# 开发平台

# 业务系统开发指导文档

目录

[dip平台业务系统开发指导文档 1](#_Toc59548895)

[1 环境准备 3](#_Toc59548896)

[1.1 jdk安装 3](#_Toc59548897)

[1.2 maven安装 5](#_Toc59548898)

[1.3 eclipse安装与配置 5](#_Toc59548899)

[2 新建maven父工程 7](#_Toc59548900)

[2.1 通过eclipse新建maven父工程，步骤如下： 7](#_Toc59548901)

[3 新建maven模块 10](#_Toc59548902)

[3.1 通过eclipse新建maven子模块，步骤如下： 10](#_Toc59548903)

[3.2 修改子模块的pom.xml文件，引入bizform和bpm依赖 13](#_Toc59548904)

[3.2.1 添加依赖的版本号 13](#_Toc59548905)

[3.2.2 添加依赖的坐标 13](#_Toc59548906)

[3.2.3 更新工程，如下图所示： 14](#_Toc59548907)

[3.2.4 在子模块如it-demo中的src/main/resources下新建application.yml和application-dev.yml配置文件 14](#_Toc59548908)

[3.2.5 修改it-demo工程的yml配置文件 14](#_Toc59548909)

[3.2.6 添加日志配置文件 18](#_Toc59548910)

[3.2.7 新建包和启动类 18](#_Toc59548911)

[3.2.8 新建业务包，如下图所示： 19](#_Toc59548912)

[3.2.9 启动xxxApplication启动类 20](#_Toc59548913)

[4 各层代码编写 21](#_Toc59548914)

[4.1 entity层 21](#_Toc59548915)

[4.2 dao层 22](#_Toc59548916)

[4.3 service接口层 24](#_Toc59548917)

[4.4 service实现层 25](#_Toc59548918)

[4.5 ui层（controller层） 26](#_Toc59548919)

# 环境准备

软件准备：

* Jdk8+ 下载地址：<http://java.sun.com/javase/downloads/index.jsp>
* Maven 下载地址：<http://maven.apache.org/download.cgi>
* Eclipse下载地址：http://www.eclipse.org/downloads/

## jdk安装

1. **下载好jdk安装程序, 双击下载的exe文件，开始安装。如下图，点击下一步**



1. **选择jdk的安装目录，推荐路径 d:\Java\jdk8 下（ 注意安装路径不要有空格） 。**



1. 直接点击下一步，出现告警信息，直接点“确定”，继续
2. 弹出个窗口，选择文件夹，安装jre，继续安装，点击下一步直到完成。



1. 配置环境变量，在桌面右击“计算机”图标，选择“属性”



1. 点击高级系统设置，在“高级”选项卡，点击下方“环境变量”按钮，如选择新建，输入变量名：JAVA\_HOME，　输入变量值：如d:\Java\jdk8  , 变量值为安装步骤b)中选择的安装路径
2. 添加Path路径， 系统变量→寻找 Path 变量→编辑，在变量值最后输入 d:\Java\jdk8\bin;

（ 注意原来Path的变量值末尾有没有;号， 如果没有先输入;号再输入上面的代码）

## maven安装

1. 加入maven环境到Path， 系统变量→寻找 Path 变量→编辑，在变量值最后输入 d:\maven\bin;（实际值为maven安装位置）
2. 配置maven仓库地址，修改conf/setting.xml文件，

<localRepository>D:/maven-repository</localRepository>

……

<mirror>

<id>alimaven</id>

<name>aliyun maven</name>

<url>http://10.22.15.121:8081/repository/maven-public/</url>

<mirrorOf>central</mirrorOf>

</mirror>

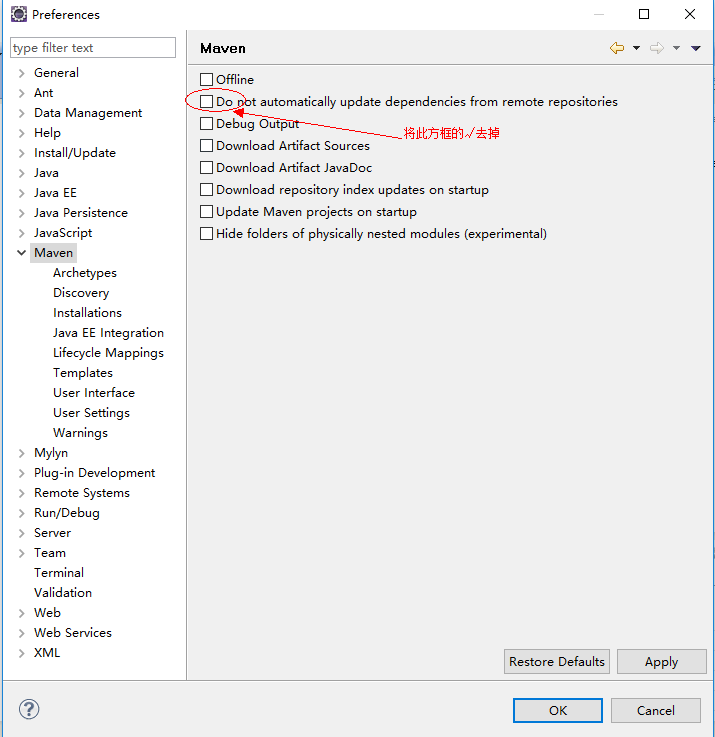


## eclipse安装与配置

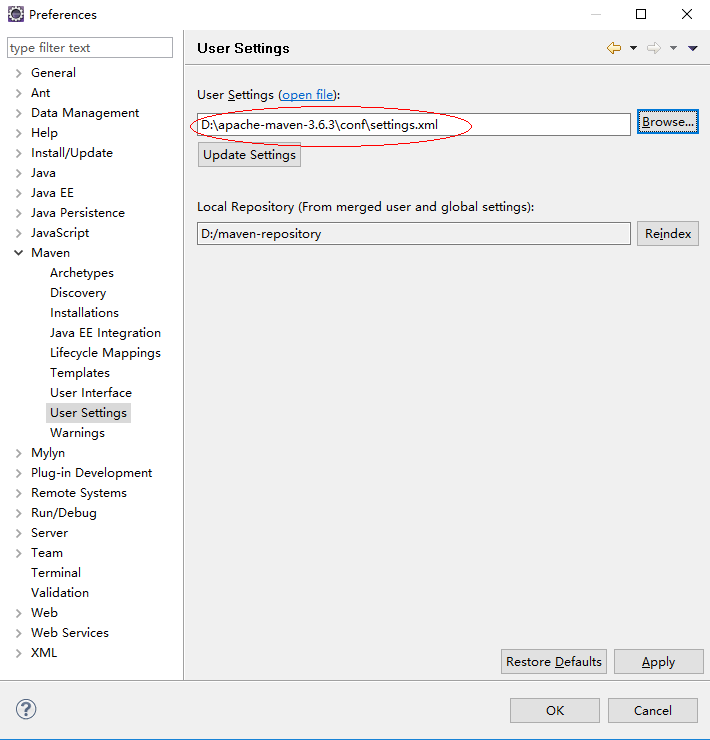
1. 配置文字编码UTF-8

(Window->Preferences->General->Workspace)

1. 配置java环境（Window->Preferences->Java-Installed JREs）
2. 配置java编译器版本为1.8（Window->Preferences->Java-Compiler）
3. 配置maven环境，（Window->Preferences->Maven）



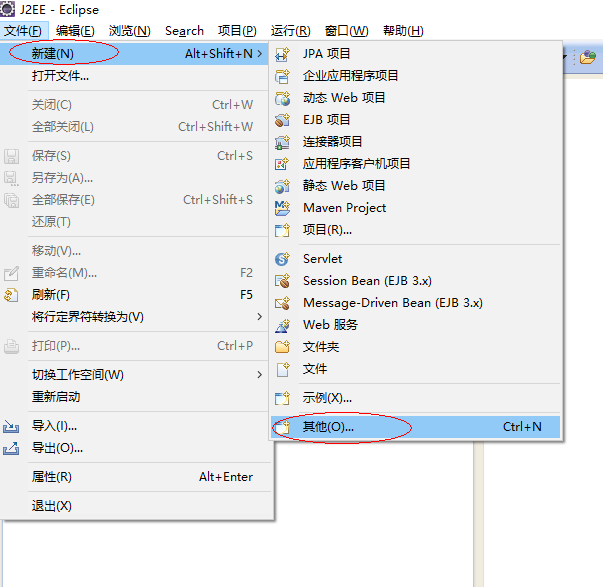
1. 配置maven的User Settings为安装的maven的settings.xml文件地址, Window->Preferences->Maven->User Settings）,



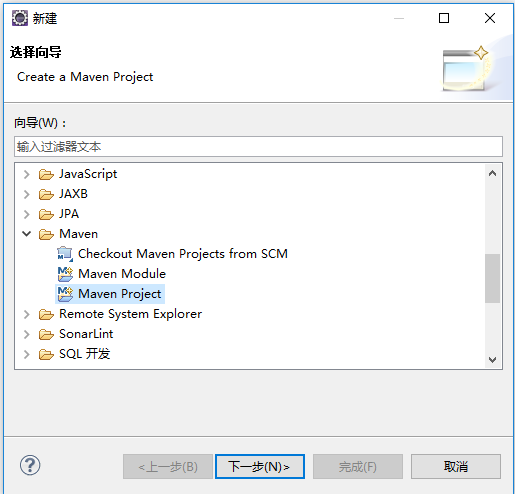
# 新建maven父工程

## 通过eclipse新建maven父工程，步骤如下：

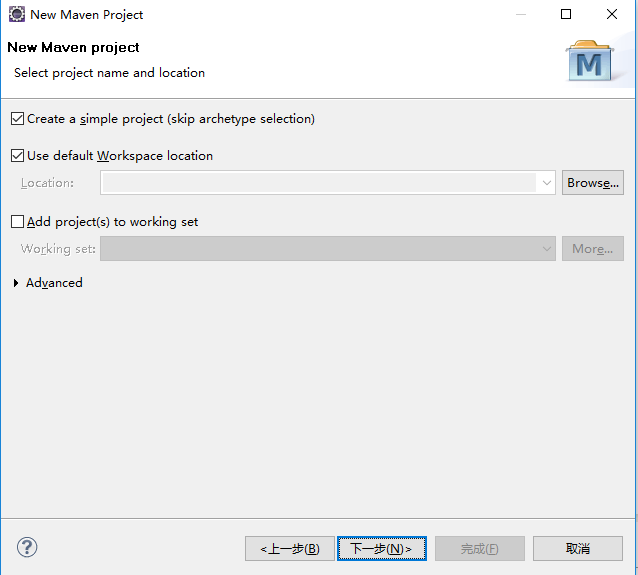
1. 点击文件—>新建—>其他，如下图所示：



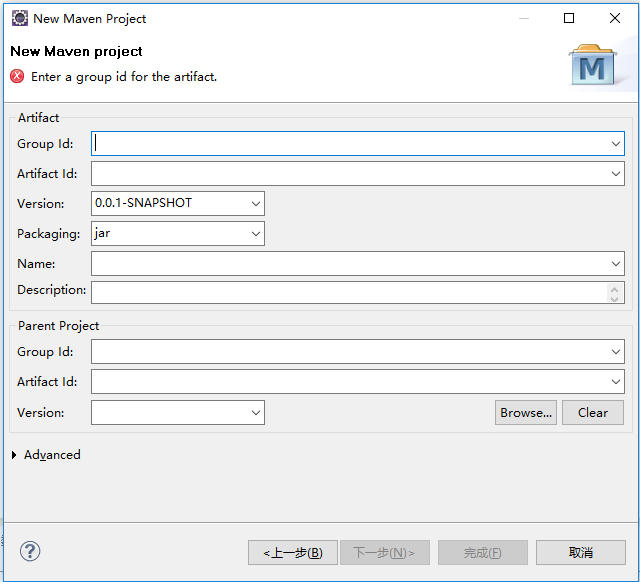
1. 在”选择向导”里选择Maven—>Maven Project，如下图所示：



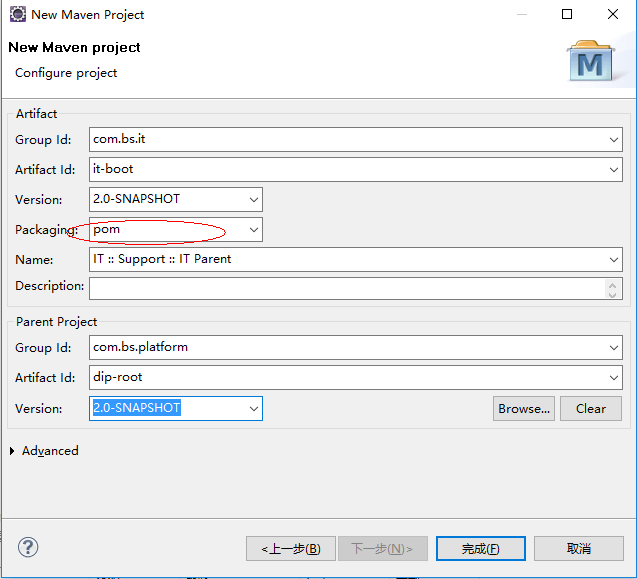
1. 点击下一步，弹出”New Maven Project” 界面，勾选Create a simple project(skip archetype selection),如下图所示：



1. 点击下一步，弹出“New Maven project”界面，填写工程坐标信息和父工程坐标信息，如下图所示：



1. 1.5 填入Group Id,Artifact Id,Version,Packaging,其中Packaging选择pom，再填入Parent Project的Group Id,Artifact Id,Version, 其中Parent Project的Group Id必须为com.bs.platform, Artifact Id必须为dip-root，Version必须为2.0-SHAPSHOT如下图所示：

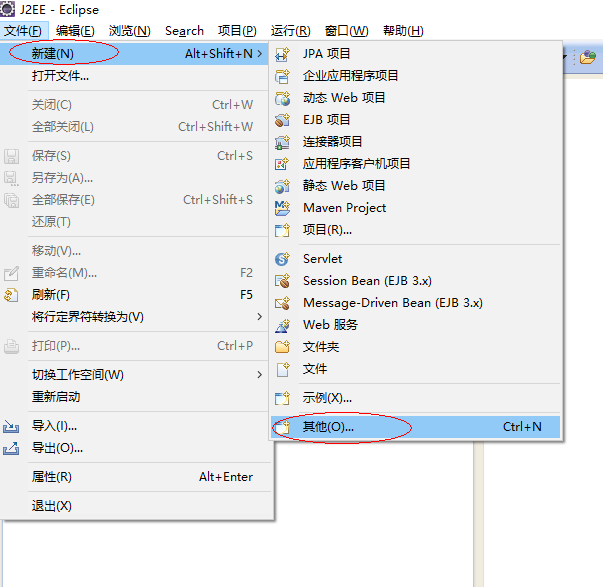


1. 点击“完成”按钮，完成父工程的创建

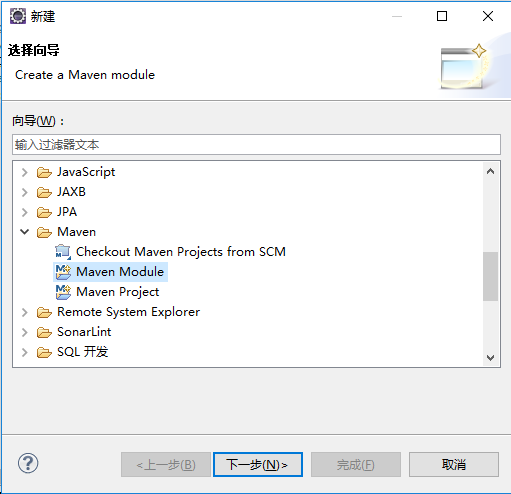
# 新建maven模块

## 通过eclipse新建maven子模块，步骤如下：

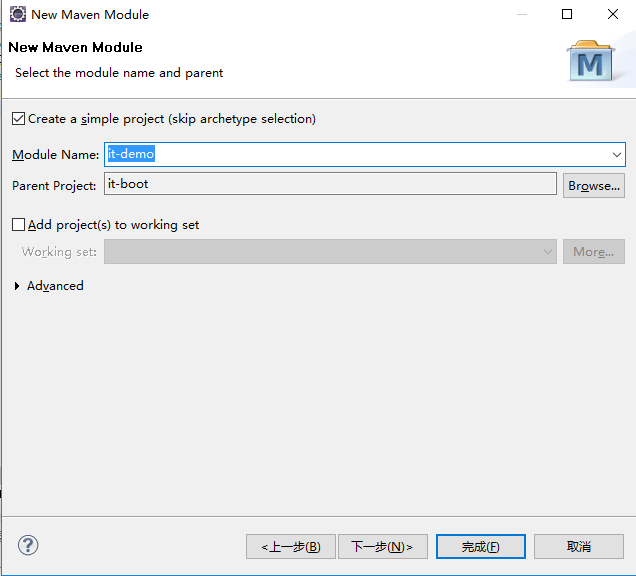
1. 点击文件—>新建—>其他，如下图所示：



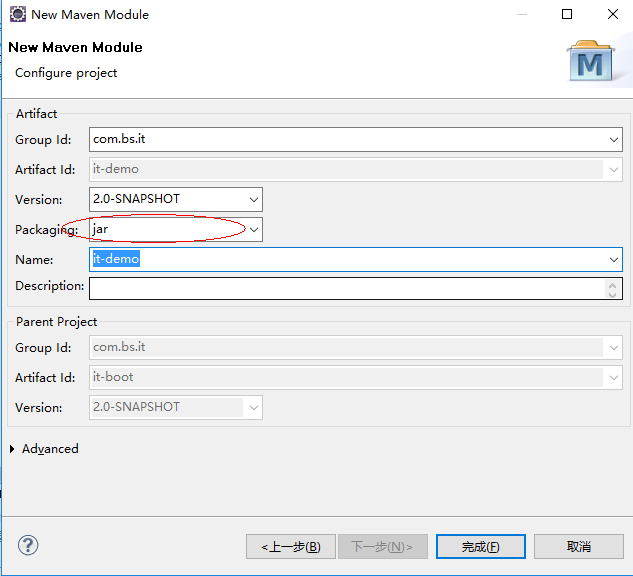
1. 在”选择向导”里选择Maven—>Maven Module，如下图所示：



1. 点击下一步，弹出”New Maven Module” 界面，勾选Create a simple project(skip archetype selection),如下图所示：



1. 填入Module Name,点击“下一步”按钮，进入New Maven Module界面，如下图所示：



1. 填入Group Id,Version,Packaging,Name,其中packaging选择jar，点击“完成”按钮，完成子模块的创建

## 修改子模块的pom.xml文件，引入bizform和bpm依赖



### 添加依赖的版本号

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

<dip.version>2.0-SNAPSHOT</dip.version>

<bpm.version>2.0-SNAPSHOT</bpm.version>

</properties>

### 添加依赖的坐标

<dependencies>

<dependency>

<groupId>com.bs.platform</groupId>

<artifactId>dip-spring-boot-starter-bizform</artifactId>

<version>${dip.version}</version>

</dependency>

<dependency>

<groupId>com.bs.platform</groupId>

<artifactId>dip-spring-boot-starter-bpm</artifactId>

<version>2.0-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>com.github.xiaoymin</groupId>

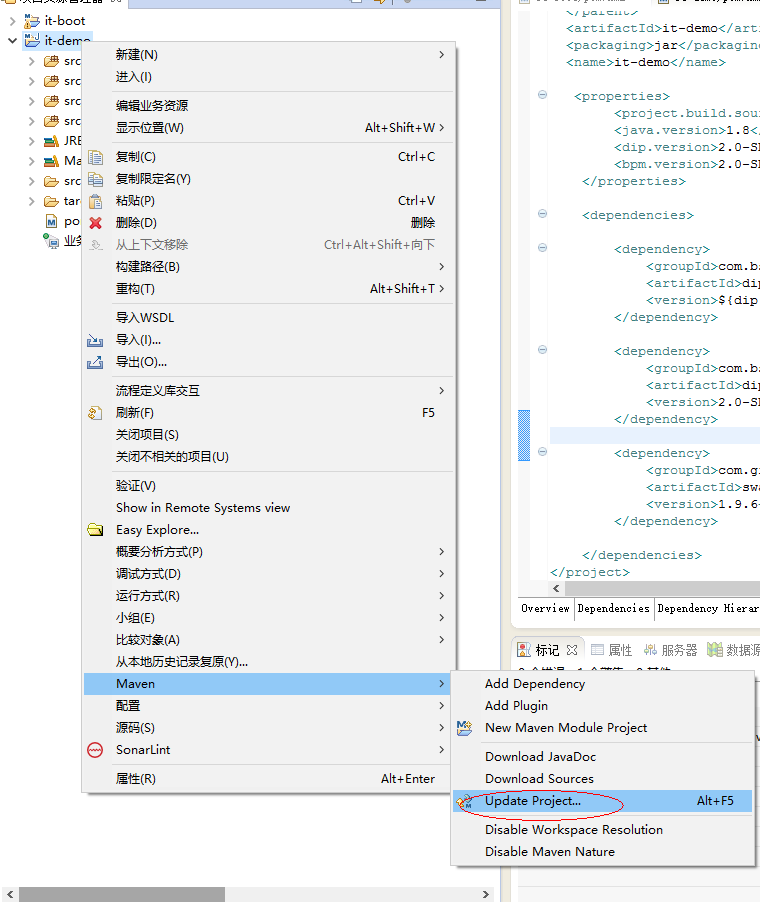
<artifactId>swagger-bootstrap-ui</artifactId>

<version>1.9.6</version>

</dependency>

</dependencies>

### 更新工程，如下图所示：



### 在子模块如it-demo中的src/main/resources下新建application.yml和application-dev.yml配置文件

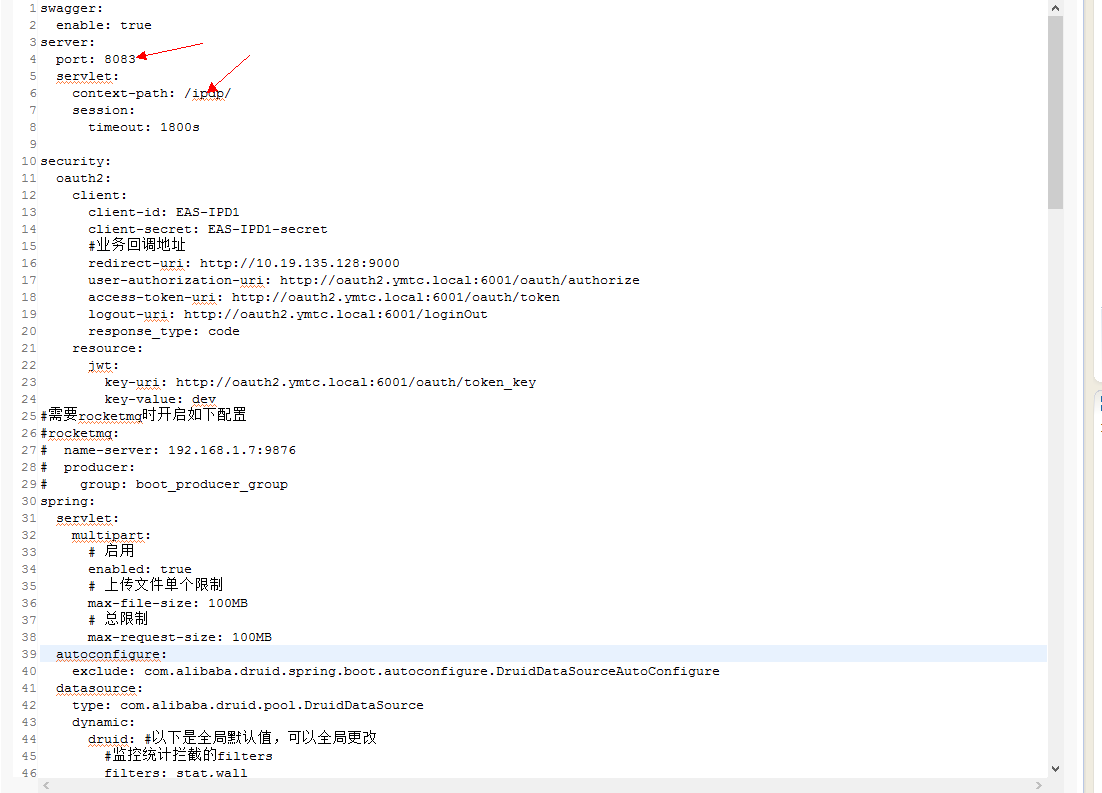


### 修改it-demo工程的yml配置文件

1. 修改application.yml文件，将spring.profile.active修改成对应的环境，如开发环境为dev，测试环境为test，生产环境为prod



1. 修改application-环境名.ym；如生产环境配置文件application-prod.yml的配置
2. 修改server.port为实际的端口号如8083
3. 修改server.servlet.context-path为实际的工程名，如ipdp，如下图所示：



1. 修改security.oauth2.client.client-id为实际的客户端id如EAS-IPD1
2. 修改security.oauth2.client.client-secret为实际的客户端密码如EAS-IPD1-secret
3. 修改security.oauth2.client.redirect-uri为实际的前端的业务回调地址，

如：http://10.19.135.128:9000

1. 修改security.oauth2.client.user-authorization-uri为实际的认证服务器的认证地址，

如：<http://oauth2.ymtc.local:6001/oauth/authorize>

1. 修改security.oauth2.client.access-token-uri为实际的认证服务器的获取token地址，如：<http://oauth2.ymtc.local:6001/oauth/token>
2. 修改security.oauth2.client.logout-uri为实际的认证服务器的登出接口地址，

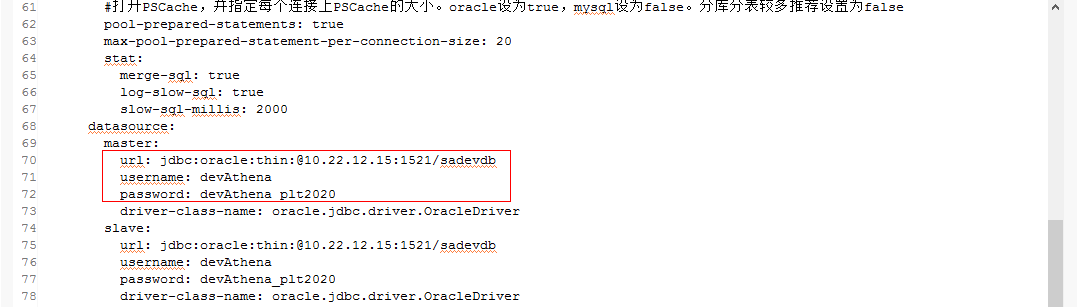
如：<http://oauth2.ymtc.local:6001/loginOut>



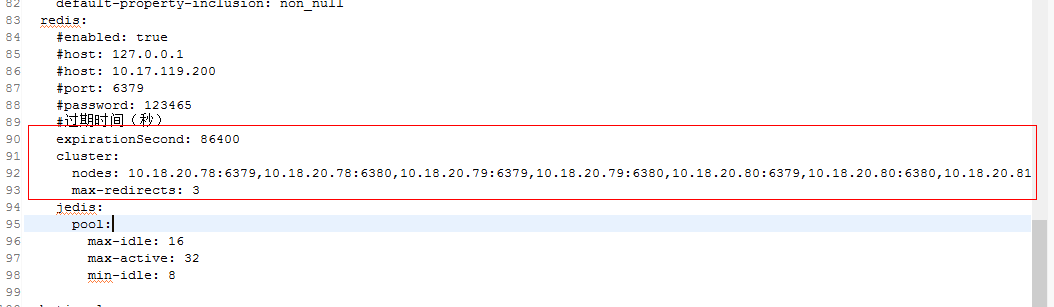
1. 修改security.oauth2.resource.jwt.key-uri为实际认证服务器的地址，

如：<http://oauth2.ymtc.local:6001/oauth/token_key>

1. 修改数据源的url,username,password为实际的值，如下图所示：



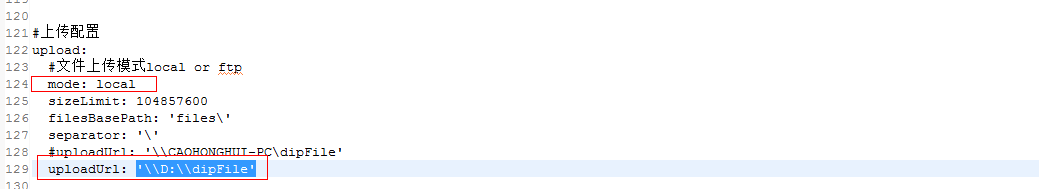
1. 修改redis的过期时间spring.redis.expirationSecond为实际的值如：86400
2. 修改redis的集群节点spring.redis.cluster.nodes为实际的ip加端口,多个节点用英文逗号分隔，如下图所示：



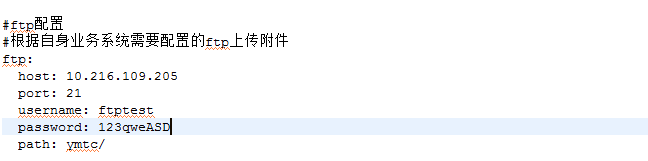
1. 修改日志配置logging.config为实际的日志配置文件地址如classpath:prod/logback.xml，如下图所示：



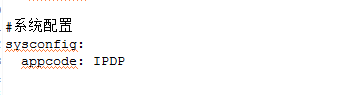
1. 修改文件上传模式upload.mode为local或ftp，local为上传到服务器本地目录，ftp为上传到ftp服务器
2. 修改本地上传模式下的文件上传路径upload.uploadUrl为实际的值如'\\D:\\dipFile'，如下图所示：



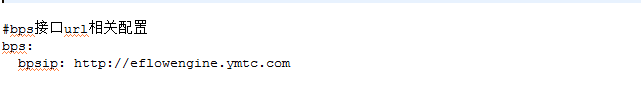
1. 修改上传模式为ftp时的ftp服务器配置信息
2. 修改[ftp.host为ftp服务器的主机地址如10.216.109.205](ftp://ftp.host为ftp服务器的主机地址如10.216.109.205)
3. 修改[ftp.port为ftp服务器的端口如21](ftp://ftp.port为ftp服务器的端口如21)
4. 修改[ftp.username为ftp服务器的登录用户名如ftptest](ftp://ftp.username为ftp服务器的登录用户名如ftptest)
5. 修改[ftp.password为ftp服务器的登录密码如123qweASD](ftp://ftp.password为ftp服务器的登录密码如123qweASD)
6. 修改[ftp.path](ftp://ftp.path)为ftp服务器的文件上传路径如ymtc/,如下图所示：



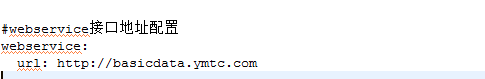
1. 修改系统编号sysconfig.appcode为实际的值如IPDP,如下图所示：



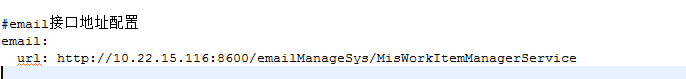
1. 修改bps接口url陪配置bps.bpsip为实际的值如<http://eflowengine.ymtc.com>,如下图所示：



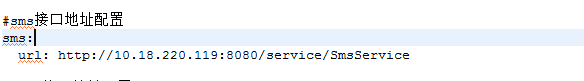
1. 修改webservice接口地址webservice.url为实际的值如<http://basicdata.ymtc.com/>，如下图所示：



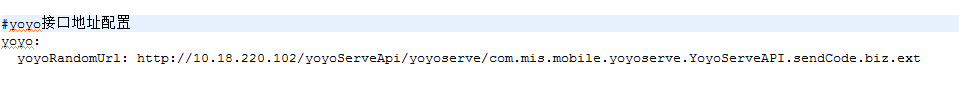
1. 修改邮件接口地址email.url为实际的值如<http://10.22.15.116:8600/emailManageSys/MisWorkItemManagerService>,如下图所示：



1. 修改短信接口地址sms.url为实际的值如<http://10.18.220.119:8080/service/SmsService>，如下图所示：



1. 修改yoyo验证码接口地址yoyo.yoyoRandomUrl为实际的值如<http://10.18.220.102/yoyoServeApi/yoyoserve/com.mis.mobile.yoyoserve.YoyoServeAPI.sendCode.biz.ext>,如下图所示：



### 添加日志配置文件

在src/main/resources下新建dev文件夹，将logback.xml配置文件拷贝进去



### 新建包和启动类

1. 新建包

在it-demo工程的src/main/java 中创建顶级包com.bs

包名必须为： com.bs.it.xxx\*\*

1. 新建启动类

启动类命名必须为xxxApplication

启动类必须放到包的最顶层com.bs中

启动类必须使用DipPlatform.run(xxxApplication.class, args);这种形式； 参考下面实例：

@SpringBootApplication

public class DemoApplication {

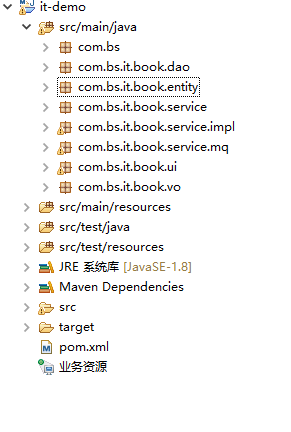
public static void main(String[] args) {

DipPlatformApp.run(DemoApplication .class, args);

}

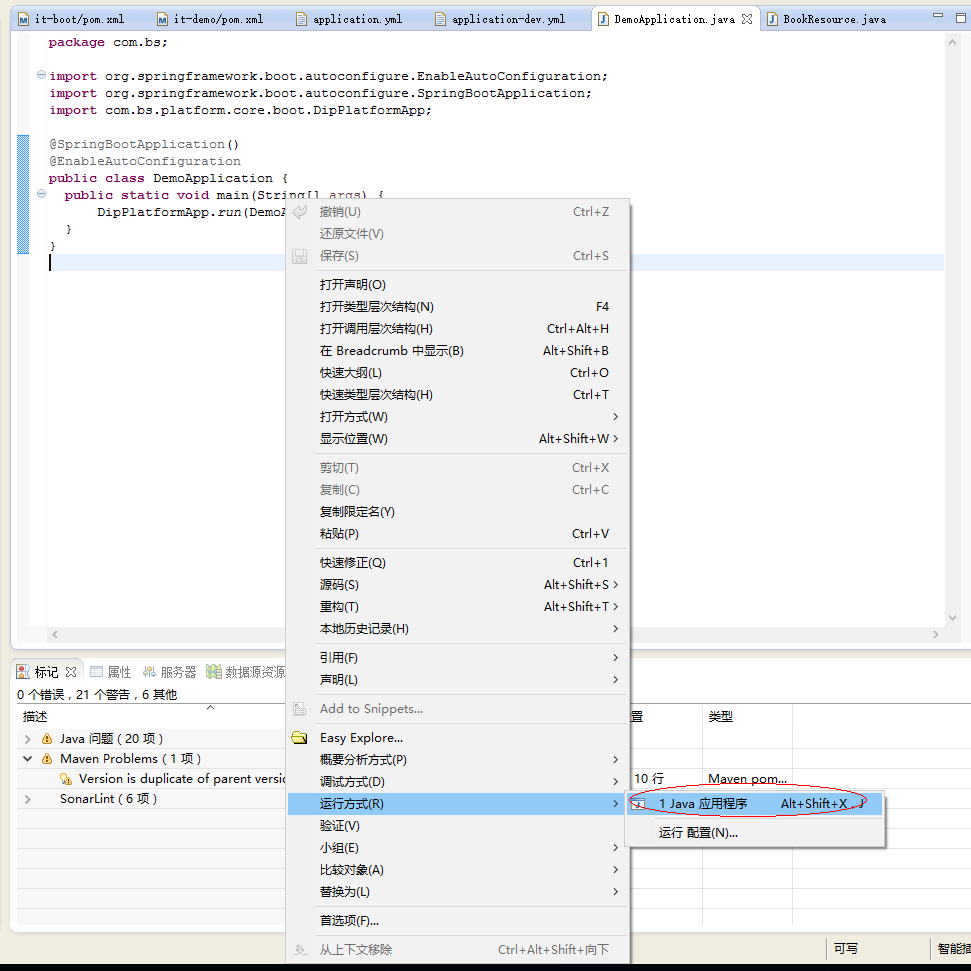
}

### 新建业务包，如下图所示：

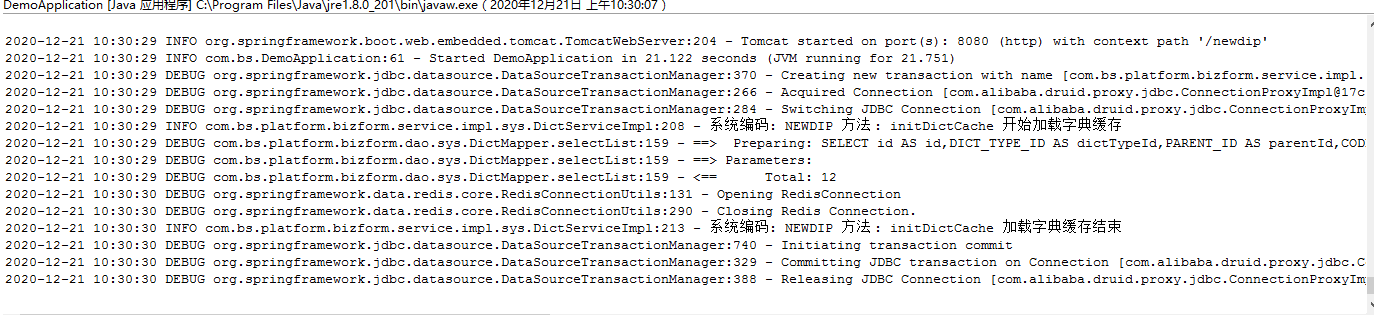


新建com.bs.it.xxx包，xxx为模块名，在com.bs.it.xxx包下新建entity包、vo包、dao包、service包、service.impl包、ui包

### 启动xxxApplication启动类



出现如下所示信息，说明启动成功：



# 各层代码编写

## entity层

**具体的Entity需继承自BaseEntity基类，以Book为例，示例代码如下：**

**package** com.bs.it.book.entity;

**import** javax.validation.constraints.NotEmpty;

**import** com.baomidou.mybatisplus.annotations.TableName;

**import** com.bs.platform.core.support.BaseEntity;

/\*\*

\* 国际化实体类，用于存储语言-键-值.

\*

\* **@author** chh

\*/

@TableName(value = "Book")

@KeySequence(value = "SEQ\_BOOK", clazz = Long.class)

**public** **class** Book **extends** BaseEntity<Book> {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 2884332738423850973L;

@NotEmpty

**private** String name;

**private** String remark;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getRemark() {

**return** remark;

}

**public** **void** setRemark(String remark) {

**this**.remark = remark;

}

}

其中@TableName(value = "Book")注解表示Book实体类对应的数据库表名为Book

@KeySequence(value = "SEQ\_BOOK", clazz = Long.class)表示Book表的主键

生成策略为序列方式，序列的名称为SEQ\_BOOK，主键对应的属性类型为Long

## dao层

**具体dao接口需继承CrudMapper接口，以book为例，示例代码如下：**

**package** com.bs.it.book.dao;

**import** java.util.List;

**import** com.bs.it.book.entity.Book;

**import** com.bs.platform.core.support.CrudMapper;

/\*\*

\* 国际化DAO.

\*

\* **@author** Bruce

\*/

**public** **interface** BookMapper **extends** CrudMapper<Book> {

/\*\*

\* 测试多数据源

\* **@return**

\*/

List<Book> list4Salve();

}

1. Mapper.xml文件编写

以book为例，示例代码如下：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace=*"com.bs.it.book.dao.BookMapper"*>

<!-- 通用查询映射结果 -->

<resultMap id=*"BaseResultMap"* type=*"com.bs.it.book.entity.Book"*>

<result column=*"id"* property=*"id"* /> <!-- id -->

<result column=*"name"* property=*"name"* /> <!-- 名称 -->

<result column=*"creator\_id"* property=*"creatorId"* /> <!-- 创建人 -->

<result column=*"create\_time"* property=*"createTime"* /> <!-- 创建时间 -->

<result column=*"last\_modifier\_id"* property=*"lastModifierId"* /> <!-- 最后修改人 -->

<result column=*"last\_modify\_time"* property=*"lastModifyTime"* /> <!-- 最后修改时间 -->

<result column=*"remark"* property=*"remark"* />

</resultMap>

<!-- 通用查询结果列 -->

<sql id=*"Base\_Column\_List"*>

id,name,remark,creator\_id,create\_time,last\_modifier\_id,last\_modify\_time

</sql>

<select id=*"list4Salve"* resultMap=*"BaseResultMap"*>

SELECT <include refid=*"Base\_Column\_List"* />

FROM book

</select>

</mapper>

其中，<mapper namespace="com.bs.it.book.dao.BookMapper">中的namespace属性的值必须为对应dao的全限定名

<resultMap id="BaseResultMap" type="com.bs.it.book.entity.Book">中的type属性必须为对应entity的全限定名

Mapper创建好后需放到src/main/resources下的对应目录如com/bs/it/book/dao下，如下图所示：

## service接口层

**service接口层需继承ICrudService接口，以book为例，示例代码如下：**

**package** com.bs.it.book.service;

**import** org.springframework.transaction.annotation.Transactional;

**import** com.bs.it.book.dao.BookMapper;

**import** com.bs.it.book.entity.Book;

**import** com.bs.platform.core.support.ICrudService;

**import** com.bs.platform.core.vo.CommonResult;

/\*\*

\* 国际化Service.

\*

\* **@author** Bruce

\*/

@Transactional()

**public** **interface** BookService **extends** ICrudService<BookMapper,Book,Long> {

/\*\*

\* **@description**: 测试多数据库查询

\* **@return**

\*/

CommonResult list4Salve();

**void** saveThis();

}

## service实现层

**service实现层需继承CrudServiceImpl基类，并实现service接口层的接口，以book为例，示例代码如下：**

**package** com.bs.it.book.service.impl;

**import** java.util.List;

**import** javax.annotation.Resource;

**import** org.springframework.stereotype.Service;

**import** com.baomidou.dynamic.datasource.annotation.DS;

**import** com.bs.it.book.dao.BookMapper;

**import** com.bs.it.book.entity.Book;

**import** com.bs.it.book.service.BookService;

**import** com.bs.platform.core.annotations.Log;

**import** com.bs.platform.core.constant.LogOutType;

**import** com.bs.platform.core.support.CrudServiceImpl;

**import** com.bs.platform.core.vo.CommonResult;

/\*\*

\* 国际化Service.

\*

\* **@author** chh

\*/

@Service

**public** **class** BookServiceImpl **extends** CrudServiceImpl<BookMapper, Book, Long> **implements** BookService {

@Resource

**private** BookMapper bookMapper;

/\*\*

\* **@description**: 测试多数据库查询

\* **@return**

\*/

@DS("slave")

@Log(outType=LogOutType.***FILE***, value="测试多数据库查询")

**public** CommonResult list4Salve(){

CommonResult result = **new** CommonResult();

List<Book> books = bookMapper.list4Salve();

result.setData(books);

**return** result;

}

**public** **void** saveThis(){

Book book = **new** Book();

book.setName("xxxx");

bookMapper.insert(book);

**int** i = 1/0;

}

}

## ui层（controller层）

**ui层需继承BaseCrudController基类，以book为例，示例代码如下：**

package com.bs.it.book.ui;

import io.swagger.annotations.Api;

import io.swagger.annotations.ApiOperation;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.InputStream;

import java.util.ArrayList;

import java.util.Collection;

import java.util.Date;

import java.util.List;

import javax.annotation.Resource;

import javax.servlet.http.HttpServletRequest;

import org.apache.rocketmq.client.producer.SendCallback;

import org.apache.rocketmq.client.producer.SendResult;

import org.apache.rocketmq.spring.core.RocketMQTemplate;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.messaging.support.MessageBuilder;

import org.springframework.util.ResourceUtils;

import org.springframework.validation.annotation.Validated;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.bind.annotation.RestController;

import com.bs.it.book.entity.Book;

import com.bs.it.book.service.BookService;

import com.bs.it.book.vo.OrderExt;

import com.bs.it.book.vo.ReadExcelDto;

import com.bs.platform.bps.processInstManager.BpsProcessInstManagerImpl;

import com.bs.platform.bps.processInstManager.BpsProcessInstManagerServiceServiceStub.CreateAndStartProcessInstance;

import com.bs.platform.bps.processInstManager.BpsProcessInstManagerServiceServiceStub.CreateAndStartProcessInstanceResponse;

import com.bs.platform.bps.workItemManager.BpsWorkItemManagerImpl;

import com.bs.platform.bps.workItemManager.BpsWorkItemManagerServiceServiceStub.QueryWorkItemParticipantInfo;

import com.bs.platform.bps.workItemManager.BpsWorkItemManagerServiceServiceStub.QueryWorkItemParticipantInfoResponse;

import com.bs.platform.core.annotations.Log;

import com.bs.platform.core.annotations.WhiteList;

import com.bs.platform.core.redis.RedisCacheUtils;

import com.bs.platform.core.support.BaseCrudController;

import com.bs.platform.core.utils.ExcelUtils;

import com.bs.platform.core.utils.RestTemplateUtils;

import com.bs.platform.core.vo.CommonResult;

import com.bs.platform.core.vo.ExcelLog;

import com.bs.platform.core.vo.ExcelLogs;

import com.bs.platform.core.vo.SheetData;

/\*\*

\* @copyright:primeton.com

\* @author: chh

\* @version: 1.0

\*/

@RestController

@RequestMapping("/book")

@Api(tags = "图书管理")

@Log("图书管理")

@WhiteList

public class BookResource extends BaseCrudController<Book, Long> {

@Resource

private BookService service;

/\*\*

\* 测试获取book列表数据

\*

\* **@return**

\*/

@ApiOperation(value = "测试获取book列表数据", notes = "查询book列表数据")

@ResponseBody

@RequestMapping(value = "/listAll")

**public** CommonResult listAll() {

List<Book> list = getService().selectList(**null**);

**return** **new** CommonResult(list);

}

}

其中@WhiteList注解表示此ui层的接口不需要登录认证就可直接访问

在浏览器中输入接口地址，如<http://localhost:8080/newdip/book/listAll>，如果返回了Book表的数据，则表示接口调用成功

样例工程如下：

