

FW 用户手册

<<1.0.0>>

恒昌德盛

2015 年 12 月 07 日

[illegible]

1 FW 介绍

fw 即 framework 的英文缩写，旨在为业务系统提供一套统一规范的基本开发框架及工具集，缩短系统搭建时间，使业务软件工程师更多的关注于业务本身，而避免陷入技术泥潭，提高开发速度、降低开发难度以运维难度、提高运营响应速度以及效率。

1.1 fw-parent

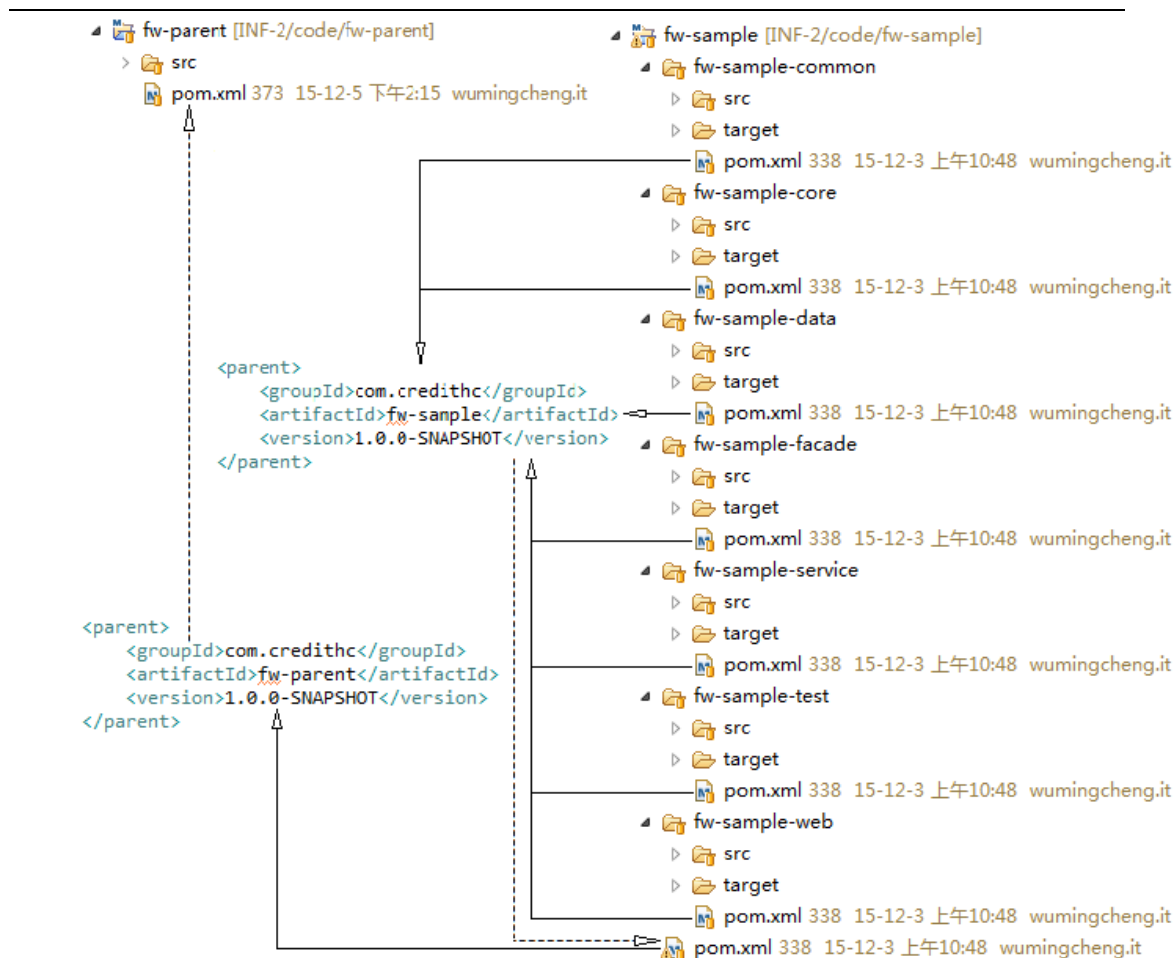
fw-parent 为所有业务系统的顶层 pom（不包括项目自己的子项目），负责统一管理 dependency 的版本号、plugins、jdk 版本等。该项目不包括任何 java 代码。

所有项目的 pom 声明中，应该按如下设置（不包括项目自己的子项目）。

```
<project xmlns=http://maven.apache.org/POM/4.0.0
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                              http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>com.credithc</groupId>
    <artifactId>fw-parent</artifactId>
    <version>1.0.0-SNAPSHOT</version>
  </parent>

  <!-- ..... -->
</project>
```



1.2 fw-common

fw-common 为所有业务系统的通用 common 组件或工具包，它本身的 parent 也是 fw-parent，它简单的包含了所有 fw-parent 中管理的 jar 包，以及框架级别抽取的公共代码及配置。

1.3 fw-sample

fw-sample 是一个项目示例，也是代码自动生成 (fw-generator) 的模板代码，后续所有业务系统都会以 fw-sample 为项目模板用代码自动生成工具 fw-generator 生成系统基础框架。

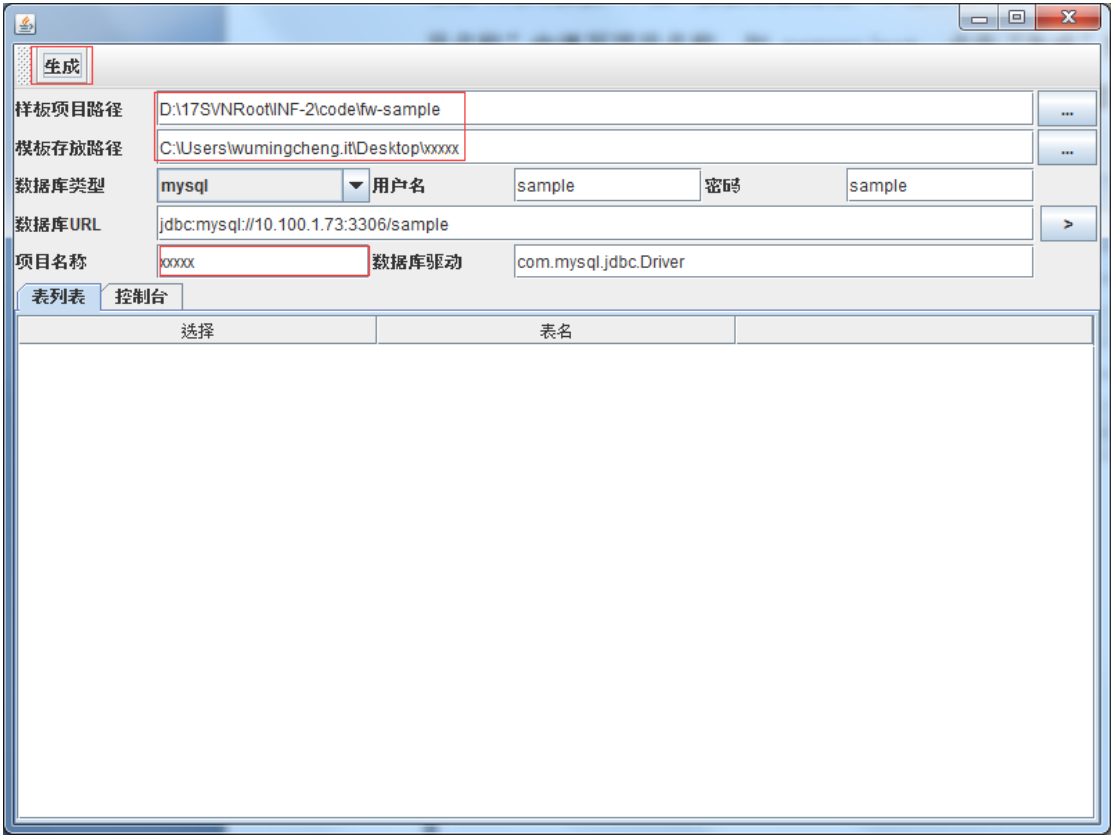
1.4 fw-generator

fw-generator 为代码自动生成工具，根据示例项目（项目模板）生成业务系统基础框架代码。

2 创建项目

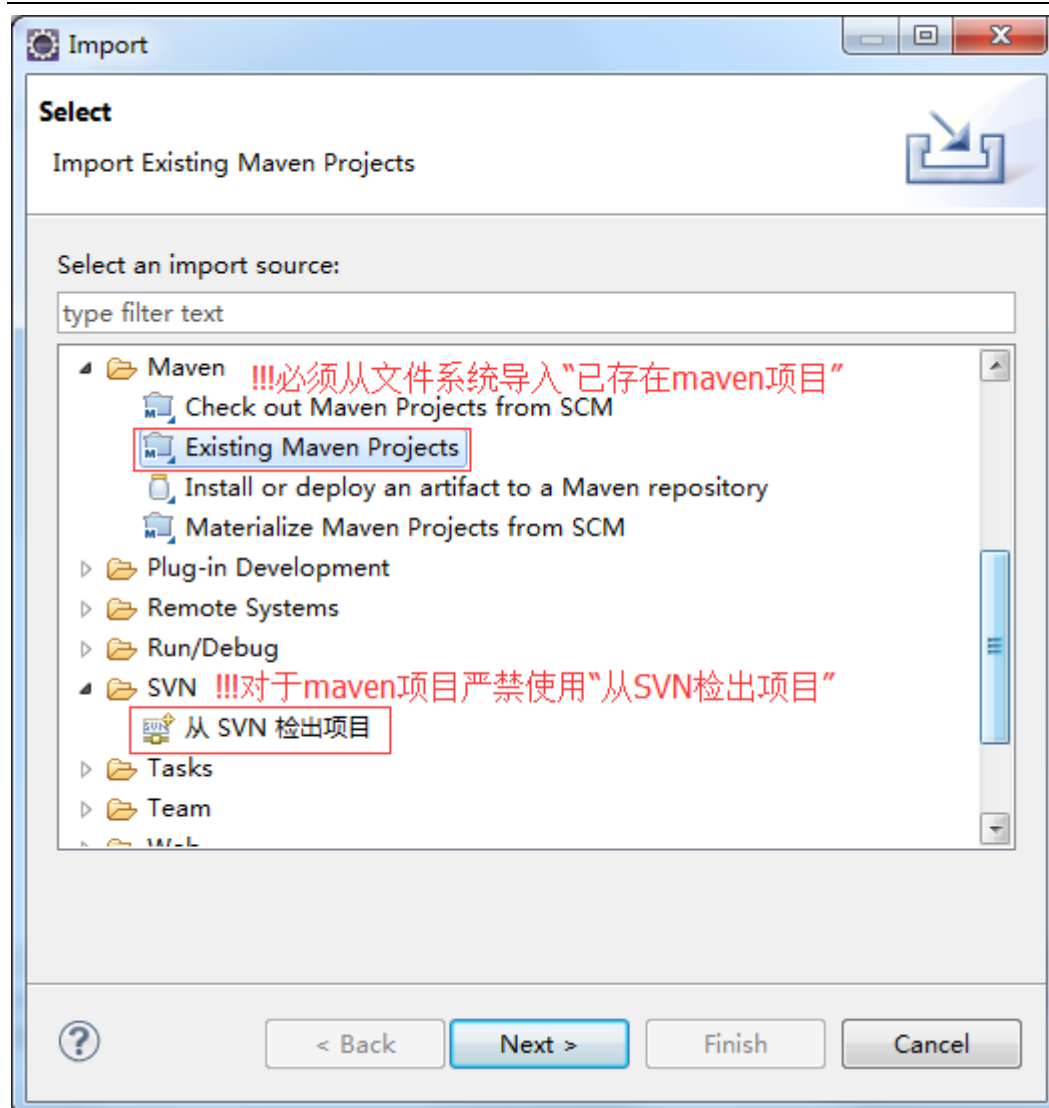
2.1 代码生成

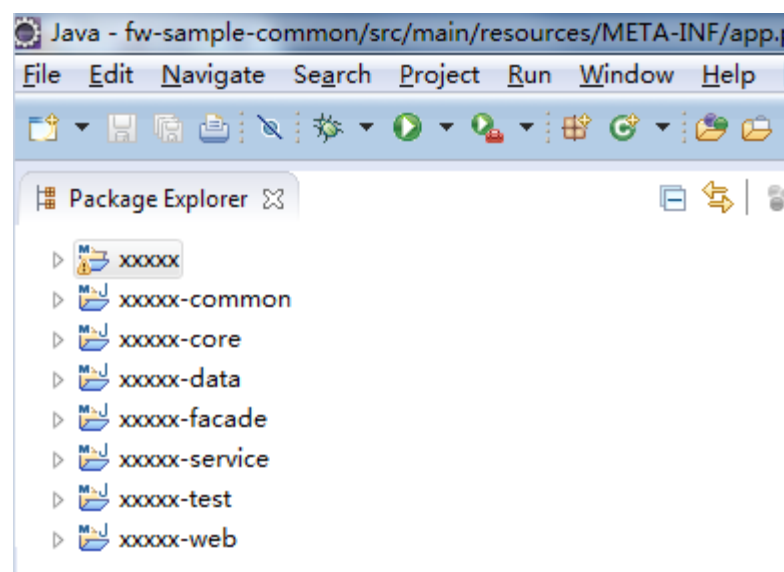
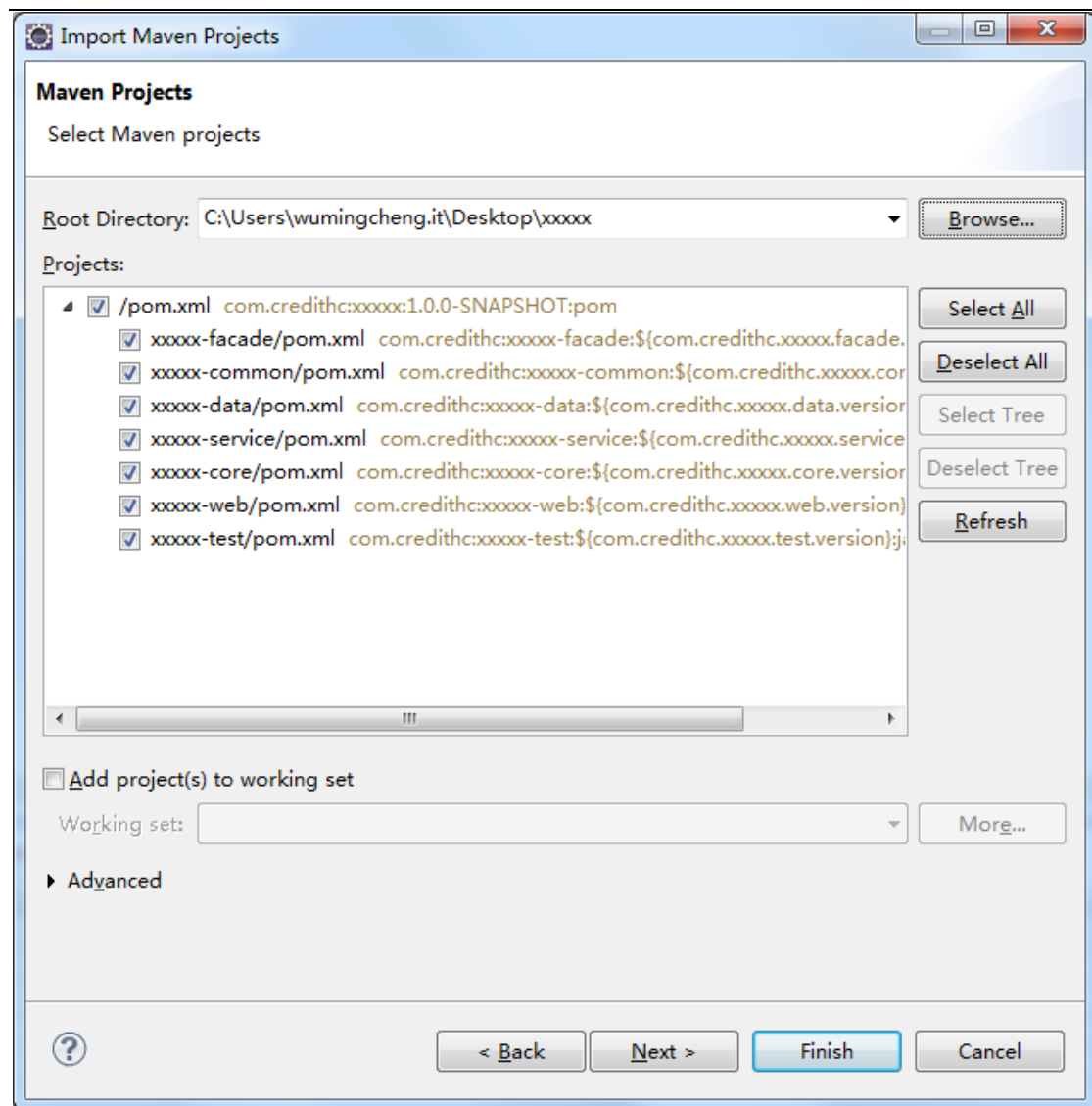
双击运行 fw-generator.jar，在“样板项目路径”中选择 fw-sample 示例项目（代码模板），在“模板存放路径”中选择自动生成的代码存放路径，在“项目名称”中填写项目名称，如 newproject，点击“生成”按钮即可，生成的模板项目放在“模板存放路径”指定的路径下（数据库部份暂时未实现，不支持，后继会补充）。



2.2 导入项目

导入项目必须从文件系统用“Import Existing Maven Projects”导入文件系统中的 maven 项目，严禁使用“从 SVN 检出项目”导入 maven 项目。如果代码存在于 SVN 上，要先用“乌龟”将代码从 SVN 检出到文件系统磁盘上，再用“Import Existing Maven Projects”导入到 IDE 中。



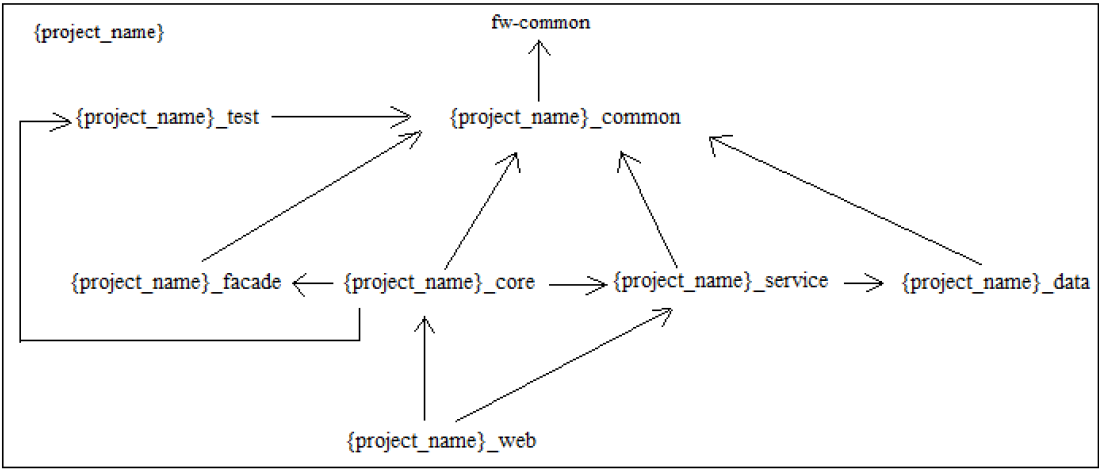


2.3 项目结构

- ▷ newproject
- ▷ newproject-common
- ▷ newproject-core
- ▷ newproject-data
- ▷ newproject-facade
- ▷ newproject-service
- ▷ newproject-test
- ▷ newproject-web

文件夹名称	功能/责任描述
{project_name}	业务项目的父项目，从文件夹层级上看，包含其它 {project_name}_*的子项目。
{project_name}_common	业务项目的通用代码、工具、配置、异常码定义等。
{project_name}_core	业务项目的核心业务逻辑、流程、触发等，同时也可以调用第三暴露的服务。
{project_name}_data	业务项目的数据持久层，负责与数据库交互。
{project_name}_facade	业务项目的对外接口层，用于对外暴露接口给第三方。
{project_name}_service	业务项目的原子服务层，调用 data 层或第三方暴露的服务。
{project_name}_test	业务项目为单元测试或自测，额外需要的工具、MOCK、配置等，一般用不到。
{project_name}_web	业务项目的 web 层，用于存放 web 或视图层相关的文件、配置、代码等，可以调用 core 层和 service 层和第三方暴露的服务。

2.4 依赖关系



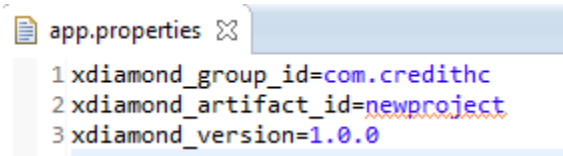
3 配置中心

3.1 本地配置

{project_name}\newproject-common\src\main\resources\META-INF\app.properties

环境变量	说明
xdiamond_group_id	项目 group id
xdiamond_artifact_id	项目 artifact id
xdiamond_version	项目 version

如下图所示：



```
1xdiamond_group_id=com.credithc
2xdiamond_artifact_id=newproject
3xdiamond_version=1.0.0
```

3.2 环境变量

环境变量	说明
xdiamond_server_host	配置中心 IP 地址
xdiamond_server_port	配置中心端口号
xdiamond_profile	标识是什么环境，如： product/dev/test
xdiamond_secret_key	配制中心密码,生产环境要求配置中心自动生成的密码

环境变量可以设置在全局级别，也可以设置在容器里（如 tomcat），设置在容器级别（JAVA_OPTS）适合一台机器运行多个窗口实际时的场景，设置方式如下：

```
Windows:

set "JAVA_OPTS= -server -Xms1024m -Xmx1024m
-Dxdiamond_server_host=10.100.1.73 -Dxdiamond_server_port=5678
-Dxdiamond_profile=product -Dxdiamond_secret_key=uWlxq2p5tWEtx9du"
```

Linux:

```
JAVA_OPTS="$JAVA_OPTS -Dxdiamond_server_host=10.100.1.73
-Dxdiamond_server_port=5678 -Dxdiamond_profile=product
-Dxdiamond_secret_key=uWlxq2p5tWEtx9du"
```

3.3 配置中心

配置名称	说明
app_name	应用名称
dubbo_registry_url	Dubbo 注册中心地址
elastic_server_list	定时任务注册中心
jdbc_password	数据库密码（加密）
jdbc_url	数据库 URL
jdbc_user	数据库用户名
rabbitmq_host	消息队列 IP
rabbitmq_port	消息队列端口
rabbitmq_pwd	消息队列密码
rabbitmq_user	消息队列用户名

如下图所示：



项目管理

UserProfile

用户管理

组管理

角色管理

权限管理

全部Config

依赖图

Metrics

Add Profile

Profile

base profile里放公共的配置，比如某个服务的端口号。所有的非base profile都会继承base profile里的Config

对于dev, test这些profile，建议不设置secretKey，便于开发

profile	access	secretKey	description
base ?	Master		查看Config
dev	Developer		查看Config
product	Master	6upamM0A4wHAVJdP	查看Config

项目管理

UserProfile

用户管理

组管理

角色管理

权限管理

全部Config

依赖图

Metrics

ThreadDump

SystemProperties

WebConsole

DruidStat

Health

Logger

Apidoc

Connections

Add Config

批量增加Config ?

Resolved Configs

以Properties格式查看

以JSON格式查看

以复制到另一个profile的格式查看

Q

Search

+ 修改/详细显示Key/Value

☐ 只显示当前项目的配置

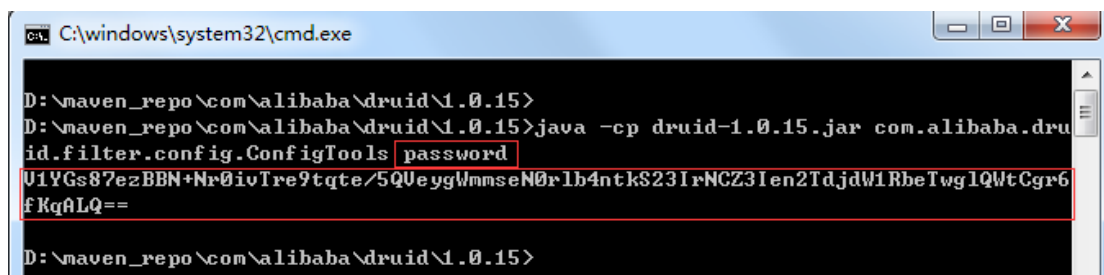
☐ 只显示从其它项目继承过来的配置

☐ 只显示当前Profile的配置

☐ 只显示非当前Profile的配置

key	value
app_name	sample
dubbo_registry_url	zookeeper://10.100.1.74:2181
elastic_server_list	10.100.1.74:2181
jdbc_password	X8nfi7tcNR0DRYX46FB7yEFAnQNIDrv3lQ/spTwaHDvEyD1C0++niPyrzkKkurCY4WRVikRwvEU+hnuIwIGEMw==
jdbc_url	jdbc:mysql://10.100.1.73:3306/sample?useUnicode=true&characterEncoding=utf-8&zeroDateTimeBehavior=convertToNull
jdbc_user	sample
rabbitmq_host	10.100.1.73
rabbitmq_port	5672
rabbitmq_pwd	123456
rabbitmq_user	admin

3.4 DB 密码加密



将加密的密码复制到配置中心的 jdbc_password 属性中。

3.5 maven settings

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
    http://maven.apache.org/xsd/settings-1.0.0.xsd">
```

```
<localRepository>d:/maven_repo</localRepository>
```

```
<servers>
```

```
<server>
```

```
<id>releases</id>
```

```
<username>admin</username>
```

```
<password>admin123</password>
```

```
</server>
```

```
<server>
```

```
<id>snapshots</id>
```

```
<username>admin</username>
```

```
<password>admin123</password>
```

```
</server>
```

```
</servers>
```

```
<mirrors>
```

```
<mirror>
```

```
<id>nexus</id>
```

```
<mirrorOf>*</mirrorOf>
```

```
<url>http://10.100.1.71:8081/nexus/content/groups/public</url>
```

```
</mirror>
```

```
</mirrors>
```

```
<profiles>
```

```
<profile>
  <id>nexus</id>
  <repositories>
    <repository>
      <id>central</id>
      <url>http://central</url>
      <releases><enabled>true</enabled></releases>
      <snapshots><enabled>true</enabled></snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>central</id>
      <url>http://central</url>
      <releases><enabled>true</enabled></releases>
      <snapshots><enabled>true</enabled></snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
</profiles>

<activeProfiles>
  <activeProfile>nexus</activeProfile>
</activeProfiles>
</settings>
```

4 配置文件

4.1 项目级别通用配置文件

{project_name}\{project_name}-common\src\main\resources\META-INF\spring\applicationContext-{project_name}-common.xml

4.2 暴露服务配置文件

{project_name}\{project_name}-core\src\main\resources\META-INF\spring\applicationContext-{project_name}-expose.xml

4.3 接口实现配置文件

{project_name}\\{project_name}-core\\src\\main\\resources\\META-INF\\spring\\applicationContext-{project_name}-facade-impl.xml

4.4 定时任务配置文件

{project_name}\\{project_name}-core\\src\\main\\resources\\META-INF\\spring\\applicationContext-{project_name}-job.xml

4.5 消息配置文件

{project_name}\\{project_name}-core\\src\\main\\resources\\META-INF\\spring\\applicationContext-{project_name}-message.xml

4.6 第三方引用配置文件

{project_name}\\{project_name}-core\\src\\main\\resources\\META-INF\\spring\\applicationContext-{project_name}-reference.xml

4.7 数据访问对象配置文件

{project_name}\\{project_name}-data\\src\\main\\resources\\META-INF\\spring\\applicationContext-{project_name}-dao.xml

4.8 数据源配置文件

{project_name}\\{project_name}-data\\src\\main\\resources\\META-INF\\spring\\applicationContext-{project_name}-datasource.xml

4.9 事务配置文件

{project_name}\\{project_name}-data\\src\\main\\resources\\META-INF\\spring\\applicationContext-{project_name}-transaction.xml

4.10 mybatis 配置文件

{project_name}\{project_name}-data\src\main\resources\META-INF\mybatis\mybatis-config.xml

4.11 Mybatis 数据实体映射配置文件

{project_name}\{project_name}-data\src\main\resources\META-INF\mybatis\sqlmapper*DOMapper.xml

4.12 原子服务配置文件

{project_name}\{project_name}-service\src\main\resources\META-INF\spring\applicationContext-{project_name}-service.xml

4.13 Spring mvc 配置文件

{project_name}\{project_name}-web\src\main\resources\META-INF\spring\dispatcherServlet.xml

4.14 日志配置文件

{project_name}\{project_name}-web\src\main\resources\META-INF\log\logback.xml

4.15 本地属性文件

{project_name}\{project_name}-common\src\main\resources\META-INF\app.properties

4.16 其它说明

配置文件可以根据项目的具体需要进行增减调整。

5 项目裁剪

自动生成的项目骨架项目集成了很多工具或功能,有些项目是不需要这些功能的,所以可以对骨架项目进行裁剪。

在 {project_name}\{project_name}-common\src\main\resources\META-INF\app.properties 中有很多开关*_enabled,用于打开或关闭某些功能。如果某

个功能关闭了，对应的 applicationContext-*.xml 也要对应的删除或修改，对应关系如下：

app.properties	applicationContext-*.xml	描述
elastic_enabled=false	applicationContext-{project_name}-job.xml	true:启用定时任务;false:禁用定时任务。默认为 false。
redis_enabled=false	无	true:启用缓存;false:禁用缓存。默认为 false。
rabbit_enabled=false	applicationContext-{project_name}-message.xml	true:启用消息队列;false:禁用消息队列。默认为 false。
xdiamond_enabled=false	无	true:启用配置中心;false:禁用配置中心。默认为 false。

如果自动生成的项目骨架结构无法满足实际项目的需要，可以对骨架项目进行修改，一般分为增加 jar 项目和 war 项目，对于此两种类型，可以分别复制一份 {project_name}-core 或者 {project_name}-web 修改相应的项目名称和包名称，并加到 {project_name}-parent 的模块列表中，然后根据建立新子项目的原因，在 pom.xml 中指定依赖关系。

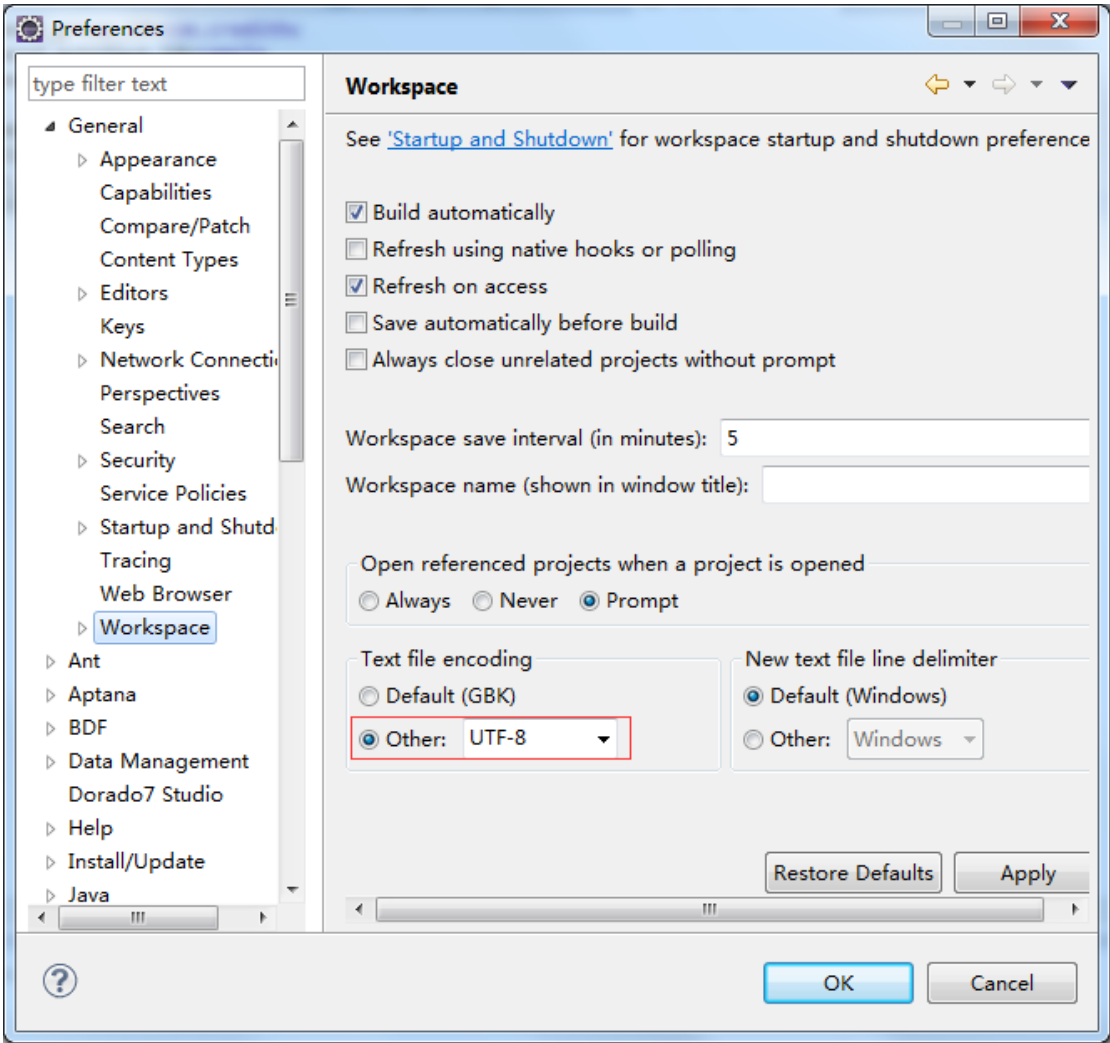
6 开发环境

6.1 开发工具

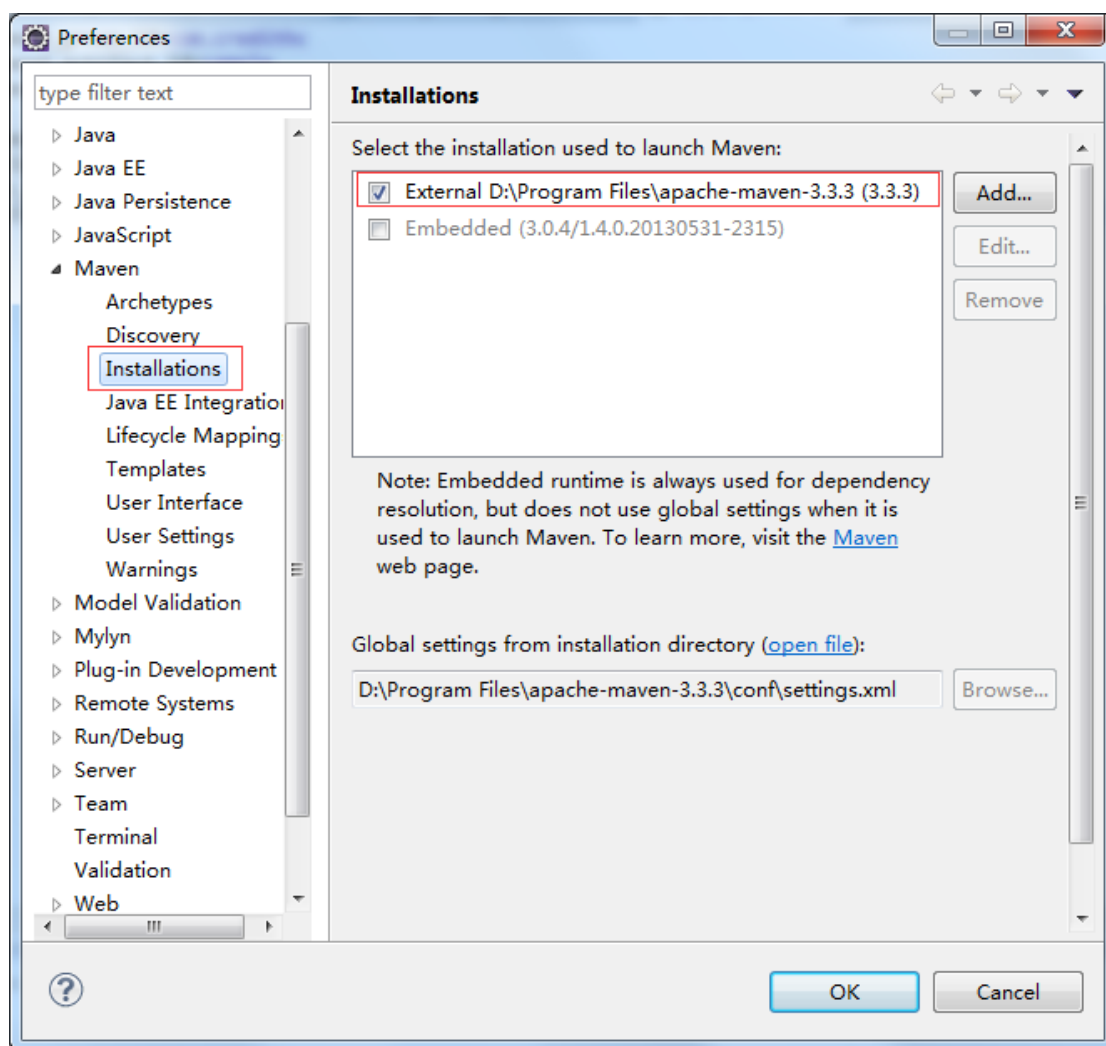
名称	版本	说明
jdk	1.7	
tomcat	7.x	
eclipse	>=eclipse-jee-kepler(4.3.2)	
subclipse	1.6.x	subclipse、tortoiseSVN 和 SVN 版本要匹配，如果不能很好的兼容，版本号自行匹配。
tortoiseSVN	1.6.7	
maven	>3.2.x	
Oracle	11.x	Oracle、oracle 客户端和 PL/SQL 要匹配兼容，如果不能很好兼容，版本号自行匹配。
Oracle 客户端	11.2	
PL/SQL	>9.x.x、10.x.x	
其它	x.x.x	自行决定。

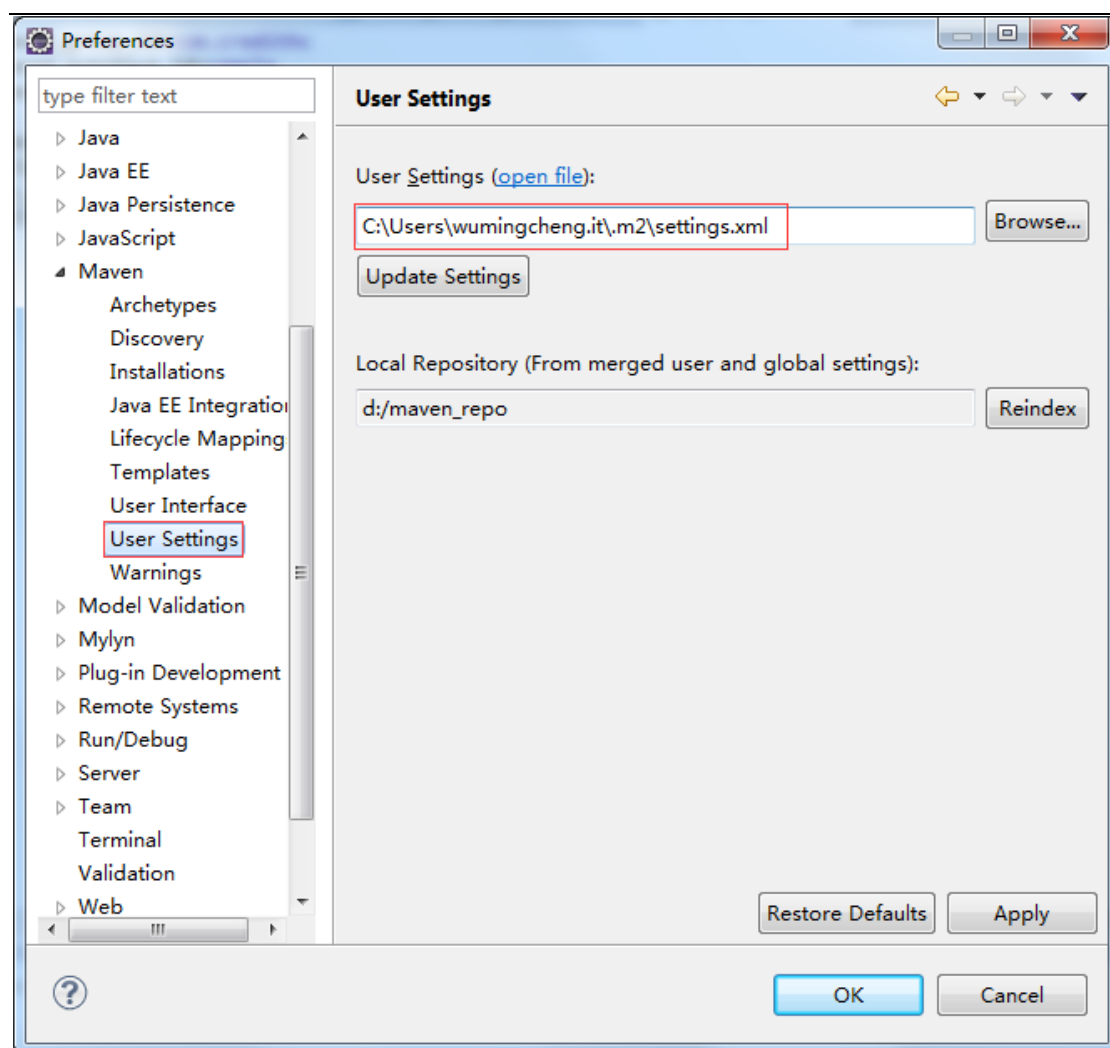
6.2 工具配置

6.2.1 全局编码设置

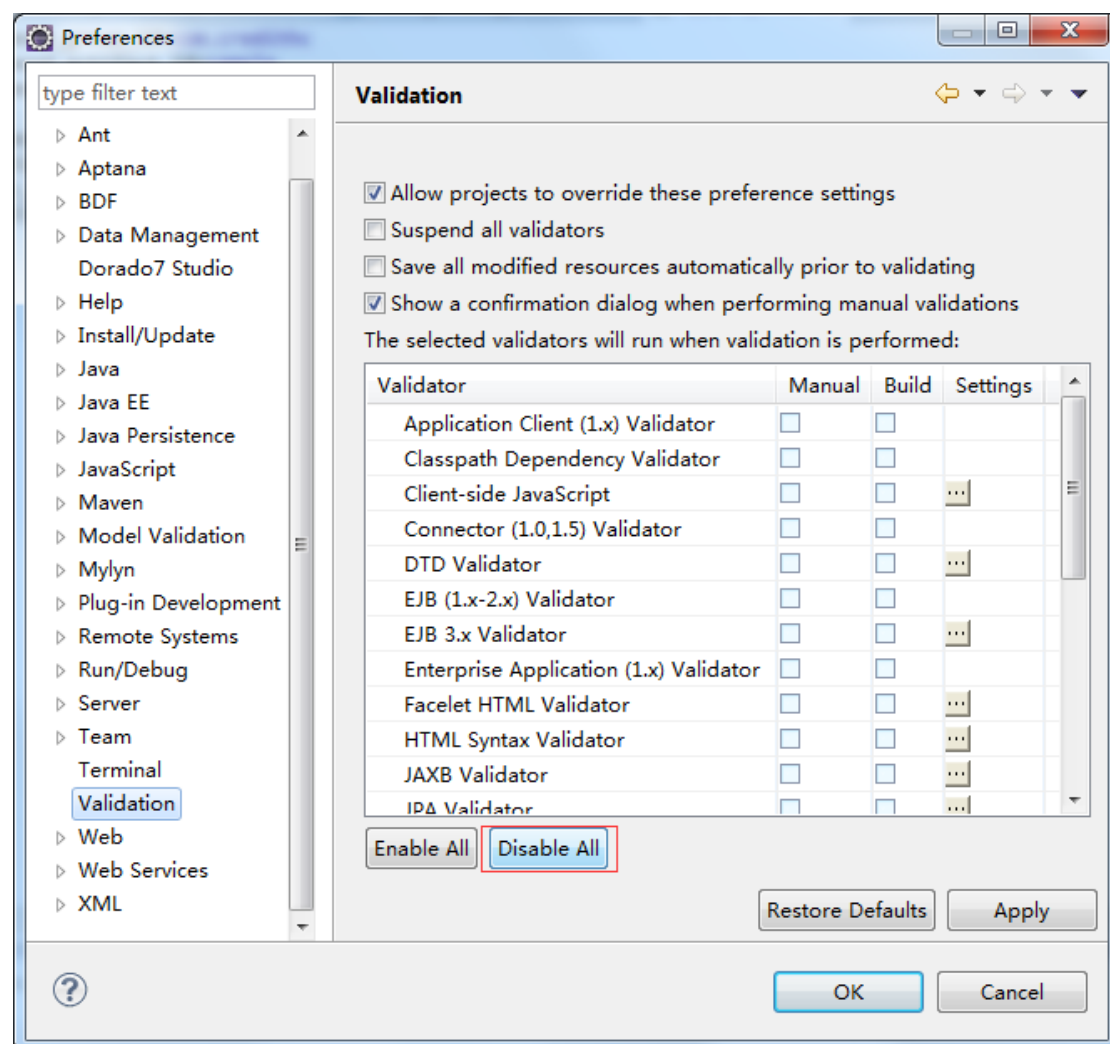


6.2.2 maven 插件设置

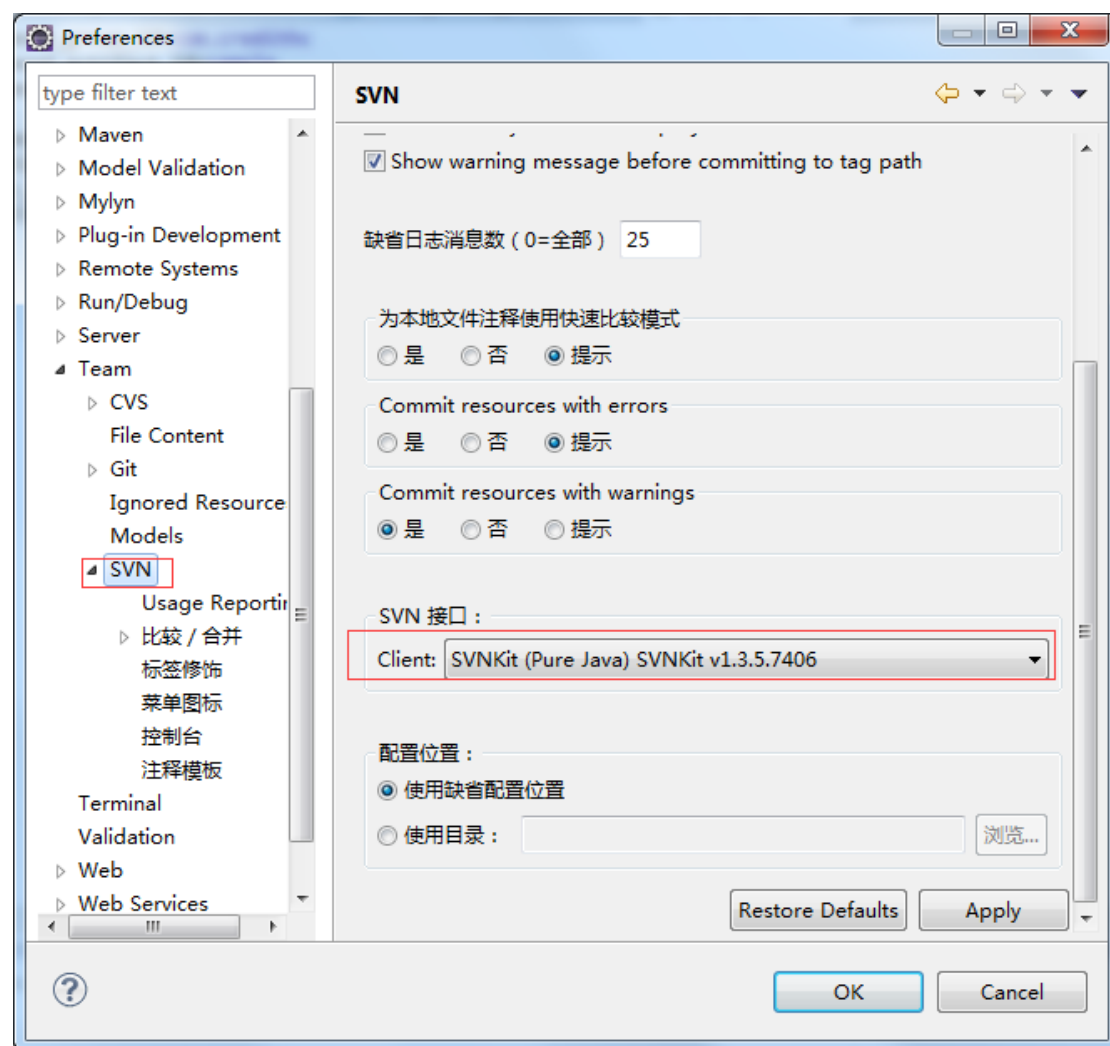


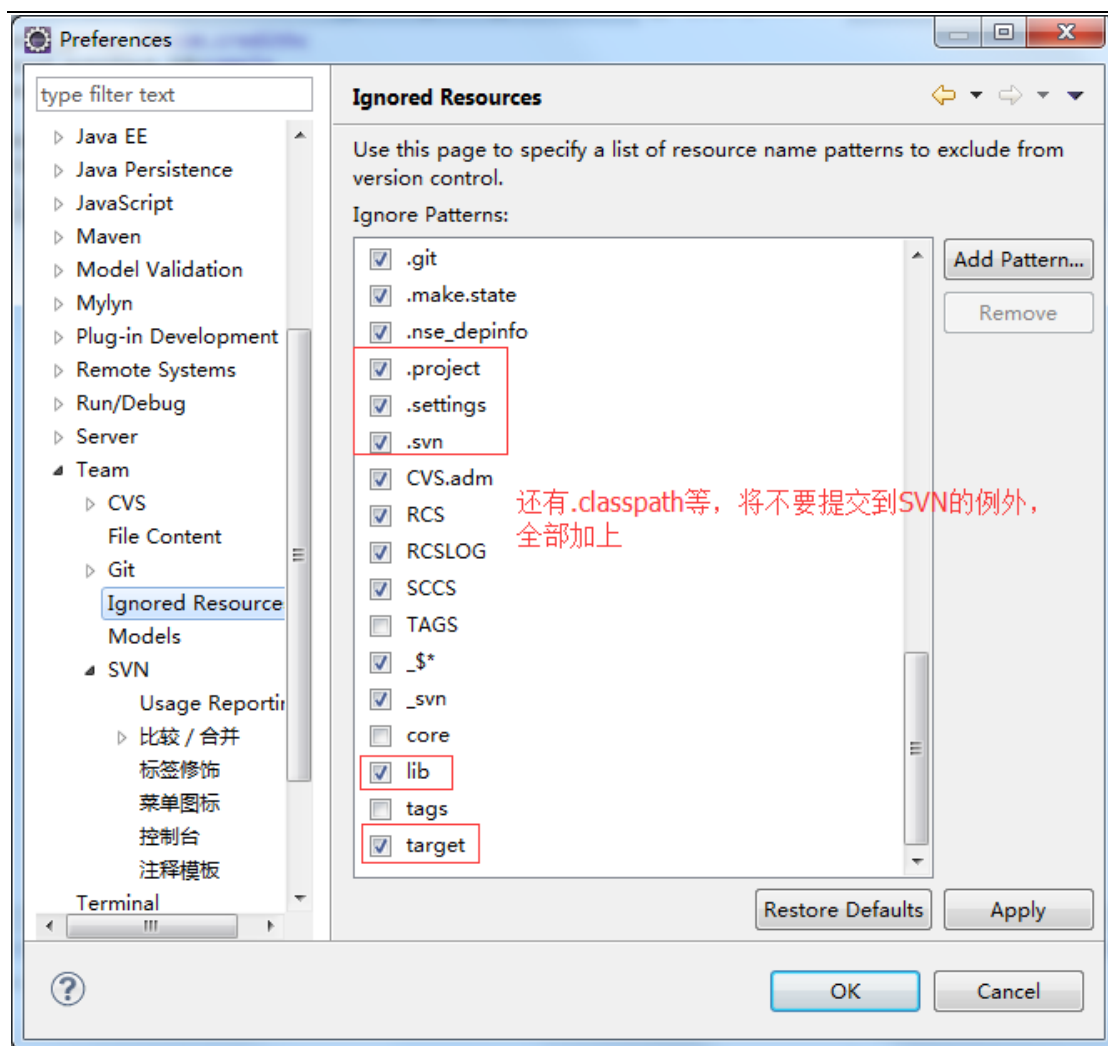


6.2.3 验证设置



6.2.4 Svn 插件设置





7 开发规范

7.1 字符集

统一为 UTF-8。

7.2 命名规范

7.2.1 项目命名规范

父项目名称用单一单词命名，全英文字母，全小写，如 notity。子项目名称用单一单词组合，全小写，中间以“-”分割，如：notify-core、notify-web、notify-service。

7.2.2 包命名规范

以 `com.credithc` 子项目名称打头，把“-”换成“.”，后面根据需要自由创建，如 `com.credithc.notify.core.xxx`、`com.credithc.notify.web.xxx`、`com.credithc.notify.facade.xxx` 等。

7.2.3 接口命名规范

用单一单词组合命名，每个单词首字母大写，如 `SampleQueryService`、`NotifySMSDao` 等。

7.2.4 抽象类命名规范

以 `Abs` 或 `Abstract` 打头，后再跟单一单词组合，每个单词首字母大写，英文缩写全部大写，如 `AbsFacade`、`AbstractService` 等。

7.2.5 类命名规范

用单一单词组合命名，每个单词首字母大写，英文缩写全部大写，如 `StringUtil`、`NotifyHelper`、`RemoteAdaptor`、`SimpleDateFormat` 等。对于实现接口或抽象类的类后加 `Impl`，如 `SampleQueryServiceImpl`、`NotifySMSDaoImpl`。

7.2.6 方法命名规范

用单一单词组合命名，除首单词首字母小写外，其它单词首字母大写，如 `querySmsList`、`updateNotifyResult` 等。

7.2.7 变量命名规范

用单一单词组合命名，除首单词首字母小写外，其它单词首字母大写，如 `resultList`、`nofityService`、`objectTemplate` 等。

7.2.8 常量命名规范

以 `public static final` + 变量名组成，变量名以单一单词以“_”分隔，变量名全部大写，如 `PAGE_SIZE`、`FILE_SUBFIX` 等。

7.2.9 数组命名规范

数据命名规范，同一般普通变量规则，声明风格采用 `java` 风格，而不是 `c` 风格，如下：

```
int [] myarray = ...; //yes
```

```
int myarray[] = ...; //no
```

7.2.10 文件名规范

除了 java 用单一单词组合命名，除首单词首字母小写外，其它单词首字母大写，如 smsList.html、app.properties、somecfg.xml 等。

Java 相关文件用单一单词组合命名，首字母全部大写。

Spring 配置文件建议用 applicationContext-*.xml 方式，如果是特殊需要或原因可以自定义其它命名方式。

7.3 排版规范

7.3.1 花括号排版规范

控制语句、方法、类等后紧跟的左花括号不换行, if 语句后花括号不能省略，如下：

```
if(x > 1){  
    //code here  
}
```

```
void fn(){  
    //code here  
}
```

```
public class A(){  
    //code here  
}
```

7.3.2 空格使用规范

在操作符前后加一个空格，包括=、==、>、<、&&、||等，如下：

```
int x = 1;  
long y = 0;  
if(x > 1 && y < 0){
```



```
//code here  
}
```

在逗号后面留一个空格，如下：

```
public int fun1(String arg0, Sring arg1, Object obj) {  
    //code here  
}
```

For 循环小括号里的分号后面加一个空格，如下：

```
For(int i = 0; i < 0; i ++){  
    //code here  
}
```

7.3.3 空行使用规范

除了抽取子方法，方法内部要有逻辑分段，使用空行分隔，不易理解的逻辑分段加上注释，如：

```
//get order count by customer  
  
int orderCount = queryOrderCount(customerId);  
if(orderCount == 0){  
    return;//if no order, do nothing, return  
}  
  
//这里空行，逻辑分段  
  
//deal with orders  
  
List<Order> orderList = queryOrderList(customerId);  
For(Order order : orderList){  
    dealOrderInnerImpl(order);  
}  
  
//这里空行，逻辑分段  
  
//do something others  
  
//code here
```

7.3.4 业务代码内聚

业务代码尽量内聚集中、不分散，不随便拆文件，每一个业务逻辑都有一个主流程方法，由主流程方法调用子方法。

7.4 注释规范

除非是非常简单、直观、易理解的地方不需要写注释，其它地方应该写明注释，说明用法。加有意义的注释，没有必要的注释或直接通过方法名看出功能的方法或变量不要加没有意义的注释。一般 setter/getter 方法上没有必要写注释。一般在方法前、代码块前、逻辑分段前都要写上注释。

注释类型	示例
文件注释	<pre>/** *
项目名: notify-service *
文件名: SmsService.java *
Copyright 2015 恒昌德盛 */</pre>
类注释	<pre>/** *
类 名: SmsService *
描 述: 短信改送服务 *
作 者: xxx *
创 建: 2015年5月5日 *
版 本: v1.0.0 *
 *
历 史: (版本) 作者 时间 注释 */</pre>
方法注释	<pre>/** * 描 述: * 作 者: <u>xxx</u></pre>

	<pre>* 历史: (版本) 作者 时间 注释 * @param date 短信日期 * @param isBatch 是否批量 * @return true:改送成功;false:改送失败 */</pre>
单行注释	<pre>//这里是单行注释 /*这里也是单行注释*/</pre>
多行注释	<pre>/* *这是一行注释 *这里又是一行注释 */</pre>

有一些用法建议如下:

```
//这里一些单行注释
```

```
Int I = 0;
```

```
Order order = ...; //这里也是单行注释
```

```
Payment payment = ...; /*这里还是单行注释*/
```

```
/*
```

```
*这里是一些对于不太好理解的方法
```

```
*的注释说明
```

```
*/
```

```
doOrderAndPayment(order, payment);
```

```
/*
```

```
*这里是一些对于不太好理解的逻辑分段
```

```
*或代码块的注释说明
```

```
*/
```

```
for(...) {
```

```
...  
}  
if() {  
    ...  
}else{  
    ...  
}
```

```
Map<String /*user code*/, String/*user name*/> userMapping = ...;
```

单行注释采用可采用//.....也可采用/*...*/，建议前都，因为使用起来比较简单，多行注释采用/*.....*/。

7.5 接口和抽象类

接口/类名	用途
com.credithc.common.web.AbsController	所有 web 层的所有 controller 都继承此类
com.credithc.common.facade.AbsReq	所有 facade 层的请求 DTO 都继承此类
com.credithc.common.facade.AbsRes	所有 facade 层的响应 DTO 都继承此类
com.credithc.common.core.AbsFacade	所有 core 层的 facade 实现都继承此类
com.credithc.common.service.AbsReqTO	所有 service 层的请求 DTO 都继承此类
com.credithc.common.service.AbsResTO	所有 service 层的响应 DTO 都继承此类
com.credithc.common.service.AbsService	所有 service 都继承此类
com.credithc.common.dao.AbsEntity	所有 data 层的实体类都继承此类
com.credithc.common.dao.BaseDao	所有 data 层的 dao 接口都继承此接口
com.credithc.common.dao.AbsBaseDao	所有 data 层的 dao 实现都继承此接口

7.6 日志规范

AbsController、AbsFacade、AbsService、AbsDao 的子类中（即在业务线中）用优先父类中提供的日志方法打印日志，如下：

```
● debug(LogableDTO, String) : void
● debug(LogableDTO, String, Object...) : void
● debug(LogableDTO, String, Throwable) : void
● info(LogableDTO, String) : void
● info(LogableDTO, String, Object...) : void
● info(LogableDTO, String, Throwable) : void
● warn(LogableDTO, String) : void
● warn(LogableDTO, String, Object...) : void
● warn(LogableDTO, String, Throwable) : void
● error(LogableDTO, String) : void
● error(LogableDTO, String, Object...) : void
● error(LogableDTO, String, Throwable) : void
```

其它不在业务线中的日志,即没有对应 LogableDTO 的地上使用原生的 SLF4J 日志。

另: 不要在代码中出现 `System.out` 及 `System.err` 这样的代码,也不要出身 `exception.printStackTrace()` 这样的代码。

7.7 消息规范

`AbsController`、`AbsFacade`、`AbsService`、`AbsDao` 的子类中（即在业务线中）用优先父类中提供的消息方法发送消息,如下:

```
● sendMessage(String, String, Object) : void
```

7.8 事务规范

事务风格优先使用 `@Transactional` 注解的方式,然后是 xml 切面配置方式,然后是编程式事务。

`AbsController`、`AbsFacade`、`AbsService`、`AbsDao` 的子类中（即在业务线中）提供了编程式事务的公用方法（非高并发场景或者对事务机制了解不深的不建议使用,这时应该优先考虑注解方式或 XML 切面配置方式）。

```
● beginTransaction() : TransactionInfo
● commitTransaction(TransactionInfo) : void
● commitAndBeginTransaction(TransactionInfo) : void
● rollbackTransaction(TransactionInfo) : void
● rollbackAndBeginTransaction(TransactionInfo) : void
● setRollBackupOnlyTransaction(TransactionInfo) : void
```

7.9 异常规范

异常分为两类，业务异常 `com.credithc.common.exception.BizException` 和系统异常 `com.credithc.common.exception.SysException`。

业务异常范围包括：业务上判断不合法，抛出此异常，属于可预知的异常，这类异常都是业务操作非法，如入参不对、没有权限、业务未开通等，是正常的异常，用户或运营人员通过传入正确的入参、开通权限等就可以正常操作，不需要技术研发参与。

系统异常范围包括：属于不可预知的异常，一般抛出此异常，是系统出现严重 Bug、网络中断等突发情况，这时是需要技术人员、网管等人员进行排查并进行修复上线。

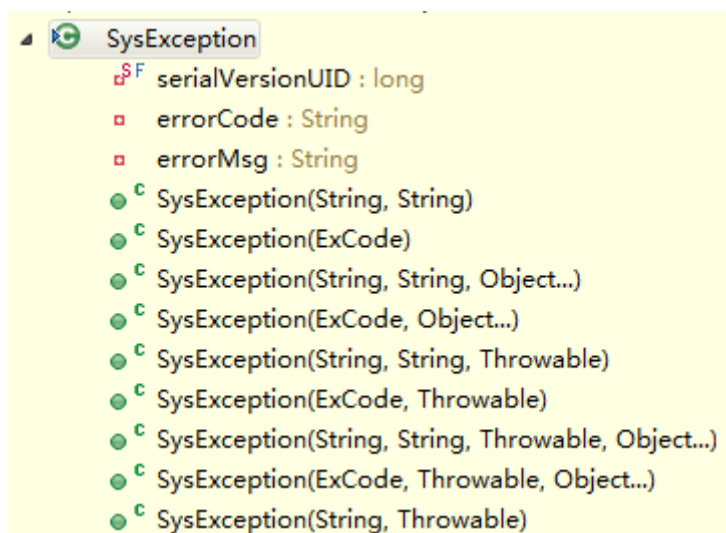
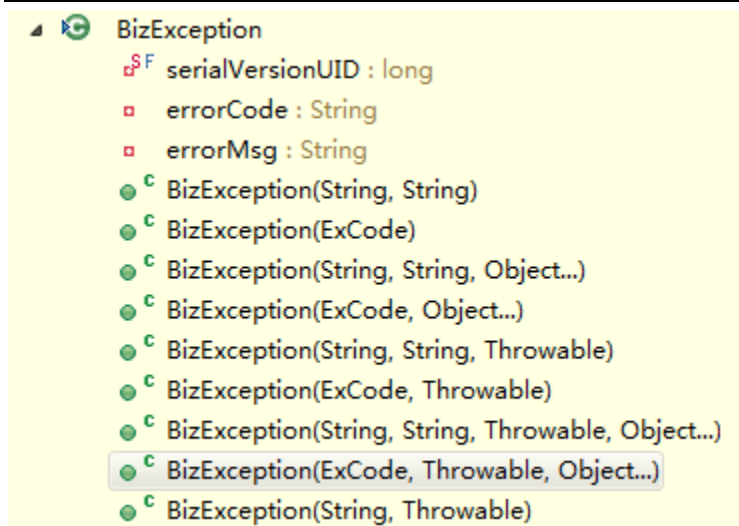
业务异常需要在方法签名上声明，系统异常无需声明。

所有下层模块只向上抛出异常，而不处理异常，异常统一由上层统一捕捉处理，并将异常代码和消息封装到响应码和响应消息中返回前端，对外暴露的接口不声明异常。

内层逻辑要考虑所有可能的异常的场景，并抛出对应的业务异常和系统异常。对于非可预知的异常（如运行时异常、网络中断等）统一包成系统异常。内层逻辑不输出异常日志，日志的输出统一在最外层输出，最内层只要抛异常即可。

对于系统异常，外层要输出 ERROR 级别的异常栈。对于业务异常，外层只要输出 WARN 级别的预先定义好的业务异常信息即可。

系统异常和业务异常的构造方法如下如示：



`errorCode` 为错误编码、`errorMsg` 为错误消息，错误消息支持 {0} 这样的占位符。`ExCode` 是对 `errorCode` 和 `errorMsg` 的一个封装，各个业务系统应该在各自的系统里实现这个接口定义枚举，业务根据是业务异常还是系统异常加上异常前缀 `BIZ_EX_PREFIX` 或 `SYS_EX_PREFIX`，如下示例。

```
1 package com.credithc.sample.common.exception;
2
3 import com.credithc.common.exception.ExCode;
4
5 public enum SampleExCode implements ExCode{
6
7     SUCCESS("000000", "成功"),
8
9     BIZ_SYS_ID_CANT_BE_NULL(BIZ_EX_PREFIX + "000001", "系统编号不能为空"),
10    BIZ_SYS_SEQ_CANT_BE_NULL(BIZ_EX_PREFIX + "000002", "请求序列号不能为空"),
11    //BIZ_...
12
13
14    SYS_LOCK_TOMEOUT(SYS_EX_PREFIX + "000001", "锁超时"),
15    //SYS_...
16    SYS_UNKNOWN_ERROR(SYS_EX_PREFIX + "999999", "未知系统异常")
17 ;
18
19 public String errorCode;
20 public String errorMsg;
```

7.10 import 规范

不要导入*, 比如 `import java.util.*;` 应该使用哪个类就导入哪个类, 不用的类不要导入, 可使用 `ctrl + shit + o` 自动识别导入和清理。

7.11 Serializable 规范

实现 `java.io.Serializable` 接口的类, 必须定义 `serialVersionUID`, 可以用 Eclipse 自动生成。有远程或有序列化需求的类都要实现 `java.io.Serializable`。

7.12 慎用模式

如果不是架构或通用代码, 慎用模式, 不要为了套用模式而用模式, 容易造成代码可读性差, 维护和交接困难。

7.13 通用工具

通用的工具类, 优先从 `apache-commons` 里面去找, 其次是 `spring framework` 里面找, 然后再是其它第三方工具, 最后才是自己实现工具类, 工具类命名一般以 `Util` 或 `Utils` 结尾, 且里面的方法是 `public static` 的, 建议类声明为 `final`

类型，以及构建方法声明为 private。

7.14 缓存规范

7.14.1 Mybatis 缓存

Mybatis 缓存有一级缓存和二级缓存，一级缓存是 session 缓存，用户无法控制，能控制的只有二级缓存。mybatis 缓存是以 namespace 为单位的，不同 namespace 下的操作互不影响，insert,update,delete 操作会清空所在 namespace 下的全部缓存。如果存在不同地方的多表关联查询，很可能出现脏数据，除非能保证所有对一张表的操作都集中在一个 namespace 下。另外，不能直接在数据库直接修改数据，这样数据无法及时同步到缓存中，也会出现脏数据问题。所以，mybatis 二级缓存，慎用。

如果要打开数据库缓存，要在 {project_name}\fw-sample-data\src\main\resources\META-INF\mybatis\mybatis-config.xml 中打开全局缓存开关，如下：

```
mybatis-config.xml
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
3 <configuration>
4   <settings>
5     <setting name="lazyLoadingEnabled" value="false" />
6     <setting name="cacheEnabled" value="true"/>
7   </settings>
8   <typeAliases>
9
10  </typeAliases>
11 </configuration>
```

全局缓存开关 cacheEnabled 默认是 true，如果它配成 false，其余各个 Mapper XML 文件配成支持 cache 也没用，各个 Mapper XML 文件，默认是不采用 cache。在配置文件加一行就可以支持 cache，且 select 语句的参数 useCache=true，如下。

```
SampleDOMapper.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
3 <mapper namespace="com.credithc.sample.dao.entity.SampleDO">
4
5   <cache type="com.credithc.common.mybatis.MybatisRedisCache"/>
6
7
8   <select id="query" parameterType="com.credithc.sample.dao.entity.SampleDO" resultMap="resultMap_SampleDO" useCache="true">
9     select
10     <include refid="Base_Column_List" />
11     from t_sample
12     <include refid="notNullWhereClause" />
13   </select>
```

7.14.2 业务缓存

AbsController、AbsFacade、AbsService、AbsDao 的子类中（即在业务线中）用优先父类中提供的缓存方法缓存数据，如下：

- redisSetValue(String, String) : String
- redisGetValue(String) : String
- redisRemoveValue(String) : Long
- redisExpire(String, int) : Long
- redisSetValueWithExpire(String, String, int) : Long
- redisTtl(String) : Long
- redisExists(String) : boolean
- redisKeys(String) : Set<String>
- redisRemoveValues(String) : long
- redisLpop(String) : String
- redisLpeek(String) : String
- redisRpush(String, String) : void
- redisBlpop(int, String) : String
- redisGetSet(String, String) : String
- redisCloseConnections(Set<Long>) : void
- redisFlushDb() : void

8 附录