力软敏捷开发框架特点

一． 框架简介：

软件开发，程序员就是不断地跟变量、方法、类、接口这些东西打交道，随着开发经验的积累，聪明的程序就会发现虽然开发出来的每个软件都不一样，但是它们在很多地方又都是一样的。每个软件的底层差不多都需要进行增删改查、文件操作、权限管理等等。最后才是根据需求把这些底层功能组合包装起来做成一个成品软件，这个底层就是传统意义上的软件开发框架。

力软敏捷开发框架就是在此基础上做了充分的优化，使开发者只用专注于业务功能的实现，便捷快速的完成开发任务。

力软敏捷开发框架有以下特点：

1. 框架的主架构为基于.net MVC 的 BS 架构。
2. 后台 ORM 支持 EF 和 dapper 两种模式。
3. 用于实现各类业务系统，如 OA、ERP、MIS、CRM、电商平台等系统的开发。框架本身是一个可二次开发的开发平台，开发者可以根据开发向导进行配置直接生成功能模块；但是他又是一套源代码，开发者也可以直接在 VS 中基于框架做发，甚至还可以对开发框架进行开发扩展。
4. 强大的权限管理组件，完成业务功能开发后，系统可以直接使用通用权限来管理业务功能的操作权限及数据权限。
5. 集成工作流引擎组件，使业务流程灵活可控。
6. 集 BS 开发、微信组件、APP 开发组件于一体，一套框架帮您解决所有问题。
7. 使用力软敏捷开发框架能帮开发者节约开发成本、提高开发效率、提升软件质量、缩短开发周期。

二． 核心优势。

1. 快速开发。开发框架中有多套开发模板，选择模板后按照向导操作可自动产生包含界面在内的所有代码。如果需要二次开发，可以直接修改生成的源代码。

说得夸张一些如果产品经理拿这套框架进行建模，模型出来了软件也基本就开发出来了，在编码效率上至少比传统开发提高 90%的效率。

2. 界面风格简洁、大气、操作便捷，非常适合中国人的审美观念，直接提升软件的印象分。

3. 采用 SOA 架构，系统核心功能均可以通过服务的方式提供给外部调用，方便系统与 ERP 系统及周边系统、硬件设备接口交互能力，解决企业信息化孤岛问题。

4. 强大的 UI 组件。UI 层基于 Jquery+Div+css+ajax 开发，没有采用第三方 UI 框架使得 UI 保持了简洁轻巧，重要的是不用支付额外的 UI 版权费用。另外也集成了大量的如勾选框、文本输入框、动态下拉框、树型组件、Grid、翻页、数据、条件查询、导出、下载等组件，即使您不是前端高手，也可以把前端做得很炫酷。

5. 开发平台稳定、成熟、高效。框架已通过上海计算机软件技术开发中心评测。

6. 系统前后台通 Ajax 交互，这样使得前台不必依赖于后台的开发语言，日后要重构成 php、JSP 后台的话，UI 层完全不用动。

7. 开发框架完美支持 Oracle、SQL Server、MYSQL 数据库。并且还支持在框架中同时操作多数据库，灵活性、稳定性都非常好。

8. 提供自定义报表功能。可通过编写 SQL、Procedure 作为数据来源进行构建不同类型的图形报表。

9. 提供工作流引擎组件，开发者可以直接在开发框架中使用自定义表单来承载业务数据进行流程审批；也可以编写代码完成复杂表单然后调用流程引擎服务进行流程审批。

10. 提供强大的权限管理组件，基于框架开发出功能后就可以直接给被授权角色授予该功能的权限。

11. 提供 SSO（单点登陆）服务，方便多系统统一登陆管理。

12. 提供微信企业号开发组件功能、让您的系统更贴近用户。

13. 提供 APP 开发功能，是的，普通的开发人员也可以基于力软敏捷开发框架进行 APP 开发了，您不用再支付高额的工资聘请安卓和 IOS 开发人员。

14. 框架支持 Websocket，让通讯变得更即时。

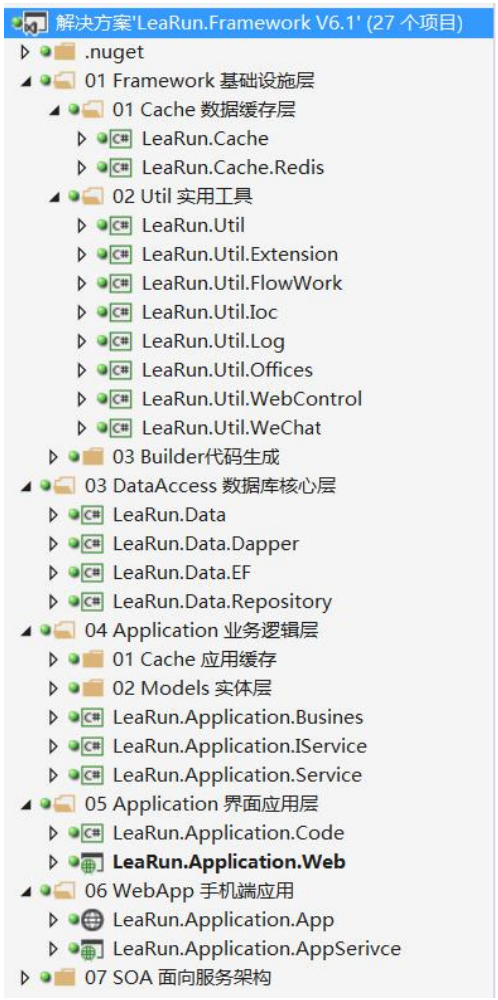
15. 框架支持 redis 缓存集群，让您的系统飞起来。

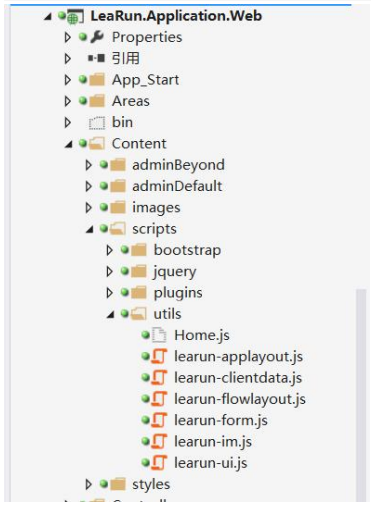
16. 优质的售后服务，由本框架的原班开发人员为您提供售后支持。

三． 开发示例

3.1 框架整体代码层次

整体采用多层工厂/依赖注入模式。



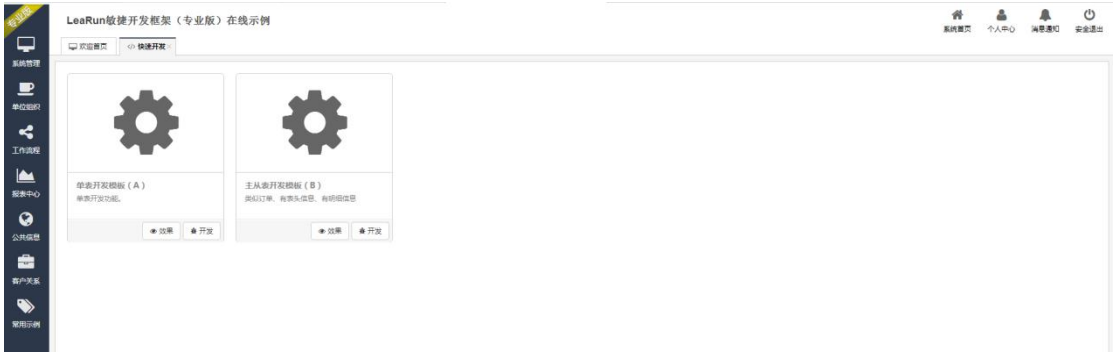


3.2 开发示例

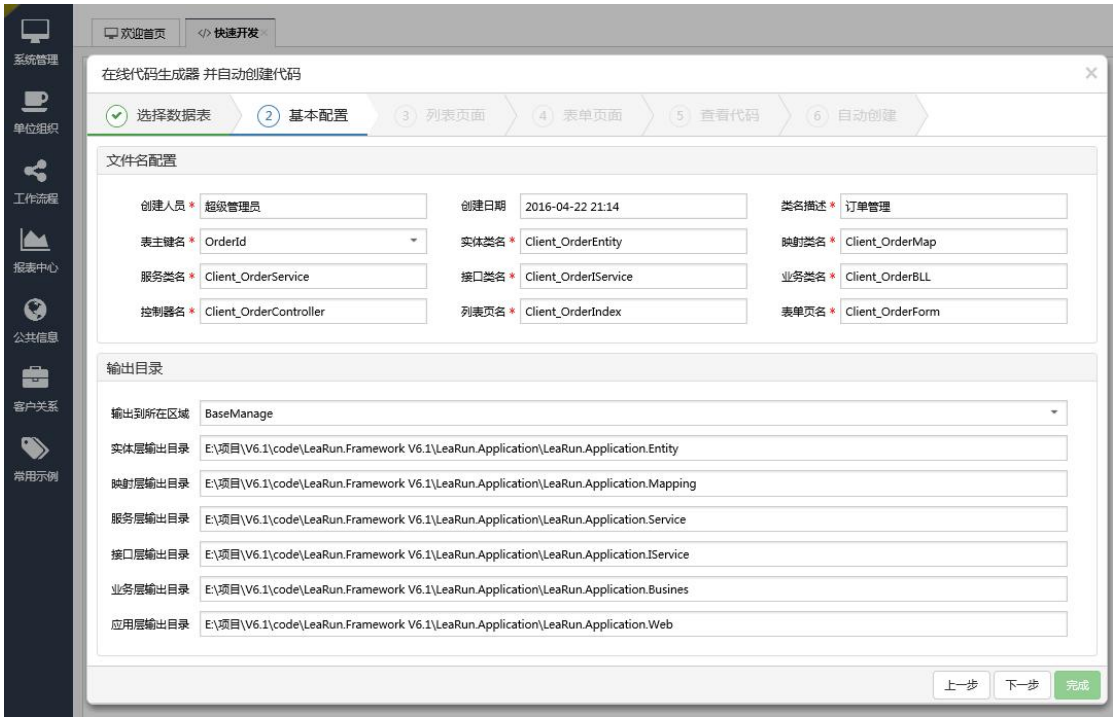
在用力软敏捷开发框架进行快速开发时有两种开发模式，一种是纯自定义表单无需编译的，一种是需要生成代码，重新编译的。

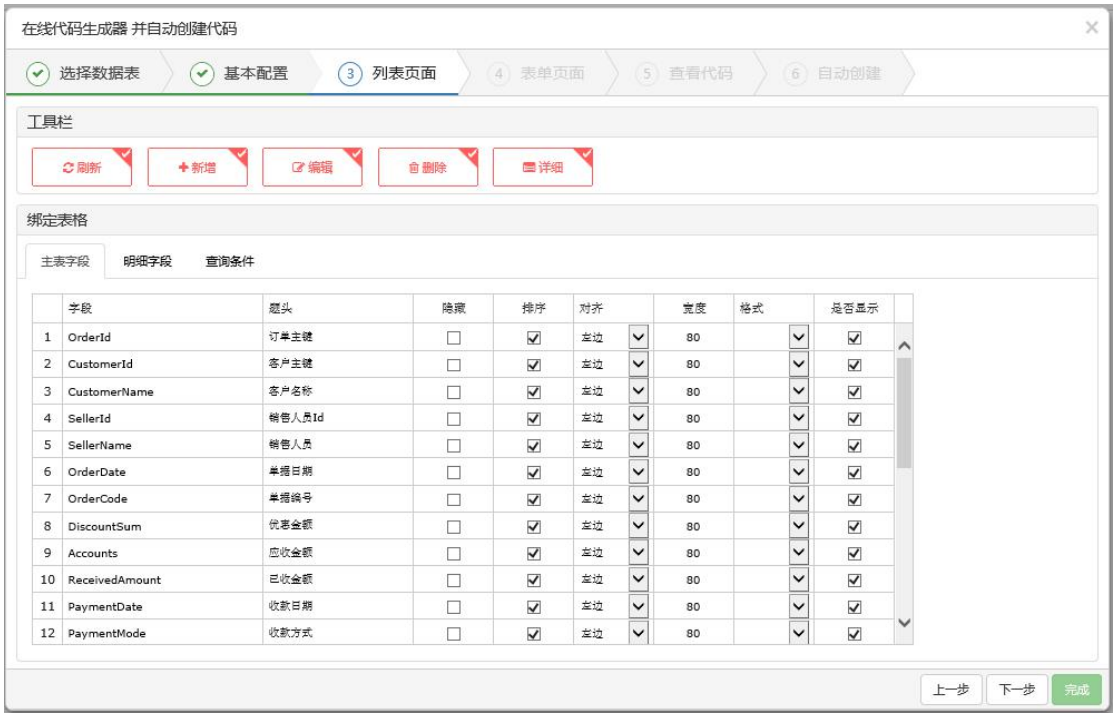
3.2.1 代码生成开发模式

1、选择一种开发向导

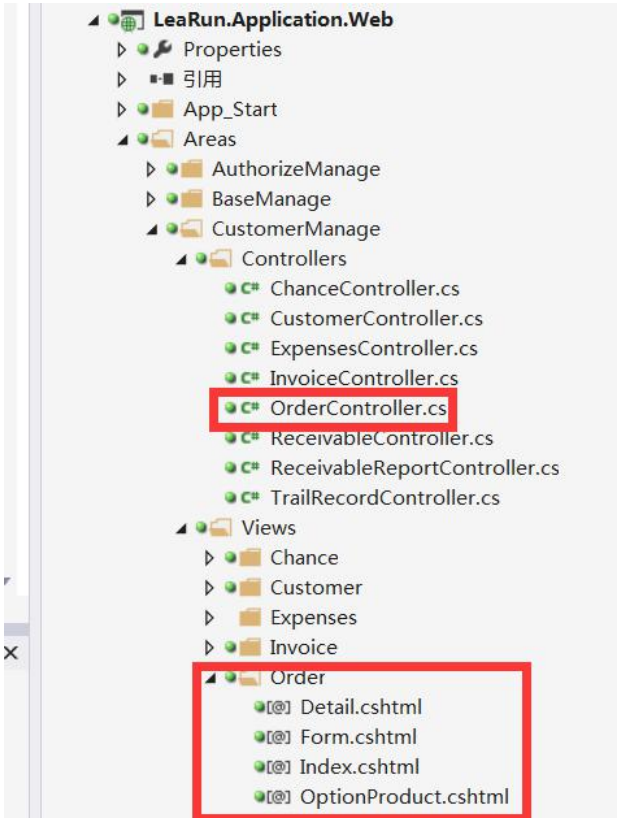


2、指定数据源、对各项开发参数进行设置

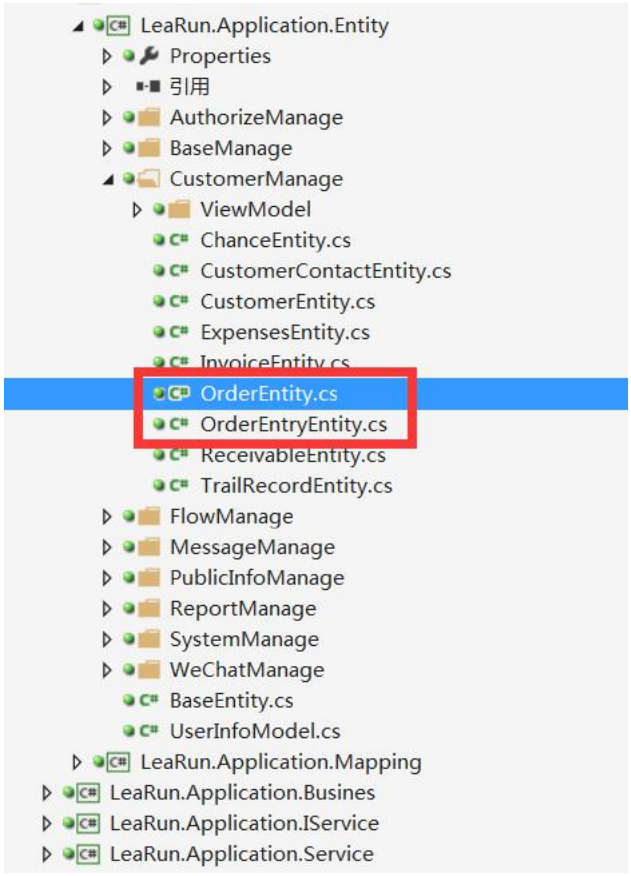




跟着开发向导一步步设置就可以自动生成代码 ,代码会根据开发者的设置放入到指定项目的指定位置。标准的 MVC 架构，表示层代码在 LeaRun.Application.Web 项目下。



实体层代码被自动放置在 Entity 下



下面是实体层代码，一看就能明白。

```
namespace LeaRun.Application.Entity.CustomerManage
{
    /// <summary>
    /// 版 本
    /// Copyright (c) 2013-2016 上海力软信息技术有限公司
    /// 创 建：佘赐雄
    /// 日 期：2016-03-16 13:54
    /// 描 述：订单管理
    /// </summary>
    24 个引用
    public class OrderEntity : BaseEntity
    {
        #region 实体成员
        /// <summary>
        /// 订单主键
        /// </summary>
        /// <returns></returns>
        5 个引用
        public string OrderId { get; set; }
        /// <summary>
        /// 客户主键
        /// </summary>
        /// <returns></returns>
        0 个引用
        public string CustomerId { get; set; }
        /// <summary>
        /// 客户名称
        /// </summary>
        /// <returns></returns>
        3 个引用
        public string CustomerName { get; set; }
```

下面是业务逻辑层，这里是按工厂模式生成的，当然框架里已经提供了 IOC 容器也可以直接调整成依赖注入模式。

```
namespace LeaRun.Application.Busines.CustomerManage
{
    /// <summary>
    /// 版 本 6.1
    /// Copyright (c) 2013-2016 上海力软信息技术有限公司
    /// 创 建：佘赐雄
    /// 日 期：2016-03-16 13:54
    /// 描 述：订单管理
    /// </summary>
    2 个引用
    public class OrderBLL
    {
        private IOrderService service = new OrderService();

        获取数据

        提交数据
    }
}
```

接口层代码

```
namespace LeaRun.Application.IService.CustomerManage
{
    /// <summary>
    /// 版 本 6.1
    /// Copyright (c) 2013-2016 上海力软信息技术有限公司
    /// 创 建：佘赐雄
    /// 日 期：2016-03-16 13:54
    /// 描 述：订单管理
    /// </summary>
    3 个引用
    public interface IOrderService
    {
        获取数据

        提交数据
    }
}
```

数据访问层，数据工厂已经将对数据库的访问提供了 EF 及 Dapper 这两种 ORM 的封，绝大部分情况下不需要写 SQL 语句，普通的 Lambda 表达式即可完成各种查询，代码整洁，可读性很好。

```
2 个引用
public class OrderService : RepositoryFactory<OrderEntity>, IOrderService
{
    private ICodeRuleService coderuleService = new CodeRuleService();

    #region 获取数据
    /// <summary>
    /// 获取列表
    /// </summary>
    /// <param name="pagination">分页</param>
    /// <param name="queryJson">查询参数</param>
    /// <returns>返回分页列表</returns>
    2 个引用
    public IEnumerable<OrderEntity> GetPageList(Pagination pagination, string queryJson)...
    /// <summary>
    /// 获取实体
    /// </summary>
    /// <param name="keyValue">主键值</param>
    /// <returns></returns>
    4 个引用
    public OrderEntity GetEntity(string keyValue)
    {
        return this.BaseRepository().FindEntity(keyValue);
    }
    /// <summary>
    /// 获取前单、后单 数据
    /// </summary>
    /// <param name="keyValue">主键值</param>
    /// <param name="type">类型（1-前单；2-后单）</param>
    /// <returns>返回实体</returns>
    2 个引用
    public OrderEntity GetPrevOrNextEntity(string keyValue, int type)
    {
        OrderEntity entity = this.GetEntity(keyValue);
        if (type == 1)
        {
            entity = this.BaseRepository().IQueryable().Where(t => t.CreateDate >entity.CreateDate).OrderBy(t => t.CreateDate).FirstOrDefault();
        }
        else if (type == 2)
        {
```

如果需要换成依赖注入模式，只需在 IOC 配置文件注册即可

```
<?xml version="1.0" encoding="utf-8"?>
<unity>
  <typeAliases>
    <typeAlias alias="IDatabase" type="LeaRun.Data.IDatabase,LeaRun.Data" />
    <typeAlias alias="EFDatabase" type="LeaRun.Data.EF.Database,LeaRun.Data.EF" />
    <typeAlias alias="IDbContext" type="LeaRun.Data.EF.IDbContext,LeaRun.Data.EF" />
    <typeAlias alias="MySql" type="LeaRun.Data.EF.MySqlDbContext,LeaRun.Data.EF" />
    <typeAlias alias="SqlServer" type="LeaRun.Data.EF.SqlServerDbContext,LeaRun.Data.EF" />
  </typeAliases>
  <containers>
    <container name="DBcontainer">
      <type type="IDatabase" mapTo="EFDatabase" ></type >
      <!-- 默认数据库软件类型：SqlServer,MySql,Oracle,Access,SQLite-->
      <type type="IDbContext" mapTo="SqlServer" ></type >
      <type type="IDbContext" mapTo="SqlServer" name="SqlServer"></type >
      <type type="IDbContext" mapTo="MySql" name="MySql"></type >
    </container>
  </containers>
</unity>
```

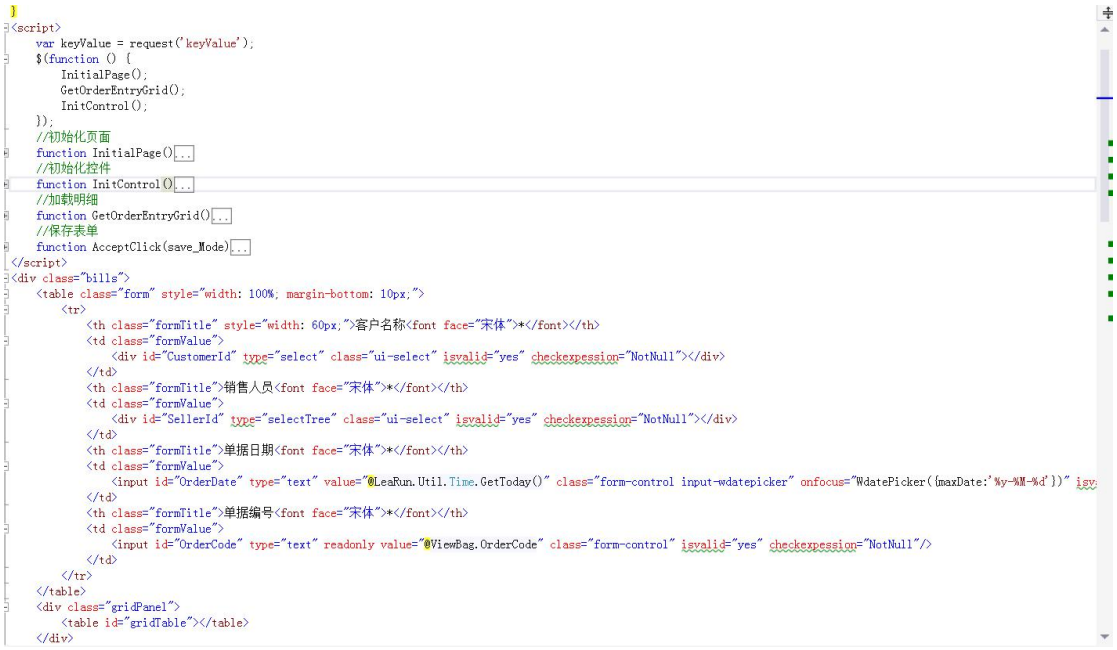
下面是 MVC 中的视图层

```
<script>
    $(function () {
        InitialPage();
        GetGrid();
    });
    //初始化页面
    function InitialPage()...
    //加载表格
    function GetGrid() {
        var selectedRowIndex = 0;
        var $gridTable = $('#gridTable');
        $gridTable.jqGrid({
            url: "../../CustomerManage/Order/GetPageListJson",
            postData: { queryJson: JSON.stringify($("#filter-form").GetWebControls()) },
            datatype: "json",
            height: $(window).height() - 136.5,
            autowidth: true,
            colModel: [
                { label: '主键', name: 'OrderId', hidden: true },
                { label: "单据日期", name: "OrderDate", width: 100, align: "left", formatter: "date", formatoptions: { newformat: 'Y-m-d' } },
                { label: "单据编号", name: "OrderCode", width: 130, align: "left" },
                { label: "客户名称", name: "CustomerName", width: 250, align: "left" },
                { label: "销售人员", name: "SellerName", width: 80, align: "left" },
                { label: "优惠金额", name: "DiscountSum", width: 80, align: "left", formatter: 'number', formatoptions: { thousandsSeparator: "", decimalPlaces
                { label: "收款金额", name: "Accounts", width: 80, align: "left", formatter: 'number', formatoptions: { thousandsSeparator: "", decimalPlaces: 2
                {
                    label: "收款方式", name: "PaymentMode", width: 80, align: "center",
                    formatter: function (cellvalue, options, rowObject) {
                        return top.clientdataItem["Client_PaymentMode"][cellvalue];
                    }
                },
```

前后端通过 ajax+json 交互。就像上面，后台返回的 json 数据，很简单的就绑定到了表格上。像数据字典的也不用写 SQL 关联，这里的数据字典，直接就可以显示来名称。当然这些代码都是可以生成出来的。需要二次开发的话可以直接修改这些代码。

```
<div class="titlePanel">
    <div class="title-search">
        <table>...</table>
    </div>
    <div class="toolbar">
        <div class="btn-group">
            <a id="lr-replace" class="btn btn-default" onclick="reload();"><i class="fa fa-refresh"></i> 刷新</a>
            <a id="lr-add" class="btn btn-default" onclick="btn_add()"><i class="fa fa-plus"></i> 新增</a>
            <a id="lr-edit" class="btn btn-default" onclick="btn_edit()"><i class="fa fa-pencil-square-o"></i> 编辑</a>
            <a id="lr-delete" class="btn btn-default" onclick="btn_delete()"><i class="fa fa-trash-o"></i> 删除</a>
            <a id="lr-detail" class="btn btn-default" onclick="btn_detail()"><i class="fa fa-list-alt"></i> 详细</a>
        </div>
        <script>$('.toolbar').authorizeButton()</script>
    </div>
</div>
<div class="gridPanel">
    <table id="gridTable"></table>
    <div id="gridPager"></div>
</div>
```

下面是表单页里的内容



```
}
<script>
    var keyValue = request('keyValue');
    $(function () {
        InitialPage();
        GetOrderEntryGrid();
        InitControl();
    });
    //初始化页面
    function InitialPage()...
    //初始化控件
    function InitControl()...
    //加载明细
    function GetOrderEntryGrid()...
    //保存表单
    function AcceptClick(save_Mode)...
</script>
<div class="bills">
    <table class="form" style="width: 100%; margin-bottom: 10px;">
        <tr>
            <th class="formTitle" style="width: 60px;">客户名称<font face="宋体">*</font></th>
            <td class="formValue">
                <div id="CustomerId" type="select" class="ui-select" isvalid="yes" checkexpession="NotNull"></div>
            </td>
            <th class="formTitle">销售人员<font face="宋体">*</font></th>
            <td class="formValue">
                <div id="SellerId" type="selectTree" class="ui-select" isvalid="yes" checkexpession="NotNull"></div>
            </td>
            <th class="formTitle">单据日期<font face="宋体">*</font></th>
            <td class="formValue">
                <input id="OrderDate" type="text" value="@LeaRun.Util.Time.GetToday()" class="form-control input-wdatepicker" onfocus="WdatePicker({maxDate:'%y-%M-%d'})" isv
            </td>
            <th class="formTitle">单据编号<font face="宋体">*</font></th>
            <td class="formValue">
                <input id="OrderCode" type="text" readonly value="@ViewBag.OrderCode" class="form-control" isvalid="yes" checkexpession="NotNull"/>
            </td>
        </tr>
    </table>
    <div class="gridPanel">
        <table id="gridTable"></table>
    </div>
```

```
        //客户名称
        $("#CustomerId").ComboBox({
            url: "../../CustomerManage/Customer/GetListJson",
            id: "CustomerId",
            text: "FullName",
            description: "==请选择==",
            height: "360px",
            width: "280px",
            allowSearch: true
        });
        //销售人员
        $("#SellerId").ComboBoxTree({
            url: "../../BaseManage/User/GetTreeJson",
            description: "==请选择==",
            height: "360px",
            width: "280px",
            allowSearch: true
        });
        //收款方式
        $("#PaymentMode").ComboBox({
            url: "../../SystemManage/DataItemDetail/GetDataItemListJson",
            param: { EnCode: "Client_PaymentMode" },
            id: "ItemValue",
            text: "ItemName",
            description: "==请选择==",
            height: "200px"
        });
```

其实里面很多功能是组件化的，像单据编码，就是通才编码规则生成的。那种下拉框的数据绑定很简单在前端只用一句代码就解决了，就像上图

看一下效果，订单管理，主从表结构的，这个是列表页


LeaRun敏捷开发框架（专业版）在线示例
系统首页
个人中心
消息通知
安全退出
欢迎首页
客户订单
系统管理
单位组织
工作流程
报表中心
公共信息
客户关系
常用示例
查询条件
2016-04-15 至 2016-04-22
今天
近7天
近1个月
近3个月
刷新
新增
编辑
删除
详细
单据日期
单据编号
客户名称
销售人员
优惠金额
收款金额
收款方式
收款状态
制单人员
备注
1
2016-04-22
XS-20160422001
百世达
刘晓雷
0.00
0.00
现金
未收款
超级管理员
第 1 页 / 共 1 页
30
检索到 1 条记录，显示 第 1 条 - 第 1 条，查询耗时 78 毫秒

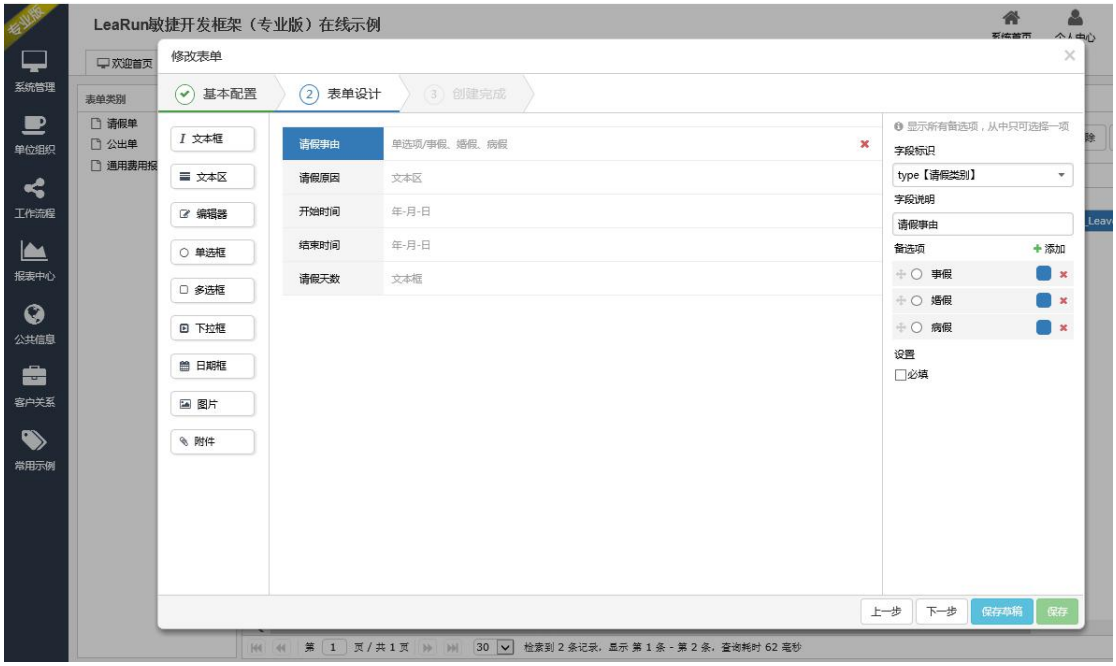
这个是表单录入的界面。


LeaRun敏捷开发框架（专业版）在线示例
系统首页
个人中心
消息通知
安全退出
欢迎首页
客户订单
编辑订单
保存并新增
保存单据
系统管理
单位组织
工作流程
报表中心
公共信息
客户关系
常用示例
客户名称 * 百世达
销售人员 * 刘晓雷
单据日期 * 2016-04-22 00:00:00
单据编号 * XS-20160422001
商品信息
价格信息
说明信息
商品名称
扫描枪录入
商品编号
单位
数量
单价
金额
税率(%)
含税单价
税额
含税金额
1 敏捷开发框架-企业增强版 lr-adms-04 套 1.00 2000000.00 2000000.00 0 2000000.00 0.00 2000000.00
2 敏捷开发框架-企业基础版 lr-adms-03 套 1.00 100000.00 100000.00 0 100000.00 0.00 100000.00
合计: 2.00 2100000.00 2100000.00 2100000.00 0.00 2100000.00
暂无备注信息
优惠金额 * 0.00
收款金额 * 0.00
收款日期 * 2016-04-22 00:00:00
收款方式 * 现金
销售费用 * 0.00
制单人员 * 超级管理员
合同编号
合同附件
摘要信息

3.2.2 表单设计器开发模式

表单设计器开发模式比较适合没有编程基础或者业务逻辑相对简单的功能开发。

看一下表单设置器



就是这样在框架开发平台里拖拖控件就好了。

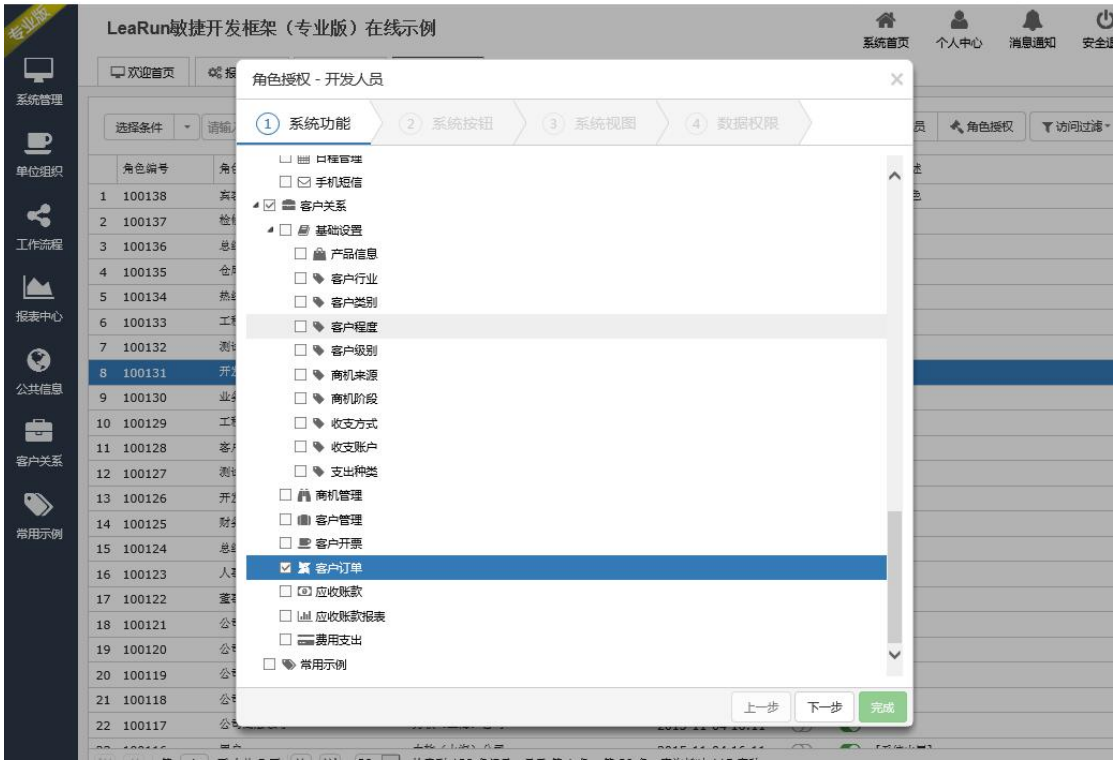
3.2.3 报表开发

在框架开发平台里，只用输入 SQL 语句也可以完成图形报表的开发。

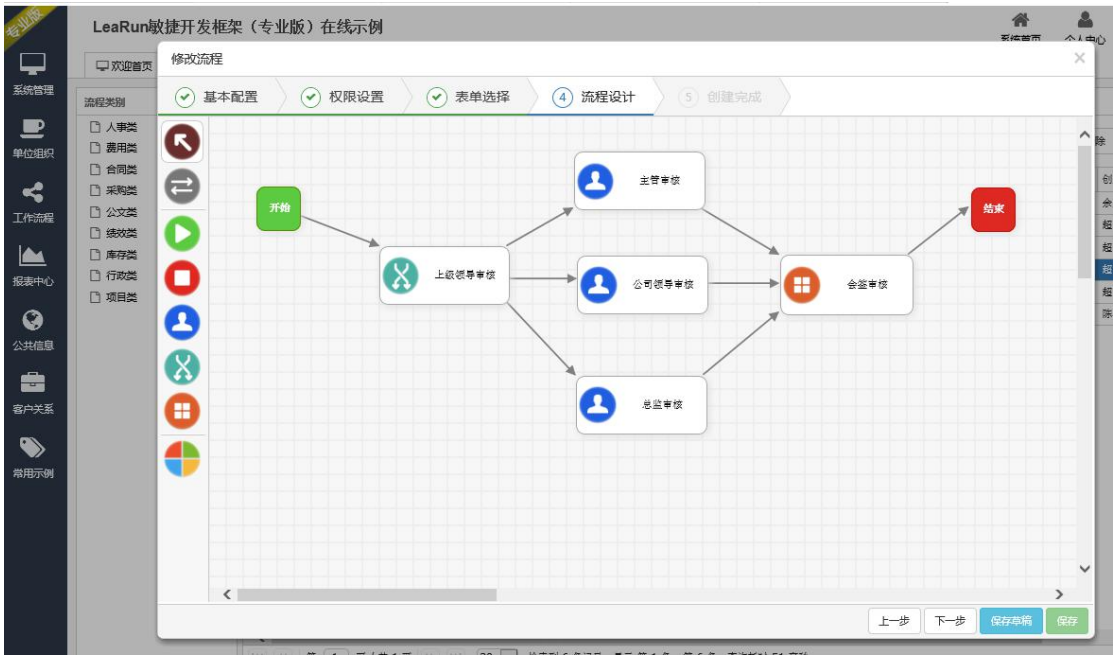
按照上图的开发向导进行设置，直接就可以得到如下的图形报表，图形组件是用的开源免费的

echarts。

开发完成后就可以直接把开发出来的功能模块进行授权操作



这个是框架内置的工作流组件，可以进行可视化流程设计。



四.　涉及第三方插件组件清单

后端

ASP.NET MVC4
EntityFramework　ORM
Dapper　ORM

NPOI Excel 操作

log4net　系统日志

Newtonsoft.Json Json 处理

signalR Websocket

unity 依赖注入容器

前端

JS 框架：Jquery-1.10.2.min、jquery-ui

CSS 框架：Bootstrap

数据表格:JqGrid

分页插件：　pagination

上传文件：Uploadify

布局：Layout

客户端验证：JQuery Validation

图表：Highcharts、echarts

字体图片：Font Awesome

富文本：ckeditor

日期控件： My97DatePicker

树结构控件：jQuery WTree

对话框：layer

代码编辑器：syntaxhighlighter

日程插件：fullcalendar

工作流流程图：flow.js

工作流表单富文本： simditor

手机端

WebApi 接口：nancy

Js 框架：ionic　angularjs mui

五． 开发及部署环境

开发环境

vs2012 及以上。

sqlserver2005\oracle11g\mysql4.5 及以上版本。

服务器端

操作系统：Microsoft Windows Server 2008R2 及以上

其它软件：IIS 7.0、.netframwork4.5