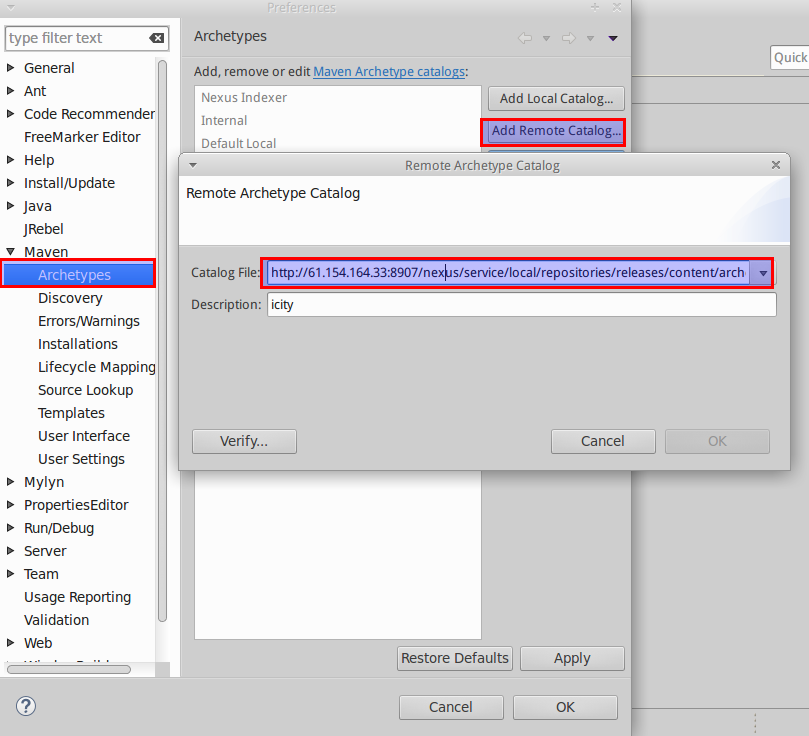
**当前框架版本: 1.0.0-beta**

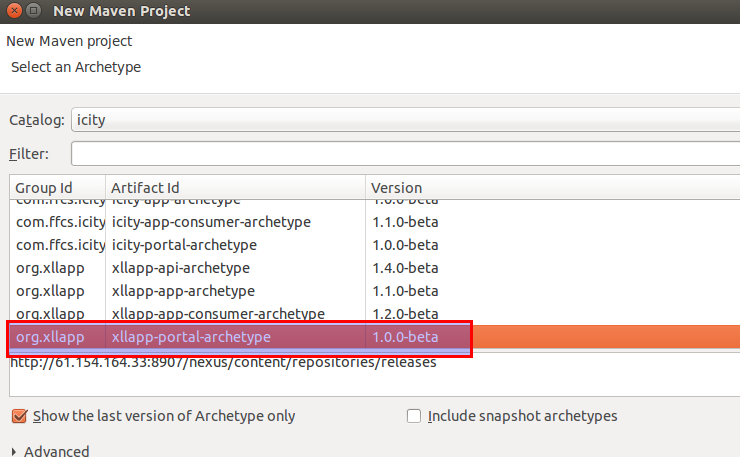
# 创建工程

在Eclipse中添加如下Archetype catalog：

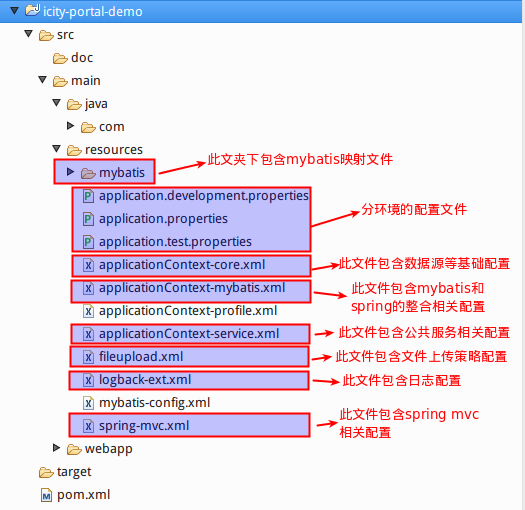
http://61.154.164.33:8907/nexus/service/local/repositories/releases/content/archetype-catalog.xml



创建工程时选择如下Archetype：



# 工程介绍



# 开发CRUD功能

数据库表：

CREATE TABLE `demo` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL COMMENT '用户名',

`password` varchar(45) DEFAULT NULL COMMENT '密码',

`icon` varchar(1024) DEFAULT NULL COMMENT '图标',

`status` int(11) NOT NULL DEFAULT '0' COMMENT '状态',

`create\_time` datetime DEFAULT NULL COMMENT '创建时间',

PRIMARY KEY (`id`),

) ENGINE=InnoDB AUTO\_INCREMENT=167 DEFAULT CHARSET=utf8;

Controller类

@RequestMapping("demo")

@Controller

public class DemoController extends SimpleController<Demo> {

private DemoDao demoDao;

@Override

protected CRUDDao<Demo> getDao() {

return this.demoDao;

}

@Autowired

public void setDemoDao(DemoDao demoDao) {

this.demoDao = demoDao;

}

}

Dao类：

@MyBatisRepository

public interface DemoDao extends CRUDDao<Demo> {

}

Entity类：

public class Demo extends IdEntity {

private String name;

private String password;

private String icon;

private Integer status;

private Date createTime;

…….

}

Mybatis Mapper文件:

<mapper namespace="com.ffcs.icity.mvc.demo.dao.DemoDao">

......

<select id="query" parameterType="map" resultMap="demoResultMap">

......

</select>

<select id="get" parameterType="long" resultMap="demoResultMap">

......

</select>

<insert id="insert" parameterType="com.ffcs.icity.mvc.demo.entity.Demo" useGeneratedKeys="true" keyProperty="id">

......

</insert>

<update id="update" parameterType="com.ffcs.icity.mvc.demo.entity.Demo">

......

</update>

<delete id="delete" parameterType="int">

......

</delete>

<delete id="deletes">

......

</delete>

</mapper>

列表页面:

<!DOCTYPE html>

<html>

......

<body>

<!-- 表格 -->

<table class="data\_panel">

<thead>

<tr>

<th><span></span></th>

<th><span>用户名</span></th>

<th><span>状态</span></th>

<th><span>创建时间</span></th>

</tr>

</thead>

<tbody>

<#list page.result as item>

<tr>

<td><#if item.id?has\_content>${item.id}</#if></td>

<td><#if item.name?has\_content>${item.name}</#if></td>

<td><#if item.status?has\_content>${statusMap[item.status?string]}</#if></td>

<td><#if item.createTime?has\_content>${item.createTime?datetime}</#if></td>

<td><a href="?act=SHOW\_FORM&id=${item.id!}">更新</a> | <a href="?act=DELETE&ids=${item.id!}">删除</a></td>

</tr>

</#list>

</tbody>

</table>

<!-- 分页栏 -->

<table class="pager\_panel">

<tr>

<td><@page\_navigator/></td>

</tr>

</table>

</body>

</html>

**注意：需要将页面放在webapp/WEB-INF/views/demo中，文件名必须为list.ftl.**

录入页面:

<!DOCTYPE html>

<html>

....

<body>

<form id="saveForm" method="post" action="?act=SAVE">

<input type="hidden" id="id" name="id" value="<#if entity?has\_content>${entity.id}</#if>"/>

<table class="save\_form\_table" border="1">

<tr>

<td class="save\_form\_field\_label"><span>用户名</span></td>

<td><input class="save\_form\_field\_input" id="name" name="name" type="text" value="<#if entity?has\_content>${entity.name}</#if>" /></td>

</tr>

<tr>

<td class="save\_form\_field\_label"><span>密码</span></td>

<td><input class="save\_form\_field\_input" id="password" name="password" type="text" value="<#if entity?has\_content>${entity.password}</#if>" /></td>

</tr>

<tr>

<td class="save\_form\_field\_label"><span>图标</span></td>

<td><input class="save\_form\_field\_input" id="icon" name="icon" type="text" value="<#if entity?has\_content>${entity.icon}</#if>" fileUploadType="image" /></td>

</tr>

<tr>

<td class="save\_form\_field\_label"><span>状态</span></td>

<td><select class="search\_field\_input search\_field\_select" id="status" name="status" >

<option value="">选择</option>

<option value="0" <#if entity?has\_content && entity.status!?string == '0'>selected='selected'</#if>>启用</option>

<option value="1" <#if entity?has\_content && entity.status!?string == '1'>selected='selected'</#if>>禁用</option>

</select></td>

</tr>

<tr>

<td class="save\_form\_field\_label"><span>创建时间</span></td>

<td><input class="save\_form\_field\_input save\_form\_field\_readonly" id="createTime" name="createTime" type="text" readonly="true" onfocus="this.blur();" value="<#if entity?has\_content && entity.createTime?has\_content>${entity.createTime?datetime}</#if>"/></td>

</tr>

<tr>

<td colspan="2" class="save\_form\_button\_panel">

<input type="submit" value="保存"/>

<input type="reset" class="reset" value="重置"/>

</td>

</tr>

</table>

</form>

</body>

</html>

**注意：需要将页面放在webapp/WEB-INF/views/demo中，文件名必须为form.ftl.**

# 异步请求

## 异步的CRUD

只需要在请求的时候添加\_responseContentType=json参数或使用jquery的ajax模块发起请求即可。

## 响应json内容

Controller类:

@RequestMapping("demo")

@Controller

public class DemoController extends SimpleController<Demo> {

@RequestMapping("test")

public View test(HttpServletRequest request, HttpServletResponse response,Model model) {

model.addAttribute("aaa", "123");

return ResponseUtils.getJsonView(true, "123");

}

}

响应内容:

{

message: "123",

data: {

aaa: "123"

},

success: true

}

页面调用:

<script type="text/javascript">

$().ready(function() {

$("#test").click(function(){

$.getJSON("<@full\_path path="/demo"/>",function(result){

alert(JSON.stringify(result));

});

});

});

</script>

<button id="test">json</button>

# 自动生成CRUD代码

## 在项目中使用代码生成器

代码生成器是一个maven插件，需要在项目的pom.xml中添加如下配置：

<plugin>

<groupId>com.ffcs.icity</groupId>

<artifactId>icity-maven-plugin</artifactId>

<version>2.0.0-beta</version>

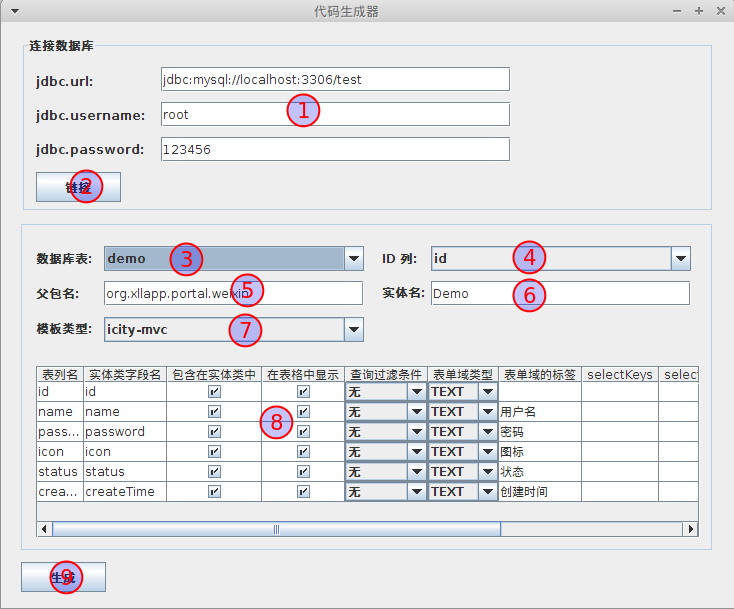
</plugin>

通过如下命令启动代码生成器：

icity:mvc-code-generate

## 代码生成器使用

以上一节中手动开发的代码为例子介绍代码生成器的使用.



使用步骤:

1，填写数据库连接相关信息(上图1);

2，点击“连接”按钮(上图2)，获取数据库表信息；

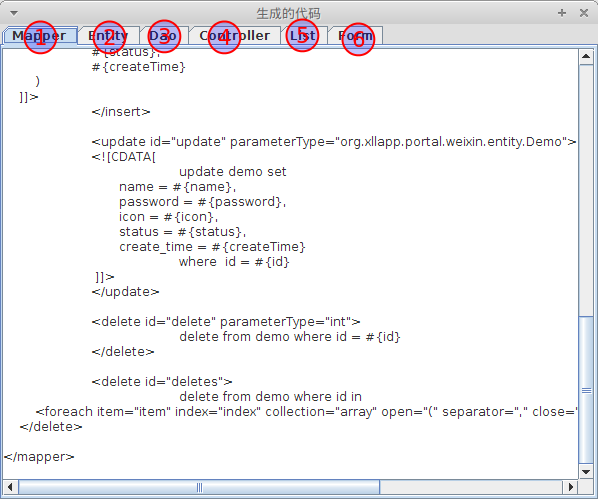
3，选择需要操作的表(上图3)后，程序会自动解析表的元信息生成实体名等(上图6)；

4，选择主键字段(上图 4) (默认第一个字段或名为”id”的字段)并填写父包名(上图5) (生成的代码在此包名的对应子包中，如Dao类在dao包中);

5，选择生成代码的模板类型(上图7)，当前支持生成基于icity-mvc框架的代码(默认)和单纯的实体类;

6，设置各个字段的相关配置(上图8)，如录入表单域的类型，列表页面查询表单的条件等；

7，点击”生成”按钮(上图9)生成代码。



说明：

上图1：mybatis映射文件的代码;

上图2：实体类的代码;

上图3：Dao类的代码;

上图4：Controller类的代码;

上图5：列表页面的代码;

上图6：录入表单页面的代码;

8,将生成的代码手动拷贝到工程中。

生成的CRUD功能界面如下：

列表页面:



录入表单页面：



# 上传组件

上传组件包含文件上传服务和jquery上传插件.使用时只需要定义好上传策略并在页面上使用上传插件上传文件即可。

## 上传策略定义

在fileupload.xml中定义上传策略，上传策略支持继承。

策略定义说明：

<fileUploadStrategy>

<id>file</id> 策略ID

<parentId>file</parentId> 父策略ID,如果提供了父策略ID,那么当策略的属性为空(maxSize属性为-1)时,会使用父策略同名属性的值

<maxSize>1048576</maxSize> 最大的上传文件大小,单位:字节, 例: 1M = 1048576 (1024\*1024)

<storageDir>file</storageDir> 上传文件的存储目录

<baseStoragePath></baseStoragePath> 上传文件存储的根目录,当为空时,值为:<应用根目录路径>/uploads

<baseFileUrl></baseFileUrl> 上传文件的URL的前缀部分,当为空时,值为:应用的根URL

<exts> 允许上传的文件类型的扩展名,当为空时,表示不限制上传文件类型

<ext>png</ext>

<ext>bmp</ext>

<ext>jpg</ext>

<ext>jpeg</ext>

<ext>gif</ext>

</exts>

</fileUploadStrategy>

## 上传插件使用

### 引入如下的js和css

<link type="text/css" rel="stylesheet" media="screen" href="<@full\_path path="js/jBox/Skins/Default/jbox.css"/>"/>

<link rel="stylesheet" type="text/css" href="<@full\_path path="js/jBox/Skins/Default/jbox.css"/>"/>

<script type="text/javascript" src="<@full\_path path="js/jquery/jquery-1.8.0.min.js"/>"></script>

<script type="text/javascript" src="<@full\_path path="js/jBox/jquery.jBox-2.3.min.js"/>"></script>

<script type="text/javascript" src="<@full\_path path="js/jBox/i18n/jquery.jBox-zh-CN.js"/>"></script>

<script type="text/javascript" src="<@full\_path path="js/uploadfile.js"/>"></script>

<script type="text/javascript" src="<@full\_path path="upload/file/js/variables"/>"></script>

### 插件的使用示例

1. 上传图片

方式一：通过添加input扩展属性的方式:

<input id="test1" name="test1" type="text" fileUploadType="image">

方式二：通过js方式：

<script type="text/javascript">

$(document).ready(function(){

$("#js-test1").xllUploadFile({

fileUploadType:"image"

});

});

</script>

<input id="test1" name="test1" type="text">

效果：



1. 上传文件

方式一：通过添加input扩展属性的方式:

<input id="test1" name="test1" type="text" fileUploadType="file">

方式二：通过js方式：

<script type="text/javascript">

$(document).ready(function(){

$("#test1").xllUploadFile({

fileUploadType:"file"

});

});

</script>

<input id="test1" name="test1" type="text">

效果：



1. 使用自定义上传策略(需要在服务端配置对应的策略)

方式一：通过添加input扩展属性的方式:

<input id="test1" name="test1" type="text" fileUploadType="file" fileUploadStrategy="abc">

方式二：通过js方式：

<script type="text/javascript">

$(document).ready(function(){

$("#test1").xllUploadFile({

fileUploadType:"file",

fileUploadStrategy:"abc"

});

});

</script>

<input id="test1" name="test1" type="text">

1. 为按钮添加class

方式一：通过添加input扩展属性的方式:

<input id="test1" name="test1" type="text" fileUploadButtonClass="upbtn">

方式二：通过js方式：

<script type="text/javascript">

$(document).ready(function(){

$("#test1").xllUploadFile({

fileUploadType:"file",

fileUploadButtonClass:"upbtn"

});

});

</script>

<input id="test1" name="test1" type="text">

效果：



1. 为图片预览框添加class

方式一：通过添加input扩展属性的方式:

<input id="test1" name="test1" type="text" fileUploadImagePreviewElementClass="uppe">

方式二：通过js方式：

<script type="text/javascript">

$(document).ready(function(){

$("#test1").xllUploadFile({

fileUploadType:"file",

fileUploadImagePreviewElementClass:"uppe"

});

});

</script>

<input id="test1" name="test1" type="text">

效果：



1. 修改上传按钮的文字

<script type="text/javascript">

$(document).ready(function(){

$("#test1").xllUploadFile({

fileUploadType:"file",

buttons:{

upload:"upload",

clear:"clear",

reset:"reset"

}

});

});

</script>

<input id="test1" name="test1" type="text">

效果：



1. 使用第三方页面文件上传组件(以ajaxFileUpload为例)

<input id="fileToUpload" type="file" size="45" name="file">

<input type="button" style="margin-left: 5px;" onclick="ajaxFileUpload()" value="上传">

<script type="text/javascript">

function ajaxFileUpload(){

$.ajaxFileUpload(

{

url:'<@full\_path path="upload/file/json/file"/>',

secureuri:false,

fileElementId:'fileToUpload',

dataType: 'json',

success: function (data, status)

{

$("#file-upload-result").append(JSON.stringify(data));

},

error: function (data, status, e)

{

alert("error:"+e);

alert("error:"+status);

alert("error:"+JSON.stringify(data));

}

}

)

return false;

}

</script>

# 日志配置

## 异常采集

框架提供将异常信息采集汇总展示并告警功能。启用此功能，需要在application.\*.properties配置文件中添加如下配置（去除这些配置即可禁用此功能）：

#日志所属模块名

log.moduleName=icity-api-client-demo

#用于记录错误日志的ActiveMQ Broker的URL

log.jms.brokerUrl=tcp://localhost:60000

#用于记录错误日志的ActiveMQ Broker的用户名

log.jms.userName=system

#用于记录错误日志的ActiveMQ Broker的密码

log.jms.password=manager

## 日志输出控制

日志配置项(application.\*.properties文件中)说明:

#日志输出级别

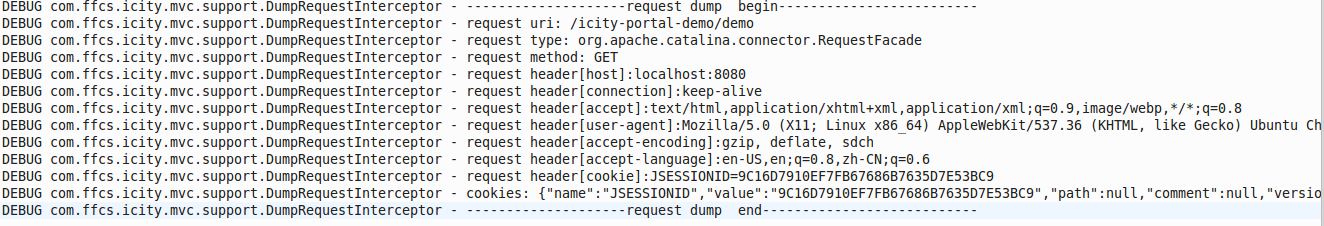
log.logger.root.level=DEBUG

#日志输出格式

log.encoderPattern=%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] IP[%X{IP}] SESSION\_ID[%X{SESSION\_ID}] REQUEST\_ID[%X{REQUEST\_ID}] %-5level %logger{80} - %msg%n

## 请求Dump功能

框架会在DEBUG级别的日志中打印出请求的详细信息，包括请求头,请求体和请求参数等。



## 如何看懂日志

每个日志条目额外包含有请求发起方的IP,会话标示(SESSION\_ID)和请求标示（REQUEST\_ID）信息。通过IP可以过滤出此IP的所有请求日志；通过会话标示(SESSION\_ID)可以过滤出同一个会话中所有请求的日志；通过请求标示（REQUEST\_ID）可以过滤出同一个请求的所有日志（包括底层框架输出的日志）。

示例：

2015-01-08 11:01:20.965 [http-8080-1] IP[127.0.0.1] SESSION\_ID[B6707E3CCDD54078E9600552489179FC] REQUEST\_ID[20150108110120-47] DEBUG com.ffcs.icity.mvc.controller.CRUDController - extract param[{}]

日志说明：

**IP[127.0.0.1]** - 表明此日志为IP为127.0.0.1的调用方发送的请求的日志**REQUEST\_ID[20150108110120-47]** - 表明此日志为标示为20150108110120-47的请求处理过程中输出的日志.可以此标示过滤出此请求处理过程中输出的全部日志。

**SESSION\_ID[B6707E3CCDD54078E9600552489179FC] -** 表明此会话标示为B6707E3CCDD54078E9600552489179FC的会话的日志，可以通过此标示过滤出此会话所有请求的处理日志。

# 配置

## 分环境配置

基于spring的profilie实现的分环境配置机制，当前支持开发,测试和生产这3种环境，分别对应application.development.properties，application.test.properties和application.properties这3个配置文件，只要将不同环境的配置写入对应的配置文件即可。可以在applicationContext-profile.xml添加新的环境配置。

默认启用的是生产环境的配置。要启用特定的环境配置，可以通过添加jvm启动参数spring.profiles.active来实现（spring.profiles.active参数值：development表示生成环境；test表示测试环境。取值的规则是配置文件中间部分:application.<spring.profiles.active参数的值>.properties）。

# 整合单点登录

## 引入统一认证中心的客户端依赖

<dependency>

<groupId>com.ffcs.icity</groupId>

<artifactId>icity-serv-client-utils</artifactId>

<version>2.8.0-20140828</version>

<exclusions>

<exclusion>

<artifactId>commons-logging</artifactId>

<groupId>commons-logging</groupId>

</exclusion>

</exclusions>

</dependency>

## 添加统一认证拦截器

在spring-mvc.xml中添加如下配置：

<bean

class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">

<property name="interceptors">

<list>

......

<ref bean="authenticationInterceptor" />

</list>

</property>

</bean>

<bean name="authenticationInterceptor" class="com.ffcs.icity.cas.client.AuthenticationInterceptor">

<property name="loginUrl" value="${cas.loginUrl}" />

<property name="retrievePrincipalUrl" value="${cas.retrievePrincipalUrl}" />

<property name="excludeUrls">

<list>

<value>/css/</value>

<value>/img/</value>

<value>/js/</value>

</list>

</property>

</bean>

在application.\*.properties配置文件中添加如下配置：

#单点登录配置

#域配置

cookie.domain=icity.com

#登录入口URL

cas.loginUrl=http://prodics-cas.icity.com:8904/icity-portal-cas/login

#获取登录用户信息接口URL

cas.retrievePrincipalUrl=http://prodics-cas.icity.com:8904/icity-portal-cas/retrievePrincipal