JSPTagEx 手册

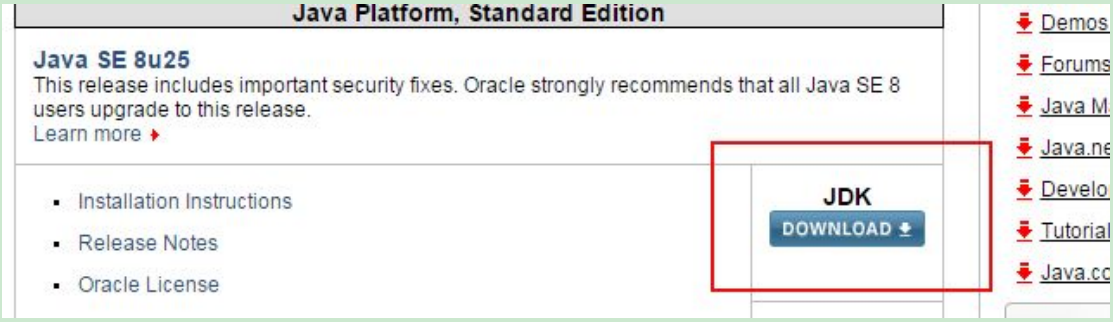
作者：Jacon.Xue

目　录

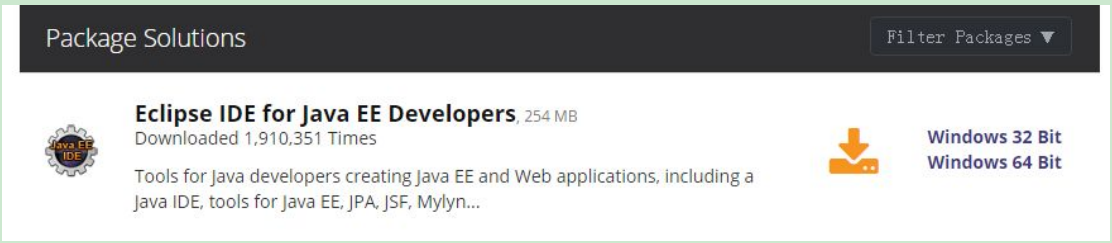
开始

环境准备

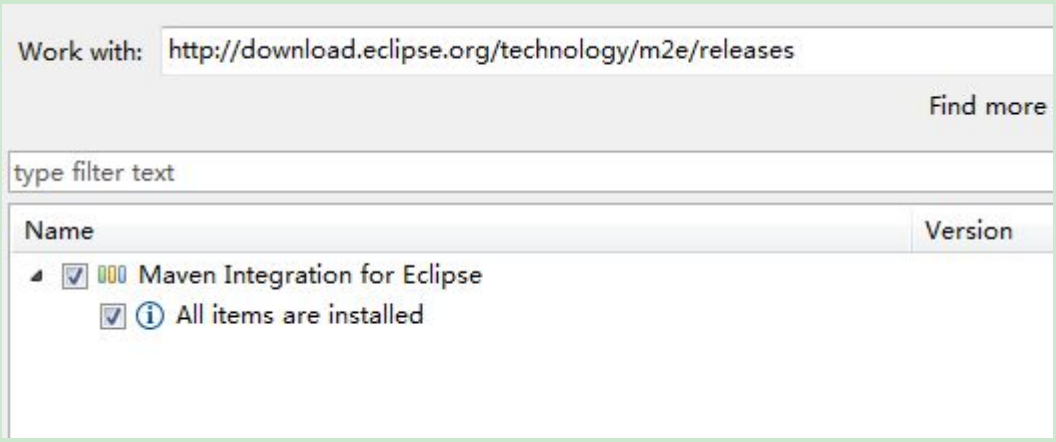
1．安装 JDK，安装 JDK1.7+版本，请到 Oracle 官网下载



2．安装 Eclipse，建议安装 luna 版本或 Eclipse Java EE 版本的最新版



3．安装 Maven 的 Eclipse 插件：m2e 插件：



4．（不是框架的提交人员请忽略）除了安装 Eclipse 的 maven 之外，还需修改.m2 下的 settings.xml（如果没有，请新建一个），以便设置本地仓库和用户授权，如下：

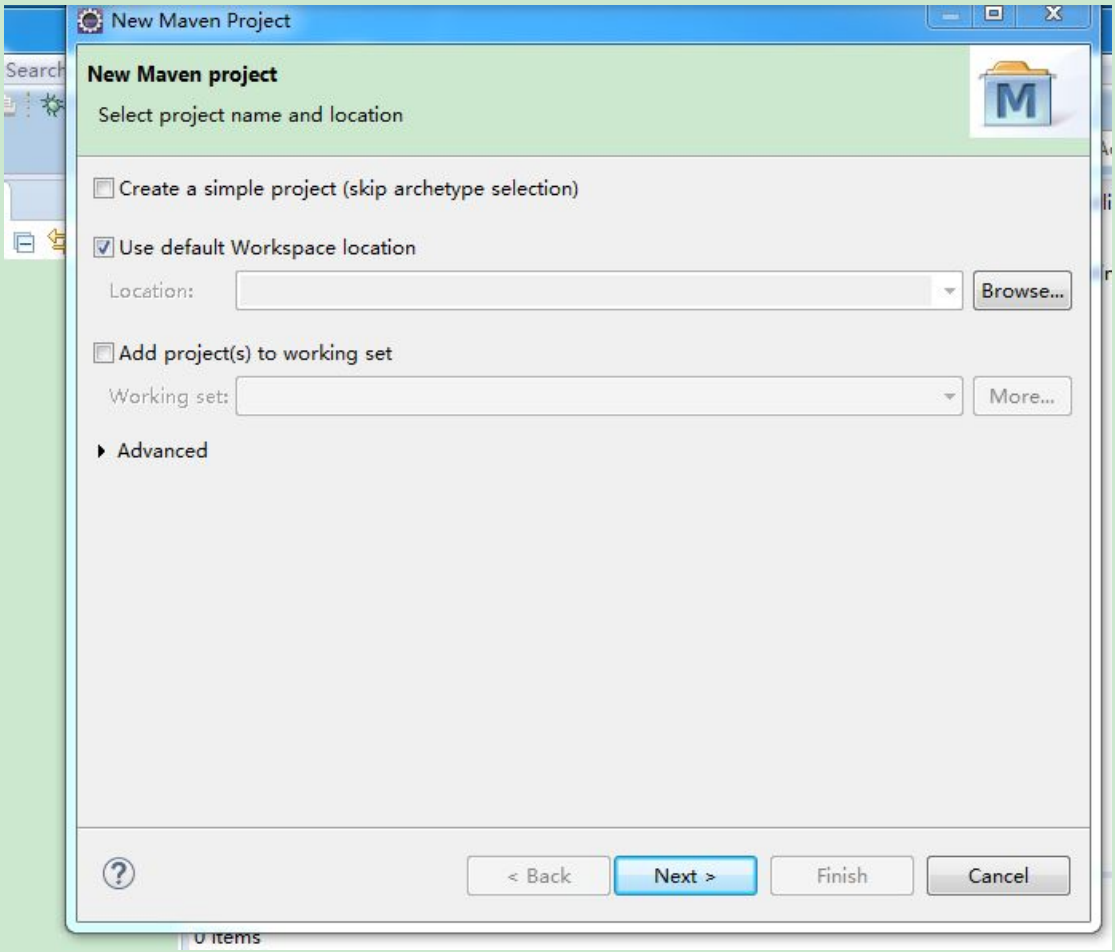
```
<servers>
  <server>
    <id>nexus</id>
    <username>[redacted]</username>
    <password>[redacted]</password>
  </server>
  <server>
    <id>nexus-snapshots</id>
    <username>[redacted]</username>
    <password>[redacted]</password>
  </server>
</servers>

<profiles>
  <profile>
    <id>nexus</id>
    <repositories>
      <repository>
        <id>nexus</id>
        <name>local private nexus</name>
        <url>[redacted]</url>
      </repository>
    </repositories>
  </profile>
  <profile>
    <id>nexus-snapshots</id>
    <repositories>
      <repository>
        <id>nexus-snapshots</id>
        <name>local private nexus snapshots</name>
        <url>[redacted]</url>
      </repository>
    </repositories>
  </profile>
</profiles>
<activeProfiles>
   <activeProfile>nexus</activeProfile>
   <activeProfile>nexus-snapshots</activeProfile>
</activeProfiles>
```

完整的 settings.xml 请到 ftp://192.168.1.200//StarndDev/settings.xml 下载，由于我们的项目要从本地数据仓库拉取 maventype 项目，所以一定要配置好本地仓库。

开始开发

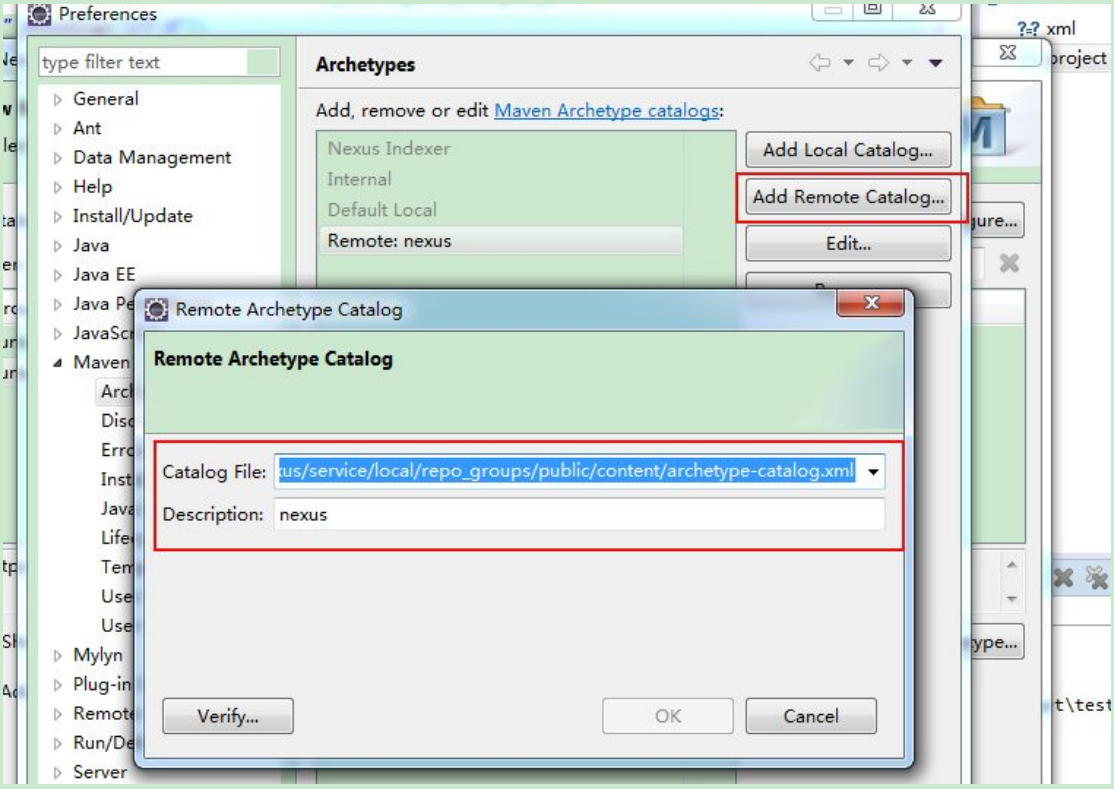
1. 新建 maven 的 webapp 原型项目：File -> New -> Maven Project



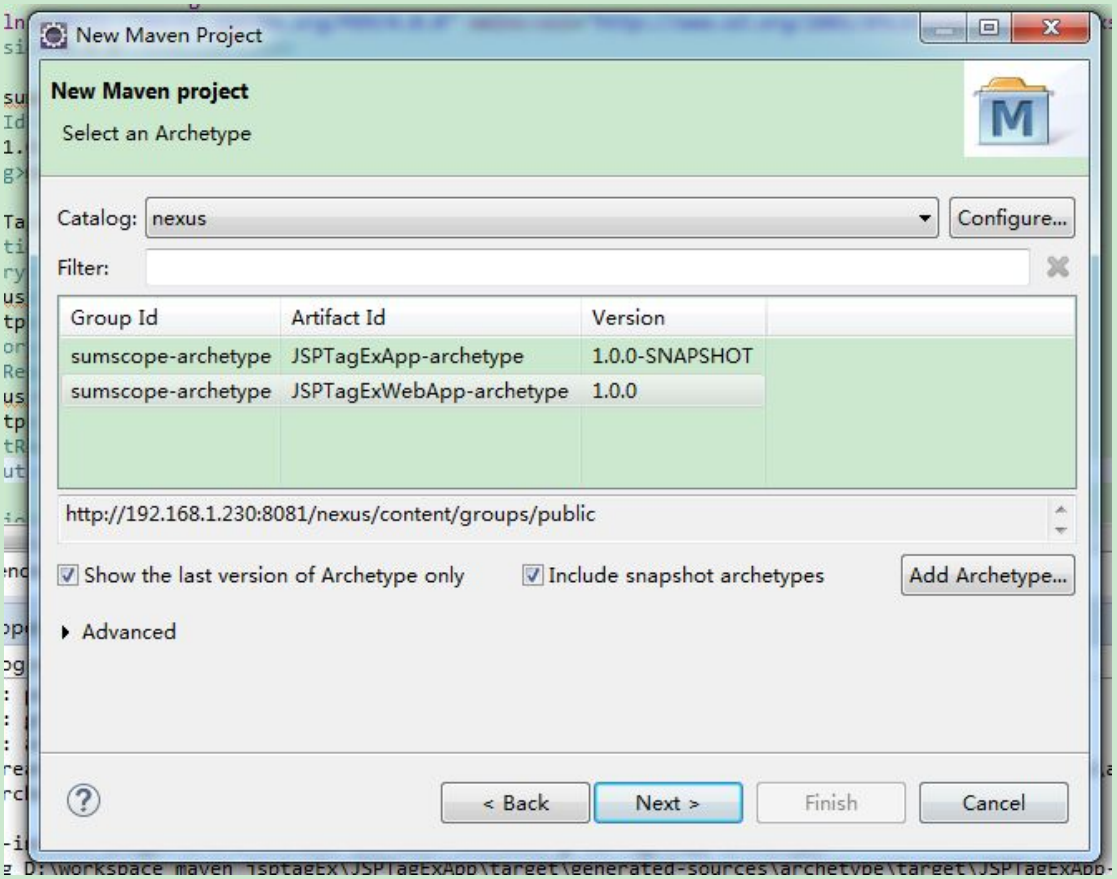
2. 如果第一次使用，请添加 maven 的 remote catalog:

http://maven.idbhost.com/nexus/service/local/repo_groups/public/content/archetype-catalog.xml，

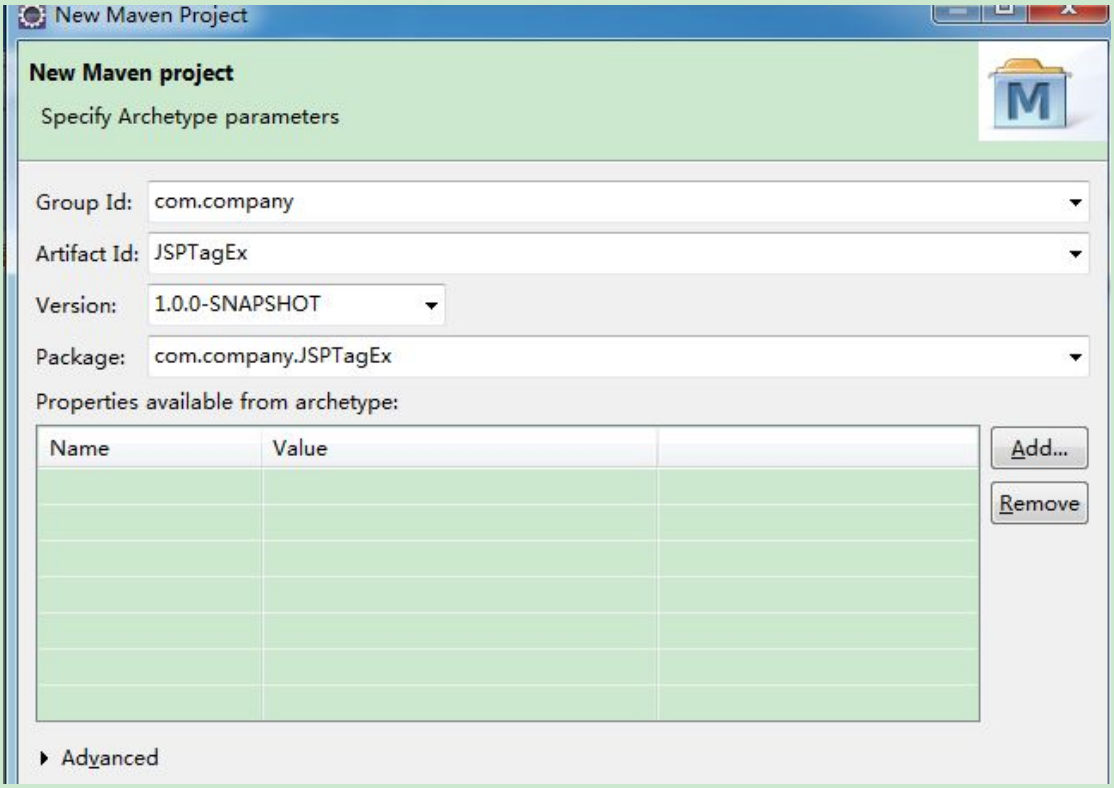
打开 Preference->Maven，选择”Archetypes”，如下图：



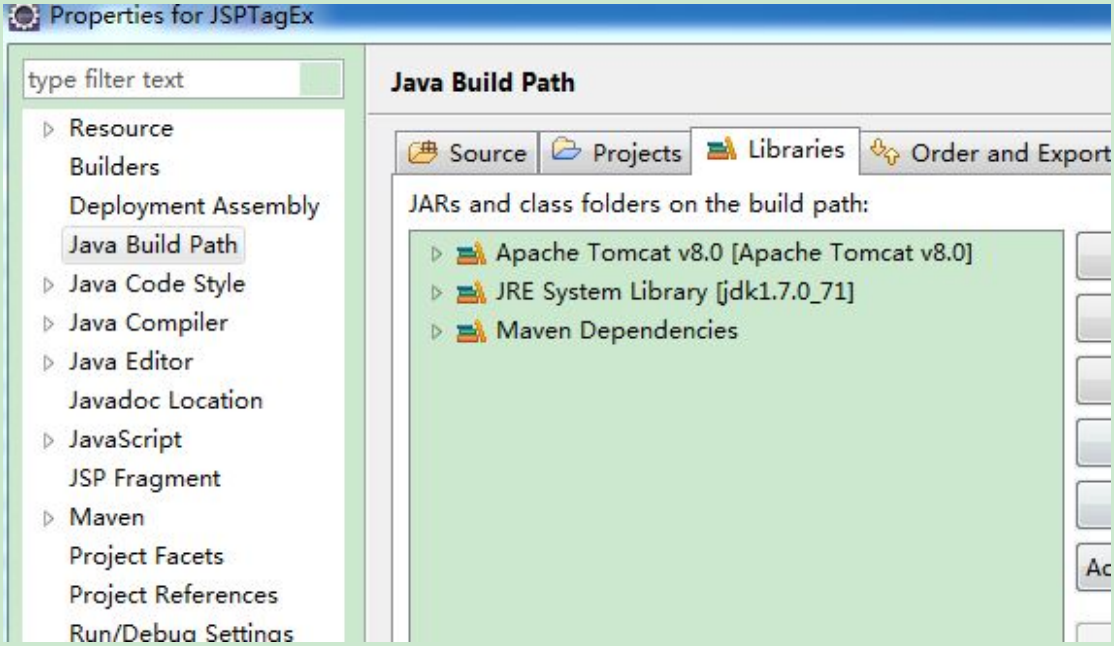
3. 点击 Next，如果你开发 Web 项目，选择 JSPTagExAPP-archetype，如果开发 Java 运行程序，请选择 JSPTagExApp-archetype，点击 Next：



4. 输入 Group Id，Artifact Id 信息，Finish

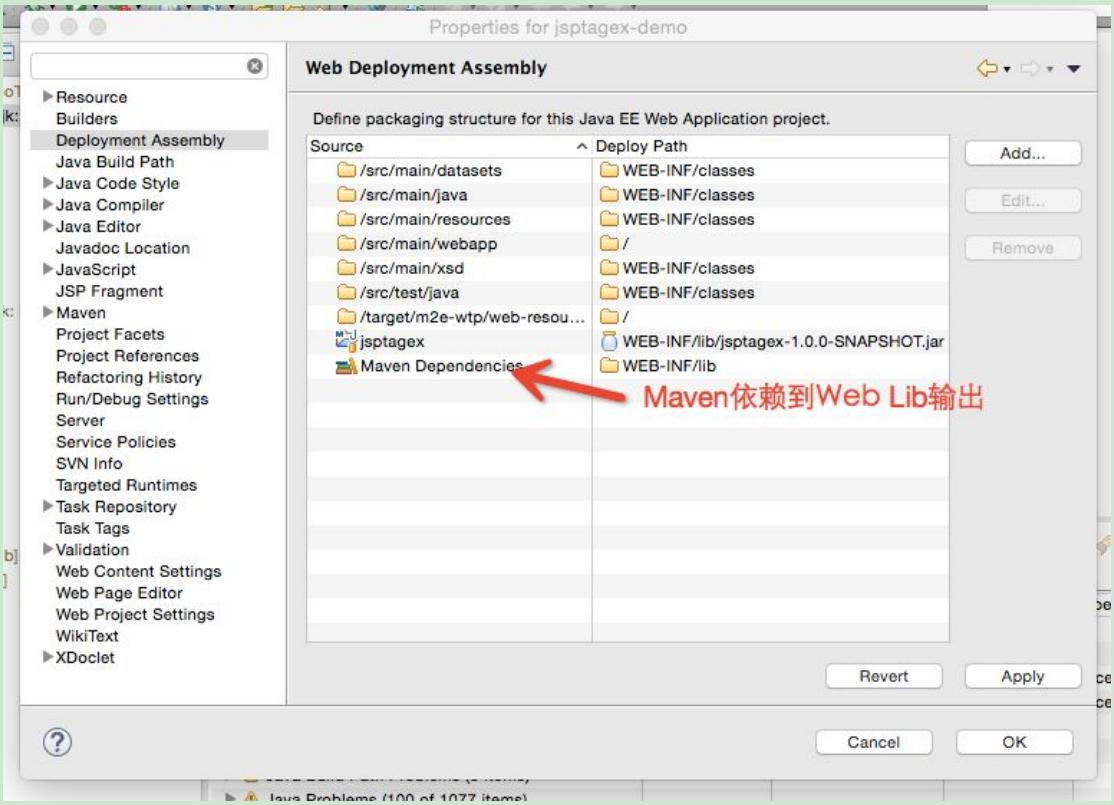


5. 设置 Project 为 UTF-8 编码，指定 Project 的 JDK：



6. 加入 maven lib 到 web 项目

maven 的 lib 要设置到 Web 下的 lib 输出，不然发布的时候可能会引起缺 lib 的提示，引入方法：



7. 运行程序

将这个 WebJSPTagEx 项目加入到 servers 的 Tomcat 或其他 Web Server 内，启动 server，在浏览器中输入：http://localhost:8080/JSPTagExWeb，登录后打开网页，即可看到 JSPTagEx 的基本功能列表演示：

是否很简单呢:)

5 分钟课程

JSPTagEx 的初衷在于简化 Web 开发，让我们花一点时间了解一下它在简化 Web 开发方面的表现。

如果你想要在页面上输出一张后台数据表的内容，并支持分页，以往我们往往需要编写 Controller，Model，Dao 以及前台页面的处理，那么现在你只需要关注前台即可，不需要编写任何 Java 代码，比如输出用户表的内容，只需要在 JSP 页面里引入 Tag 并循环输出即可，其中{param.username}是表单传入的网页 username 的参数值：

```
<sn:dataset name="ds1">select * from base_user where username like {param.username}</sn:dataset>
<sn:repeat varStatus="sts" var="item" items="${ds1 }">
    ${sts.index+1 } ${item.username } - ${item.truename }<br/>
</sn:repeat></li>
```

其次如果你想编写一个 Controller 类用来处理网页业务，那么你只需要在 web.xml 定义好 controller 的 package 的扫描路径，是否需要全局拦截这些 controller，支持多个 controller package 的定义，这样针对团队开发是极其有利的，如下：

```
<!-- 支持restful映射的servlet配置 -->
<servlet>
    <servlet-name>dispatchServlet</servlet-name>
    <servlet-class>com.sumscope.tag.rest.servlet.TagDispatchServlet</servlet-class>
    <init-param>
        <param-name>prefix</param-name>
        <param-value>restful,restful2</param-value>
    </init-param>
    <init-param>
        <param-name>scanPath</param-name>
        <param-value>com.company.controller,com.company.other.controller</param-value>
    </init-param>
    <init-param>
        <param-name>interceptor</param-name>
        <param-value>com.company.interceptor.ControllerInterceptor</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatchServlet</servlet-name>
    <url-pattern>*.jhtml</url-pattern>
</servlet-mapping>
```

然后就可以在 com.company,controller 或 com.company.other.controller 下新建 Controller 类，并且集成 BaseController 就可方便的操作数据了，如下图：

```
public class TestController extends BaseController{
    public String gotoJSP()
    {
        Map map = TagJDBCInstance.getInstance().queryOne("select 1", null);
        request.setAttribute("map", map);
        return "/test.jsp";
    }
    public String gotoFtl()
    {
        List list = TagJDBCInstance.getInstance().queryList("select * from base_user", null);
        request.setAttribute("users", list);
        return "/test.ftl";
    }
    @URIAlias(value="d")
    public String restd() throws Exception
    {
        request.setAttribute("test", "actionKey :d的");
        return "/test.jsp";
    }
}
```

那么我们分别可以用

http://localhost:8080/WebJSPTagEx/restful/TestController/gotoJSP.html，

http://localhost:8080/WebJSPTagEx/restful/TestController/gotoFtlhtml

http://localhost:8080/WebJSPTagEx/restful/TestController/d.html

访问到这三个方法，其中 restd 使用了注解，当然你也可以在 Controller 类上使用注解，以便让 URL 更加友好：

```
@URIAlias(value="other")
public class TestOtherController extends BaseController{
    @URIAlias(value="d")
    @Inject(value=TestMethodAnnoInterceptor.class)
    public String restd() throws Exception
    {
        request.setAttribute("test", "actionKey :d的"+TestOtherController.class.getName());
        return "/test.jsp";
    }
}
```

这个 Controller 的访问地址就是：

http://localhost:8080/WebJSPTagEx/restful/other/d.html

可以看到，在 Controller 里可以很方便的使用 BaseController 封装好的一系列函数，并且可以使用 TagJDBCInstance 单例方便的提取数据并支持多数据源，如果想让 d 访问只支持 Post，那么在方法上加上@Post 注解，如果只支持 Get，在方法上加入@Get 注解，另外还有@Put 和@Delete。

在一个网站的开发中，可能还需要使用 AOP，在上图的 restd 方法中，有一个@Inject(value=TestMethodAnnoInterceptor.class) 的注解，表名这个方法的执行注入了 TestMethodAnnoInterceptor 这个类的执行，这个类必须实现 IAOPCallback 这个接口，如下图：

```
public class TestMethodAnnoInterceptor implements IAOPCallback{

    @Override
    public void before(Object obj, Object[] args) {
        System.out.println("before:"+obj);

    }

    @Override
    public void after(Object obj, Object[] args) {
        System.out.println("after:"+obj);
    }

}
```

这样在执行 restd 函数的开始处，会调用 before 方法，在执行函数返回值之前会调用 after 函数。通过使用 AOP 我们可以很方便提供注入方法调用次数统计，通知，日志记录等功能。

本框架中所有的数据集定义被独立在单独的 XML 文件中，当然也可以使用 TagJDBCInstance 直接执行 SQL 语句，如果在 XML 中定义数据集，程序启动的时候会自动扫描 app.xml 定义中的数据集扫描路径下扫描所有数据集并缓存，数据集的定义可以是一个 SQL 语句，或一个存储过程，或一个 JavaScript 函数（拼接 SQL）甚至是一个 Java 类，这是有区别于 mybatis 框架的地方。在任何地方都可以通过类似：

TagConst.datasetMap.get("xx.xxx");这样的语句获取 DataSet 对象，DataSet 对象了一系列的方法用于执行数据集。当然所有的数据集都支持缓存，可以在数据集或 API 函数中传入 cacheName 即可，数据集就会根据 cacheName 在 app.xml 中定义的缓存策略执行。具体的使用在下文中将会有详细的介绍。当然我们提供了另外一种操作数据库的方式，通过 ActiveRecordGenCode 单例类可实现所有 DAO 代码的自动生成，并在这些 DAO 中利用 OneOne 和 OneMany 注解提供单表的一对一和一对多接口，从而给单表提供更多的附加属性，这种方式操作数据库将变得更加简单。

作为一个网站，可以很方便的编写 Controller，又可以很方便的定义数据集了，那么就类似完成了 Spring+SpringMVC+Mybatis 的工作了，而且比这个框架组合编写代码起来更方便，更容易维护，更精简，那么剩下的工作就是如何更加简化 Web 开发，让它不仅仅局限在仅仅是一个框架而已，所以 JSPTagEx 将集成 Web 开发中经常会使用到的一些公共模块，这些模块有些被整合在框架内部，有些以框架的插件形式提供。

配置文件概述

在本框架中，只有 1 个配置文件 app.xml，Web 程序在启动的时候会自动读入 app.xml 的配置，完成初始化的工作（主要初始化全局变量、数据源和扫描数据集配置文件等）。

全局配置一览表

名称	值	说明
show-sql	false	是否显示调试的 SQL
登录有关的设置		
login.type		目前只支持 db，未来可能会添加 LDAP 等
login.getuser		获取用户信息的 SQL 语句
login.getroles		获取用户角色列表
login.param.username		登录中网页表单用户名 name
login.param.password		登录中网页表单密码 name
login.field.password		数据库中的密码字段名
login.url.login		登录页面地址
login.url.security.ignore		无需登录的 url 地址，/**表示全部放行
login.url.security.deny		必须登录的 url 地址，高于 login.url.security.ignore 的设置

login.url.success		登录成功转向的页面地址
login.url.failed		失败页面地址
login.url.logout		登出后跳转的地址
Security 标签设置		
sec.disableUISecurity	true	是否需要嵌入不显示标签时的 HTML 片段
sec.tip.prefix		sec.disableUISecurity=True 时有效，HTML 片段前缀
sec.tip.suffix		sec.disableUISecurity=True 时有效，HTML 片段后缀
WebSocket 插件的配置信息		
websocket.server.enable		是否启动 websocket 服务
websocket.server.port		Websocket 对外提供的服务端地址
websocket.client.enable		是否启动内部通信的 client
websocket.client.ip		Client 的 ip 地址
websocket.client.port		Client 的端口
导出标签配置		
export.dir.tmp		导出文件地址
Freemarker 自定义标签		
tags.dataset	com.sumscope.tag.freemark.tag.DataSetDirective	固定值，dataest 的 freemarker 自定义标签
tags.selectOne	com.sumscope.tag.freemark.tag.SelectOneDirective	固定值，selectOne 的 freemarker 自定义标签
ActiveRecord 方式操作数据库配置		
dao-scan-package		配置扫描包名，多个用,分开

缓存配置

```
<cache name="cacheDD" cache2disk="false" idletime="3000" max-num="1000" livetime="6000" policy="LRU"/>
```

Name：缓存名，与操作数据集的 cacheName 配对；
Cache2disk：是否缓存到磁盘；
Idletime：空闲时间，单位：秒；
Max-num：最大元素；
Livetime：存活时间，单位：秒；
Policy：缓存策略，支持：LRU/FIFO 等；

定时任务

```
<job name="jobTest" class="com.sumscope.web.cms.job.ScanQMAnalysis" ds-rel="default" cron="0/30 * * * * ?" title="每隔5s执行一次"/>
```

Name：唯一任务名，不支持中文；

Class：任务实现类，必须集成 TagJob；
Ds_rel：数据源名称，与 datasource 定义的 name 一致；
Cron：cron 表达式；
Title：任务描述；

数据源定义

```
<datasource dbtype="mysql" name="default" class="org.apache.commons.dbcp2.BasicDataSource">
    <property name="driver" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://192.168.1.242:3306/blog?useUnicode=true&amp;characterEncoding=utf8&amp;autoReconnect=true" />
    <property name="user" value="root" />
    <property name="password" value="123456" />

    <property name="maxActive" value="50" />
    <property name="maxIdle" value="10" />
    <property name="minIdle" value="0" />
    <property name="maxWait" value="100000" />
    <property name="timeBetweenEvictionRunsMillis" value="600000" />
    <property name="minEvictableIdleTimeMillis" value="300000" />
</datasource>
```

略，都懂的:)

数据集定义

```
<datasets>
    <dataset ds-rel="default" recursive="true" pattern="true" name="cms" title="CMS数据" file="abs:D:/workspace_qb/cms_yezi/datasets/(cms_)(\S*?)(\.xml)" />
</datasets>
```

Ds-rel：数据源名称；
Recursive：目录是否递归；
Pattern：是否支持正则；
Name：数据集的 namespace，该 namespace 下扫描到的数据集的真实数据源名都必须带上这个 name;
Title：数据集描述
File：支持 abs 和 classpath 前缀，分别对应在绝对路径下查找和在 classpath 下查找；

web.xml

如果你使用本框架用于 Web 开发，并且需要使用 MVC，那么在 web.xml 还需要几个配置，如下：

- 安装编码过滤器，为了避免 Web 项目中产生乱码，将所有请求进行统一 UTF-8 编码：

```
<filter>
    <filter-name>Encoding</filter-name>
    <filter-class>
        com.sumscope.tag.filter.SetCharacterEncodingFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>Encoding</filter-name>
    <url-pattern>*</url-pattern>
</filter-mapping>
```

- Restful 配置，如果需要使用类似微博/a/b/xx/yyy 的 URL 风格，请安装这个 Filter，既可

以使用注解也可以在 XML 配置。

```
<!-- 支持Restrul风格的URL配置 -->
<filter>
    <filter-name>TagRestFilter</filter-name>
    <filter-class>com.sumscope.tag.filter.TagRestFilter</filter-class>
    <init-param>
        <param-name>scanPackage</param-name>
        <param-value>/tag.rest.xml,com.sumscope.web.cms.servlet</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>TagRestFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

- 如果需要默认的登录功能，请安装这个 Filter：

```
<!-- 如果要使用登录功能，请配置这个filter -->
<filter>
    <filter-name>TagLoginFilter</filter-name>
    <filter-class>com.sumscope.tag.filter.TagLoginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>TagLoginFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

- 请安装初始化 Listener：

```
<listener>
    <listener-class>com.sumscope.tag.listener.NBSListener</listener-class>
</listener>
```

- 请安装 Servlet 的 MVC 分发器，scanPath 支持多个 package，用英文逗号分开；

```
<!-- 支持restful跳转的servlet配置 -->
<servlet>
    <servlet-name>dispatchServlet</servlet-name>
    <servlet-class>com.sumscope.tag.rest.servlet.TagDispatchServlet</servlet-class>
    <init-param>
        <param-name>prefix</param-name>
        <param-value>restful</param-value>
    </init-param>
    <init-param>
        <param-name>scanPath</param-name>
        <param-value>com.sumscope.controller</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatchServlet</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>
```

MVC

在 web.xml 定义好 Dispatch Servlet 后即可使用 MVC 功能，MVC 的设计思路主要是注

解大于约定，在 web.xml 定义好 Servlet 的 MVC 分发器后即可使用，在该 package 或子 package 下定义任务 Class 都具有 URL 访问能力，特别注意：public 方法才能使用 URL 访问。MVC 的访问 URL 有以下几种情况（以 http://localhost:8080/web 为站点名，web.xml 配置中 prefix 为 restful，url-patter 为*.jhtml 为举例）：

- 类没注解，方法名没注解：该情况下 URL 的访问地址：

http://localhost:8080/web/restful/x.y.z.className/funcName.jhtml

注意 x.y.z 为该 package 的子 package 路径，如果直接在 package 下，则直接改为：className，注意类名第一个字母系统会自动转为小写，比如你类型为：Test，则 URL 上实际上为：test，方法名为函数名，区分大小写；

- 类没注解，方法名有注解：该情况下 URL 的访问地址：

http://localhost:8080/web/restful/x.y.z.className/funcAlias.jhtml

注意 x.y.z 为该 package 的子 package 路径，如果直接在 package 下，则直接改为：className，注意类名第一个字母系统会自动转为小写，方法名为函数名上用 URIAlias 注解的名字，此时 restful 后的 URL 全为小写；

- 类有注解，方法名没注解：该情况下 URL 的访问地址：

http://localhost:8080/web/restful/classAlias/funcName.jhtml

注意 classAlias 为类上 URIAlias 注解的值，后跟类中的函数名，此时 restful 后的 URL 全为小写；

- 类有注解，方法名有注解：该情况下 URL 的访问地址：

http://localhost:8080/web/restful/classAlias/funcAlias.jhtml

注意 classAlias 为类上 URIAlias 注解的值，后跟类中的函数名的注解 funcAlias，此时 restful 后的 URL 全为小写；

Controller 中的方法支持重载，不像 SpringMVC，需要针对函数中的参数需要加入注解，本框架函数自动反射，并且网页传来的顺序也无需保持一致，只需要函数的形参名和网页参数一致即可：

```
public String testparam(String p1)
{
    request.setAttribute("p1", p1);
    return "/test.jsp";
}
public String testparam(String p1,String p2)
{
    request.setAttribute("p1", p1);
    request.setAttribute("p2", p2);
    return "/test.jsp";
}
//测试重载函数
public String testparam(String p1,String p2,String p3)
{
    request.setAttribute("p1", p1);
    request.setAttribute("p2", p2);
    request.setAttribute("p3", p3);
    return "/test.jsp";
}
```

网页传入的参数方式：

```
测试重载函数1：testController/testparam.html?p1=abc
测试重载函数2：testController/testparam.html?p1=abc&p2=中国人
测试重载函数3(参数可随意跌倒)：testController/testparam.html?p1=abc&p3=中国人&p1=p1value
```

请注意：目前仅支持 String 类型的形参。

另外为了开发 SPA 应用方便（将所有的符合规定的 URL 路由到同一个 Servlet 处理），框架提供了 TagRest 注解，能为任何的 Servlet 提供 Restful 风格的 API，无需在 Web.xml 定义，其中**代码第 N 个参数，***代表所有的请求转发到这个 servlet 处理，如遇到冲突，优先映射到**的 servlet，最后映射到***的 servlet 处理，如`@TagRest(value="teams/***")`表示 /teams/xxx ， /teams/x/y,/teams/x/y/z 等 URL 都会被这个 Servlet 处理，再如`@TagRest(value="teams/api/user/**")`，表示/teams/api/user/xxx，会被这个 servlet 处理，并且通过 request.getAttribute(“$1”)可拿到参数 xxx，可以有任意多个**，依次通过$1,$2,……等拿到参数值。

总之，类似 SpringMVC 但不同 SpringMVC，你甚至不需要在类和函数上使用任何注解功能就能让 Controller 类中的方法具有 URL 访问能力，简化开发，Controller 类下的任何方法都能使用 request 和 response 两个成员变量，以便从网页提取参数和向网页输出内容。使用上去除了 SpringMVC 繁杂的配置和注解（只用到 1 个 URIAlias 注解），目前 Controller 下的方法 URL 默认访问方式是 Post 和 Get 都可以，如果仅支持 Post，请使用@Post 注解限定，如果仅支持 Get 访问，请使用@Get 限定。

注意：所有的 Controller 类必须继承 BaseController，记住：URL 有一个约定，不管是类还是函数，只要有 1 个用到了注解，则 URL 实际访问地址都是小写，如果要从跟开始访

问，则直接 URIAlias=”/”即可。

Logging

日志处理是所有系统非常重要的特性之一，JSPTagEx 使用 Log4j2 进行日志记录，并附件一整套日志工具函数，日志解析程序与日志报表生成工具，让每个使用 JSPTagEx 框架的项目都能做到日志实时监控，汇总归并日志，按需生成各类业务日志报表，框架提供了 LoggerManager 单例类来记录标准化的日志，这些标准化的日志通过 agent 将所有 Log4j 日志归并到统一的日志分析处理器执行。让所有项目都暴露在可控范围之内。

ActiveRecord

使用 ActiveRecord 的方式操作数据库，一切将变得更加简单，该功能从 1.1.0 版本开始提供，实现步骤：

1. 在 app.xml 配置 Model 的扫描包路径，直接多个，用,分开，如：

```
<item name="dao-scan-package" value="com.sumscope.test.dao.bo,com.sumscope.test.dao.bo2"/>
```

2. 在指定包下创建 Model，统一 extends Model 即可，如：

```
@Table("base_app")
@PrimaryKey("app_id")
@DataSource("default")
public class BaseApp extends Model<BaseApp>{
    public static BaseApp me = new BaseApp();
}
```

当然，你可以针对 BaseApp 应用 getter 和 setter 方法，如下：

```
public String getApp_id() {
    return get("app_id");
}

public BaseApp setApp_id(String app_id) {
    set("app_id",app_id);
    return this;
}
```

注意：建议将 set 方法返回 this 指针，以便链式调用。
这样也许你调用起来更直接：

```
new BaseApp().setApp_id("1").add();
```

支持三个注解：

- @Table：（可选）如果表名和类名不一样，请指定；
- @PrimaryKey：（可选）如果表名的主键字段不是 ID，多个主键用,分开，请指定（区分

大小写）；

- @DataSource：（可选）如果表对应的数据源不是默认数据员名称，请指定，与 app.xml 中定义的数据源名称保持一致；

另外你可以通过 GenDao 工具类一次性生成所有的这些 DAO。

以下代码是针对 base_app 表的增删改查：

```
//根据ID查询BaseApp记录
BaseApp app = BaseApp.me.findById("a", "*");
List<BaseApp> appList = BaseApp.me.find("select * from base_app where app_name = ?","test");
//保存数据
app.set("app_name", "test").save();
//新增记录
BaseApp.me.set("app_id", "abc").set("app_name", "hellow world!").add();
//删除记录
app.deleteById("a");
```

另外，框架提供了 `ActiveRecordGenCode` 单例类可自动生成这些 DAO，您无需编写一行代码。自动生成的代码只包含了单表的所有字段信息，针对该 DAO 还需知道的其他信息需要你通过 OneMany 和 OneOne 注解在这个 DAO 里扩展编写，比如在一个团队用户的表里，针对用户的 DAO 还需知道他属于哪些团队，则可以利用 OneMany 注解（一对多），比如还需要知道他属于详细的地址信息（一对一）则可以使用 OneOne 注解，以 OneMany 注解为例：

```
@OneMany(sql="select * from t_team where id in (select teamid from t_team_user where userid=:id)",type=T_teamDAO.class)
public List<T_teamDAO> getTeams(){
    return (List)get("teams");
}
```

注意:id 是待参数名的 SQL 写法，这个变量最终会被 User 对象的 id 值替代，整个过程 SQL 是无法注入的，因为采用的是预定义 SQL 方法，通过解析这个 SQL，:id 会被解析成?，然后框架通过编排预定义的 SQL 执行获取结果集，接下去在 Controller 里就可以拿到这个用户列表了：

```
List<T_userDAO> userList = T_userDAO.dao
        .find(new String[]{"teams"},"select id,username, realname from "
                + T_userDAO.dao.getTable().getName()
                + whereSql.toString()
                + "  order by username limit 0, 10");
```

可以在 String[]里拿去任意多个扩展属性，由于在 OneMany 注解中使用了:id，所以在提取用户列表时，select id 是必须的，否则 teams 将无法获取到正确的列表，这里请特别注意。

是不是很简单呢:)

事务处理

事务处理主要通过原生 JDBC 事务机制进行，非常简单：

```
DbUtils.tx(new ITx() {
    @Override
    public boolean run(Connection conn) throws SQLException {
        new BaseApp().set("app_id", "1").set("app_name","abc").add(conn);
        new Base_Org().set("app_id", "1").set("org_id", "1").add(conn);
        return true;
    }
});
```

使用工具类 DbUtils 的 tx 即可，在 run 方法中返回 true 表示事务提交，false 表示回滚事

物，注意，用事务中 add 和 save 方法，必须传入当前的 Connetion。

目前本框架暂不支持声明式事务。

AOP

目前框架针对 AOP 仅支持 Controller 全局拦截和 Controller 里的单个函数安装特定拦截器，这两个特性能满足大部分 Web 开发的需求。

JSP Taglib

为了简化开发，框架提供了一套类似 struts 的标签库，主要包括了常量提取标签，数据集提取标签，判断标签，循环标签，权限判定标签，工具类标签等，后续可能还会按需加入更多标签。

1. 认识基础标签

```
<!-- dateset start -->
<sn:choose>
    <sn:when test="${param.c=='1' }">c:1</sn:when>
    <sn:when test="${param.c=='2' }">c:2</sn:when>
    <sn:else>c:other</sn:else>
</sn:choose>
<!-- dateset end -->
```

if-elseif-else标签

```
<p>sn:repeat标签的For循环：
<sn:repeat var="ss" begin="3" end="10" step="2">
        <span>${ss }  </span>
</sn:repeat>
</p>
```

for循环标签

```
<p>sn:authorize标签：
<sn:authorize var="isHaveRole1OrRole3" ifAnyGranted="role1,role3">
<p>这是有role1，role3任意角色能看的内容</p>
</sn:authorize>
<sn:authorize ifAllGranted="role1,role3">
<p>这是同时具有role1，role3角色能看的内容</p>
</sn:authorize>
<sn:authorize ifNotGranted="role1">
<p>这是不具有role1,role3任何一个角色能看的内容</p>
</sn:authorize>
<p>输出角色判定的变量:${isHaveRole1OrRole3 }</p>
</p>
```

资源访问受限标签

2. 数据集标签

```
<li>数据集演示
    <ol>
        <li>页面里直接使用数据集：<br/>
        按用户名模糊查询:<input name="user" id="user" type="text" /><input type="button" value="search" onclick="doQuery()"/><br/>
        <sn:dataset name="ds1">select * from base_user where username like {param.username}</sn:dataset>
        <sn:repeat varStatus="sts" var="item" items="${ds1 }">
            ${sts.index+1 } ${item.username } - ${item.truename }<br/>
        </sn:repeat></li>
        <li>
            脚本SQL：<br/>
            按用户名模糊查询：
            <select id="islock">
                <option value="all">不限</option>
                <option value="0">正常</option>
                <option value="1">锁定</option>
            </select>
            <input type="button" value="search" onclick="doQuery2()"/><br/>
            <br/>
            <sn:dataset name="dsscript" datasetRel="tag.test" bizRel="sqlScript" />
            <sn:repeat varStatus="sts" var="item" items="${dsscript }">
                ${sts.index+1 } ${item.username } ${item.islock=='0'?"正常状态":"锁定状态" }<br/>
            </sn:repeat>
        </li>
        <li>
            引用后台定义的数据集并使用了缓存策略testCache:<br/>
            <sn:dataset name="ds2" datasetRel="tag.test" bizRel="sql1" />
            <sn:repeat varStatus="sts" var="item" items="${ds2 }">
                ${sts.index+1 } ${item.username } - ${item.truename }<br/>
            </sn:repeat>
        </li>
        <li>
            数据集是java类并且使用了testCache,特别注意items的写法:<br/>
            <sn:dataset name="ds_java" datasetRel="tag.test" bizRel="javabiz" />
            <sn:repeat varStatus="sts" var="item" items="${ds_java.users }">
                ${sts.index+1 } ${item.menu_title }<br/>
            </sn:repeat>
        </li>
    </ol>

</li>
```

以上展现了直接利用 SQL 定义数据集，利用数据集 XML 配置文件定义数据集，缓存使用，java 类数据集等功能用法，要理解这些用法并不难，一看属性便知；

3. 工具类标签

导出标签：

```
<sn:export datasetRel="dstest1.dataset1" expType="text" title="<b>导出成txt</b>">
    [
        {name:"CODE",label:"股票代码"},
        {name:"SHORT_NAME",label:"股票名称"},
        {name:"MARKET",label:"交易市场"},
        {name:"TRADE_DATE",label:"上市日"}
    ]
</sn:export>
<sn:export datasetRel="dstest1.dataset1" expType="csv" title="导出成CSV">
    [
        {name:"CODE",label:"股票代码"},
        {name:"SHORT_NAME",label:"股票名称"},
        {name:"MARKET",label:"交易市场"},
        {name:"TRADE_DATE",label:"上市日"}
    ]
</sn:export>
<sn:export datasetRel="dstest1.dataset1" expType="word" title="导出成Word">
    [
        {name:"CODE",label:"股票代码"},
        {label:"股票基本信息",cols:[
            {name:"SHORT_NAME",label:"股票名称"},
            {name:"MARKET",label:"交易市场"},
            {label:"日期信息",cols:[
                {name:"TRADE_DATE",label:"上市日"},
                {name:"CREATED_DATE",label:"创建日期"}
            ]}
        ]}

    ]
</sn:export>
<sn:export datasetRel="dstest1.dataset1" expType="excel" title="导出成EXCEL" expTitle="多行表头定义格式">
    [
        {name:"CODE",label:"股票代码"},
        {label:"股票基本信息",cols:[
            {name:"SHORT_NAME",label:"股票名称"},
            {name:"MARKET",label:"交易市场"},
            {label:"日期信息",cols:[
                {name:"TRADE_DATE",label:"上市日"},
                {name:"CREATED_DATE",label:"创建日期"}
            ]}
        ]}

    ]
```

展现了纯文本，csv，word 和 excel 的导出方式，数据自动按数据集定义导出，其中 Excel 可支持多表头。

Freemarker Tag

dataset 标签

用于获取数据集，典型的应用如下，直接指定 SQL 语句：

```
<@dataset name="aa" sql="select * from wc_channel where website_id=1" sortdatafield="id" sortorder="asc" >
<ul>
<#list aa as a>
  <li><a href="http://${a.CNL_PATH}.domain.com">${a.CNL_NAME}</a></li>
</#list>
</ul>
</@dataset>
```

也可以直接引用数据集：

```
<@dataset name="aa" bizRel="list" datasetRel="cms.article" datasourceRel="default"  sortdatafield="id" sortorder="asc" >
<ul>
<#list aa as a>
  <li><a href="http://${a.CNL_PATH}.domain.com">${a.CNL_NAME}</a></li>
</#list>
</ul>
</@dataset>
```

后者是被推荐的方式，SQL 的集中管理有利于系统的可维护性。

selectOne 标签

用于获取某个单条结果集，可以使用本标签：

```
<@selectOne name="article" id="${articleId}" datasetRel="cms.article" bizRel="getArticleById">
  <h1>${article.TITLE}</h1>
  <p>
      ${article.CONTENT}
  </p>
</@selectOne>
```

同 dataset 标签，你也可以直接编写 sql 属性，不过我们不建议你这么做。

注解

本框架注解主要被应用在 ActiveRecord 的定义，URL 的定义，Plugin 的扩展以及 Rest URL 风格等地方。合理的使用注解，可使得面向终端使用这套框架的用户降低学习成本，减少配置以及让代码更具有可读性。大家熟知的 Spring 框架在后续的版本中，也大量使用了注解，下表罗列了目前框架包含的注解：

注解	Scope	说明
@URIAlias	注解在 Controller 的类或方法上	URL 改写，提供更友好的 URL 访问地址

@Inject	Controller 类的方法上	拦截该方法，value 是一个类名，如：
		`@Inject(value=TestMethodAnnoInterceptor.class)`
		`public String restd() throws Exception`

@TagRest	Rest Servlet 上	支持/*变量，如/a/b/*/*，参数通过 getAttribute(“$1”),$2 等获取
@Post	Controller 类的方法上	该方法只能通过 http post 方式访问
@Get	Controller 类的方法上	该方法只能通过 http get 方式

		访问
@Put	Controller 类的方法上	该方法只能通过 http put 方式访问
@Delete	Controller 类的方法上	该方法只能通过 http delete 方式访问
@ContentTypeJSON	Controller 类的方法上@Put	表示通过 Header 头里拿去参数信息，参数统一放到方法的 JSONObject 形参中
ActiveRecord 注解		
@Table		表名注解
@PrimaryKey		表的关键字注解
@DataSource		数据源注解
@OneMany	Type：DAO 类名 Sql：提取 List 的 SQL 语句，用:xx 形式提供参数值	一对多
@OneOne	Type：DAO 类名 Sql：提取 DAO 对象的 SQL 语句，用:xx 形式提供参数值	一对一

数据集定义

本框架支持三种数据集的定义格式，分别是普通 SQL 语句，JavaScript 脚本和 Java 类，根据不同的应用场景可以选用不同的数据集定义方式。

普通 SQL 语句

普通 SQL 语句定义是最常见的方式，适用于大多数场景，当然这里的 SQL 语句也可以是直接调用存储过程。

普通 SQL 语句：

```
<select init-arg="{n:1}" cache-rel="testCache">
    <![CDATA[
        select * from master where short_name like ${param.n}
    ]]>
</select>
```

调用存储过程

```
<select>
    <![CDATA[
      call proTest(${sessionScope.username})
    ]]>
</select>
```

其中 cache-rel 指定缓存名，init-arg 指定 SQL 语句的初始化参数，JSON 字符串格式。与 app.xml 配置的 cache name 对应，SQL 语句中提取网页参数的写法是：${param.xxx}，提取 Session 中的参数值是:${sessionScope.xxx}，类似 EL 表达式。系统在执行这个 SQL 时，会自动将其替换为?，利用 PreparedStatement 的方式执行 SQL 语句，用来防范 SQL 注入。

JavaScript 脚本

在处理有些应用场景下，单条 SQL 可能涉及到一些业务判断，以便组装相应的 SQL 语句，这种应用场景也较为常见，不同于 mybatis 等框架，本框架采用 Javascript 写法：

```
<biz id="sqlScript">
    <script>
    <![CDATA[
            var ret = "select * from base_user where 1=1";
            if("${param.islock}" !='all'){
                ret += " and islock = '${param.islock}'";
            }
        ]]>
    </script>
</biz>
```

请特别注意，js 的写法采用的 SQL 拼凑法，请注意防范 SQL 注入的处理，无需返回值，固定将 ret 赋值即可，在框架中会自动提取 ret 字符串，执行 SQL。

Java 类

在更复杂的业务场景下，数据集可能无法通过数据库直接查询得出，部分数据可能来自于内存或其他地方，如 LDAP 等，这时，我们就需要通过定义 Java 类来定义数据集，数据集的定义，固定继承 AbsDataSetExec 即可。

Restful Style

插件体系

插件的整体设计思路基于 Web 应用的功能扩展，但这些扩展应用并不是每个 Web 应用所必须的，为了实现更加简单的功能扩展以及功能使用，我们配套了针对框架的插件扩展，编写一个插件比较简单，只需要简单的继承 Plugin 即可，并通过 maven 脚本将其打包成一个 zip 文件，然后将该 zip 文件放到 Web 应用的 classpath 下的 plugins 目录即可。

插件与 Web 应用互不影响，按需加载，如果要使用插件，比如引入 plugin-api 的 maven 模块，以便 Web 应用和插件通信。最典型的有以下应用场景：

1. Web 应用调用插件方法：
2. 插件对外提供 HTTP 服务：

Connector 插件

Connector 插件为 Web 项目提供了连接 SFTP，Qpid，ActiveMQ，LDAP 等中间件的能力，通过简单的在 init.xml 配置，即可实现与这些中间件的连接以及业务处理。

验证码插件

为网站提供验证码服务。

WebSocket 插件

为网站提供 WebSocket 的能力。

全文检索插件

提供基于 lucene 的全文检索。

RBAC 插件

完善的基于 RBAC 的权限体系插件。

工作流插件

基于 Activiti5 的工作流引擎。

上传插件

基于图片和文件的通用上传接口。

前端框架整合

Bootstrap

首先非常感谢 Bootstrap 团队提供的丰富的 CSS 和 JS 组件，通过与 Bootstrap 整合，可开发完成绚丽的后台效果和灵活的布局变更，并支持响应式布局，在浏览器兼容性方面，支持 IE8+，Chrome，Firefox 等，单纯的用 Bootstrap 还无法达到快速开发后台项目的任务，所以有必要在此基础上封装一层，以便更好的与 jsptagex 框架融合，详见 jsdoc。