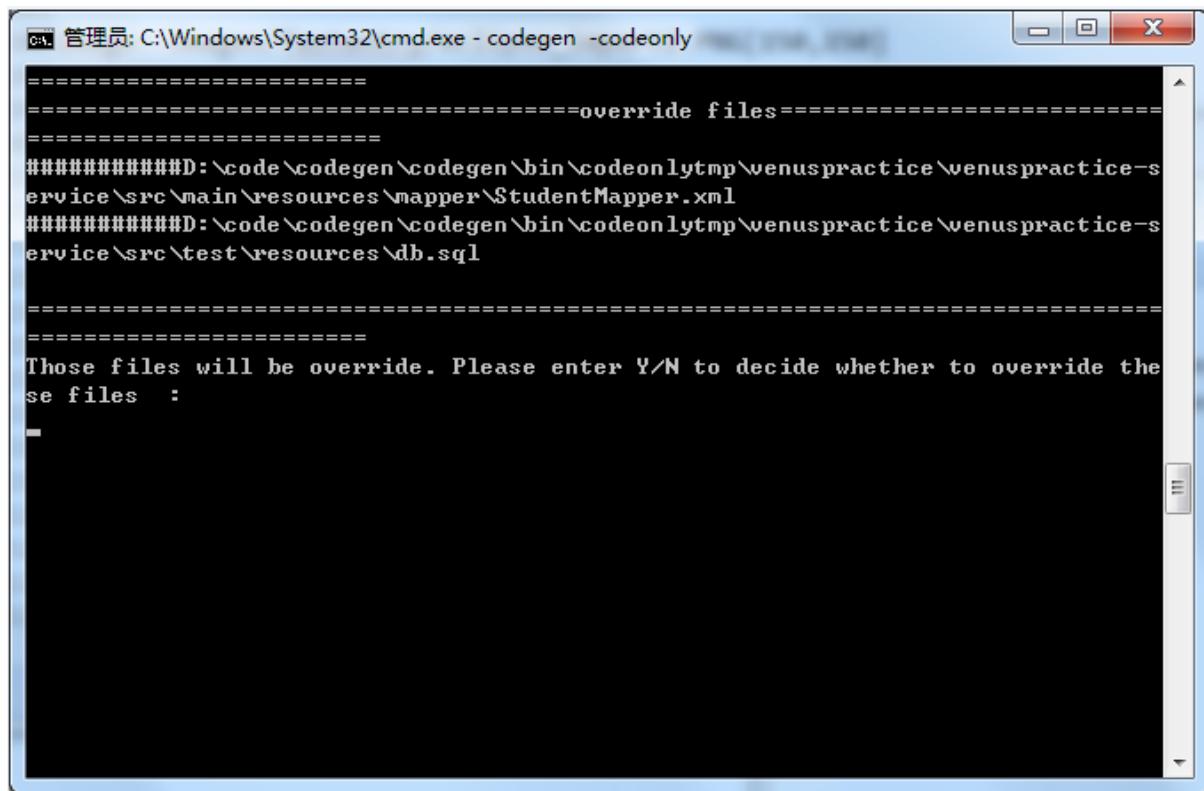
step2: 运行

cmd进入codegen/bin, 运行codegen.bat时加上`-codeonly`参数: 'codegen -codeonly'



```
D:\code\codegen\codegen\bin>codegen -codeonly
14:45:24.636 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT
  find resource [logback.groovy]
14:45:24.651 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT
  find resource [logback-test.xml]
14:45:24.651 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Found res
  ource [logback.xml] at [jar:file:/D:/code/codegen/codegen/lib/codegen-core-1.2.4
-SNAPSHOT.jar!/logback.xml]
14:45:24.651 |-INFO in ch.qos.logback.core.joran.spi.ConfigurationWatchList@ffea
a19 - URL [jar:file:/D:/code/codegen/codegen/lib/codegen-core-1.2.4-SNAPSHOT.jar
!/logback.xml] is not of type file
14:45:24.698 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction -
  debug attribute not set
14:45:24.698 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction -
  Setting ReconfigureOnChangeFilter scanning period to 1 seconds
14:45:24.698 |-INFO in ReconfigureOnChangeFilter{invocationCounter=0} - Will sca
n for changes in [] every 1 seconds.
14:45:24.698 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction -
  Adding ReconfigureOnChangeFilter as a turbo filter
14:45:24.698 |-INFO in ch.qos.logback.core.joran.action.AppenderAction - About t
o instantiate appender of type [ch.qos.logback.core.ConsoleAppender]
14:45:24.698 |-INFO in ch.qos.logback.core.joran.action.AppenderAction - Naming
```

如果是对原有表进行重新生成，这时候会提醒有文件会被覆盖，如果确认要覆盖则输入Y，覆盖文件，输入N就取消生成代码。



The screenshot shows a Windows command prompt window titled "管理员: C:\Windows\System32\cmd.exe - codegen -codeonly". The window displays the following output:

```
=====
=====override files=====
=====
#####D:\code\codegen\codegen\bin\codeonlytmp\venuspractice\venuspractice-s
ervice\src\main\resources\mapper\StudentMapper.xml
#####
#####D:\code\codegen\codegen\bin\codeonlytmp\venuspractice\venuspractice-s
ervice\src\test\resources\db.sql
=====
=====
Those files will be override. Please enter Y/N to decide whether to override the
se files :
```

44. 3#UI版本版本使用说明

生成venus项目功能

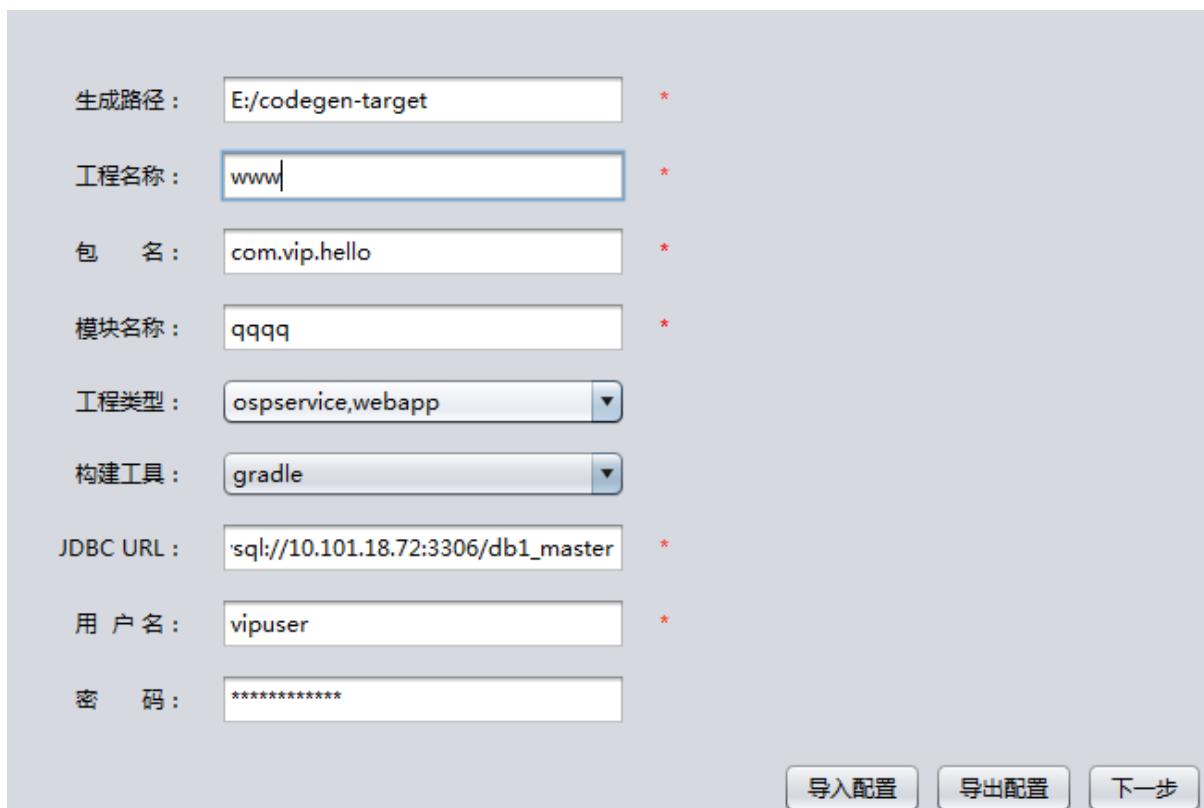
step1: 在数据库中建立数据库database和数据表table

step2: 下载codegen-gui.jar

下载<http://venus.vip.vip.com/tools/codegen/1.2.3/codegen-gui.jar> [codegen-gui.jar]

step3: 运行使用

首先双击codegen-gui.jar开始运行，设置项目配置，可以手工设置，也可以导入配置文件 config.properties



然后进入下一步，选择要使用的表，如图：

列名	类型	长度	是否为空	主键	默认值
id	BIGINT	20	NULL	PRIMARY	NULL
package_name	VARCHAR	50	NULL		NULL
service_name	VARCHAR	50	NULL		NULL
version	VARCHAR	50	NULL		NULL
add_time	DATETIME	19	NULL		NULL
update_time	DATETIME	19	NOT NULL		NULL
remark	VARCHAR	200	NOT NULL		NULL
artifact_id	BIGINT	20	NULL		NULL

表名 对象名
tb_service_in_artifact

表名 对象名
user

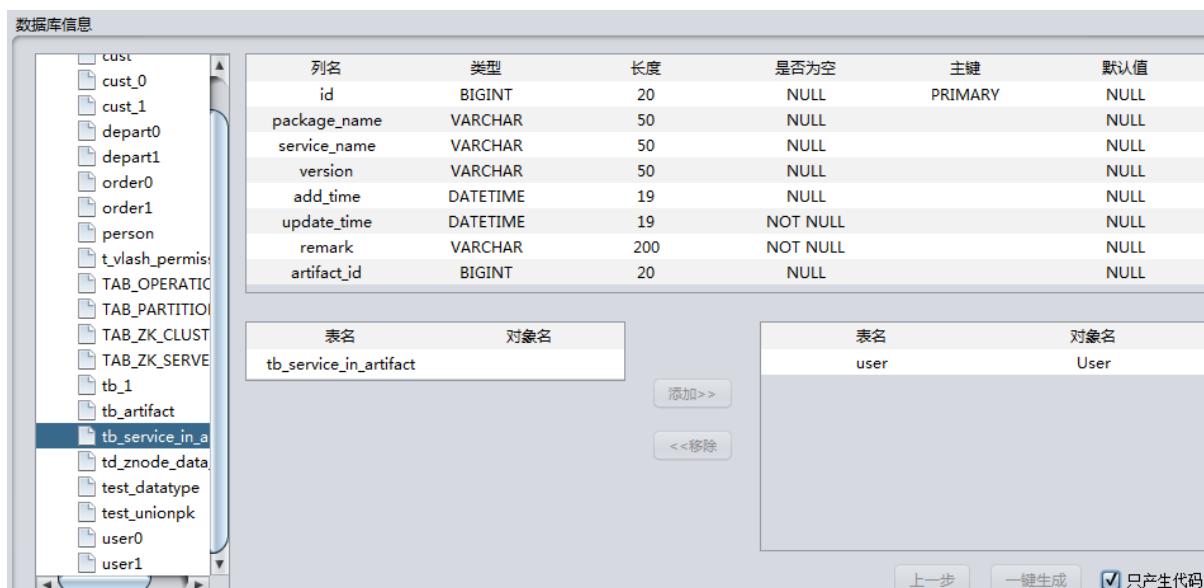
上一步 一键生成 只产生代码

然后点击一键生成就可以生成项目了。

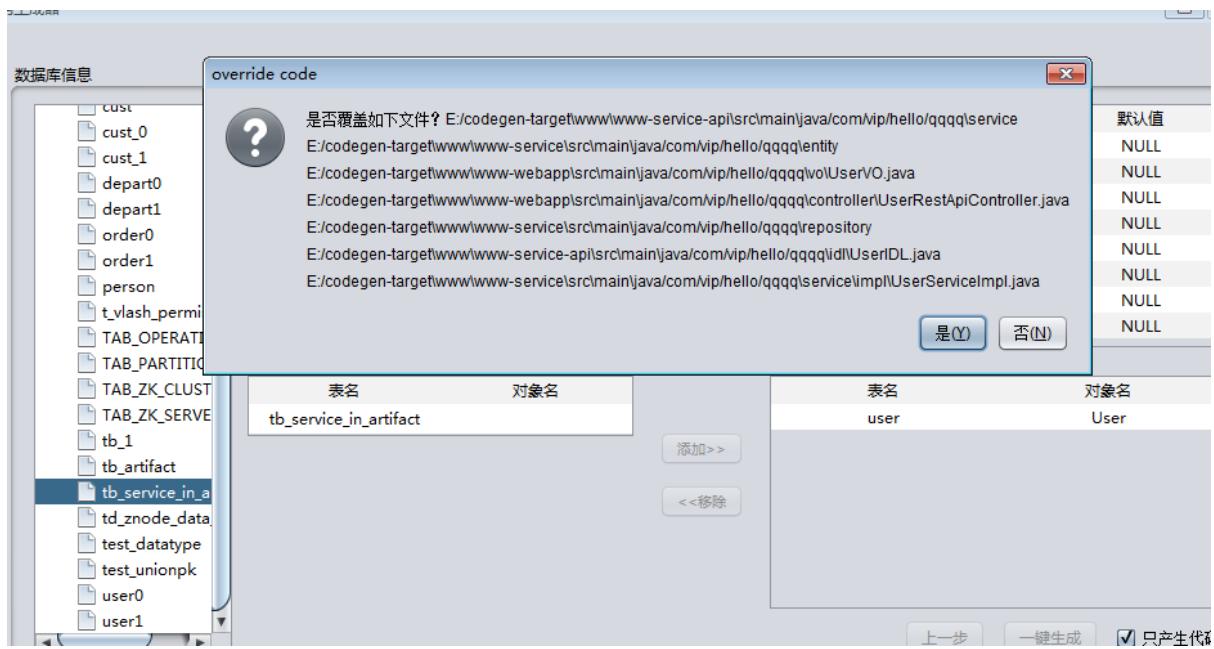


只生成代码功能

如果以后要生成代码到已有项目中，其他步骤不变，需要选上“只生成代码”勾选框，再点击一键生成。



生成的代码会提示是否需要被覆盖，点是后就会覆盖原先生成的代码，点否不会做任何改变。如图：



44. 4#codegen使用注意事项

主键暂时支持MEDIUMINT, INT (或INTEGER), BIGINT四种类型,

普通变量暂不支持YEAR(4), LOB类型, TEXT类型,

详细请看下面表格:

MySQL数据类型	JAVA数据类型	JDBC TYPE	普通变量类型	主键类型
BIGINT	Long	BIGINT	支持	支持
TINYINT	Byte	TINYINT	支持	不支持
SMALLINT	Short	SMALLINT	支持	不支持
MEDIUMINT	Integer	INTEGER	支持	支持
INTEGER	Integer	INTEGER	支持	支持
INT	Integer	INTEGER	支持	支持
FLOAT	Float	REAL	支持	不支持
DOUBLE	Double	DOUBLE	支持	不支持
DECIMAL	BigDecimal	DECIMAL	支持	不支持
NUMERIC	BigDecimal	DECIMAL	支持	不支持
CHAR	String	CHAR	支持	不支持
VARCHAR	String	VARCHAR	支持	不支持
TINYBLOB	DataTypeWithBLOBs.byte[]	BINARY	不支持	不支持
TINYTEXT	String	VARCHAR	支持	不支持

MySQL数据类型	JAVA数据类型	JDBC TYPE	普通变量类型	主键类型
BLOB	DataTypeWithBLOBs. byte[]	BINARY	不支持	不支持
TEXT	DataTypeWithBLOBs. String	LONGVARCHAR	不支持	不支持
MEDIUMBLOB	DataTypeWithBLOBs. byte[]	LONGVARBINARY	支持	不支持
MEDIUMTEXT	DataTypeWithBLOBs. String	LONGVARCHAR	不支持	不支持
LONGBLOB	DataTypeWithBLOBs. byte[]	LONGVARBINARY	支持	不支持
LONGTEXT	DataTypeWithBLOBs. String	LONGVARCHAR	不支持	不支持
DATE	Date	DATE	支持	不支持
TIME	Date	TIME	支持	不支持
YEAR	Date	DATE	不支持	不支持
DATETIME	Date	TIMESTAMP	支持	不支持
TIMESTAMP	Date	TIMESTAMP	支持	不支持

Part#IX. #注意事项

45. properties中编辑json格式	227
46. 项目依赖	228
47. 配置编译后的代码兼容JDK1.6	229

45. #properties中编辑json格式

properties中如果json写成一整行，不容易阅读，我们可以如下编辑：

```
/resource/RDBMS/matrix/dataSource={"matrixName": "dataSource","state": "online","type": "MySQL",\
"groups": [\n    {"groupName": "rwds1","state": "online","loadBalance": "roundRobin",\n        "atoms": [{\"atomName": "write01","host": "10.101.18.191","port": "3306","username":\n            "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": true,"state":\n            "online"},\n            {"atomName": "read101","host": "10.101.18.209","port": "3306","username":\n            "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": false,"weight":\n            1,"state": "online"},\n            {"atomName": "read102","host": "10.101.18.163","port": "3306","username":\n            "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": false,"weight":\n            1,"state": "online"}],\n        {"groupName": "rwds2","state": "online","loadBalance": "roundRobin",\n            "atoms": [{\"atomName": "write02","host": "10.101.18.211","port": "3306","username":\n                "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": true,"state":\n                "online"},\n                {"atomName": "read201","host": "10.101.18.208","port": "3306","username":\n                "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": false,"weight":\n                1,"state": "online"},\n                {"atomName": "read202","host": "10.101.18.177","port": "3306","username":\n                "vipuser","password": "xR54PUU8GWia","dbName": "db_sharding","param": "",\"isMaster": false,"weight":\n                1,"state": "online"}]\n    ]}\n}
```

换行方便我们阅读，但需要在每行的末尾加上\，这样在解析properties文件时才能正确解析json

46. #项目依赖

很多项目团队开发项目的时候可能会分隔多个有依赖的项目 比如 项目A依赖项目B, 然后B可能很频繁的出SNAPSHOT版本的jar给A依赖使用, 虽然我们不赞成这样对SNAPSHOT版本的依赖, 但是基于现实情况, 给出在gradle下对于这样的情况下的依赖更新办法

step1. 找到项目根目录下/gradle/javaProjects.gradle文件

step2. 找到configurations.all{ }代码块

step3. 在代码块中加上: resolutionStrategy.cacheChangingModulesFor 0, 'seconds'

之后对于依赖的更新, 只需要在eclipse里 右键选择项目→Gradle→Refresh Dependencies 就会更新最新依赖

47. #配置编译后的代码兼容JDK1.6

修改`/gradle/javaProjects.gradle`文件如下：

```
compileJava { sourceCompatibility=1.6 targetCompatibility=1.6 }

compileTestJava { sourceCompatibility=1.6 targetCompatibility=1.6 }
```

修改文件后，执行`gradlew build`，编译后的代码就能兼容jdk1.6了。