

开发指导书V1.0

文档首页

版本	时间	修改人	说明
1.1.1	2018-06-05	柯尊超	增加pom文件的配置截图参考
1.1.0	2018-06-05	柯尊超	1.增加 spring 和 maven 的基础知识点 2.增加Mybatis-generator的 GUI工具 3.增加 服务发布地址 配置说明
1.0.0	2018-05-30	柯尊超	--

文档首页

一、框架介绍

二、基础环境

三、知识要点

1.maven节点

2.Spring的Bean定义与注入

2.1.Bean定义

2.2 Bean注入

四、工程代码

1.项目主体结构

2.子模块的结构

2.1 命名规范

2.2 sp服务包代码结构

2.3 object实现包结构

2.4 maven使用

3.应用的配置文件

1. 全局配置文件

2.与spring集成

3.JAVA获取配置

4.workspace使用配置文件

4.Mybatis-generator插件使用

1.GUI工具（推荐）

2.ideal插件

五、开发规范

开发规约

插件安装

六、环境搭建与配置

Step1:github克隆种子项目

Step2:cityserver基础包

Step3:配置私有maven仓库

Step4:maven打包路径

一、框架介绍

框架是以spring和cxf为基础，集成了mybatis，将服务配置信息集中化，充分利用Spring的特性，按照MVC的模型进行开发。与之前的开发模式相比，更换了持久层为Mybatis，不再推荐使用之前的workspace，但是还是兼容之前的workspace的用法。

种子文件地址:<https://github.com/zizhengzhuan/cf-service-seed>

二、基础环境

- Git: 种子文件托管空间 ([git介绍](#))
- Maven: 包依赖管理 ([maven介绍](#))
- JAVA: java编译，版本统一为 jdk1.8
- tomcat: 程序运行环境，版本为 8.x
- IDE: 统一为 IntelliJ IDEA，不反对使用MyEclipse，但是不提供支持。

三、知识要点

1.maven节点

在maven的对象管理至少包含 groupId、artifactId、version，这三个要素确定唯一——个项目。

- groupId: 组织ID，如 com.ecity
- artifactId: 对象名，通常是项目名或者模块名
- version: 版本，分releases和snapshots

POM文件介绍：

```
<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/maven-v4_0_0.xsd"
">

    <!-- 父项目的坐标。如果项目中没有规定某个元素的值，那么父项目中的对应值即为项目的默认值。
        坐标包括group ID, artifact ID和 version。 -->
    <parent>
        <!-- 被继承的父项目的构件标识符 -->
        <artifactId>com.zzht</artifactId>

        <!-- 被继承的父项目的全球唯一标识符 -->
        <groupId>cf-project</groupId>
```

```

<!-- 被继承的父项目的版本 -->
<version>1.0.0-releases</version>

<!-- 父项目的pom.xml文件的相对路径。相对路径允许你选择一个不同的路径。默认值是../pom.xml。
Maven首先在构建当前项目的地方寻找父项目的pom，其次在文件系统的这个位置（relativePath位置），
然后在本地仓库，最后在远程仓库寻找父项目的pom。 -->
<relativePath>xxx</relativePath>
</parent>

<!-- 声明项目描述符遵循哪一个POM模型版本。模型本身的版本很少改变，虽然如此，但它仍然是必不可少的，
这是为了当Maven引入了新的特性或者其他模型变更的时候，确保稳定性。 -->
<modelVersion> 4.0.0 </modelVersion>

<!-- 项目的全球唯一标识符，通常使用全限定的包名区分该项目和其他项目。并且构建时生成的路径也是由此生成，
如com.mycompany.app生成的相对路径为： /com/mycompany/app -->
<groupId>com.zzht</groupId>

<!-- 构件的标识符，它和group ID一起唯一标识一个构件。换句话说，你不能有两个不同的项目拥有同样的
artifact ID
和groupId；在某个特定的group ID下，artifact ID也必须是唯一的。构件是项目产生的或使用的一个东西，Maven
为项目产生的构件包括：JARs，源码，二进制发布和WARs等。 -->
<artifactId>sp-demo</artifactId>

<!-- 项目产生的构件类型，例如jar、war、ear、pom。插件可以创建他们自己的构件类型，所以前面列的不是全部构件类型 -->
<packaging> jar </packaging>

<!-- 项目当前版本，格式为:主版本.次版本.增量版本-限定版本号 -->
<version> 1.0-SNAPSHOT </version>

<!-- 以值替代名称，Properties可以在整个POM中使用，也可以作为触发条件（见settings.xml配置文件里
activation元素的说明）。格式是<name>value</name>。 -->
<properties>
  <jdk.version>1.8</jdk.version>
</properties>

<!-- 模块（有时称作子项目） 被构建成项目的一部分。列出的每个模块元素是指向该模块的目录的相对路径 -->
<modules>
  <!--子项目相对路径-->
  <module></module>
</modules>

<!-- 该元素描述了项目相关的所有依赖。 这些依赖组成了项目构建过程中的一个个环节。它们自动从项目定义的
仓库中下载。
要获取更多信息，请看项目依赖机制。 -->
<dependencies>
  <dependency>
    <!-- 依赖的group ID -->
    <groupId> org.apache.maven </groupId>

```

```

<!-- 依赖的artifact ID -->
<artifactId> maven-artifact </artifactId>

<!-- 依赖的版本号。在Maven 2里，也可以配置成版本号的范围。 -->
<version> 3.8.1 </version>

<!-- 依赖类型，默认类型是jar。它通常表示依赖的文件的扩展名，但也有例外。一个类型可以被映射成
另外一个扩展
名或分类器。类型经常和使用的打包方式对应，尽管这也有例外。一些类型的例子：jar, war,
ejb-client和test-jar。
如果设置extensions为 true，就可以在plugin里定义新的类型。所以前面的类型的例子不完整。
-->
<type> jar </type>

<!-- 依赖的分类器。分类器可以区分属于同一个POM，但不同构建方式的构件。分类器名被附加到文件名
的版本号后面。例如，
如果你想要构建两个单独的构件成JAR，一个使用Java 1.4编译器，另一个使用Java 6编译器，你
就可以使用分类器来生
成两个单独的JAR构件。 -->
<classifier></classifier>

<!-- 依赖范围。在项目发布过程中，帮助决定哪些构件被包括进来。欲知详情请参考依赖机制。
- compile：默认范围，用于编译
- provided：类似于编译，但支持你期待jdk或者容器提供，类似于classpath
- runtime：在执行时需要使用
- test：用于test任务时使用
- system：需要外在提供相应的元素。通过systemPath来取得
- systemPath：仅用于范围为system。提供相应的路径
- optional：当项目自身被依赖时，标注依赖是否传递。用于连续依赖时使用 -->
<scope> test </scope>

<!-- 仅供system范围使用。注意，不鼓励使用这个元素，并且在新的版本中该元素可能被覆盖掉。该元
素为依赖规定了文件
系统上的路径。需要绝对路径而不是相对路径。推荐使用属性匹配绝对路径，例如${java.home}。
-->
<systemPath></systemPath>

<!-- 当计算传递依赖时，从依赖构件列表里，列出被排除的依赖构件集。即告诉maven你只依赖指定的
项目，不依赖项目的
依赖。此元素主要用于解决版本冲突问题 -->
<exclusions>
  <exclusion>
    <artifactId> spring-core </artifactId>
    <groupId> org.springframework </groupId>
  </exclusion>
</exclusions>

<!-- 可选依赖，如果你在项目B中把C依赖声明为可选，你就需要在依赖于B的项目（例如项目A）中显式
的引用对C的依赖。
可选依赖阻断依赖的传递性。 -->
<optional> true </optional>
</dependency>

```

```

</dependencies>

<!-- 构建 -->
<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <!-- 编译的jdk版本 -->
                    <encoding>utf-8</encoding>
                    <source>${jdk.version}</source>
                    <target>${jdk.version}</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <configuration>
                    <skip>true</skip>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>

</build>
</project>

```

2.Spring的Bean定义与注入

2.1.Bean定义

Bean的定义分java注解和xml文件方式

重要提示： bean定义时候指定的id不能重复，否则会报错。

通常如果不涉及依赖注入的引用，不需要指定Bean的ID

```

import org.springframework.stereotype.Component;
import org.springframework.stereotype.Repository;
//①通过Repository定义一个DAO的Bean

@Component("userDao")
public class UserDao {

}

```

在①处，我们使用@Component注解在 UserDao 类声明处对类进行标注，它可以被Spring容器识别，Spring容器自动将POJO转换为容器管理的Bean。

它和以下的XML配置是等效的：

```
<bean id="userDao" class="com.baobaotao.anno.UserDao"/>
```

@Component以外，Spring提供了3个功能基本和@Component等效的注解，它们分别用于对DAO、Service及Web层的Controller进行注解，所以也称这些注解为Bean的衍型注解：

- @Repository：用于对DAO实现类进行标注；
- @Service：用于对Service实现类进行标注；
- @Controller：用于对Controller实现类进行标注；

2.2 Bean注入

Bean注入分属性注入、构造函数注入和工厂方法注入，这里只介绍属性注入。

- xml方式：

```
<bean id="logDao" class="com.xxxx.xxx.LogDao"/>
<bean id="userDao" class="com.xxxx.xxx.UserDao"/>
<bean class="com.xxxx.xxx.LogonService">
    <property name="logDao" ref="logDao"></property>
    <property name="userDao" ref="userDao"></property>
</bean>
```

- 注解方式：

使用@Autowired注入或者@Resource和@Inject注解，区别：

- 1.@Autowired注入是按照类型注入的，只要配置文件中的bean类型和需要的bean类型是一致的；
- 2.@Resource标签是按照bean的名字来进行注入的，如果我们没有在使用@Resource时指定bean的名字，同时Spring容器中又没有该名字的bean,这时候@Resource就会退化为@Autowired即按照类型注入

```
//① 定义一个Service的Bean（不需要在XML中定义Bean）
@Service
public class LogonService implements BeanNameAware{
    //② 分别注入LogDao及UserDao的Bean（不需要在XML中定义property属性注入）
    //如果注入的接口实现类有多个实现，需要@Qualifier指定对应名称
    @Autowired(required=false)
    private LogDao logDao;
    @Autowired
    @Qualifier("userDao")
    private UserDao userDao;

    public LogDao getLogDao() {
        return logDao;
    }
    public UserDao getUserDao() {
```

```

        return userDao;
    }

    public void setBeanName(String beanName) {
        System.out.println("beanName:"+beanName);
    }
    //.....
}

```

四、工程代码

1.项目主体结构

服务包和业务实现包以子项目形式存在。

- sp-demo: 单独的服务包
- demo-object: 服务对应的业务实现
- pom.xml: 基础依赖，具体服务包会继承这个pom的依赖，不要重复引用

父项目pom.xml

```

<modelVersion>4.0.0</modelVersion>

<groupId>com.zzht</groupId>
<artifactId>cf-project</artifactId>
<packaging>pom</packaging>
<version>1.0.0-snapshots</version>
<modules>
    <module>sp-demo</module>
    <module>demo-object</module>
</modules>

```

2.子模块的结构

2.1 命名规范

- 服务包以 `sp-` 开头，如 `sp-demo`；
- 服务实现包以 `-object` 结尾，如 `demo-object`；
- 服务包的基础报名前缀为 `com.zzht.service.xxxx`；
- 业务实现包的报名前缀为 `com.zzht.component.xxxx`；

示例代码：

```

// #1.服务包
package com.zzht.service.demo.common;
//.....
//....
// #2.实现包
package com.zzht.service.demo.webservice
//....
//....

```

2.2 sp服务包代码结构

```

## main
- common          公共部分
    - ServiceCore  实现IService, , 用来secore发布服务
- constanst       常量等基本定义
- exception       异常或者返回的消息定义
- webservice      服务接口
    - restful      服务接口restful
    - soap        服务接口soap
## resource
- Service.xml      服务发布依赖的bean
- Service-REST-http.xml  REST服务地址绑定
- Service-SOAP-http.xml SOAP服务地址绑定
- pom.xml

```

- pom.xml:模块的POM


```

<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.apac

  <parent>
    <artifactId>cf-project</artifactId>
    <groupId>com.zzht</groupId>
    <version>1.0.0-snapshots</version>
  </parent>
  <packaging>jar</packaging>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>sp-demo</artifactId>
  <version>1.0.0-snapshots</version>
  <dependencies>
    <dependency>
      <groupId>com.zzht</groupId>
      <artifactId>demo-object</artifactId>
      <version>1.0.0-snapshots</version>
    </dependency>
  </dependencies>

  <!-- 构建 -->
  <build>
    <plugins...>
  </build>
</project>

```

- Service.xml: 基本的bean配置
- Service-REST-http.xml: 服务发布地址配置
 - `jaxrs:server` 表示一个服务配置, 可以配置多个
 - **id和address全局不能和其他的服务相同, 否则发布不成功**
 - 建议id和address按照自己的项目或者模块名称作为前缀

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxrs="http://cxf.apache.org/jaxrs"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd
    http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd
    http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd">

  <jaxrs:server id="demoService" address="/rest/personService">
    <!--serviceBeans: 暴露的WebService服务类-->
    <jaxrs:serviceBeans>
      <ref bean="personRestService"/>
    </jaxrs:serviceBeans>

    <jaxrs:providers>
      <bean id="jsonProvider"
        class="org.apache.cxf.jaxrs.provider.json.JSONProvider">

```

```

        <property name="dropRootElement" value="true"/>
        <property name="dropCollectionWrapperElement" value="true"/>
    </bean>
</jaxrs:providers>

<jaxrs:inInterceptors>
    <bean class="org.apache.cxf.transport.common.gzip.GZIPInInterceptor"/>
</jaxrs:inInterceptors>

</jaxrs:server>
</beans>

```

2.3 object实现包结构

```

## main:com.zzht.component.xxx
- api          jdk级别的API调用, 包含逻辑实现层
- common       公共部分抽离
- dao          数据访问层
- entity       实体类
- exception    自定义异常
## resource
- com/zzht/component/xxx/dao          mybatis数据访问配置文件目录
    - XxxMapper.xml    mapper配置
- spring/spring-mabaties.xml mybatis与spring集成配置
- pom.xml

```

- pom.xml:实现包的模块POM

```

<?xml version="1.0" encoding="UTF-8"?>
<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
    <parent>
      <artifactId>cf-project</artifactId>
      <groupId>com.zzht</groupId>
      <version>1.0.0-snapshots</version>
    </parent>
    <packaging>jar</packaging>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>demo-object</artifactId>

    <version>1.0.0</version>
    <dependencies>
      <dependency>
        <groupId>com.microsoft.sqlserver</groupId>
        <artifactId>sqljdbc4</artifactId>
        <version>4.0</version>
        <scope>test</scope>
      </dependency>
    </dependencies>

    <!-- 构建 -->
    <build...>
  </project>

```

2.4 maven使用

maven的私有仓库配置：参照《[maven私服使用-1.0.2.pdf](#)》

```

mvn clean install:将打包的jar添加到仓库（本地）
mvn clean package: 打包

```

3.应用的配置文件

1. 全局配置文件

应用的配置文件统一放在 `cityserver/conf` 里面， `application.yml`，也允许自己项目增加额外配置的文件，非必要情况下不推荐。

[YAML规范](<http://www.ruanyifeng.com/blog/2016/07/yaml.html>)（尤其注意缩进和冒号之后的空格）

[YAML校验](<https://codebeautify.org/yaml-validator>)

文件 `application.yml`

```

## 数据库配置
###数据库类型: mysql、mssql、oracle

### driverClassName:

```

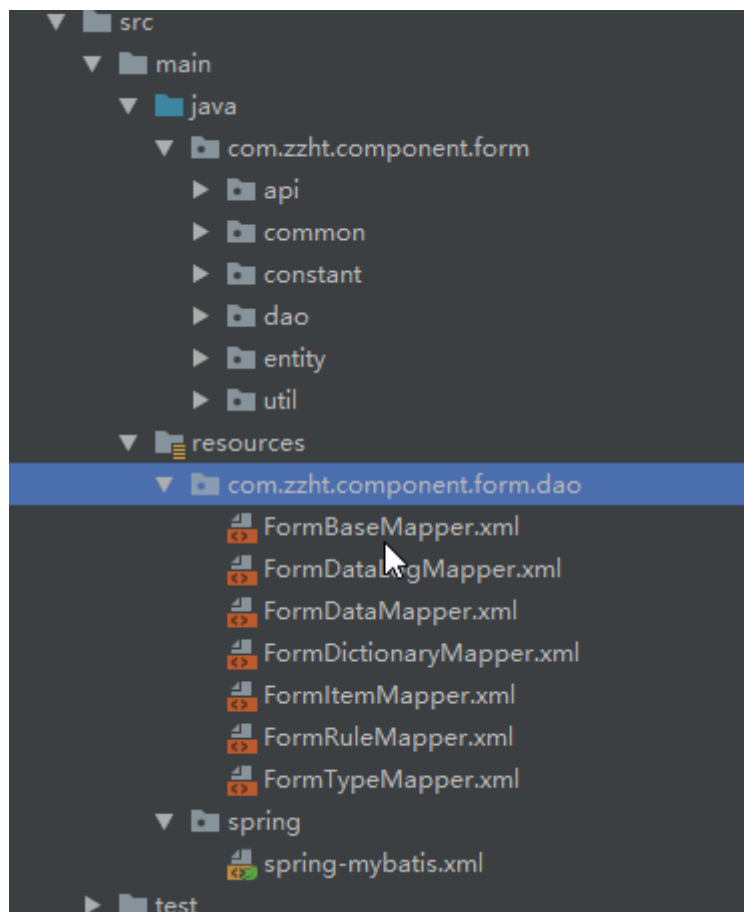
```
### mysql:com.mysql.jdbc.Driver
### mssql:com.microsoft.sqlserver.jdbc.SQLServerDriver
### oracle:oracle.jdbc.driver.OracleDriver

## 基础数据库配置
db: &base1
  type: mssql
  driver: com.microsoft.sqlserver.jdbc.SQLServerDriver
  url: jdbc:sqlserver://192.168.8.183:1433;DatabaseName=cole_flower
  username: zzht
  password: zzht
  initialSize: 0
  maxActive: 200
  maxIdle: 20
  minIdle: 1
  maxWait: 6000

##具体的项目配置文件可以直接引用基本数据库配置
oms: *base1
form:
  <<: *base2
  url: 192.168.8.183:1433:ecity
```

2.与spring集成

所有与spring的集成配置文件建议放在spring下面，classpath:spring/*.xml都会默认加入到context中。如果您不想您的配置文件集成xml自动加入context，放在除spring之外的位置。



例如：spring/spring-mybatis.xml

注意：bean的id是应用内唯一的，各项目需要添加前缀，把xxx替换掉。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd">

    <!-- 配置数据源 -->
    <!-- 配置数据源 -->
    <bean id="xxxDataSource" destroy-method="close"
        class="org.apache.commons.dbcp2.BasicDataSource"
        p:driverClassName="${form.driver}"
        p:url="${form.url}"
        p:username="${form.username}"
        p:password="${form.password}"
        p:initialSize="${form.initialSize}"
        p:maxTotal="${form.maxActive}"
        p:maxIdle="${form.maxIdle}"
        p:minIdle="${form.minIdle}"
    />

    <!-- spring和MyBatis完美整合，不需要mybatis的配置映射文件 -->
    <bean id="xxxSqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"
        p:dataSource-ref="xxxDataSource"
        p:mapperLocations="classpath:com/zzht/component/form/dao/*.xml"/>

    <!-- DAO接口所在包名，Spring会自动查找其下的类 --><!-- 只扫描带@Repository标签类-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"
        p:annotationClass="org.springframework.stereotype.Repository"
        p:basePackage="com.zzht.component.form.dao"
        p:sqlSessionFactoryBeanName="xxxSqlSessionFactory"/>

    <!-- 事务管理 -->
    <bean id="xxxTransactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
        p:dataSource-ref="xxxDataSource"/>

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

</beans>
```

3.JAVA获取配置

- 获取所有配置(Properties): `AppConfig.Init(homePath).getApplicationProperties()`

- 按照前缀获取: `getApplicationProperties("oms")`
- AppConfig 依赖 `cole-flower-common-1.0.0-RELEASE.jar`

4.workspace使用配置文件

方式一：使用JAVA代码根据应用根地址获取

```
Properties props = AppConfig.Init(homePath).getApplicationProperties();
String dbType = props.getProperty("oms.type");
String dbUrl = props.getProperty("oms.url");
String name = props.getProperty("oms.username");
String psw = props.getProperty("oms.password");
String minIdle = props.getProperty("oms.minIdle");
String maxIdle = props.getProperty("oms.maxIdle");
int minnum = minIdle==null?1:Integer.valueOf(minIdle);
int maxnum = maxIdle==null?1:Integer.valueOf(maxIdle);

if(dbType.equalsIgnoreCase("mssql")) {
    dbType = "sql";
}
//.....其他的和之前没有区别
//=====分割线:按照前缀获取=====
//参数: (String prefix)
Properties props = AppConfig.Init(homePath).getApplicationProperties("oms");
String dbType = props.getProperty("type");
// ohthers.....
```

方式二：注解从全局的配置bean(`yamlProperties`)获取

```
package com.zzht.component.form.common;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/**
 * 获取form的配置
 * @author : kunhour
 * @version :1.0
 * @since : 2018/5/30 9:24
 */
@Component
public class Configuration {

    @Value("#{yamlProperties['form.type']}")
    private String dbType;

    @Value("#{yamlProperties['form.wsUrl']}")
    private String dbUrl;

    @Value("#{yamlProperties['form.username']}")
    private String dbUsername;

    @Value("#{yamlProperties['form.password']}")
```

```
private String dbPassword;

public String getDbType() {
    return dbType;
}

public String getDbUrl() {
    return dbUrl;
}

public String getDbUsername() {
    return dbUsername;
}

public String getDbPassword() {
    return dbPassword;
}

public void setDbType(String dbType) {
    this.dbType = dbType;
}

public void setDbUrl(String dbUrl) {
    this.dbUrl = dbUrl;
}

public void setDbUsername(String dbUsername) {
    this.dbUsername = dbUsername;
}

public void setDbPassword(String dbPassword) {
    this.dbPassword = dbPassword;
}
}
```

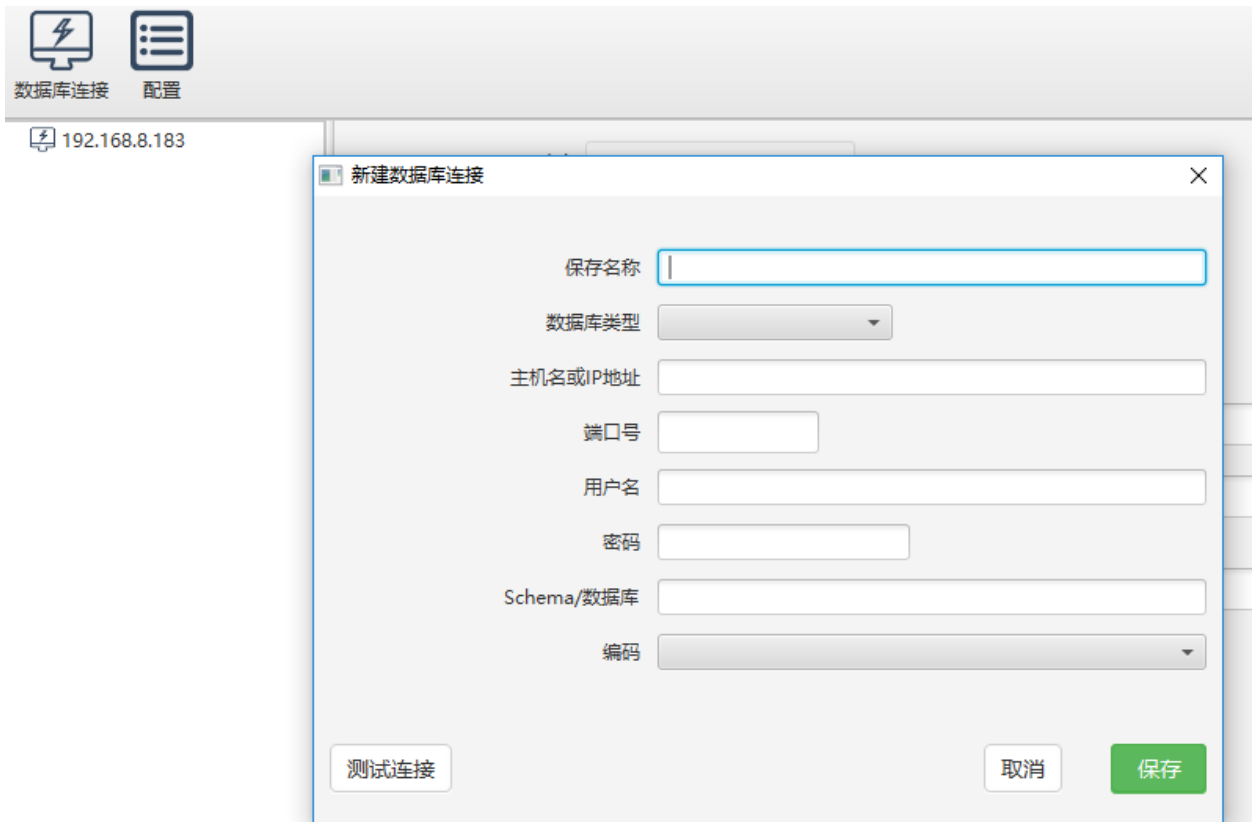
4.Mybatis-generator插件使用

1.GUI工具（推荐）

GUI工具地址: <https://github.com/zizhengzhuan/CityServer/blob/master/GeneratorUI.zip>

下载后解压，运行startup.bat

- 第一步: 首先点击左上角的Connections按钮新建数据库连接，在Connection Name输入框处填入一个好记的名字，比如mysql-local，然后其它字段像连接数据库一下，请参考如下图：



- 填好了所有字段可以先点击“Test Connection”看连接是否成功，如果成功保存连接则主界面左侧会生成一颗数据库连接数。
- 第二步: 双击刚刚保存的连接节点，然后再展开的所有表中双击选择你要生成代码的数据库表，右侧的Table Name和Domain Object Name将会自动填充。
- 第三步: 事先准备好对应数据库的connector的jar包放在你的电脑任意文件夹中，然后在右侧Connector Jar字段右边的Choose按钮，选择你刚刚准备好的jar包。
- 第四步: 选择你的项目所在的目录，例如D:\workspace\example-project或者/Users/youname/workspace/example-prject
- 第五步: 在Model Package输入框中输入你的数据库表对应的Java模型的model的包名，例如：`com.zzht.component.demo.entity`，右侧的Target Folder是你的model在项目中的source目录, 如果你的项目是一个maven项目的，那就是 `src/main/java`，如果是一个普通项目则一般是src。最后注意这个target folder目录一定要存在，否则代码将不会生成。
- 第六步: 在DAO package输入你的generator mapper文件生成的目录，例如 `com.zzht.component.demo.deo`，Target Folder意思同上。
- 第七步: 最后输入你生成的XML的包名，一般我们使用项目包名, 例如 `com.zzht.component.demo.deo`，Target Folder意义同上，如果是maven项目一般是 `src/main/resoures`。
- 第八步: 最后点击Generate按钮，生成代码。如果显示Generation Completed，刚表示代码生成成功。

数据库连接

配置

192.168.8.183

表名

person

Java实体类名

Person

定制列

主键(选填)

primary key, such as id

项目所在目录

D:\workspace\example

选择

实体类名包名

com.zzht.component.demo

存放目录

src/main/java

接口包名

com.zzht.component.demo

存放目录

src/main/java

自定义接口名称(选填)

PersonDAO

映射XML文件包名

com.zzht.component.demo

存放目录

src/main/resources

☒ 分页

☒ 生成实体域注释(来自表注释)

☒ 生成toString/hashCode/equals方法

☐ 生成JPA注解

☐ 使用实际的列名

代码生成

保存配置

2.ideal插件

1.在object模块的pom.xml中添加插件:

```
<plugins>
  <plugin>
    <!--Mybatis-generator插件,用于自动生成Mapper和POJO-->
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-maven-plugin</artifactId>
    <version>1.3.2</version>
    <configuration>
      <!--配置文件的位置-->
      <configurationFile>src/main/resources/mybatis-generator-config.xml</configurationFile>
      <verbose>true</verbose>
      <overwrite>true</overwrite>
    </configuration>
    <executions>
      <execution>
        <id>Generate MyBatis Artifacts</id>
        <goals>
          <goal>generate</goal>
        </goals>
      </execution>
    </executions>
    <dependencies>
      <dependency>
        <groupId>org.mybatis.generator</groupId>
        <artifactId>mybatis-generator-core</artifactId>
        <version>1.3.2</version>
      </dependency>
    </dependencies>
  </plugin>
</plugins>
```

```
</plugins>
```

2. 配置 mybatis-generator-config.xml

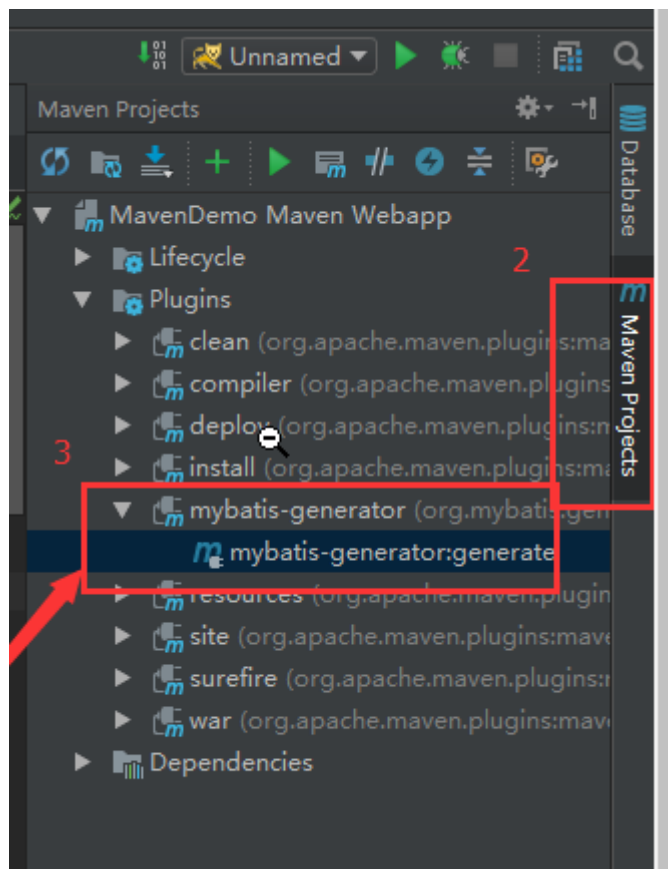
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration
    PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
    "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">

<generatorConfiguration>
    <!--mysql 连接数据库jar 这里选择自己本地位置-->
    <classPathEntry location="D:/mysql-connector-java-5.1.20-bin.jar" />
    <context id="testTables" targetRuntime="MyBatis3">
        <commentGenerator>
            <!-- 是否去除自动生成的注释 true: 是 : false:否 -->
            <property name="suppressAllComments" value="true" />
        </commentGenerator>
        <!--数据库连接的信息：驱动类、连接地址、用户名、密码 -->
        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
            connectionURL="jdbc:mysql://localhost:3306/ecps" userId="root"
            password="root">
        </jdbcConnection>
        <!-- 默认false, 把JDBC DECIMAL 和 NUMERIC 类型解析为 Integer, 为 true时把JDBC DECIMAL 和
            NUMERIC 类型解析为java.math.BigDecimal -->
        <javaTypeResolver>
            <property name="forceBigDecimals" value="false" />
        </javaTypeResolver>

        <!-- targetProject:生成PO类的位置 -->
        <javaModelGenerator targetPackage="com.ecps.seckill.pojo"
            targetProject="src/main/java">
            <!-- enableSubPackages:是否让schema作为包的后缀 -->
            <property name="enableSubPackages" value="false" />
            <!-- 从数据库返回的值被清理前后的空格 -->
            <property name="trimStrings" value="true" />
        </javaModelGenerator>
        <!-- targetProject:mapper映射文件生成的位置
            如果maven工程只是单独的一个工程, targetProject="src/main/java"
            若果maven工程是分模块的工程, targetProject="所属模块的名称", 例如:
            targetProject="ecps-manager-mapper", 下同-->
        <sqlMapGenerator targetPackage="com.ecps.seckill.mapper"
            targetProject="src/main/java">
            <!-- enableSubPackages:是否让schema作为包的后缀 -->
            <property name="enableSubPackages" value="false" />
        </sqlMapGenerator>
        <!-- targetPackage: mapper接口生成的位置 -->
        <javaClientGenerator type="XMLMAPPER"
            targetPackage="com.ecps.seckill.mapper"
            targetProject="src/main/java">
            <!-- enableSubPackages:是否让schema作为包的后缀 -->
            <property name="enableSubPackages" value="false" />
        </javaClientGenerator>
    </context>
</generatorConfiguration>
```

```
<!-- 指定数据库表 -->
<table schema="" tableName="seckill"></table>
<table schema="" tableName="success_killed"></table>
</context>
</generatorConfiguration>
```

3.在 idea 的右侧栏点击Maven,选中添加的Mybatis-generator插件并运



五、开发规范

开发规约

citysever的开发规范经过评估, 决定采用阿里巴巴的 p3c规约。《阿里巴巴Java开发手册》涵盖编程规约、单元测试规约、异常日志规约、MySQL规约、工程规约、安全规约等, 使用这个规范希望能够帮助开发团队在Java开发上更高效、容错、有协作性, 提高代码质量, 降低项目维护成本。

p3c地址: <https://github.com/alibaba/p3c>

p3c完整PDF文档: [在线地址](#)

插件安装

参照p3c插件安装方法: <https://github.com/alibaba/p3c/tree/master/idea-plugin>

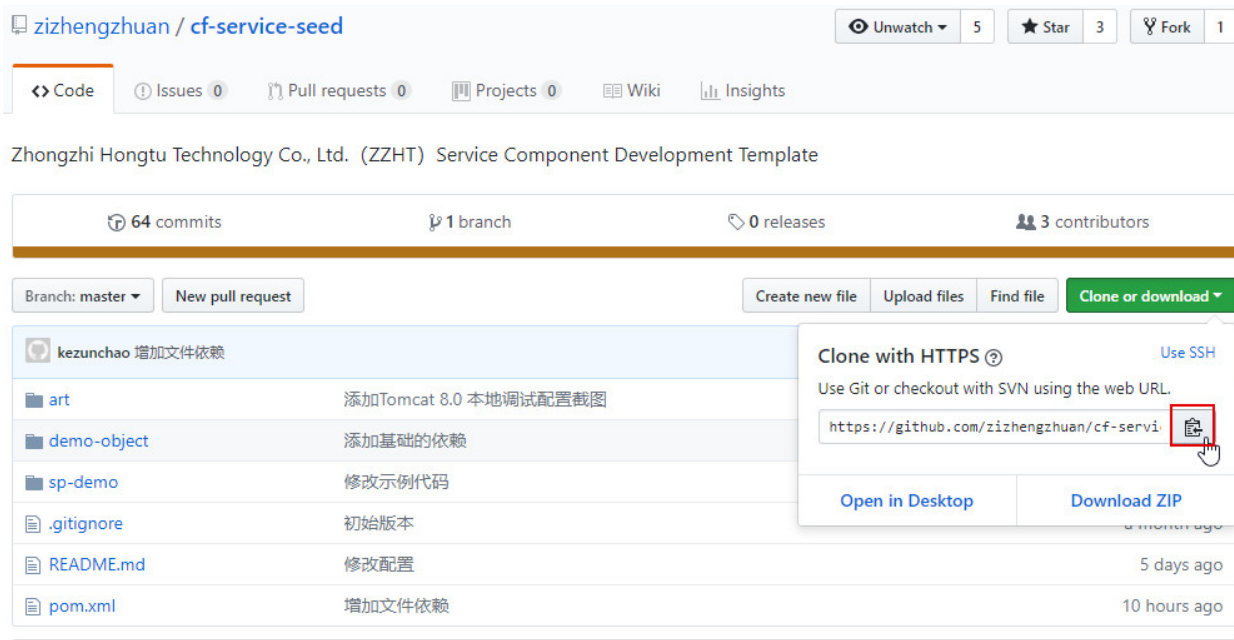
六、环境搭建与配置

务必保证[基础环境](#)中的提到的环境都安装完成并做好了相关的配置。

Step1:github克隆种子项目

种子项目地址: <https://github.com/zizhengzhuan/cf-service-seed>

1. 复制项目地址



zizhengzhuan / cf-service-seed

Unwatch 5 Star 3 Fork 1

<> Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

Zhongzhi Hongtu Technology Co., Ltd. (ZZHT) Service Component Development Template

64 commits 1 branch 0 releases 3 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

Clone with HTTPS Use SSH

Use Git or checkout with SVN using the web URL.

<https://github.com/zizhengzhuan/cf-service-seed>

Open in Desktop Download ZIP

art	添加Tomcat 8.0 本地调试配置截图
demo-object	添加基础的依赖
sp-demo	修改示例代码
.gitignore	初始版本
README.md	修改配置
pom.xml	增加文件依赖

2. 打开Git的Bash



3. clone项目到本地

```
kunhour@kunhour MINGW64 /d/WXAPP
$ git clone https://github.com/zizhengzhuan/cf-service-seed.git
Cloning into 'cf-service-seed'...
remote: Counting objects: 643, done.
remote: Compressing objects: 100% (89/89), done.
remote: Total 643 (delta 28), reused 156 (delta 25), pack-reused 473
Receiving objects: 100% (643/643), 481.35 KiB | 68.00 KiB/s, done.
Resolving deltas: 100% (191/191), done.

kunhour@kunhour MINGW64 /d/WXAPP
$ cd cf-service-seed/
```

```
MINGW64:/d/WXAPP/cf-service-seed
kunhour@kunhour MINGW64 /d/WXAPP
$ git clone https://github.com/zizhengzhuan/cf-service-seed.git
Cloning into 'cf-service-seed'...
remote: Counting objects: 643, done.
remote: Compressing objects: 100% (89/89), done.
remote: Total 643 (delta 28), reused 156 (delta 25), pack-reused 473
Receiving objects: 100% (643/643), 481.35 KiB | 68.00 KiB/s, done.
Resolving deltas: 100% (191/191), done.

kunhour@kunhour MINGW64 /d/WXAPP
$ cd cf-service-seed/

kunhour@kunhour MINGW64 /d/WXAPP/cf-service-seed (master)
$
```

Step2:cityserver基础包

CityServer地址: <https://github.com/zizhengzhuan/CityServer>

将cityserver的包解压到tomcat的webapps下, 作为一个应用发布。cityserver的结构说明:

- conf -- 配置文件
- application.yml -- 项目全局配置文件
- services -- 服务包的目录
- sp-xxx.jar
-
- lib -- 依赖包地址
- xxxxx.jar
- WEB-INF -- 应用的入口
- beans.xml
- web.xml

Step3:配置私有maven仓库

参照 [《maven私服使用-1.0.2.pdf》](#)

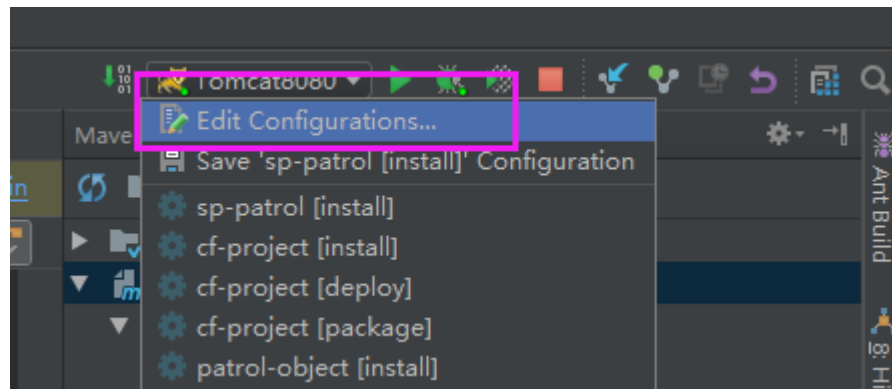
Step4:maven打包路径

修改pom.xml中 `output.dir` 路径.

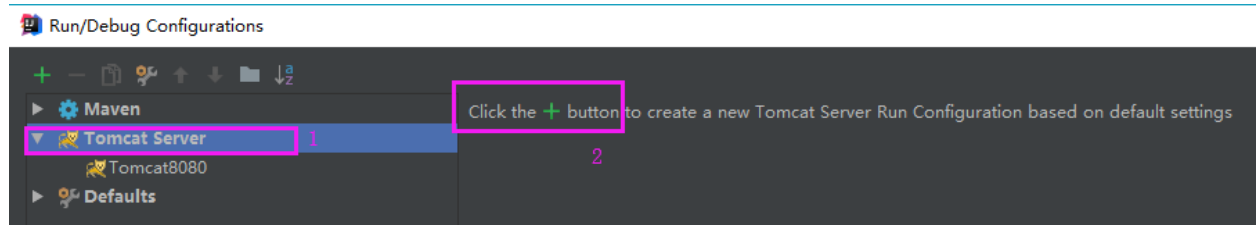
```
<dbcp.version>2.1.1</dbcp.version>
<!-- jar包输出路径, 默认是${project.build.directory}, 也就是target
    也可以自定义输出路径, 如: D:/tomcat85_30/webapps/ServiceEngine/WEB-INF
-->
<!--<output.basedir>${project.build.directory}</output.basedir>-->
<output.basedir>d:/tomcat85_30/webapps/ServiceEngine/WEB-INF</output.basedir>
```

Step5:IDEA下代码调试

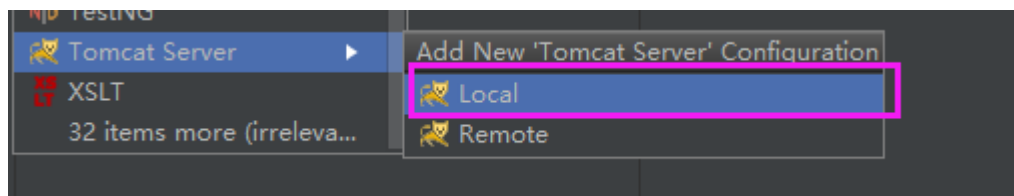
1. IDEA调试工具栏中编辑调试配置，选择Edit Configurations



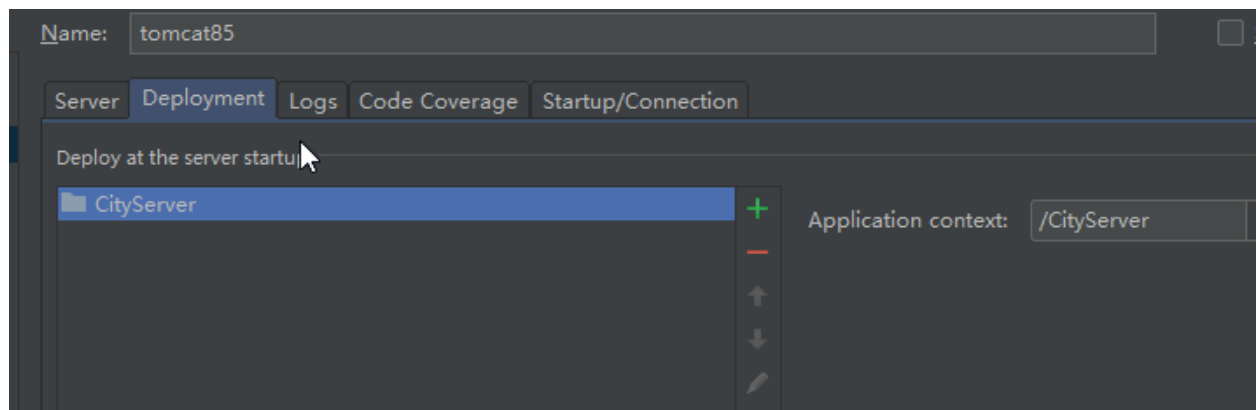
2. 选择Tomcat Server，点击绿色 + 号



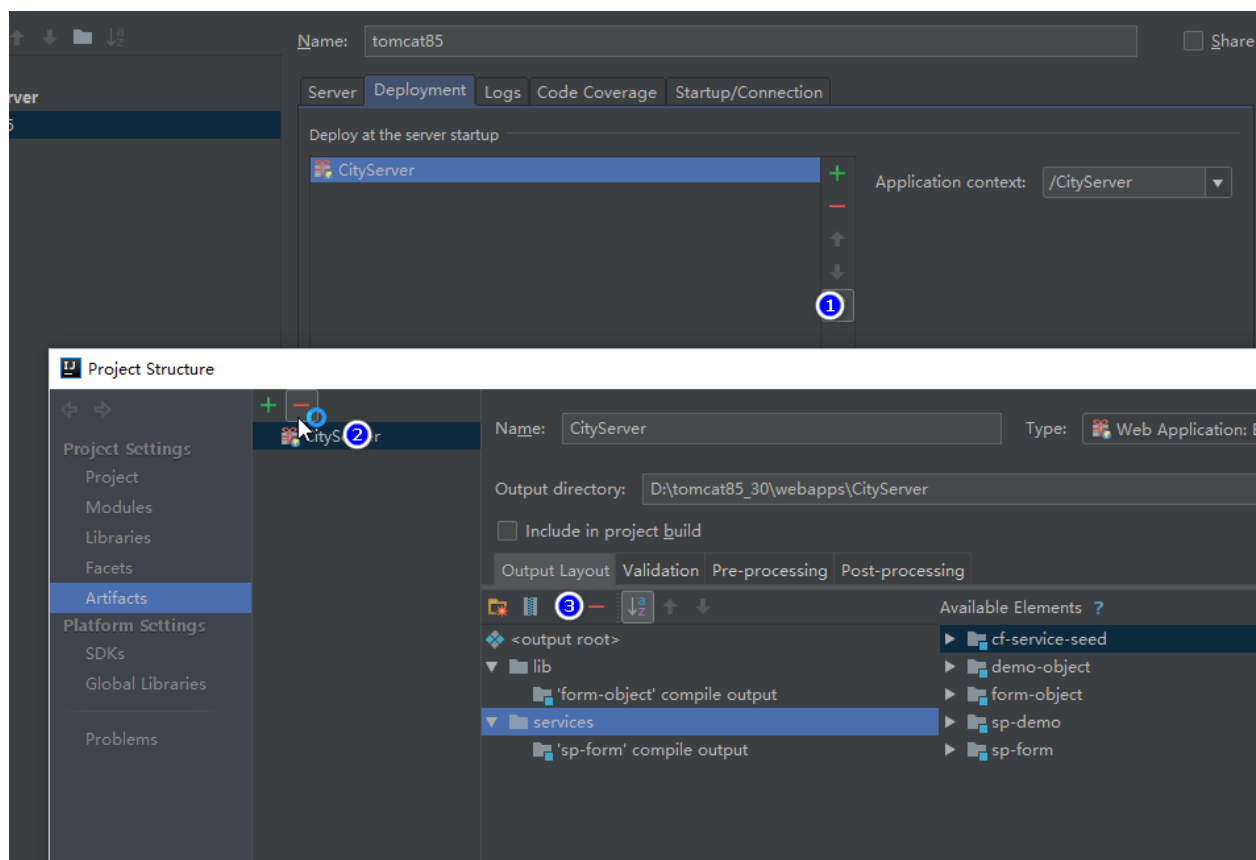
3. Tomcat IDEA下有 local 和 remote 两种模式，选择local



4. 填写Application Context，目前约定使用cityserver



5. 添加编译关联



6. 验证

This XML file does not appear to have any style information associated with it. The document

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://wadl.dev.java.net/2009/02" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <grammars/>
  <resources base="http://localhost:8080/CityServer/rest/personService">
    <resource path="/person">
      <resource path="/getPerson">
        <method name="GET">
          <response>
            <representation mediaType="application/json"/>
            <representation mediaType="application/xml"/>
            <representation mediaType="application/javascript"/>
            <representation mediaType="text/html"/>
          </response>
        </method>
      </resource>
    <resource path="/queryPerson">
      <method name="POST">
        <doc>查询人员列表</doc>
        <request>
          <representation mediaType="application/xml">
            <param name="userId" style="query" type="xs:string">
              <doc>用户ID</doc>
            </param>
          </representation>
        </request>
        <response>
          <representation mediaType="application/json"/>
          <representation mediaType="application/xml"/>
          <representation mediaType="application/javascript"/>
          <representation mediaType="text/html"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

附录

附录一 Git介绍

1. 分布式管理更可靠

git是分布式的，svn是集中式的。什么是集中式？以git举例来说，有一个远程服务器来托管代码，同时本地机器也是一个服务器。优点就是当远程服务器出现问题时，可以将本地服务器推送到远程，这样远程服务器不会丢失任何东西。所以git提交代码分为两部分，先要执行commit命令，将代码提交到本地服务器，然后通过push命令将本地服务器代码推送到远程服务器。

2. 更简单的分支操作

更简单的分支操作。分支的作用是方便多个团队协同合作。默认分支为master分支。我们一般讲master分支做为稳定分支，在各分支开发完成，测试通过，将代码合并到master分支，在master分支出版本。git合并分支很简单，运行一下pull命令即可，pull命令等同于fetch+merge。当有新的任务或者临时修改BUG可以快速的切换代码。

关于git的操作教程：[git教程](#)

附录二 Maven介绍

现象:在实际开发或者学习中你可能遇到过下面的这些问题

- 同样的代码，为什么在别人那里可以正常编译和运行，拷贝到我本地之后就报错了呢？
- 在使用其他技术的时候需要导入一些jar包，有可能你导入的这些jar包又依赖于另一个技术的jar包，你还需要导入这些jar包。
- 随着项目中使用技术的增多，项目中的jar包也越来越多，这样就会可能会存在一些jar包的冗余。
- 你自己编写了一款jar包，在公司内部有多个项目使用了这块jar包，倘若某天你发现该jar包存在bug，修正后你需要把这个jar包更新到所有相关的项目中。

1.什么是Maven

Maven是Apache旗下一款开源自动化的项目管理工具，它使用java语言编写，因此Maven是一款跨平台的项目管理工具。Maven主要功能：

- 项目构建
在实际开发中，不仅仅是写完代码项目就算完成了，后面还有一些诸如：编译，打包，部署等工作要做，这些工作都可以使用maven来完成。
- 依赖管理
说的简单一点就是对jar包的管理，开发者不用再手动的下载所需要的jar包，而是将想要的jar包通过配置一个叫做pom.xml的文件中，之后maven会自动的下载相关的jar包。

2.Maven的安装

1. 下载

你可以通过maven的官网下载：<http://maven.apache.org/>

注意：在安装前请确保机器上已经安装了jdk，并且jdk的版本最好是7以上的。

2. 解压

将maven解压，解压的目录中最好不要含有空格、中文或者其他特殊符号。

解压后目录如下：

bin：maven的命令

boot：含有一个类加载器，通常情况下不使用

conf：maven的配置文件

lib：maven的jar包，这里是maven运行时需要的jar包，并非用户在项目中的jar包

3. 配置maven环境变量

添加一个环境变量：

变量名：MAVEN_HOME

变量值：填写你的maven的解压目录，我本地的是：D:\apache-maven-3.5.2

之后在path中添加;%MAVEN_HOME%\bin

注意前面使用“;”与其他值隔开。

4. 验证是否配置成功

在cmd中输入mvn -v

如果显示出当前mvn的版本号，则说明maven的安装成功

3.maven配置

安装完成后分为 **默认配置** 和 **用户配置** **默认配置** 在maven的安装目录的conf文件夹下有一个settings.xml文件，打开后，可以看到有一项：

```
<!-- localRepository
  | The path to the local repository maven will use to store artifacts.
  |
  | Default: ${user.home}/.m2/repository
<localRepository>/path/to/local/repo</localRepository>
-->
```

该配置是默认注释掉的，其意思是默认情况下，maven仓库的目录地址是在你的 *user.home/.m2/repository* 文件中，我的地址是： *C:\Users\Administrator.m2\repository*。{user.home}表示的是你本地电脑的用户名。你可以在下面自己写一个localRepository标签来为其指定一个目录。maven仓库目录，就是maven将你项目中所用到的jar包下载的目录地址。

用户配置 可能在你的Windows操作系统中有多个用户，你可以为每个用户设定一个该用户自己的maven仓库地址，即在该用户的.m2文件夹下复制一份settings.xml文件，然后在文件中指定其仓库地址。

如果设置了用户配置，则默认配置会失效。