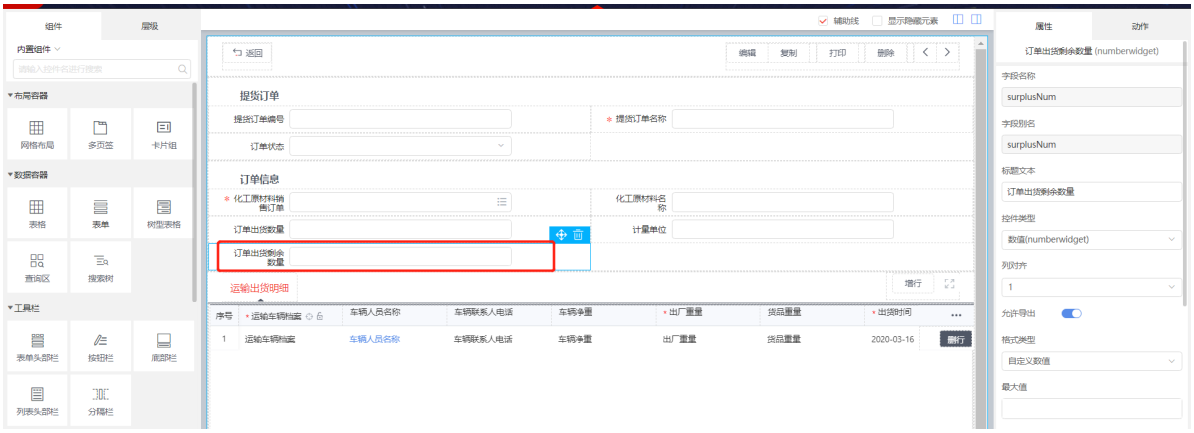
# 业务场景



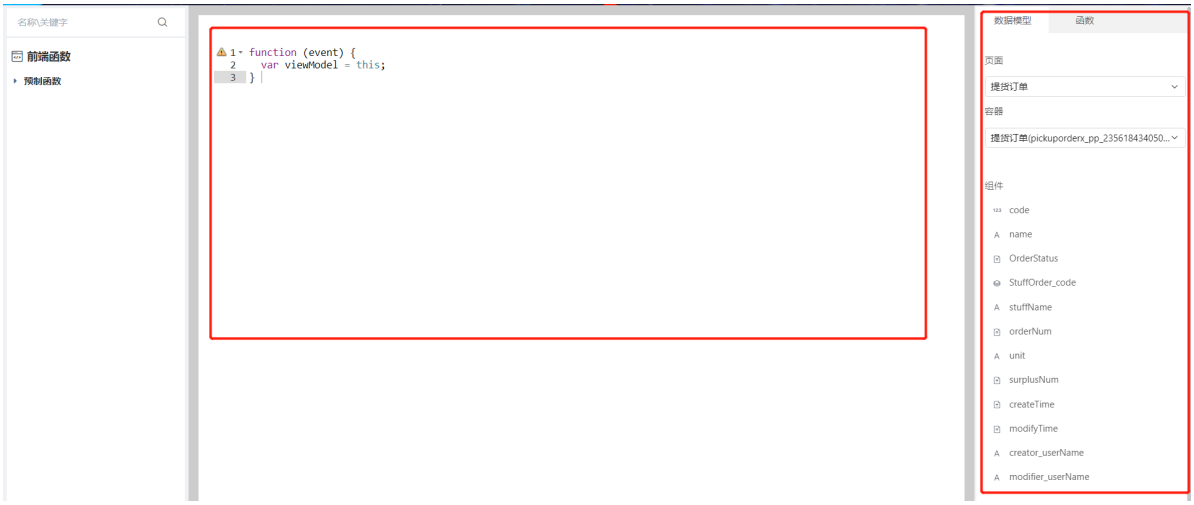
# 1.计算订单出货剩余数量

## 1.1 通过子行运输出货明细数据来改变订单出货剩余数量

点击【货物重量】——选择【动作】——点击【值改变-点击设置】——选择【前端函数】

选择【前端函数】之后，如下图所示：

点击【配置】后，如图所示：




注意：

左侧区域：预知函数；中间区域：编写使用函数；右侧区域：数据模型控件名称

## 1.2 低代码开发实现

常用函数：

```
//获取控件对象
let 控件对象 = viewModel.get('控件名称')
//获取控件的value值
let val = 控件对象.getValue();
//设置控件的value值
控件对象.setValue(值);
```

代码

```
function (event) {
  var viewModel = this;
  //ShippingDetails_ppList
  //获取所有子元素数据
  var ShippingDetails_ppList =
viewModel.get('ShippingDetails_ppList').getData();
  //获取商品重量集合
  let list = ShippingDetails_ppList.map(e => {return e.stuffWeight});
  //商品重量和
  let stuffWeightSum =list.reduce((pre,cur)=>{
    return pre+cur;
  },0);
  //获取订单数量
  let orderNum =  viewModel.get('orderNum').getValue();
  //计算出剩余数量
  let  surplusNum = orderNum -  stuffWeightSum;
  viewModel.get('surplusNum').setValue(surplusNum);
  if(surplusNum < 0){
    cb.utils.alert("出货运输总量不能超过订单数量");
    return false;
  }
}
```

编辑函数，最后保存函数


注意：不保存函数功能无法实现

# 1.3 功能测试

点击页面预览


输入数据

订单出货剩余数量 = 订单出货数量 - 货品重量和


# 2.设置车辆出厂重量>车辆净重量


# 3.每次运输货物重量不能小于0

```
function (event) {
  var viewModel = this;
  //保存之前   每次运输货物重量不能小于0
  viewModel.on('beforeSave',function(args){
    //console.log("----------------------");
    //获取子选项数据
    let data  = viewModel.get('ShippingDetails_ppList').getData();
    if(data){
       //获取所有的货物重量
      let stuffWeightList =  data.map(e =>{return e.stuffWeight});
      //排序函数
      stuffWeightList.sort();
      if(stuffWeightList[0] < 0){
        cb.utils.alert("商品重量不能为负数");
        return false;
      }
    }

    //剩余出货量大于等于零
    let surplusNum =viewModel.get('surplusNum').getValue();
    if(surplusNum < 0){
      cb.utils.alert("商品剩余重量不能为负数");
      return false;
    }

  });
}
```


# 4.车辆号牌唯一性校验

需求：车辆号牌唯一性校验，需使用后端函数，再数据保存之前验证

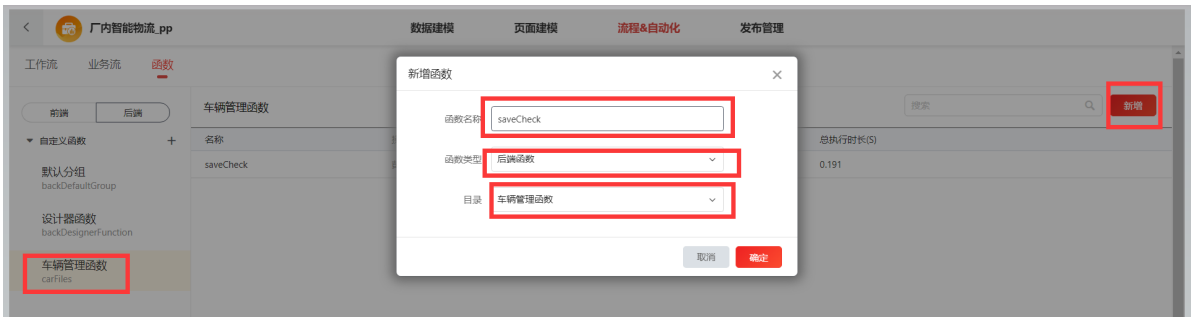
1.创建后端函数

创建函数管理文件夹

【流程&自动化】——【函数】——【后端】——【+】新增——【车辆管理函数】文件夹

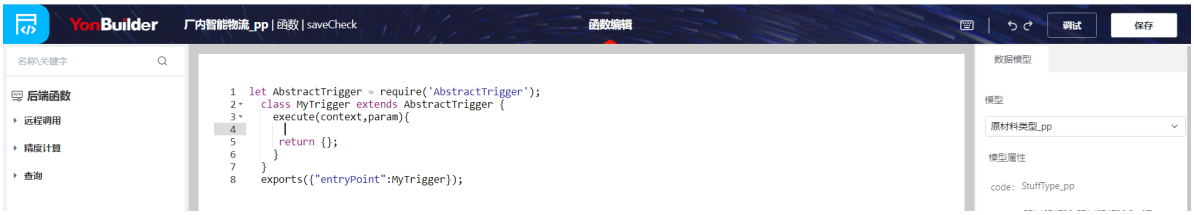

点击【新增】创建函数

函数名称：saveCheck 函数类型：后端函数 目录：我们前面创建的【车辆管理函数】




设计函数


进入函数如图所示：




编写函数：

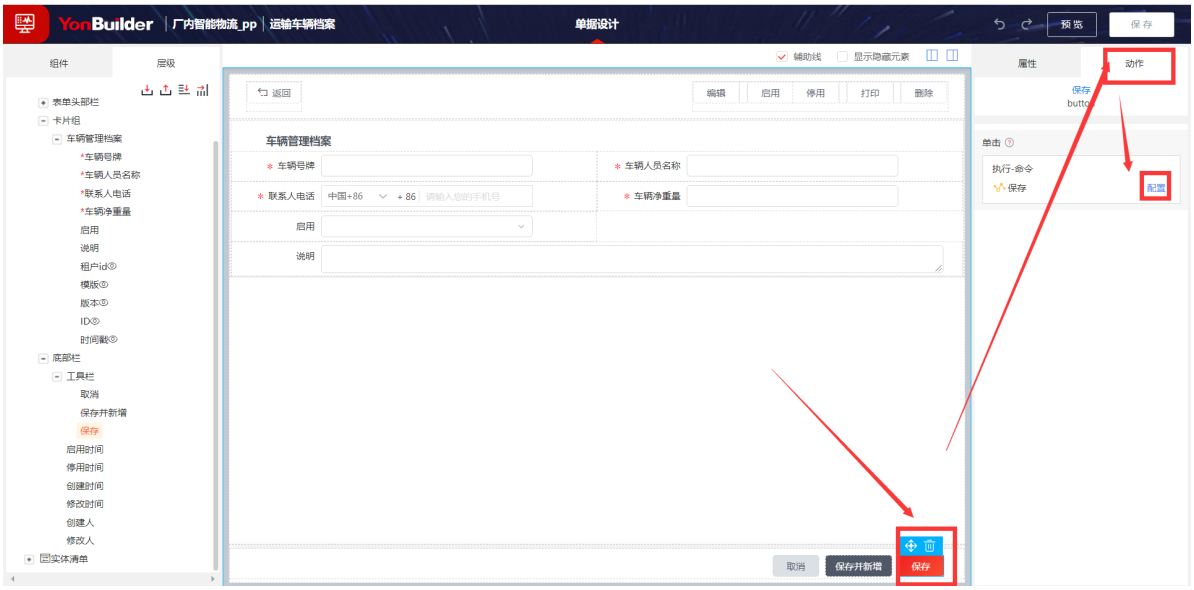
代码

```
let AbstractTrigger = require('AbstractTrigger');
  class MyTrigger extends AbstractTrigger {
    execute(context,param){
      //在动作前可以获取当前业务数据
      let data = param.data[0];
```

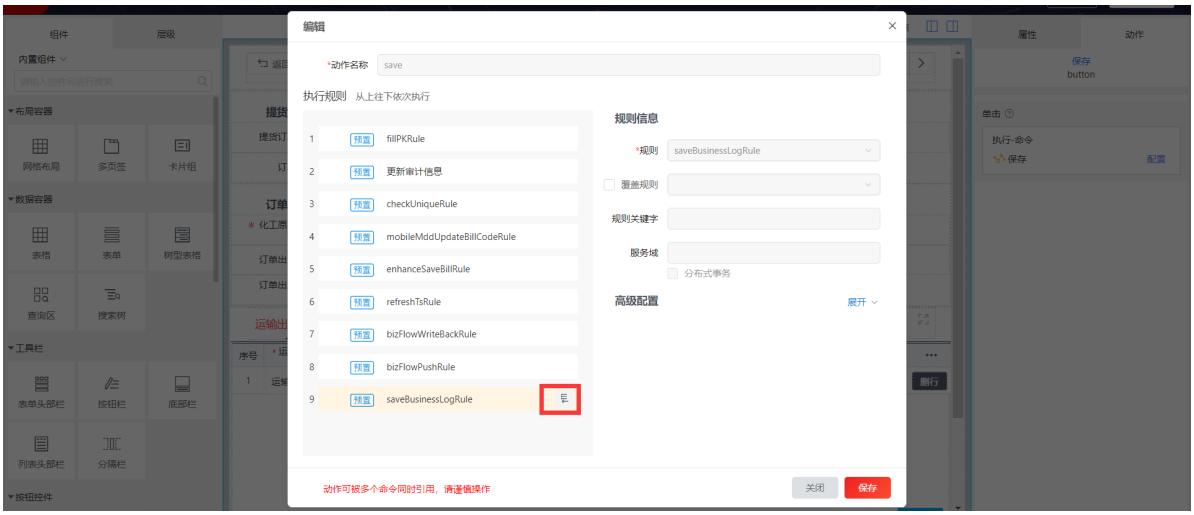
```
        //获取保存之前的   车辆号牌和电话
        let {code,tel} = data;
        //根据车辆号牌是否有数据
        let sqlCode =  "select * from GT1467AT20.GT1467AT20.CarFiles_pp where
code = '"+code+"' ";
        //根据电话查询是否有数据
        let sqlTel =  "select * from GT1467AT20.GT1467AT20.CarFiles_pp where  tel
= '"+tel+"' ";
        //查询
        let respCode =ObjectStore.queryByYonQL(sqlCode);
        respCode.map(v=>{
          if(v.id!==null){
            throw new Error('当前车辆号牌已存在，请检查！');
          }
        });
        let respTel =ObjectStore.queryByYonQL(sqlTel);
        respTel.map(v=>{
          if(v.id!==null){
            throw new Error('当前电话已存在，请检查！');
          }
        });
       return {};
      }
    }
    exports({"entryPoint":MyTrigger});
```

编写完成：

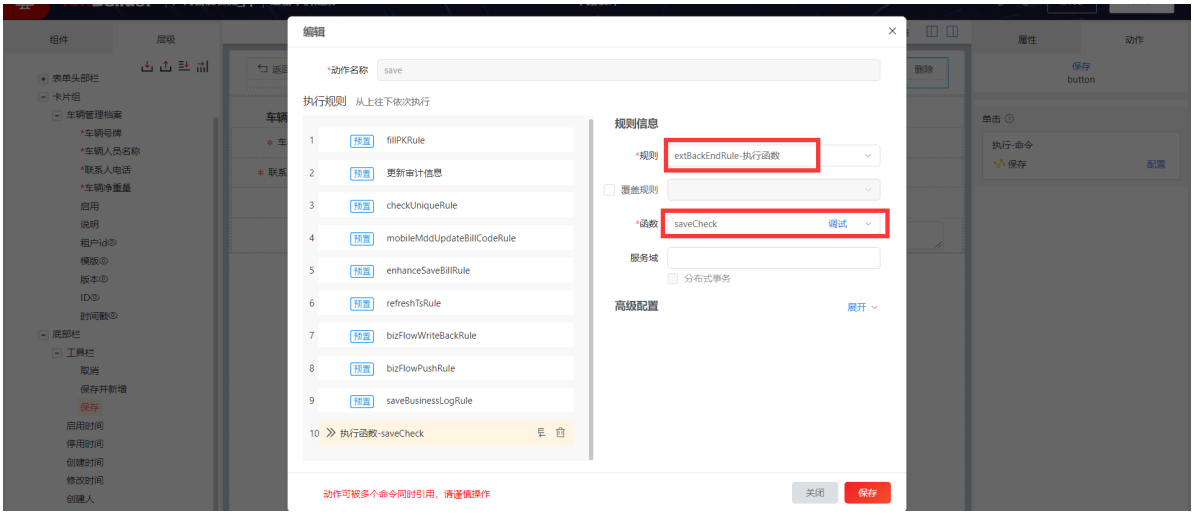

给【保存】按钮配置动作



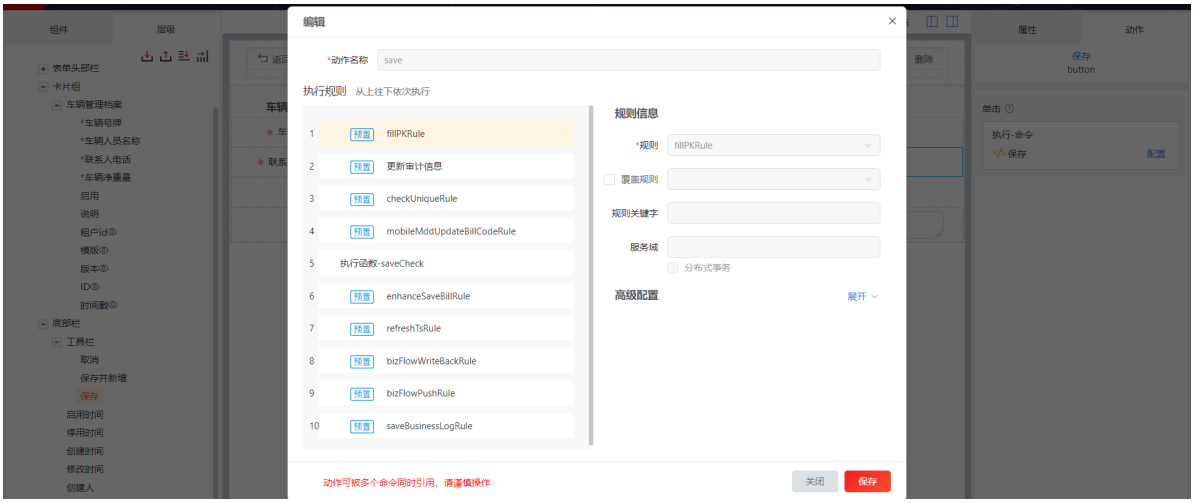

点击图中图标新增一行规则




选择规则：【extBackEndRule-执行函数】选择我们前面创建后台函数：【saveCheck】



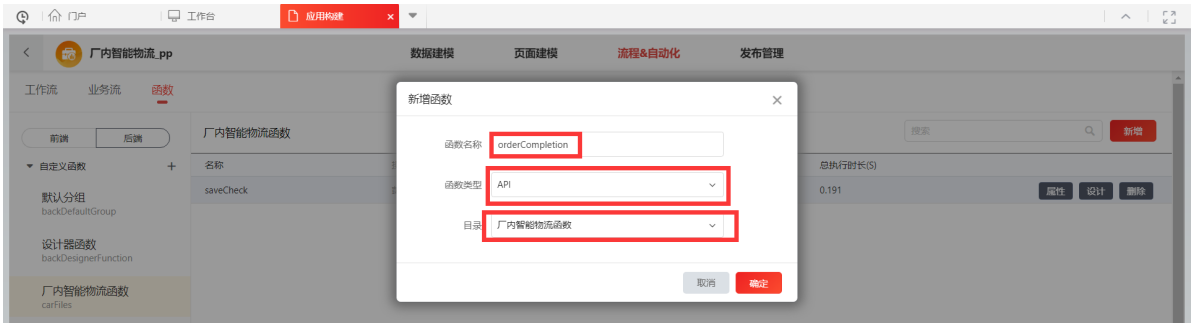

调整自定义规则位置到5

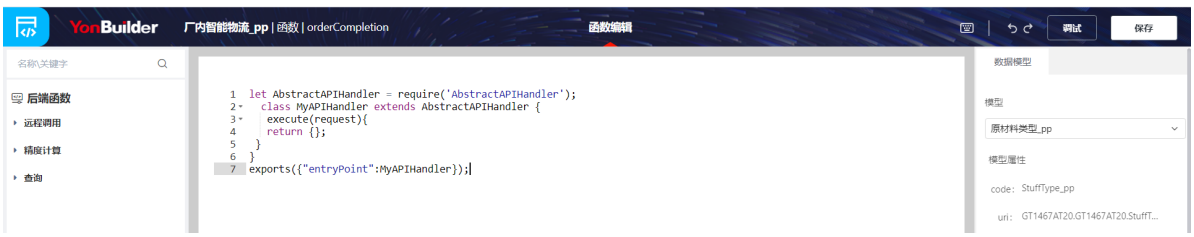



保存，测试添加数据，如图所示：


# 5.在显示页面新增“提货完成按钮”，当订单出货剩余数量为0时，订单状态改为“提货完成”

1.创建后端函数 API orderCompletion 订单完成




```
let AbstractAPIHandler = require('AbstractAPIHandler');
 class MyAPIHandler extends AbstractAPIHandler {
  execute(request){
    //获取请求中的数据
    let data = request.data;
    //debugger
    //设置修改订单状态为   订单完成
    data.OrderStatus = "2";
    //执行修改
    ObjectStore.updateById("GT1467AT20.GT1467AT20.PickUpOrderx_pp",data);
    //获取修改后的数据
    let retObj =ObjectStore.selectById("GT1467AT20.GT1467AT20.PickUpOrderx_pp",{"id":data.id});
    //判定数据是否存在
    if (retObj !== null) {
      //存在   返回获取的数据
      return  {"data" : retObj};
    } else {
      //不存在  返回原有数据
      return  {"data" : data};
    }
  }
}
exports({"entryPoint":MyAPIHandler});
```

```
//获取当前数据
let data = viewModel.getAllData();
//获取当前行状态
let  orderStatus =viewModel.get('OrderStatus').getValue();
let  surplusNum =viewModel.get('surplusNum').getValue();
//状态不为提货中时
if (orderStatus !=='1') {
    cb.utils.alert("只有状态为提货中的单据才可以完成！");
    return false;
}
//运输货物未完成时
if (surplusNum !== 0 ) {
    cb.utils.alert("只有货物运输完毕的单据才可以完成！");
    return false;
}
//放满足条件后      调用后端我们创建的API
cb.rest.invokeFunction('e5eb42568ef742458f086b9ecbb14104',{"data" : data},
(err,res)=>{
    let data = res.data;
    if(data !== null){
        viewModel.setData(data);
    }
})
```

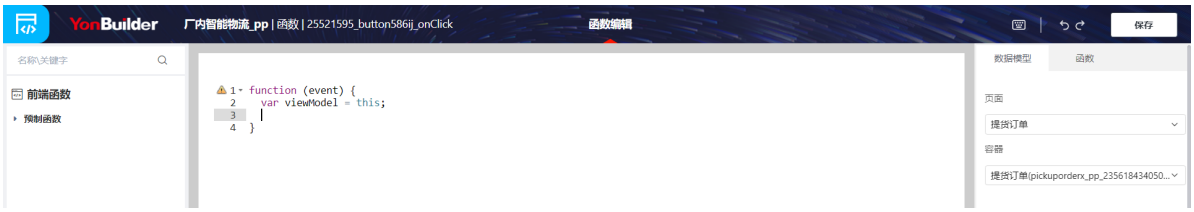
2.配置前端函数

打开提货订单配置器，创建完成订单标签

点击【层级】——选择【提货订单】——在工具栏中插入按钮改名【完成订单】


给【完成订单】按钮添加单击函数


点开配置，如图所示：




编写函数：

代码：

```
//获取当前数据
let data = viewModel.getAllData();
//获取当前行状态
let  orderStatus =viewModel.get('OrderStatus').getValue();
let  surplusNum =viewModel.get('surplusNum').getValue();
//状态不为提货中时
if (orderStatus !=='1') {
cb.utils.alert("只有状态为提货中的单据才可以完成！");
return false;
}
//运输货物未完成时
if (surplusNum !== 0 ) {
cb.utils.alert("只有货物运输完毕的单据才可以完成！");
return false;
}
//放满足条件后      调用后端我们创建的API
cb.rest.invokeFunction('e5eb42568ef742458f086b9ecbb14104',{"data" : data},
(err,res)=>{
let data = res.data;
if(data !== null){
      viewModel.setData(data);
    }
  })
```


3.设置订单完成按钮 添加不可见、编辑时当订单状态为完成时不可见

打开提货订单页面初始化函数,编写功能代码：


代码：

```
let mode = viewModel.getParams().mode;
//添加页面/修改页面
if(mode.toLocaleLowerCase() == 'add' || mode.toLocaleLowerCase() == 'edit'){
//隐藏 订单完成按钮
viewModel.get('button586ij').setVisible(false);
}
```

扩展知识

前端函数

http://tinper.org/mdf/docs#/uzuanu.html

前端调用后端

```
cb.rest.invokeFunction(befunctionId,parameters ,fefunction);
参数列表
    befunctionId     后端函数id  "fd7db308a0be4b47840437ca456723e9"
    parameters       前端传入后端的参数     {r:2}
    fefunction       处理后端函数返回结果的函数    function(err, res) {
        console.log(err);
        console.log(res);
    }
```

后台函数操作实体所用API

https://www.yuque.com/docs/share/56e1399e-e689-452b-b50b-613e7737f2f0#3o3BQ


事件名称

http://tinper.org/mdf/docs#/ak8rsd.html

## 基础模型通用扩展汇总

| 模型 | 对应控件 | 集成设计器 | 方法名称 | 使用范围 | 方法含义 | 参数 |
| --- | --- | --- | --- | --- | --- | --- |
| SimpleModel | 输入框、<br>数值框、<br>日期、定位、<br>按钮<br>手机号 | 需要 | beforeValueChange | 通用 | 值改变以前的事件 | 对象格式：value 要改变的 值，oldValue 原有值 |
| | | 需要 | `afterValueChange` | 通用 | 值改变后的事件 | 对象格式：value 要改变的 值，oldValue 原有值 |
| | 按钮 | 需要 | click | 按钮 | 按钮点击事件 | |
| | 输入框、<br>输入框多语、<br>日期组件 | 需要 | blur | 输入框、<br>输入框多语<br>、<br>日期组件 | 失去焦点的回调 | 输入框、<br>输入框多语控件 |
| | 输入框、<br>输入框多语控件、<br>手机号组件 | 需要 | valueChange | 输入框、<br>输入框多语控件 | 输入值改变之后的回调 | |
| | 日期 | 需要 | toggleOpen | 日期面板收起/<br>展开状态改变时的回调 | | |
| | 下拉项、<br>复选框、 | 需要 | beforeSetDataSource | 通用 | 设置数据源前的事件 | 数组格式，数据源 |
| | | 需要 | afterSetDataSource | 通用 | 设置数据源后的事件 | 数组格式，数据源 |
| | | 需要 | beforeSelect | 通用 | 选择前的事件 | 格式：对象数组（多选），对象（单选） |
| | | 需要 | `select` | 通用 | 选择的事件 | 格式：对象数组（多选），对象（单选） |

动作规则清单

http://tinper.org/mdf/docs#/zqdto8.html

| | | | |
| --- | --- | --- | --- |
| | search | 过滤区搜索 | |
| | Add | 新增操作 | 关联方法 _hasList _addVoucher（支持ArchiveList,TreeList,VoucherList,compare） _addCard（report,treearchive) |
| | batchDelete | 批量删除操作 | 顺序执行batchDo方法和_batchDo 方法 |
| | edit | 编辑操作 | 根据billtype判断，执行dblClick或者common.edit |
| | batchaudit | 批量审查 | 调用batchDo方法 |
| | `batchunaudit` | 批量取消审查 | 调用batchDo方法 |
| | batchsubmit | 批量提交 | 调用batchDo方法 |
| | batchunsubmit | 批量取消提交 | 调用batchDo方法 |
| | batchdo | 批量处理操作 | 调用batchDo方法 |
| | `doservice` | 公共方法，调用服务 | 先判断有没有lastMode（针对树卡和树表），有，<br>获取选中的树，执行common.doService,更新页面。没有，<br>执行SingleDo 方法 |
| | copy | 行功能，类似新增带复制的行数据 | |
| | open | 启用 | 内部调用_doService |
| | close | 停用 | 内部调用_doService |
| | save | 保存 | 校验数据，收集数据，根据页面状态和是否存在树模型，<br>拼接参数。接口成功，返回数据与原有数据合并渲染 |
| | push | 推单 | |
| | singlepush | 行功能，推单 | |
| | dblclick | 行功能，编辑 | 判断 是树还是表，拼接参数，打开编辑卡片页 |