## 管控平台3.0开发API手册

|  |  |
| --- | --- |
| 文档编号 | -- |
| 软件名称 | 管控平台3.0 |
| 相关系统 | -- |
| 参与人员 | 李磊 |
| 编写人员 | 李磊 |
| 版本号 | 1.0 |
| 编写时间 | 2015-08-27 |

### 引言

本文主要目的是引导服务开发人员做系统功能开发。文档主要关注JAVA环境的配置和讲解，让想深入的开发人员有更好的发挥空间。

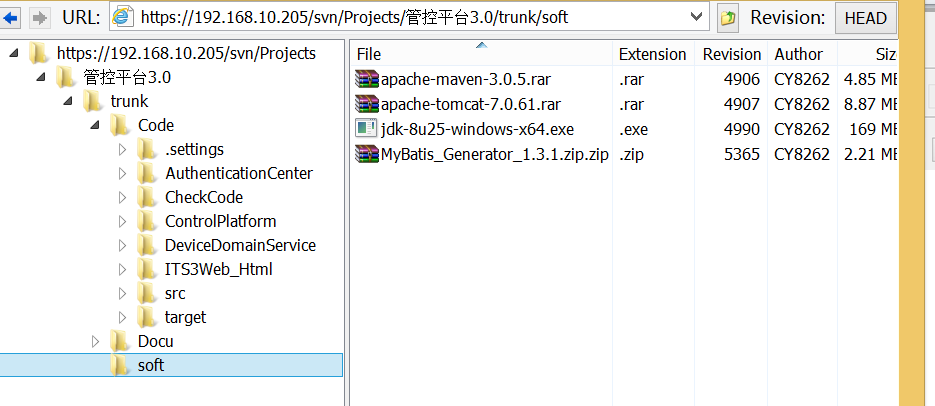
### 插件安装

#### 软件版本

|  |  |
| --- | --- |
| Eclipse | 4.4(或以上) |
| JDK | 1.8.0\_25(或以上) |
| TOMCAT | apache-tomcat-7.0.61(+) |
| maven | 3.0.5(+) |

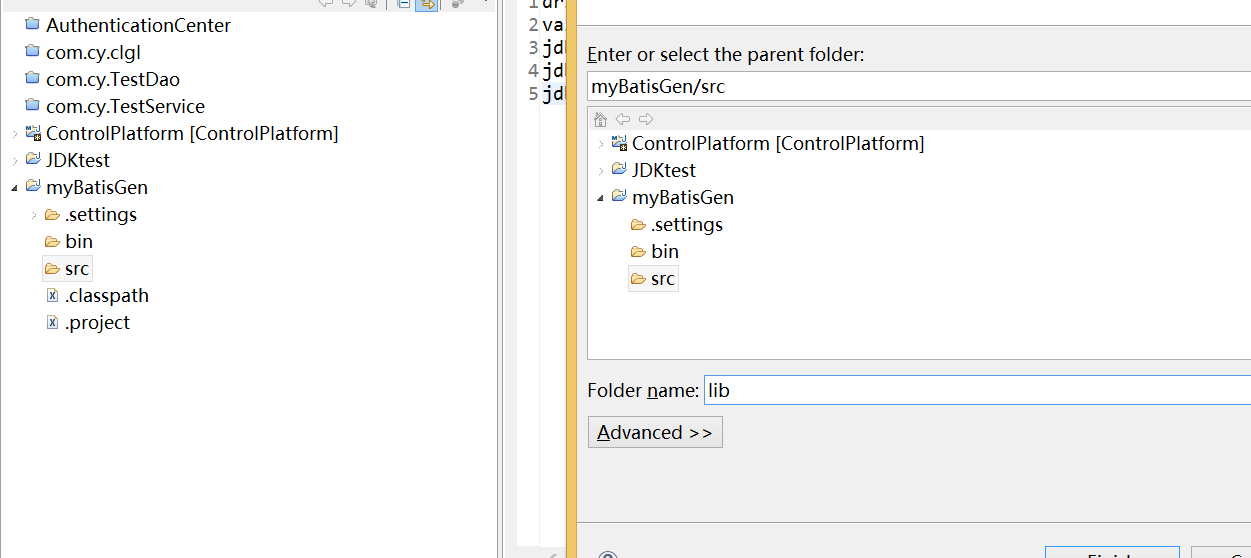
#### Mybatis插件

1. 从SVN 上下载插件:

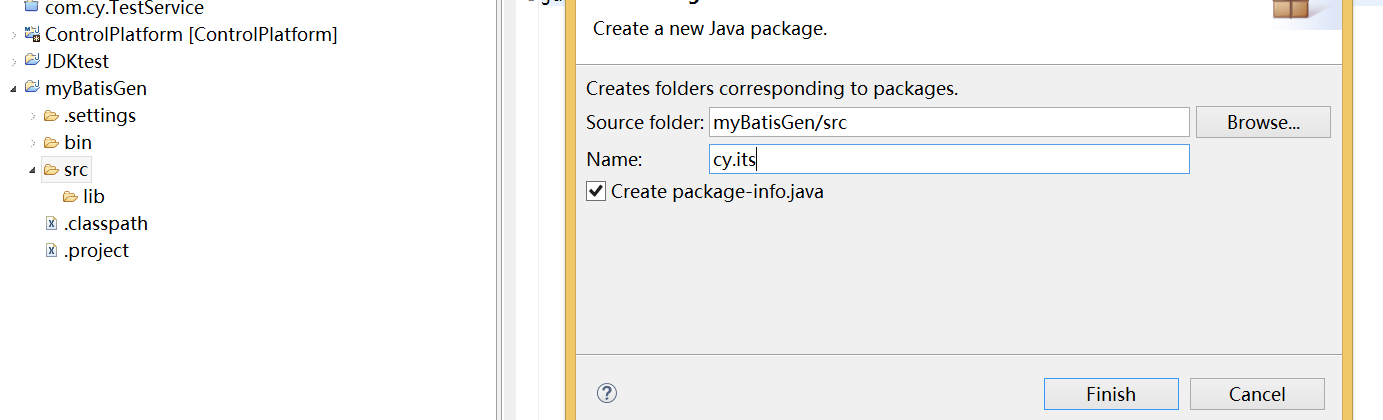


解压到任意目录下，复制eclipse文件夹，替换 eclipse 安装目录eclipse文件夹，重启eclipse。

1. 在eclipse下新建Java工程（名字任意），如: myBatisGen
2. 工程项目新建LIB目录:



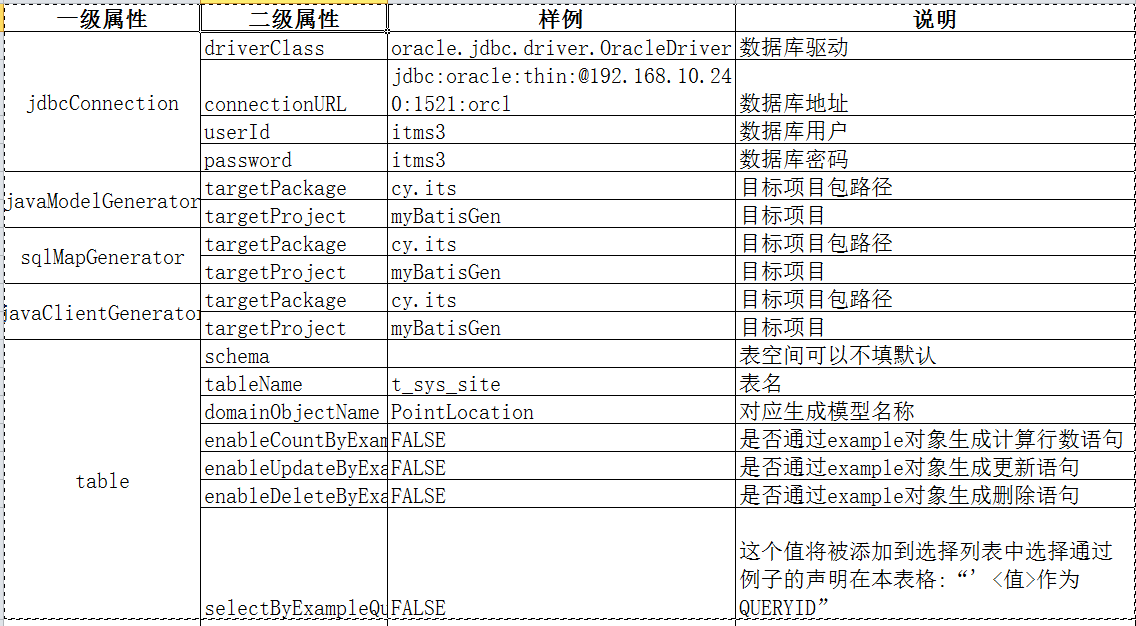
1. 工程项目新建代码包目录:



1. 复制generatorConfig.xml到项目跟目录下，配置如下:

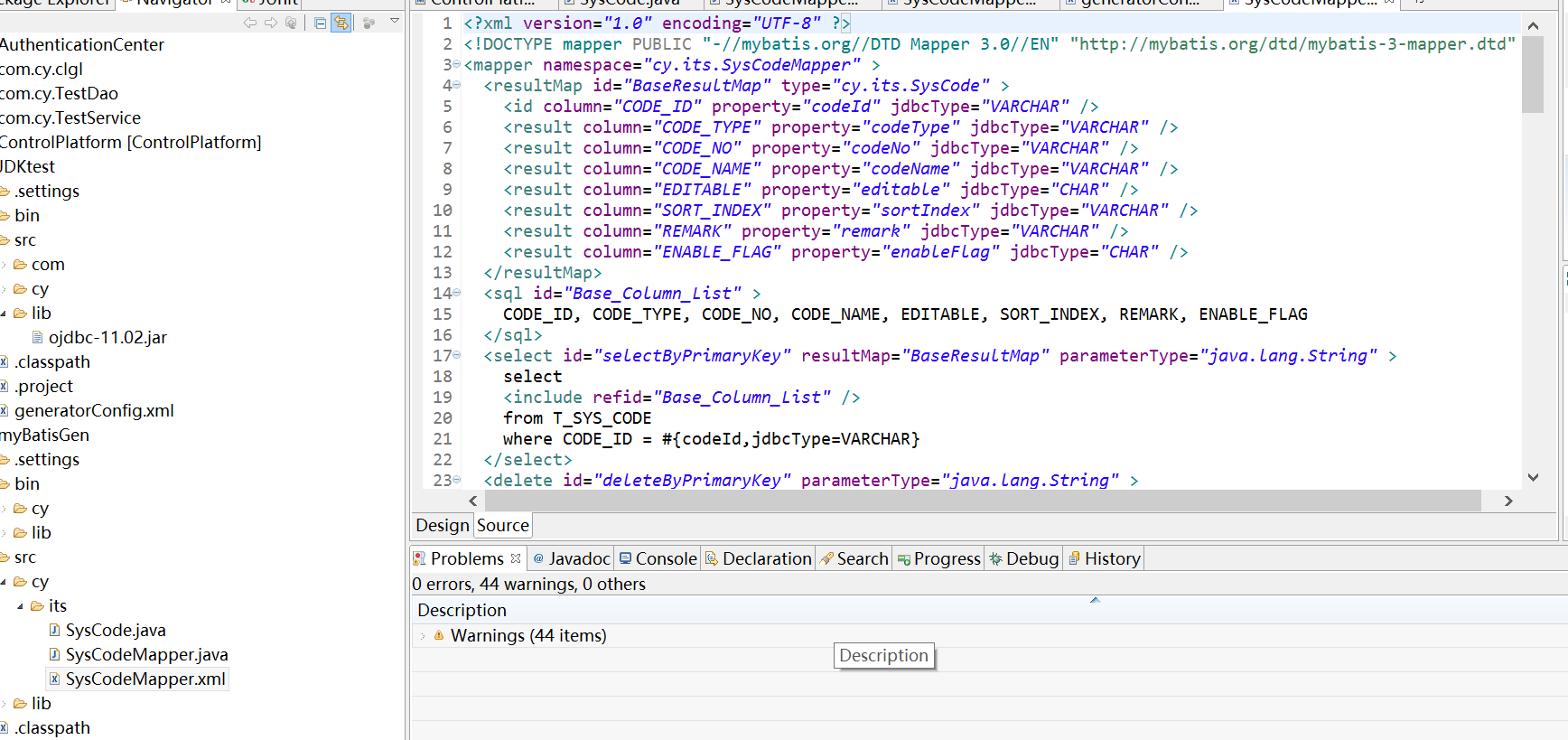


1. 把ojdbc-11.02.jar复制到项目LIB下。
2. generatorConfig.xml配置文件说明:



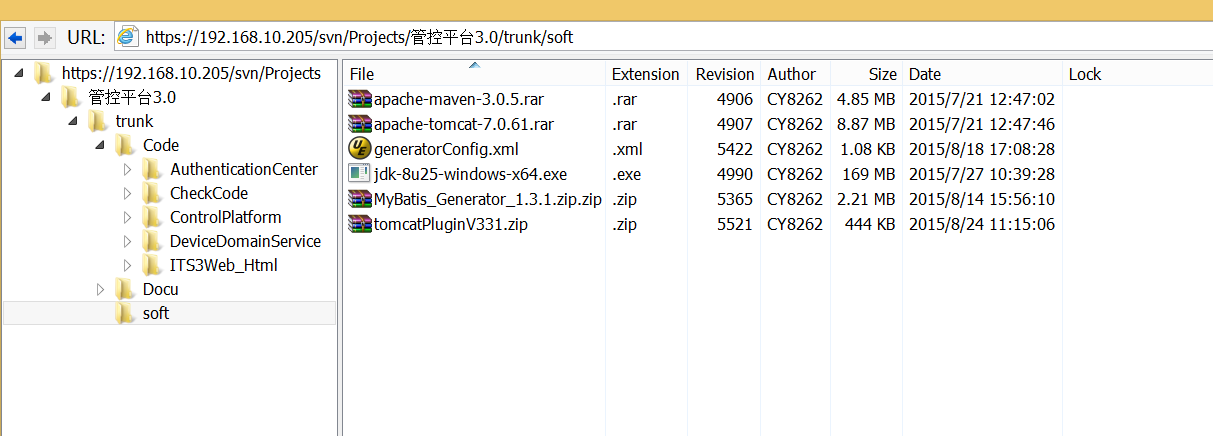
1. 选中generatorConfig.xml右键，选择：

generate Mybatis/ibatis Artifacts，生成代码SysCode.java、SysCodeMapper.java、SysCodeMapper.xml如下:

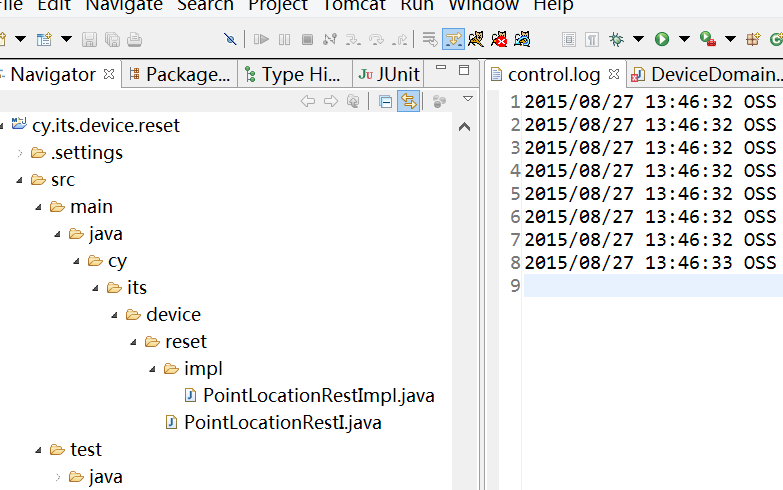


#### tomcat插件

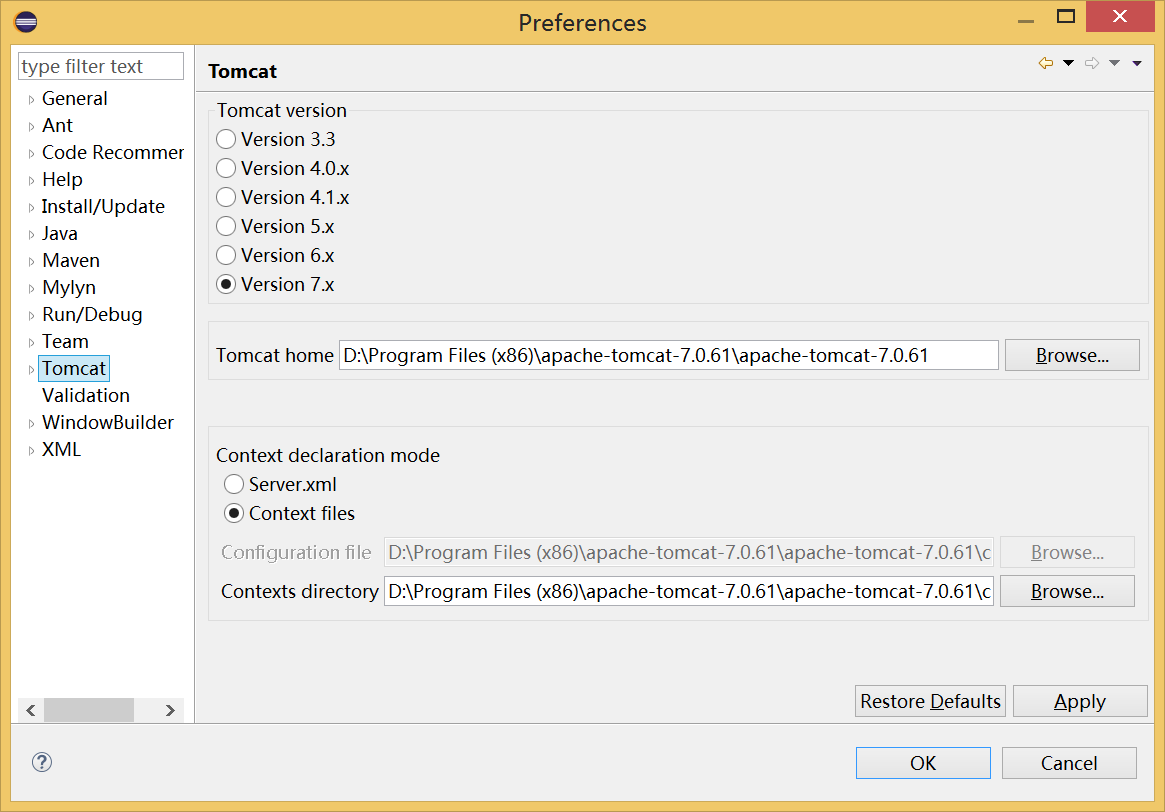
1. 下载TOMCAT插件：
2. 地址:https://192.168.10.205/svn/Projects/管控平台3.0/trunk/soft



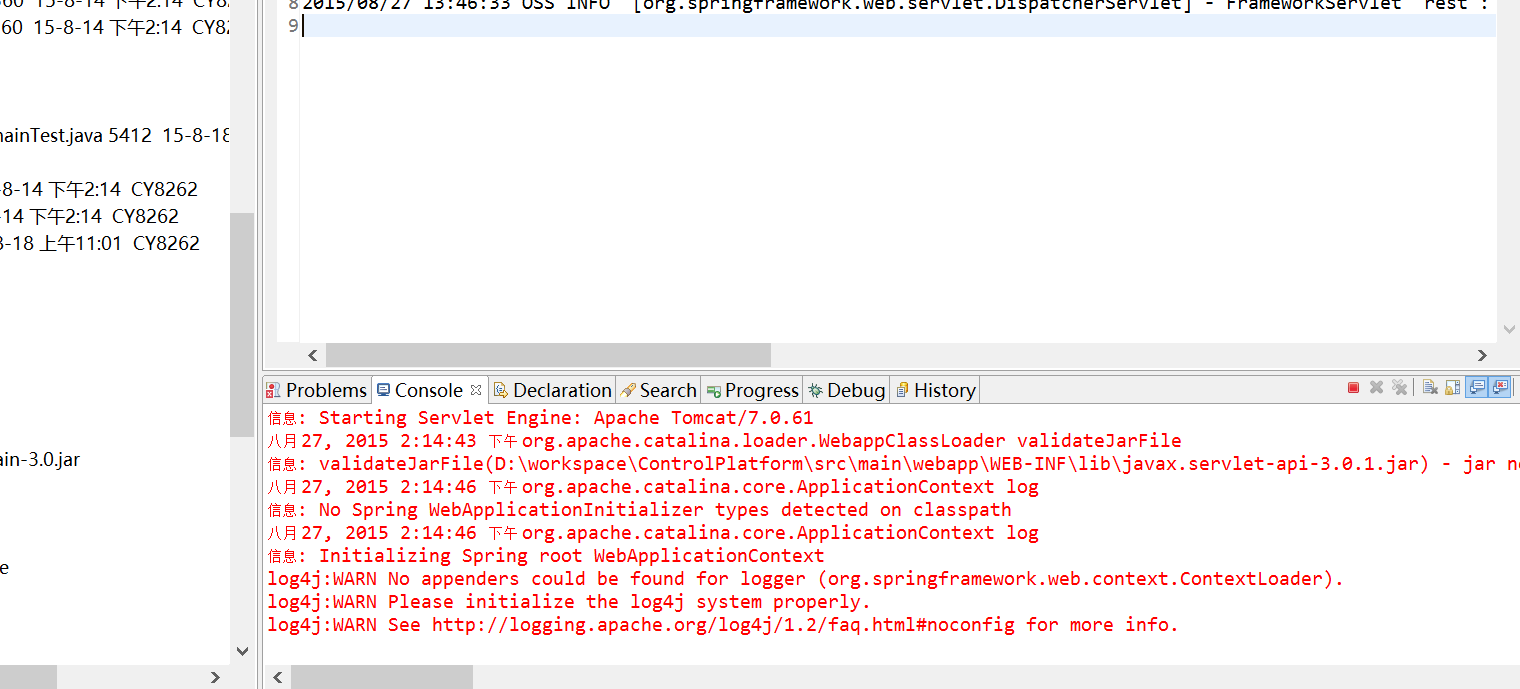
1. 解压得到文件com.sysdeo.eclipse.tomcat\_3.3.1
2. 放入安装eclipse plugins下
3. 重启eclipse显示如下三个猫型图标，说明安装成功:



1. 配置window->preference->tomcat找到tomcat解压的路径:



1. 点击启动，console展示如下，则配置成功。如点击不起作用，要设置eclipse以管理员的身份运行即可。

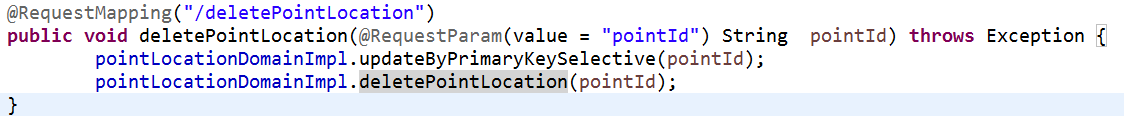


### 开发API说明

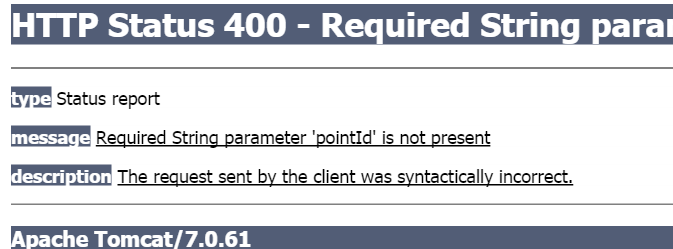
#### 参数传递

* 有参数接口，此方式适合参数简单而且数量少的接口

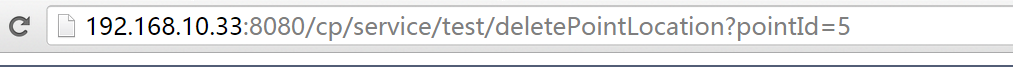
Resetful接口接收参数方式如下:



其中：@RequestParam 表示需要接收参数，参数名称为: pointId，如果参数没传，则会报：

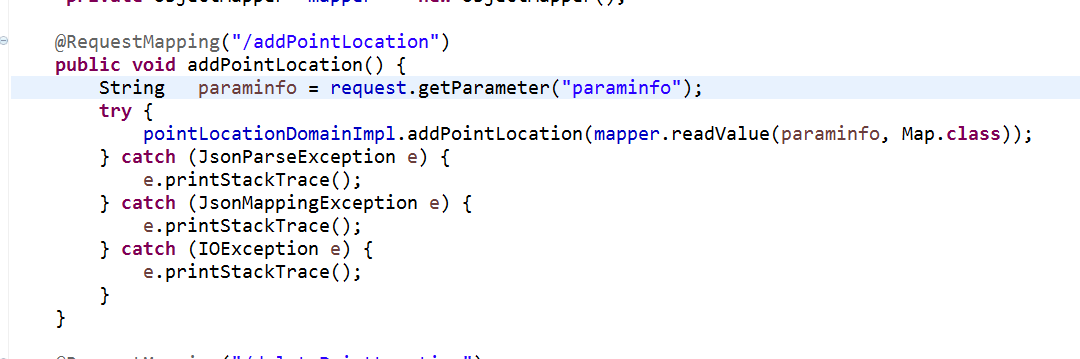


正确地址如下:



* 无参数接口，此方式适合无参数或是复杂多样的参数传递

Resetful无参数接口，即接口方法无需参数:



无参接口不代表不需要参数，复杂的参数传递需要用到request，必须在调用之前被初始化，一般采用spring 注入的方式:



通过request可以获得你所需要的所有参数，如果有复杂的参数需要转化为DTO的模型，可以采用以下的方式:

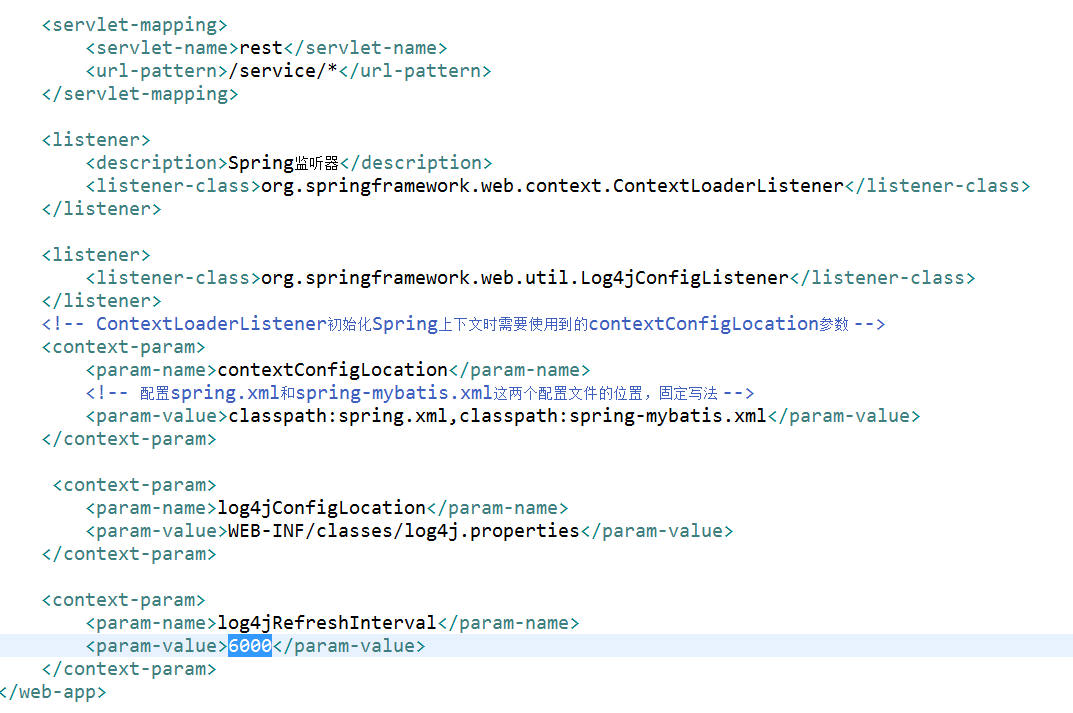


把参数封装到类中，而不需要做太多的操作。

#### WEB配置文件

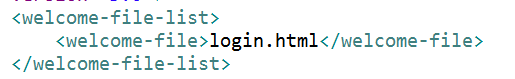
项目中web.xml配置如下:





WEB.XML配置文件做了5件事情:

1. 指定打开项目默认界面



意思是，如果浏览器访问工程项目而没有指定界面，则默认访问:

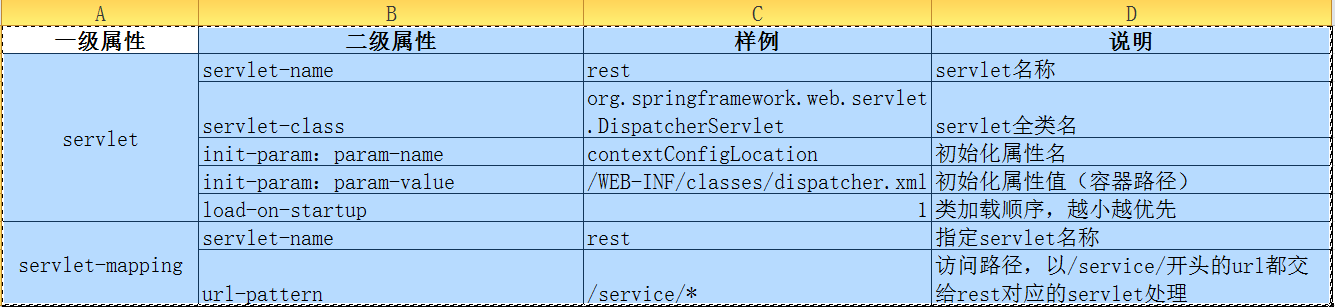
Login.html界面。

1. 启动resetful容器





属性说明:

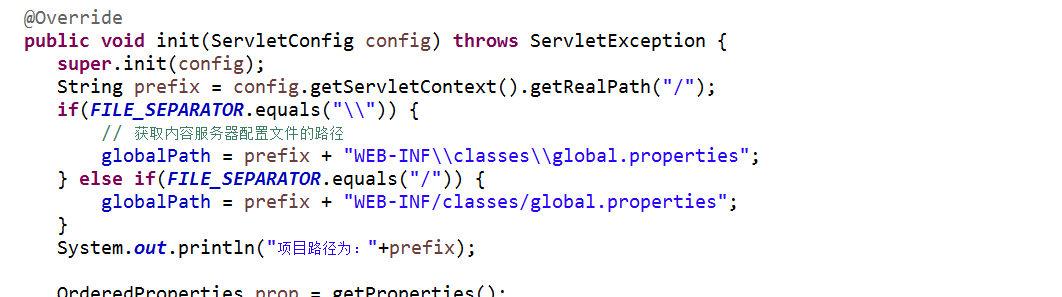


1. 项目启动，初始化的servlet

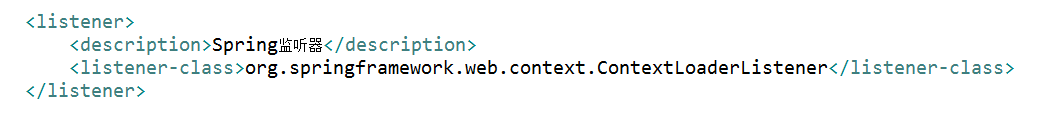


项目启动时，执行initservlet 中init 方法，主要的作用是：

可以在项目启动，初始化一些必要的参数。代码展示如下:

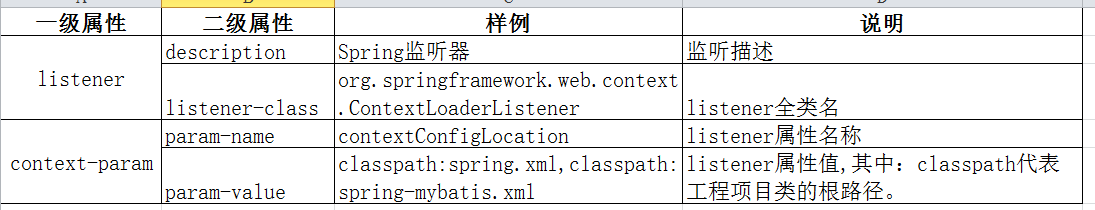


1. 启动spring容器

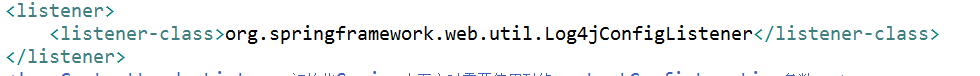


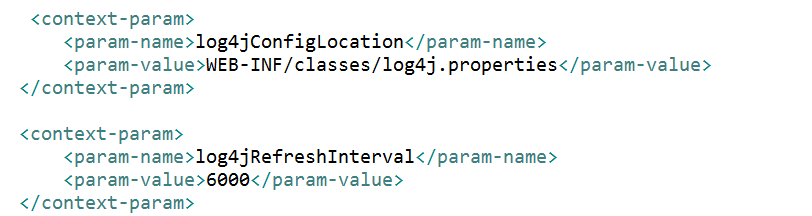


属性说明:



1. 指定日志配置





属性参考4的说明:

log4jRefreshInterval：表示 开一条watchdog线程每6秒扫描一下配置文件的变化，可根据实际情况调整大小。

Log4j.properties文件如下:



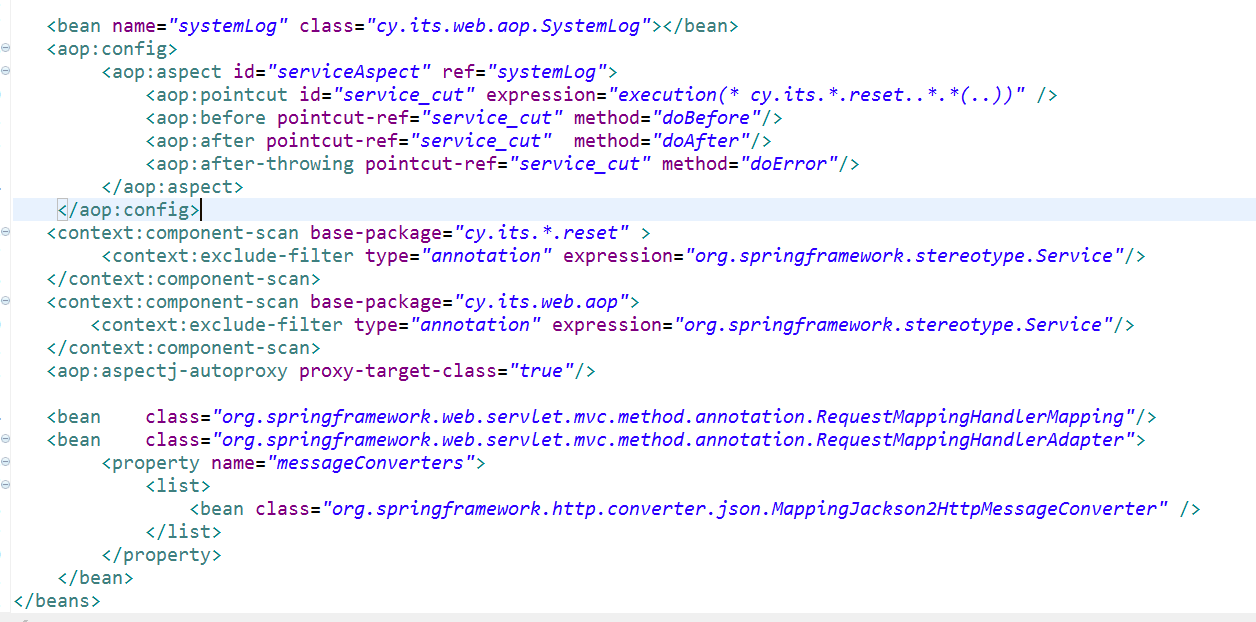
各个属性已经做了说明，但是注意其中一项：

${webapp.root}：属性是运行时动态变化的，只要在项目初始化时才会替被替换掉，其路径为工程项目根路径。

#### Dispatcher.xml配置文件

Dispathcer.xml是spring mvc 核心配置文件，DispatcherServlet是前端控制器设计模式的实现，提供Spring Web MVC的集中访问点，而且负责职责的分派，且与Spring IoC容器无缝集成。

工程项目配置如下:



其中AOP:CONFIG会在后面做讲解，此处不再讲述。

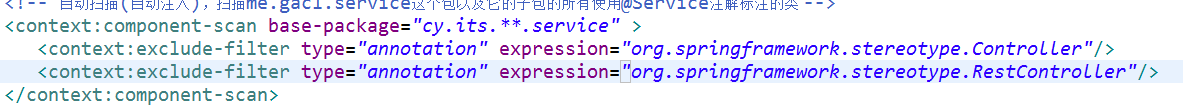
context:component-scan配置如下:



该属性是一个扫描器，扫描指定包下所有的类，有注解@ RestController、@ Controller、@ Component 注解的类会被注册到spring框架里面管理。其中:

这属性意思是：排除@service 注解类，即以@service注解的类不再被注册，防止重复实例化@ service 注解的类。

项目开发过程中要严格遵守开发规范，@service 注解的类不能出现在spring mvc 层，同样@ RestController、@ Controller 注解的类不能出现在业务层中。为了防止重复，业务层也对@ RestController、@ Controller进行排除:



aspectj-autoproxy指定代理方式，



proxy-target-class属性，默认为false，表示使用jdk动态代理织入增强，当配为<aop:aspectj-autoproxy  poxy-target-class="true"/>时，表示使用CGLib动态代理技术织入增强。

注意: 使用jdk动态代理方式AOP 在MVC 层不起作用。

#### SRING.xml配置文件

SPRIN.XML样例如下:



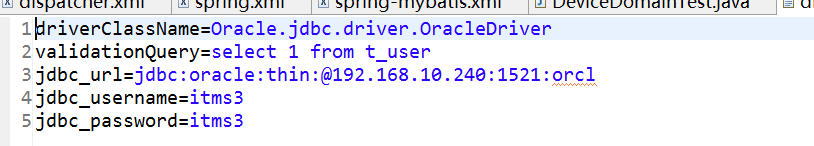
配置文件做了两件事

1. 把资源配置信息，注入到spring框架中。



Classpath：项目类根路径

dbconfig.properties数据库相关信息，展示如下:



1. 配置扫描器，对该包下所有的类进行扫描。



其中: cy.its.\*\*.service包里\*\*是通配符，可匹配一个或多个包，比如:device或是device.domain等，如下两种都是可以匹配：

cy.its.device.service和cy.its.device.domain.service

context:exclude-filter：是排除注解@Controller，@RestController，即这两种注解的类在上述包下不会被注入SRING容器中。

#### SRING-MYBATIS.XML配置文件

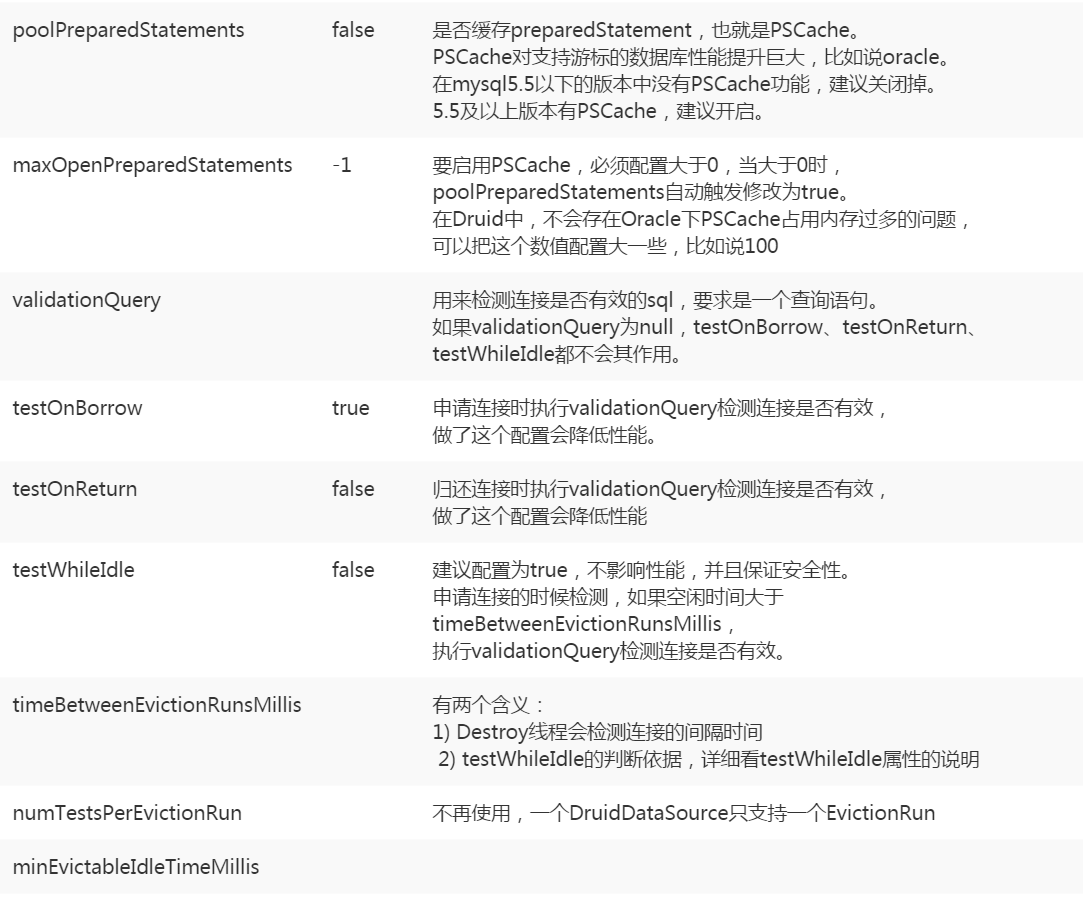
主要的作用：1、数据库连接管理 2、spring集成mybatis 3、事务管理。

* 数据库配置：固定配置方式，基本不需要修改。展示如下:



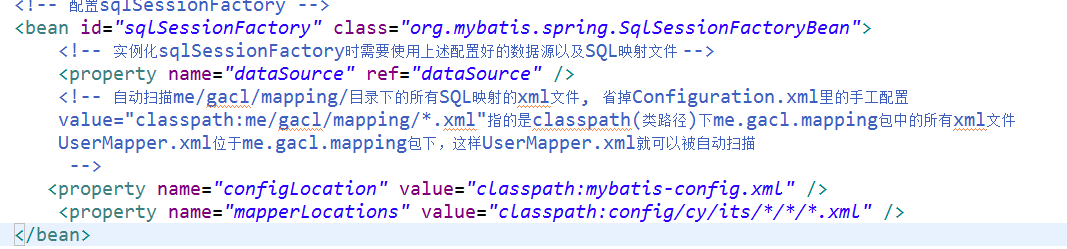
详细说明，有兴趣的可深入研究:

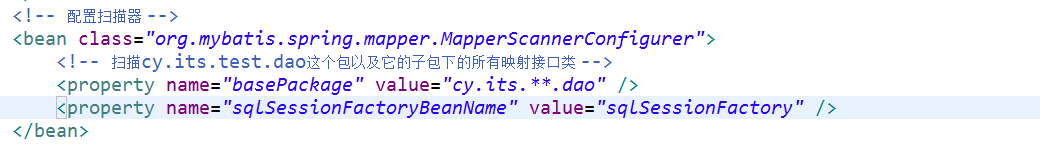






* Spring集成mybatis

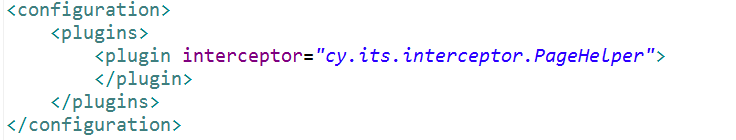




在定义SqlSessionFactoryBean的时候，dataSource属性是必须指定的，它表示用于连接数据库的数据源。当然，我们也可以指定一些其他的属性，下面简单列举几个：  
l  mapperLocations：它表示我们的Mapper文件存放的位置，当我们的Mapper文件跟对应的Mapper接口处于同一位置的时候可以不用指定该属性的值。

l  configLocation：用于指定Mybatis的配置文件位置。如果指定了该属性，那么会以该配置文件的内容作为配置信息构建对应的SqlSessionFactoryBuilder，但是后续属性指定的内容会覆盖该配置文件里面指定的对应内容。  
l  typeAliasesPackage：它一般对应我们的实体类所在的包，这个时候会自动取对应包中不包括包名的简单类名作为包括包名的别名。多个package之间可以用逗号或者分号等来进行分隔。  
l  typeAliases：数组类型，用来指定别名的。指定了这个属性后，Mybatis会把这个类型的短名称作为这个类型的别名，前提是该类上没有标注@Alias注解，否则将使用该注解对应的值作为此种类型的别名。

l  plugins：数组类型，用来指定Mybatis的Interceptor。如下图:



上面pageHelper 是分页的Interceptor。

l  typeHandlersPackage：用来指定TypeHandler所在的包，如果指定了该属性，SqlSessionFactoryBean会自动把该包下面的类注册为对应的TypeHandler。多个package之间可以用逗号或者分号等来进行分隔。  
l  typeHandlers：数组类型，表示TypeHandler。

* Spring事务处理

Spring事务处理，采用AOP方式，专门在3.7章讲解，请参阅

#### SRING IOC说明

Ioc—Inversion of Control，即“控制反转”，不是什么技术，而是一种设计思想。在Java开发中，Ioc意味着将你设计好的对象交给容器控制，而不是传统的在你的对象内部直接控制。如何理解好Ioc呢？理解好Ioc的关键是要明确“谁控制谁，控制什么，为何是反转（有反转就应该有正转了），哪些方面反转了”，那我们来深入分析一下：

谁控制谁，控制什么：传统Java SE程序设计，我们直接在对象内部通过new进行创建对象，是程序主动去创建依赖对象；而IoC是有专门一个容器来创建这些对象，即由Ioc容器来控制对象的创建；谁控制谁？当然是IoC 容器控制了对象；控制什么？那就是主要控制了外部资源获取（不只是对象包括比如文件等）。

为何是反转，哪些方面反转了：有反转就有正转，传统应用程序是由我们自己在对象中主动控制去直接获取依赖对象，也就是正转；而反转则是由容器来帮忙创建及注入依赖对象；为何是反转？因为由容器帮我们查找及注入依赖对象，对象只是被动的接受依赖对象，所以是反转；哪些方面反转了？依赖对象的获取被反转了。

项目中怎么用到IOC下面做个简单介绍:

* 通过扫描器把写好的JAVA POJO注入到spring框架中去:



记得包下要注入的类需要加: @Service

* 通过注解的方式把对象引入JAVA类中：



记住这个是必须的。

#### SRING AOP说明

AOP(Aspect Oriented Programming)，也就是面向切面编程的技术。AOP基于IoC基础，是对OOP的有益补充。

　AOP将应用系统分为两部分，核心业务逻辑（Core business concerns）及横向的通用逻辑，也就是所谓的方面Crosscutting enterprise concerns，例如，所有大中型应用都要涉及到的持久化管理（Persistent）、事务管理（Transaction Management）、安全管理（Security）、日志管理（Logging）和调试管理（Debugging）等。

　　让我们从定义一些重要的AOP概念开始。

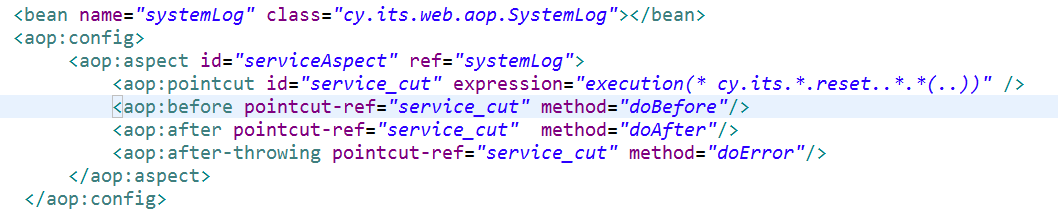
* 切面（Aspect）：一个关注点的模块化，这个关注点实现可能另外横切多个对象。事务管理是J2EE应用中一个很好的横切关注点例子。方面用Spring的Advisor或拦截器实现。
* 连接点（Joinpoint）：程序执行过程中明确的点，如方法的调用或特定的异常被抛出。
* 通知（Advice）：在特定的连接点，AOP框架执行的动作。各种类型的通知包括“around”、“before”和“throws”通知。通知类型将在下面讨论。许多AOP框架包括Spring都是以拦截器做通知模型，维护一个“围绕”连接点的拦截器链。
* 切入点（Pointcut）：指定一个通知将被引发的一系列连接点的集合。AOP框架必须允许开发者指定切入点，例如，使用正则表达式。
* 引入（Introduction）：添加方法或字段到被通知的类。Spring允许引入新的接口到任何被通知的对象。例如，你可以使用一个引入使任何对象实现IsModified接口，来简化缓存。
* 目标对象（Target Object）：包含连接点的对象，也被称作被通知或被代理对象。
* AOP代理（AOP Proxy）：AOP框架创建的对象，包含通知。在Spring中，AOP代理可以是JDK动态代理或CGLIB代理。
* 编织（Weaving）：组装方面来创建一个被通知对象。这可以在编译时完成（例如使用AspectJ编译器），也可以在运行时完成。Spring和其他纯Java AOP框架一样，在运行时完成织入。

各种通知类型包括：

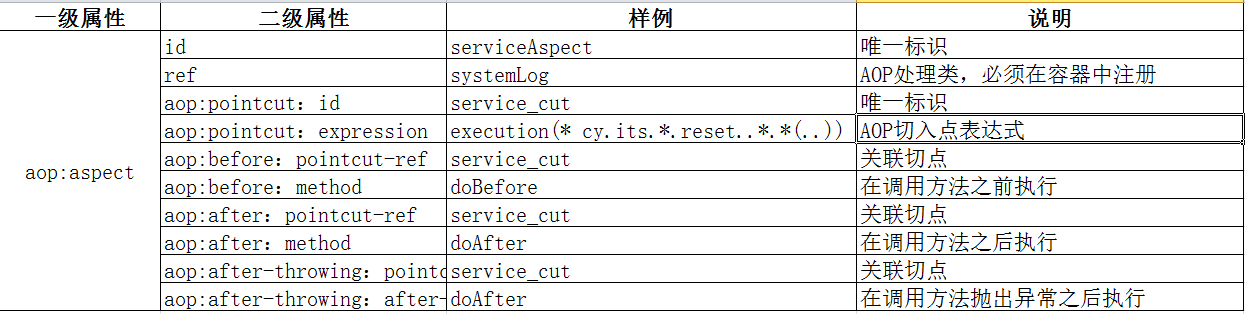
* Around通知：包围一个连接点的通知，如方法调用。这是最强大的通知。Aroud通知在方法调用前后完成自定义的行为，它们负责选择继续执行连接点或通过返回它们自己的返回值或抛出异常来短路执行。
* Before通知：在一个连接点之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。
* Throws通知：在方法抛出异常时执行的通知。Spring提供强制类型的Throws通知，因此你可以书写代码捕获感兴趣的异常（和它的子类），不需要从Throwable或Exception强制类型转换。
* After returning通知：在连接点正常完成后执行的通知，例如，一个方法正常返回，没有抛出异常。

系统中以日志框架和事务管理为例来说明AOP用法:

* 操作日志管理
* 配置文件，在dispather.xml中



参数说明:



execution切入点指示符:

execution（modifiers-pattern? ret-type-pattern declaring-type-pattern? name-pattern（param-pattern）throws-pattern?）

除了返回类型模式（上面代码片断中的ret-type-pattern），名字模式和参数模式以外， 所有的部分都是可选的。返回类型模式决定了方法的返回类型必须依次匹配一个连接点。 你会使用的最频繁的返回类型模式是\*，它代表了匹配任意的返回类型。 一个全限定的类型名将只会匹配返回给定类型的方法。名字模式匹配的是方法名。 你可以使用\*通配符作为所有或者部分命名模式。 参数模式稍微有点复杂：()匹配了一个不接受任何参数的方法， 而(..)匹配了一个接受任意数量参数的方法（零或者更多）。 模式(\*)匹配了一个接受一个任何类型的参数的方法。 模式(\*,String)匹配了一个接受两个参数的方法，第一个可以是任意类型， 第二个则必须是String类型。

下面给出一些通用切入点表达式的例子。

* 任意公共方法的执行：

execution（public \* \*（..））

* 任何一个名字以“set”开始的方法的执行：

execution（\* set\*（..））

* AccountService接口定义的任意方法的执行：

execution（\* com.xyz.service.AccountService.\*（..））

* 在service包中定义的任意方法的执行：

execution（\* com.xyz.service.\*.\*（..））

* 在service包或其子包中定义的任意方法的执行：

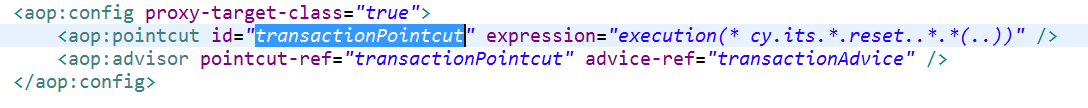
execution（\* com.xyz.service..\*.\*（..））

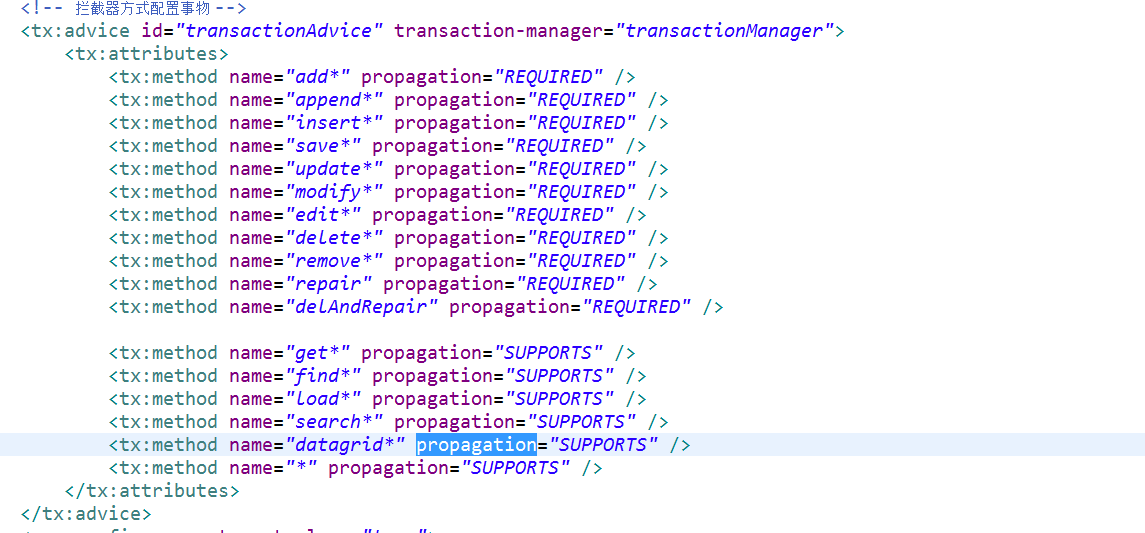
配置文件的用意是,所有符合execution表达式的类的所有方法都会被

SystemLog处理，在方法之前、之后或是抛出异常都会记录日志，当然不是所有方法都需要记录，则后期会加入一些配置，让日志之记录符合要求的。

* 数据操作事务处理

事务处理也采用AOP声明式处理，无任何代码的侵入，开发人员只要开发业务而不需要去处理复杂的事务，事务控制在rest层，这就要AOP在MVC层处理事务。注意开发时候需要事务介入的方法需要抛出异常，不能做try catch处理。配置如下:





先配置切点，execution表达式参考上节讲述。

在Spring AOP中，有3个常用的概念，Advices、Pointcut、Advisor，解释如下:

* Advices：表示一个method执行前或执行后的动作。
* Pointcut：表示根据method的名字或者正则表达式去拦截一个method。
* Advisor：Advice和Pointcut组成的独立的单元，并且能够传给proxy factory 对象。

<tx:advice/> 标签来指定不同的事务性设置，详解如下:



* read-only

事务属性中的readOnly标志表示对应的事务应该被最优化为只读事务。如果值为true就会告诉Spring我这个方法里面没有insert或者update，你只需要提供只读的数据库Connection就行了，这种执行效率会比read-write的Connection高，所以这是一个最优化提示。在一些情况下，一些事务策略能够起到显著的最优化效果，例如在使用Object/Relational映射工具（如：Hibernate或TopLink）时避免dirty checking（试图“刷新”）。

* timeout

在属性中还有定义“timeout”值的选项，指定事务超时为几秒。一般不会使用这个属性。在JTA中，这将被简单地传递到J2EE服务器的事务协调程序，并据此得到相应的解释。

* Isolation Level(事务隔离等级)的5个枚举值

为什么事务要有Isolation Level这个属性？先回顾下数据库事务的知识：

第一类丢失更新(lost update)：在完全未隔离事务的情况下，两个事物更新同一条数据资源，某一事物异常终止，回滚造成第一个完成的更新也同时丢失。

第二类丢失更新(second lost updates)：是不可重复读的特殊情况，如果两个事务都读取同一行，然后两个都进行写操作，并提交，第一个事务所做的改变就会丢失。

第三脏读(dirty read)：如果第二个事务查询到第一个事务还未提交的更新数据，形成脏读。因为第一个事务你还不知道是否提交，所以数据不一定是正确的。

第四虚读(phantom read)：一个事务执行两次查询，第二次结果集包含第一次中没有或者某些行已被删除，造成两次结果不一致，只是另一个事务在这两次查询中间插入或者删除了数据造成的。

第五不可重复读(unrepeated read)：一个事务两次读取同一行数据，结果得到不同状态结果，如中间正好另一个事务更新了该数据，两次结果相异，不可信任。

当遇到以上这些情况时我们可以设置isolation下面这些枚举值：

* DEFAULT：采用数据库默认隔离级别
* SERIALIZABLE：最严格的级别，事务串行执行，资源消耗最大；
* REPEATABLE\_READ：保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的数据。避免了“脏读取”和“不可重复读取”的情况，但是带来了更多的性能损失。
* READ\_COMMITTED:大多数主流数据库的默认事务等级，保证了一个事务不会读到另一个并行事务已修改但未提交的数据，避免了“脏读取”。该级别适用于大多数系统。
* READ\_UNCOMMITTED：保证了读取过程中不会读取到非法数据。隔离级别在于处理多事务的并发问题。
* 关于propagation属性的7个传播行为
* REQUIRED：指定当前方法必需在事务环境中运行，如果当前有事务环境就加入当前正在执行的事务环境，如果当前没有事务，就新建一个事务。这是默认值。
* SUPPORTS：指定当前方法加入当前事务环境，如果当前没有事务，就以非事务方式执行。
* MANDATORY：指定当前方法必须加入当前事务环境，如果当前没有事务，就抛出异常。
* REQUIRES\_NEW：指定当前方法总是会为自己发起一个新的事务，如果发现当前方法已运行在一个事务中,则原有事务被挂起,我自己创建一个属于自己的事务,直我自己这个方法commit结束,原先的事务才会恢复执行。
* NOT\_SUPPORTED：指定当前方法以非事务方式执行操作，如果当前存在事务，就把当前事务挂起，等我以非事务的状态运行完，再继续原来的事务。
* NEVER：指定当前方法绝对不能在事务范围内执行，如果方法在某个事务范围内执行，容器就抛异常，只有没关联到事务，才正常执行。
* NESTED：指定当前方法执行时，如果已经有一个事务存在,则运行在这个嵌套的事务中.如果当前环境没有运行的事务，就新建一个事务，并与父事务相互独立，这个事务拥有多个可以回滚的保证点。就是指我自己内部事务回滚不会对外部事务造成影响，只对DataSourceTransactionManager事务管理器起效。