**准备工作**

       xweb是个Maven项目，需要安装JDK1.6+，MAVEN2.0+。数据库目前只支持MySQL，WEB服务器可使用Jetty，也可以使用tomcat等。

**初始化数据**

       初始化的SQL脚本是src/main/resource/sql.mysql目录下的schema.sql，包括包括xweb\_user、xweb\_user\_role、xweb\_role、xweb\_role\_menu、xweb\_menu、xweb\_test共6张表。  
数据字典如下：

* xweb\_user:用户表

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **字段类型** | **主键** | **备注** |
| id | BIGINT | 是 | 主键 |
| login\_name | VARCHAR(255) | NOT NULL | 登录名 |
| name | VARCHAR(50) |  | 用户名 |
| password | VARCHAR(255) |  | 密码 |
| salt | VARCHAR(64) |  | 加密密文 |
| email | VARCHAR(128) |  | 邮箱 |
| status | VARCHAR(32) |  | 状态 |
| staff\_no | VARCHAR(50) |  | 工号 |
| phone | VARCHAR(128) |  | 电话 |
| mobile | VARCHAR(128) |  | 手机号 |

* xweb\_user\_role:用户角色关联表

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **字段类型** | **主键** | **备注** |
| user\_id | BIGINT | 是 | 用户ID |
| role\_id | BIGINT | 是 | 角色ID |

* xweb\_role:角色表

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **字段类型** | **主键** | **备注** |
| id | BIGINT | 是 | 主键 |
| name | VARCHAR(255) |  | 角色名称 |
| detail | VARCHAR(255) |  | 明细 |

* xweb\_role\_menu:角色菜单关联表

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **字段类型** | **主键** | **备注** |
| role\_id | BIGINT | 是 | 角色ID |
| menu\_id | BIGINT | 是 | 菜单ID |

* xweb\_menu:菜单表

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **字段类型** | **主键** | **备注** |
| id | BIGINT | 是 | 主键 |
| parent\_id | BIGINT | NOT NULL | 父类ID |
| parent\_ids | VARCHAR(255) | NOT NULL | 所有父类编号 |
| name | VARCHAR(100) |  | 菜单名称 |
| href | VARCHAR(255) |  | 链接名称 |
| sort | INTEGER | NOT NULL | 排序 |
| is\_show | CHAR(1) | NOT NULL | 是否展示 |
| permission | VARCHAR(200) |  | 权限 |

* xweb\_test:测试表，用于模块展示

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **字段类型** | **主键** | **备注** |
| id | int(10) | 是 | 主键 |
| msg | varchar(200) |  | 消息 |

**项目运行**

       由于是Maven项目，使用mvn eclipse:eclipse构建成为Eclipse项目；  
       运行项目使用mvn jetty:run -Djetty.port=8080，表示在8080端口启动xweb，访问地址如下<http://localhost:8080/xweb>；  
       使用mvn package可以完成项目的打包过程，方便到测试机上部署war；  
       跳过单元测试，使用-Dmaven.test.skip=true；

**自定义开发**

* 已完成的模块  
         系统完成了基本用户-角色-菜单的一套权限系统，即先创建用户，然后给用户赋予角色信息，然后角色关联菜单，实现了不同用户展示不同的菜单。

1. 进入权限模块的菜单
2. 创建用户
3. 创建角色
4. 创建菜单
5. 创建菜单

注意：

* 目前系统只支持三级菜单展示，超过三级只展示三级菜单信息，如果只有二级菜单的话，只能展示到二级菜单信息。
* 如果需要页面级的权限控制，也就是说想控制页面上的某个按钮或链接展示给不同的用户，只需要在创建菜单的时候，选择页面中“未显示”并且不输入“链接”，就可以把这个菜单名当成权限名使用。框架使用的是shiro权限管理，推荐的命名规则一般是模块：用户：操作，如普通用户在菜单模块的添加的权限可以命名为menu:user:create，修改可以命名为menu:user:modify，删除可以命名为menu:user:delete，查询可以命名为menu:user:search。jsp页面使用的使用可以标签
* <shiro:hasPermission name="menu:user:create">
* <div class="row"><a class="btn btn-default" href="${ctx}/account/menu/create">创建菜单</a></div>
* </shiro:hasPermission> name="menu:user:create">

Shiro的用法可以参考技术介绍的Shiro章节。

* DEMO模块  
  目前只完成了一个简单的展示模块，数据库表对应xweb\_test，相应的控制器、业务逻辑层、数据操作层和领域层都是以Test开头。  
  系统使用的主要开发框架是Spring MVC作为控制器层，Spring为业务逻辑层，MyBatis作为数据连接层，前端使用jQuery+Bootstrap。

1. 实际开发的过程中，先完成基本设计后，创建数据表，然后根据数据表设计领域类，包名在com.renren.infra.xweb.entity下，基本的POJO即可；
2. 数据连接层位于com.renren.infra.xweb.repository，是各种类的接口，必须使用@MyBatisRepository，Spring会按照该注册扫描成DAO注入并与对应的XML配置文件关联；同时需要在src/main/resources/mybatis下创建与接口名相同的Mapper.xml配置文件，并需要保证方法名的一致。建议以XXXMybatisDAO的方式命名，XML文件最好以XXXMapper.xml来命名。  
   分页的信息，使用Mybatis自带的RowBounds，传入limit和offset，分别表示行数和起始位置。只需要在DAO的interface创建方法加入该参数，XML不需要修改，就实现了分页功能。
3. 业务逻辑层可按照需要按照接口和实现类的方法来创建，样例中为求简单没有创建接口，而是直接使用了实现类，建议以XXXService的方式命名。使用到DAO的时候需要使用@Autowired来自动注入，同时XXXService的类名要使用@Component注册，表示是将类注入到Spring容器中。@Transactional(readOnly = true)表示不使用事务控制，@Transactional(readOnly = false)这是默认方式，表示方法接受事务处理，操作失败会自动回滚。  
   位于com.renren.infra.xweb.service包下。
4. 控制器层一般以XXXController来命名，位于com.renren.infra.xweb.web包下，需要使用@Controller表示是标注控制层组件，各个方法使用@RequestMapping表示方法拦截的信息，其中value表示页面URL，method表示对应方法，可以是POST或是GET。方法返回string为对应jsp页面路径信息。  
   用到业务逻辑类时，可使用@Autowired注入业务逻辑类。  
   如果需要用到ajax，使用@ResponseBody表示json字符串，使用
5. JsonMapper binder = JsonMapper.nonDefaultMapper();
6. Map<String, Object> jsonMap = new HashMap<String, Object>();
7. jsonMap.put("属性名", 变量);
8. return binder.toJson(jsonMap);
9. jsp页面  
   文件路径在WEB-INF/views，建议以不同的业务创建不同的文件夹，一般业务包括列表页最好以XXXList.jsp结尾，增加或修改表单页面最好以XXXForm.jsp结尾。  
   使用到列表页查询的时候，只需要按照这样命名表单Form中的输入框信息
10. <label for="search\_name">菜单名：</label>
11. <input type="text" id="search\_name" name="search\_name" class="form-control" style="width: 150px;" value="${param.search\_name}" placeholder="菜单名" />

其中的search\_name中的search表示查询的前缀，name表示要查询的数据表中的字段信息，value为查询字段的值。在Contoller中使用如下代码：

Map<String, Object> searchParams = Servlets.getParametersStartingWith(request, "search\_");

将查询参数封装为Map，map的key为name，value为对应的name查询值，直接调用Service的查询方法。

1. 权限信息  
   有的时候需要使用到系统的权限，可以在Controller类中直接使用注解。@RequiresRoles("Admin")表示需要Admin的角色，@RequiresUser表示需要用户。系统使用用户信息可以通过ShiroUser user = (ShiroUser) SecurityUtils.getSubject().getPrincipal();来获取。
2. 单元测试  
   java代码位于src/test/java下，src/test/resources为配置文件信息，目前有一些基本的DAO测试。如需要测试的话，请自动添加。

**参考**

感谢开源社区，感谢Springside，Jeesite，Appfuse，提供了很多灵感，项目还在一个逐步完善的过程中，希望越来越壮大。

# [MyBatis物理分页实现](http://wiki.d.xiaonei.com/pages/viewpage.action?pageId=25331823)

### 分页原因

因MyBatis的分页使用的内存分页，一次从数据库中读入所有数据，然后按照RowBound中的limit开始记录和offset偏移量来完成分页。一旦数据量过大，可能会导致内存溢出等问题，故需要改为内存分页。

### 实现原理

Mybatis提供了Interceptor拦截器，可以通过自定义拦截器，在执行SQL语句前做一些SQL转换工作。

### 实现

1. 修改applicationContext.xml文件，加入自定义的Interceptor
2. <!-- 自定义paginationInterceptor的拦截器 -->
3. <bean id="paginationInterceptor" class="com.renren.infra.xweb.util.PaginationInterceptor"></bean>
4. <!-- MyBatis配置 -->
5. <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
6. <property name="dataSource" ref="dataSource" />
7. <!-- 自动扫描entity目录, 省掉Configuration.xml里的手工配置 -->
8. <property name="typeAliasesPackage" value="com.renren.infra.xweb.entity" />
9. <!-- 显式指定Mapper文件位置 -->
10. <property name="mapperLocations" value="classpath:/mybatis/\*Mapper.xml" />
11. <!-- 自定义Plugins的Interceptor -->
12. <property name="plugins">
13. <ref bean="paginationInterceptor"/>
14. </property>
15. </bean>
16. 创建自定义的Interceptor，代码如下：
17. @Intercepts({ @Signature(type = StatementHandler.class, method = "prepare", args = { Connection.class }) })
18. public class PaginationInterceptor implements Interceptor {
19. private final static String SQL\_SELECT\_REGEX = "(?is)^\\s\*SELECT.\*$";
20. private final static String SQL\_COUNT\_REGEX = "(?is)^\\s\*SELECT\\s+COUNT\\s\*\\(\\s\*(?:\\\*|\\w+)\\s\*\\).\*$";
21. @Override
22. public Object intercept(Invocation invocation) throws Throwable {
23. StatementHandler statementHandler = (StatementHandler) invocation.getTarget();
24. BoundSql boundSql = statementHandler.getBoundSql();
25. String sql = boundSql.getSql();
26. if (StringUtils.isBlank(sql)) {
27. return invocation.proceed();
28. }
29. //select sql do
30. if (sql.matches(SQL\_SELECT\_REGEX) && !Pattern.matches(SQL\_COUNT\_REGEX, sql)) {
31. Object obj = FieldUtils.readField(statementHandler, "delegate", true);
32. // 反射获取 RowBounds 对象。
33. RowBounds rowBounds = (RowBounds) FieldUtils.readField(obj, "rowBounds", true);
34. // 分页参数存在且不为默认值时进行分页SQL构造
35. if (rowBounds != null && rowBounds != RowBounds.DEFAULT) {
36. FieldUtils.writeField(boundSql, "sql", newSql(sql, rowBounds), true);
37. // 一定要还原否则将无法得到下一组数据(第一次的数据被缓存了)
38. FieldUtils.writeField(rowBounds, "offset", RowBounds.NO\_ROW\_OFFSET, true);
39. FieldUtils.writeField(rowBounds, "limit", RowBounds.NO\_ROW\_LIMIT, true);
40. }
41. }
42. return invocation.proceed();
43. }
44. /\*\*
45. \* 构造加入分页的SQL语句,暂时只是针对MySQL数据库
46. \* @param sql
47. \* @param rowBounds
48. \* @return SQL语句
49. \*/
50. private String newSql(String sql, RowBounds rowBounds) {
51. StringBuffer sb = new StringBuffer();
52. sb.append(sql).append(" limit ").append(rowBounds.getOffset()).append(",").append(rowBounds.getLimit());
53. return sb.toString();
54. }
55. @Override
56. public Object plugin(Object target) {
57. return Plugin.wrap(target, this);
58. }
59. @Override
60. public void setProperties(Properties properties) {
61. System.out.println(properties);
62. }
63. }

其中，@Intercepts表示拦截器，@Signature表示修改的方法信息，包括type类名，method方法，args参数。  
实现方法在intercept中，完成拦截后的SQL替换，将SQL语句中有select语句后面加入RowBound，生成LIMIT语句，谨记替换完SQL之后，需要将limit和offset替换为0。  
setProperties表示设置配置文件信息。

注：此处存在继续完善的地方，目前只支持MySQL，可以修改setProperties读取配置文件，来表示不同的数据库。之后再根据设计不同数据库下，生成不同分页的语句。

# [Excel导入和导出](http://wiki.d.xiaonei.com/pages/viewpage.action?pageId=25334522)

### 功能描述

excel作为通用的数据格式，在业务系统中大量使用，故在xweb系统中加入excel的导入和导出功能。

### 实现原理

excel的导入和导出使用的apache的POI包，导入是将excel写入到系统数据库中，导出是将系统的数据写入到excel中。

##### 导出部分的实现

1. 项目增加jar包
   * maven项目增加如下配置信息
   * <!-- poi -->
   * <dependency>
   * <groupId>org.apache.poi</groupId>
   * <artifactId>poi</artifactId>
   * <version>${poi.version}</version>
   * </dependency>
   * <dependency>
   * <groupId>org.apache.poi</groupId>
   * <artifactId>poi-ooxml</artifactId>
   * <version>${poi.version}</version>
   * </dependency>
2. 实体类字段加入@ExcelField  
   在想要导出的实体类的字段加上@ExcelField注解，其中各个属性含义为value字段值、title字段标题、type为字段导入导出、align为对齐方式、sort为是否排序。
3. @ExcelField(title = "消息", align = 0, sort = 0, value = "msg")
4. private String msg;
5. public String getMsg() {
6. return msg;
7. }
8. public void setMsg(String msg) {
9. this.msg = msg;
10. }
11. 修改spring-mvc的配置文件
12. <!-- 加入excel视图 -->
13. <bean id="excel" class="com.renren.infra.xweb.web.excel.DefaultExcelView"></bean>
14. 修改控制器Controller
15. @RequestMapping(value = "export")
16. public ModelAndView export(Model model, RedirectAttributes redirectAttributes){
17. List<Test> testList = service.getAllTest();
18. model.addAttribute("testList", testList);
19. return new ModelAndView(new DefaultExcelView("导出Test信息", "testList", Test.class));
20. }

注：其中的导出Test信息为导出的文件名，testList为传入的数据库查询之后的数据信息，Test.class为注解后的实体类。

最后，可以参数测试模块和TestController，其中包含了具体的实现代码。

##### 导入部分的实现

1. 项目增加jar包
   * maven项目增加如下配置信息
   * <!-- file upload 文件上传-->
   * <dependency>
   * <groupId>commons-fileupload</groupId>
   * <artifactId>commons-fileupload</artifactId>
   * <version>${commons-fileupload.version}</version>
   * </dependency>
   * <dependency>
   * <groupId>commons-io</groupId>
   * <artifactId>commons-io</artifactId>
   * <version>${commons-io.version}</version>
   * </dependency>
   * <!-- poi -->
   * <dependency>
   * <groupId>org.apache.poi</groupId>
   * <artifactId>poi</artifactId>
   * <version>${poi.version}</version>
   * </dependency>
   * <dependency>
   * <groupId>org.apache.poi</groupId>
   * <artifactId>poi-ooxml</artifactId>
   * <version>${poi.version}</version>
   * </dependency>
2. 页面使用了jquery-file-upload这个开源插件，目的是将文件上传到附件，增强了html的file功能，具体介绍如下，<https://github.com/blueimp/jQuery-File-Upload>  
   需要使用的jsp页面加入如下代码：
3. <!-- 引入css样式和js文件 -->
4. <link rel="stylesheet" href="${ctx}/static/jquery-fileupload/css/jquery.fileupload.css" type="text/css" />
5. <script type="text/javascript" src="${ctx}/static/jquery-fileupload/js/vendor/jquery.ui.widget.js"></script>
6. <script type="text/javascript" src="${ctx}/static/jquery-fileupload/js/jquery.iframe-transport.js"></script>
7. <script type="text/javascript" src="${ctx}/static/jquery-fileupload/js/jquery.fileupload.js"></script>
8. <script type="text/javascript">
9. $(document).ready(function(){
10. $("#fileupload").fileupload({//file的初始化代码，加入了进度条展示
11. url : '${ctx}/test/upload',
12. type : 'post',
13. dataType : 'json',
14. acceptFileTypes : '/(\.|\/)(xls?x)$/i',
15. maxFileSize : '5000000',//5M
16. maxNumberOfFiles : 10,
17. done: function (e, data) {
18. console.log(data);
19. console.log(data.result);
20. console.log(data.result.files);
21. },
22. add: function (e, data) {
23. $("#progress").css("display", "block");
24. data.submit();
25. },
26. progressall: function (e, data) {
27. $('#progress .progress-bar').css('width', '');
28. var progress = parseInt(data.loaded / data.total \* 100, 10);
29. $('#progress .progress-bar').css('width', progress + '%');
30. }
31. });
33. });
34. </script>
35. <!-- 导入按钮 -->
36. <span class="btn btn-default fileinput-button pull-right">
37. <span>导入</span>
38. <input id="fileupload" type="file" name="files" multiple="" accept="xls|xlsx" />
39. </span>
40. <!-- 进度条 -->
41. <div id="progress" class="row progress" style="display: none;">
42. <div class="progress-bar progress-bar-success"></div>
43. </div>
44. spring-mvc的配置信息修改：
45. <!-- 定义上传文件对应的MultipartResolver，支持文件上传 -->
46. <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
47. <property name="defaultEncoding" value="UTF-8" />
48. <property name="maxUploadSize" value="5242880" />
49. <property name="uploadTempDir" value="/tmp" />
50. </bean>

注：maxUploadSize为上传的最大文件，uploadTempDir为文件的临时保存路径。

1. 控制器Controller端的代码修改：
2. @RequestMapping(value = "upload")
3. @ResponseBody
4. public String upload(@RequestParam("files") MultipartFile[] files,
5. RedirectAttributes redirectAttributes) throws Exception {
6. }

注：MultipartFile表示上传的文件，之后就可以对上传的文件进行操作。

1. 最后，贴一段excel的读取代码，这部分还需要继续完善，毕竟每个用户可能上传的文件不同，可以是excel、txt等不同文件类型。
2. private void importToDataBase(InputStream stream) throws InvalidFormatException, IOException{
3. try {
4. Workbook workbook = WorkbookFactory.create(stream);
5. Sheet sheet = workbook.getSheetAt(0);//取第一个sheet
6. List<Test> testList = new ArrayList<Test>();
8. int rowStart = Math.max(1, sheet.getFirstRowNum());//从第二行开始查找
9. int rowEnd = Math.min(Integer.MAX\_VALUE, sheet.getLastRowNum());
10. for (int rownum = rowStart; rownum < rowEnd; rownum++) {
11. Row row = sheet.getRow(rownum);
12. int lastColumn = Math.max(row.getLastCellNum(), 1);
13. Test test = new Test();
14. for (int columnnum = 0; columnnum < lastColumn; columnnum++) {
15. Cell c = row.getCell(columnnum, Row.RETURN\_BLANK\_AS\_NULL);
16. if (c == null) {//记录为空
17. continue;
18. } else {//batch
19. test.setMsg(setCellValue(c).toString());
20. testList.add(test);
21. }
22. }
23. }
25. service.saveBatchTest(testList);
26. } finally{
27. try {
28. stream.close();
29. } catch (IOException e) {
30. e.printStackTrace();
31. }
32. }
33. }

注：POI使用的时候，建议大家使用WorkbookFactory.create()方法来创建，可以兼容HSSFWorkbook和XSSFWorkbook两种格式。org.apache.poi.ss.usermodel包下的为最新的类。

# [Quartz集成](http://wiki.d.xiaonei.com/pages/viewpage.action?pageId=25334524)

### 功能描述

定时触发执行相关类的方法，实现一些重复的工作。

### 实现原理

基于spring框架的定时任务大致有3种实现方式，一是使用java的Timer来定时执行，二是使用Spring的schedule来执行，最后是与Quartz集成执行定时任务。本文只是介绍如何使用Quartz与Spring集成。

### 实现

代码修改内容如下：

1. 项目增加jar包
   * maven项目增加如下配置信息.
   * <!-- Quartz -->
   * <dependency>
   * <groupId>org.quartz-scheduler</groupId>
   * <artifactId>quartz</artifactId>
   * <version>${quartz.version}</version>
   * </dependency>
   * <dependency>
   * <groupId>org.quartz-scheduler</groupId>
   * <artifactId>quartz-jobs</artifactId>
   * <version>${quartz.version}</version>
   * </dependency>
2. 修改applicationContext-quartz.xml的配置信息
3. <?xml version="1.0" encoding="UTF-8"?>
4. <beans xmlns="http://www.springframework.org/schema/beans"
5. **xmlns:xsi**="http://www.w3.org/2001/XMLSchema-instance"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd"
7. default-lazy-init="true">
8. <description>quartz定时任务配置</description>
9. <!-- 执行定时任务的Service服务类 -->
10. <bean id="initializeDBScheduler" class="com.renren.infra.xweb.service.SchedulerService" />
11. <!-- Spring与Quartz集成的FactoryBean-->
12. <bean id="schedulerFactory"
13. class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
14. <property name="triggers">
15. <list>
16. <ref bean="initializeDBCronTrigger" />
17. </list>
18. </property>
19. <!-- 配置Quartz初始化信息 -->
20. <property name="quartzProperties">
21. <props>
22. <prop key="org.quartz.threadPool.threadCount">5</prop>
23. </props>
24. </property>
25. <!-- 启动时延期2秒开始任务 -->
26. <property name="startupDelay" value="2" />
27. </bean>
28. <!-- 触发执行任务的Cron表达式 -->
29. <bean id="initializeDBCronTrigger" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
30. <property name="jobDetail" ref="initializeDBJob" />
31. <!-- 每天凌晨一点执行任务 -->
32. <property name="cronExpression" value="0 0 1 \* \* ?" />
33. </bean>
34. <!-- 服务类的定时任务触发的实现方法 -->
35. <bean id="initializeDBJob"
36. class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
37. <property name="targetObject" ref="initializeDBScheduler" />
38. <property name="targetMethod" value="initializeDB" />
39. </bean>
40. </beans>
41. 具体的定时任务触发方法
42. public class SchedulerService {
43. private static final Logger logger = LoggerFactory.getLogger(SchedulerService.class);
44. @Autowired
45. private TestMybatisDao testDao;
46. /\*\*
47. \* 初始化DB
48. \*/
49. public void initializeDB() {
50. initializeTest();
51. logger.info(new Date() + ": refresh data finish.");
52. }
53. /\*\*
54. \* 初始化Test表
55. \*/
56. private void initializeTest() {
57. Test test = new Test();
58. test.setId(1L);
59. test.setMsg("test");
60. testDao.deleteAll();
61. testDao.insert(test);
62. logger.info(new Date() + ": initialize table Test.");
63. }
64. }

注：Quartz会根据applicationContext-quartz配置文件中的Cron表达式值执行对应SchedulerService下的initializeDB()方法。

1. 关于Quartz框架  
   Quartz是一个非常优秀的定时任务框架，介绍地址为：<http://www.quartz-scheduler.org/>  
   有三个核心的类即Job、Trigger、Scheduler，分别表示作业、触发器和执行器。Job的实现类对需要执行的具体任务，Trigger为触发任务执行的规则，可以是Cron或是日期时间等，最后封装到Scheduler中，由Scheduler触发执行任务。  
   Scheduler中的scheduleJob()方法表示装配job和trigger，start()表示触发任务，shutdown()表示关闭作业。

# [发送邮件实现](http://wiki.d.xiaonei.com/pages/viewpage.action?pageId=25335447)

### 功能描述

发送邮件功能是一个web项目必备可少的功能点，xweb使用的是spring发送邮件功能，做了一层封装，底层使用的仍旧java mail的jar包。

### 代码路径

内容位于src/main/java下com.renren.infra.xweb.util.mail包下，SimpleMailService表示发送纯文本邮件，MimeMailService表示发送带html或是附件的邮件。  
测试用例位于src/test/java下的com.renren.infra.xweb.util.mail包下的MailServiceTest，测试了发送纯文本邮件，发送带有html内容的邮件，带有附件的邮件。  
注：此处打算集成GreenMail实现邮件测试的自动发送和接收测试。

### 实现

代码修改内容如下：

1. 项目增加jar包
   * maven项目增加如下配置信息
   * <!-- email -->
   * <dependency>
   * <groupId>javax.mail</groupId>
   * <artifactId>mail</artifactId>
   * <version>${mail.version}</version>
   * </dependency>
   * <dependency>
   * <groupId>javax.activation</groupId>
   * <artifactId>activation</artifactId>
   * <version>${activation.version}</version>
   * </dependency>
   * <!-- spring support -->
   * <dependency>
   * <groupId>org.springframework</groupId>
   * <artifactId>spring-context-support</artifactId>
   * <version>${spring.version}</version>
   * </dependency>
2. 修改applicationContext-email.xml的配置信息
3. <!-- mail配置信息 -->
4. <bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
5. <property name="host" value="smtp.renren-inc.com" />
6. <property name="username" value="邮箱" />
7. <property name="password" value="密码" />
8. <property name="defaultEncoding" value="UTF-8" />
9. <property name="javaMailProperties">
10. <props>
11. <prop key="mail.smtp.auth">true</prop>
12. <!-- 使用人人邮箱无需配置此处，如果使用gmail，请配置如下信息：<prop key="mail.smtp.starttls.enable">true</prop> -->
13. </props>
14. </property>
15. </bean>
16. 如果想要发送html格式的附件，可以自己手写html代码，然后传入到对应的SimpleMailMessage实例中。  
    也可以使用Freemarker模板来生成html代码，将Freemarker的模板拷贝到src/main/resources/email目录下，传入文件名和参数，即可生成html代码。
17. SimpleMailMessage message = buildSimpleMessage();
18. String filename = "mailTemplate.ftl";
19. Map<String, String> params = Collections.singletonMap("userName", "yong.cao");
20. mimeMailService.sendMailTemplate(message, filename, params);

注：SimpleMailMessage包括发送人，收件人，内容，标题等信息。

1. 如果想要发送带有附件的邮件，需要将附件拷贝到src/main/resources/email目录下。
   * 单一附件
   * SimpleMailMessage message = buildSimpleMessage();
   * String[] files = new String[] { EMAIL\_ATTACHMENT };//附件名
   * mimeMailService.sendMailFiles(message, files);
   * 多附件
   * SimpleMailMessage message = buildSimpleMessage();
   * String[] files = new String[] { EMAIL\_ATTACHMENT, EMAIL\_ATTACHMENT };
   * mimeMailService.sendMailFiles(message, files);
2. 控制器层Contoller的使用  
   在相应的Controller中，使用@Autowired注入相应的邮件Service，调用其中的方法。
3. @Autowired
4. private SimpleMailService simpleMailService;
5. @Autowired

private MimeMailService mimeMailService;