如何打造陆金所营销活动配置发布平台

日渐增多的活动带来的问题

陆金所作为一个高度活跃的金融交易投资平台，日渐频繁的活动需求逐渐使原本低效的活动开发团队捉襟见肘。而以下问题也成为我们主要的问题：

1. 开发周期长。完成一个活动页面往往需要业务人员，产品经理，项目经理，UED，开发人员，运营人员等等的参与。其中的沟通，流程成本不言而喻，这和活动的高实时性大相径庭。

2. 可变性差。一个活动页面完成之后，往往还会根据活动情况进行调整。这种调整无疑又需要经过业务员，产品经理，UED，开发人员的参与。

3. 如何兼容多平台分享。如何通过分享活动可以更广泛的扩大活动的影响范围以及影响程度。而不同平台的分享方式各异，比如微信App分享，陆金所App分享等。

4. 如何跟踪活动成果。如何判断活动是否达到预期的效果，需要我们统计活动页的访问量以及活动页的转化率。

如何解决这些个问题？

答案就是：打造一款面向业务人员、运营人员的营销活动发布平台。

这样的一个平台，不再需要项目经理、产品经理以及开发人员的参与，业务人员和运营人员可以随时改变活动的状态，修改活动的属性，调整活动的内容。这样极大的缩短了开发的周期；并结合陆金所App、微信等平台的分享功能相提供活动分享，整合陆金所大数据业务平台统计能力，可以跟踪活动的成果。

设计原则

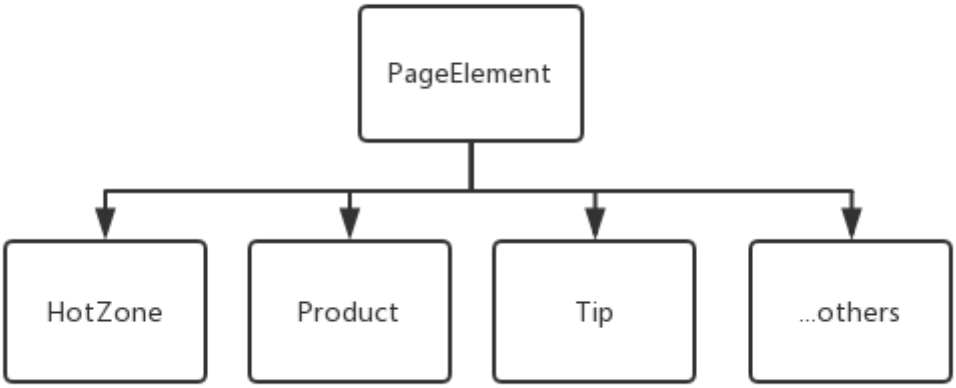
一个面向业务人员、运营人员的平台要求我们在设计的时候需要遵循两个原则：

1. 使用面向业务人员和运营人员的术语。说白了就是使用业务人员、运营人员日常使用的话语。

2. 实时可见、操作简洁。也就是说需要让业务人员、运营人员在制作活动页的时候可以通过简单的操作完成所需要的效果，并且可以随时的看到活动页的效果。

面向元数据的设计

在制作活动页的过程中，业务人员、运营人员所作的一切行为都是生产活动页的元数据的过程。元数据记录了整个活动页的状态，由哪些页面元素构成，每一个元素如何展示，有哪些行为，比如：由几张背景图组成，热区在哪、范围多大，业务统计的key值是什么等等。



在制作活动页的工具（我们称之为活动引擎）中，我们根据制作的需要，通过不同的形式来操作活动页的元数据。

在viewport中，展示活动页可编辑的形态，可以通过拖拽修改背景图片位置、顺序以及热区大小、位置等等。

在页面元素属性编辑区域中，可以修改每一个页面元素的属性，同时修改产生的效果会在viewport中实时呈现出来。

实现

活动引擎主要分组件选区、工具条、页面编辑区以及页面、页面元素属性编辑区四部分。

组件选区提供制作页面的各种组件元素。

- 工具条，提供全屏、放大、缩小、预览、保存等操作；
- 页面编辑区，展示活动页面的效果图，并提供编辑各个页面元素位置、设置元素大小以及设置页面背景等操作；
- 页面、页面元素属性编辑区，提供页面以及各个页面元素的编辑。

  活动页根据平台的不同分为PC端和移动端两种。因此，活动引擎根据PC和移动两个不同平台的特性，分别提供了PC和移动两个页面制作工具。

页面元素组件

每一个活动页面由多个不同的页面元素组成，而每一个页面元素都与一个组件相对应。每一个组件需要制定对应的模型、属性编辑视图、页面元素编辑区视图以及生成活动页面元素的渲染器。

- 页面元素模型，定义了页面元素的基本属性，而这些基本属性描述了组件最终展示的效果以及行为。例如HotZone组件的模型：

```
// file: hotzone-model.js
 class HotZoneModel extends PageElement {
    constructor(id, index, type, page) {
    super(id, index, type);

    this.addAttribute(new Attribute('action', 'emum', 'link', [
      {display: '跳转页面', value: 'link'},
      {display: '弹出层', value: 'popup'},
    ]));
    this.addAttribute(new Attribute('popup', 'text', ''));
```

```
    this.addAttribute(new Attribute('href', 'text', page.platform
=== 'mobile' ? 'https://m.lu.com' :'https://www.lu.com'));
    this.addAttribute(new Attribute('target', 'emum', '_self', [
      {display: '新窗口中打开', value: '_blank'},
      {display: '当前窗口打开', value: '_self'},
    ]));
    this.addAttribute(new Attribute('hotpot', 'text',
`${page.platform}_hotpot_${index}`));
  }
```

- 属性编辑视图，描述编辑组件的各个属性的视图，通过该视图可以设置每一个属性的值。比如HotZone组件的为：

```
<!-- file: hotzone-editor.ejs -->
 <section
  data-id="<%= locals.id %>"
  class="element-editor hotzone <%= locals.id %> <% if
(locals.action === 'link') { %> link <% } else if (locals.action
=== 'popup') { %>popup<% } %>">
  <header class="header">
    <span class="type-flag"></span>
    <span>
      <% if (locals.name) { %>
        <%= locals.name %>
      <% } else { %>
        热区 <%= locals.index %>
      <% } %>
    </span>
    <i class="fa fa-exclamation-circle editor-warning"></i>
    <span class="float-right fui-cross btn-tool btn-tool-remove">
</span>
    <i class="float-right fa fa-angle-up fa-2x btn-tool btn-tool-
collapse"></i>
    <i class="float-right fa fa-eye btn-tool btn-tool-toggle">
</i>
  </header>
  <div class="content form">
    <div class="form-group">
      <label>数据埋点名称：</label>
      <span class="information"><%= locals.hotpot %></span>
    </div>
    <div class="form-group rm-margin-top">
      <label>热区行为<span class="required">*</span>：</label>
      <select class="form-control select select-primary"
        data-toggle="select"
        data-attribute="action"
        value="<%= locals.action %>">
        <option value="link" <% if (locals.action == "link") { %>
selected <% } %> >跳转页面</option>
        <option value="popup"  <% if (locals.action == "popup") {
```

```
%> selected <% } %> >弹出窗口</option>
      </select>
    </div>
    <div class="form-group only-link">
      <label>页面链接地址<span class="required">*</span>: </label>
      <input class="form-control"
        data-attribute="href"
        value="<%= locals.href %>" >
    </div>
    <div class="form-group only-popup">
      <label for="popup-radios">选择弹层<span class="required">*
</span>: </label>
      <div class="popups-group"></div>
    </div>
  </div>
</section>
```

- 页面元素编辑区视图，定义了编辑区组件展示的样子，根据模型呈现当前组件的状态。与属性编辑视图一样采用ejs模板。

```
<!-- file: hotzone-viewport.ejs -->
 <div class="element movable-resizable hotzone <%= locals.id %>"
data-id="<%= locals.id %>">
  <div class="content">
    <a href="<%= locals.href %>" target="<%= locals.target %>">
</a>
  </div>
  <div class="mask">
    <div class="index"><%= locals.index %></div>
  </div>
</div>
```


- 页面元素渲染器，通过渲染器根据组件属性在活动页面生成最终的展示效果，并与组件行为绑定到一起。

```
<!-- file: hotzone-renderer.ejs -->
 <%
  var element = locals.element;
  var attributes = element.attributes;
  var page = locals.page;
  var styles = [
    'left:', attributes.x, 'px;',
    'top:', attributes.y, 'px;',
    'width:', attributes.width, 'px;',
    'height:', attributes.height, 'px;',
    'position:', 'absolute;',
    'background-color:', '#fff;',
    'filter:' , 'alpha(opacity=0);',
```

```
    'opacity:', '0;',
    'cursor:', 'pointer;',
    '-webkit-tap-highlight-color:', 'rgba(0, 0, 0, 0);'
    ].join('');
%>
<% if (attributes.action === 'link') { %>


    <% if (page.platform !== 'mobile') {%>
      <a
      id="<%= element.id %>"
      class="element hotzone"
      style="<%= styles %>"
      data-sk="<%= page.attributes.actionId  %><%=
attributes.hotpot %>"
      data-type="click"
      data-t-category="<%= page.attributes.actionId  %>"
      data-t-title="<%= attributes.hotpot %>"
      data-t-url="<%= attributes.href %>"></a>

    <% } else if (/^lufax:\/\//i.test(attributes.href)) { %>
      <a
      id="<%= element.id %>"
      class="element hotzone"
      style="<%= styles %>"
      data-sk="<%= page.attributes.actionId  %><%=
attributes.hotpot %>"
      data-type="click"
      data-t-category="<%= page.attributes.actionId %>"
      data-t-title="<%= attributes.hotpot %>"
      data-t-otag="eventEngine"
      data-t-schema="<%= attributes.href %>"></a>
    <% } else { %>
      <a
      id="<%= element.id %>"
      class="element hotzone"
      style="<%= styles %>"
      data-sk="<%= page.attributes.actionId  %><%=
attributes.hotpot %>"
      data-type="click"
      data-t-category="<%= page.attributes.actionId  %>"
      data-t-title="<%= attributes.hotpot %>"
      data-t-url="<%= attributes.href %>"></a>
    <% } %>
    <script>
      $(function() {
        var $hotzone = $('#<%= element.id %>');
        var href = null;
        if (PlatformSupport.mobile) {
          if (PlatformSupport.lu) {
            // Open in App
            href = $hotzone.data('href');
```

```
            //$hotzone.tap(function() {
              //window.Bridge.call({task: 'url', url: href});
            //});
          }
        }
      });

    </script>
<% } else { %>
  <a
    id="<%= element.id %>"
    class="element hotzone"
    style="<%= styles %>"
    data-popup="<%= attributes.popup %>"
    href="javascript:void(0);"></a>
  <script>
    $(function() {
      var NoticePopup = $.Widget.NoticePopup;
      var TipPopup = $.Widget.TipPopup;

      var $hotzone = $('#<%= element.id %>');
      var popupId = $hotzone.data('popup');
      var $popup = $('#' + popupId);


      if($popup.hasClass('notice-popup')){
        var noticePopup = new NoticePopup($popup);
        $hotzone.click(function() {
          noticePopup.show();
        });
      }else if($popup.hasClass('tip-popup')){
        var tipPopup = new TipPopup($popup);
        $hotzone.click(function() {
          tipPopup.toast();
        });
      }
    });
  </script>
```


成果跟踪

自1月份上线以来至3月底总计活动页166个，覆盖95%以上活动页面；大部分活动页面直接由运营人员生产，活动开发周期也从原来的一个星期缩减到不到一天。

问题难点

依赖后端的问题

陆金所的绝大多数项目使用的后端开发语言是Java，然后该平台的后端使用的是PHP，考虑到开发成本以及平台的移植性，我们决定不使用Smarty模板，而采用ejs，所有的接口请求使用ajax的方式，最后生成一个index.html，以及相应的静态资源，部署的时候直接放到对应的运行目录就可以。前端可以专注配置页面和组件的开发，本地调试的时候可以通过localStorage来存放页面的元数据，真实环境的后端也只需要提供相应的保存和获取数据的接口。

响应式的组件开发和活动页面展示

在陆金所，视觉稿的尺寸规范是有标准的，PC和H5的宽度分别为1920px和640px，所以我们的活动引擎默认的页面编辑区域分别就是这两个尺寸，然而真是环境的屏幕尺寸是十分丰富的，我们使用css transform的scale来进行相应的缩放。这样，我们就可以完全按照设计稿中的尺寸和位置关系来开发组件，减少了计算的复杂度，同比例缩放也能保证页面最终的展现效果和设计稿尽量保持一致。

如何进行移动端调试

我们的活动页面需要满足在微信，陆金所App以及手机浏览器端进行展示，免不了要进行手机端的页面调试。我们主要借助了Charles进行抓包调试接口，mock请求的返回，分别使用Safari和Chrome inspect功能连接手机进行页面调试。

使用scale导致的问题

当然，使用scale也遇到了一些问题，比如说父容器进行scale后会导致原先fixed的元素失效，我们通过将fixed的元素添加到 `<body>` 标签下；容器进行使用scale缩小之后，底部会有大片留白，高度并不会自动缩小，我们可以在容器外层添加一层div，计算缩放之后容器的高度，将可视区域高度设置成需要的大小，超出部分设置overflow:hidden就可以了。

内嵌JQuery代码导致Velocity模版解析报错

我们生成的活动页面会固化成一个个的vm文件存放到文件服务器，使用 `#parse('xxx.vm')` 去解析一个模板的时候，由于我们页面上使用了内嵌的JQuery代码，使用了很多的$，而类似 `$('#id')` 这种选择器的js代码被解析成了java的方法去执行，自然就会报错。其实解决的方法也是很简单的，直接在我们的js代码前后，或者说任何我们不希望Velocity进行解析的内容前后增加 `#[[...the code shouldn't be parsed...]]#` 就可以了。