**介绍蚂蚁体系内各种前端框架的特性**

金蝉：因为不再进行维护了，所以这里简单介绍一下，因为之前一直使用金蝉来开发渠道手续费和国际返佣两个项目，所以对金蝉的使用是最为熟悉的。

金蝉与传统的开发模式不同：

传统的开发模式需要自己搭建前端项目，无论是路由的配置、组件的引入、单项数据流的管理与控制都需要自己来维护，需要注意的细节较多，属性配置的地方也很多，涉及到复杂的业务逻辑或者数据的集中管理时，开发起来就要一边考虑到项目的架构层面又要考虑到业务的逻辑层面，这样的话，开发者就不会只专心于业务的编写。而使用金蝉就不会有这些问题的出现，因为开发者在金蝉专用的IDE中进行开发的话，金蝉就已经对架构层面的东西做了整合，只需要按照金蝉的使用方式进行开发业务即可，全局逻辑与私有逻辑拆分/路由的配置/数据的集中管理/权限的集中管理/项目的打包等细节都不需要开发者自己来设计。另外当我们想要使用某个组件，只需要在IDE右侧的组件管理面板中追加一个就可以了，这点有点类似于html的标签嵌套，通过组件的嵌套来完成基本的页面初始化布局，这样就省略了开发者引入组件的步骤，也避免了传统开发中由于组件拼写错误导致的页面报错。另外，在金蝉中，每个组件有哪些属性，在金蝉的属性版面中也是全部展示出来的，这样避免了开发者因不熟悉组件到底有哪些属性而浪费时间去翻阅文档，造成开发时间的耽误。比如一个Input输入框，金蝉会很明显的展示出他有哪些事件，如提示有键盘输入事件、鼠标焦点事件，这样你就不会想它会不会有一个自动补全事件了，那如果你想要使用自动补全事件，你就会很自然的去寻找其他组件了，不会在Input组件上浪费时间了。

总结来讲，金蝉提供了在线的视图搭建以及模型设计，通过金蝉IDE的特性，在IDE中通过标签的嵌套以及标签属性的设置来进行页面的布局、排版以及简单的自定义样式设置。通过标签中事件的绑定，这里的事件就是一个方法，通过编写方法来实现业务中的交互逻辑。通过一键更新jar包，可以观测到哪些接口修改了，哪些入参变化了，哪些出参变化，都很明显的可以被感知到。在多人协作时，也不会出现别人修改我的代码造成代码冲突，因为当别人在编辑页面或逻辑的时候，自己是无法同时进行编辑相同文件的，当前文件谁在编辑都会有明显的提示。

总之给我的感觉就是用金蝉编写项目很快，错误排查也很快。很遗憾，金蝉也不再进行维护了，以后使用的人也会越来越少，主要原因可能还是做出来的项目，用户的体验不是很好，有时候还会出现页面的卡顿，因为在金蝉的体系里，对开发者限制较多，也导致使用金蝉开发交互过于复杂的应用或者复杂的组件，就会使开发者无从下手，这也是金蝉灵活性不足的体现吧。所以作为前端开发还是会寻找其他的蚂蚁体系中的前端框架，比如bigfish或者云凤蝶。

云凤蝶：与金蝉类似，也是通过页面IDE来进行开发，给人直观的体现就是无需像金蝉那样嵌套组件完成预览，而是通过直接拖拽组件，来完成页面布局的设计，宽度、高度等也是通过拖拽来调整。让人感觉好像在使用一个绘图工具。这样的设计，使得初步上手构建页面较为容易，更适合PD或UI来进行页面的静态布局与简单的页面逻辑交互，而开发者可以在其基础上进行页面交互、联动、接口联调与数据渲染等工作。云凤蝶最强大的地方就是智能向导。比如我们想生成一个新增用户的表单，只需要将正确的接口地址输入到IDE中，就会自动根据入参的类型生成一个表单，如入参name是一个string类型，那么就会生成一个名为name的Input，如入参hobby是一个List类型, 那么就会生成一个名为hobby的下拉框组件，智能向导就是根据字段的类型来生成合理化的组件，当然如果觉得有些不合理(比如后端传来一个age, 它会自动生成一个Input,你也可以改成一个仅能输入数字的Input)，或者需要做表单校验，也可以通过微调来实现表单校验的功能。但是如果涉及到较为复杂的项目或复杂的交互，可能会让开发者显得力不从心，个人认为，如果项目中选择使用云凤蝶，那么需要满足两个条件：1.有熟悉云凤蝶的PD使用云凤蝶来进行PRD的产出，因为当PD使用云凤蝶进行拖拽绘图的时候，他就很明显的能感知到这个交互云凤蝶是否可以实现，当然也可以通过PD与前端的沟通，来一起尝试功能实现的可能性。如果项目中没有设计稿，那么使用云凤蝶会有可能会让前端开发开发过程中遇到问题。(简单的页面除外，比如简单的表单提交，简单的CURD等，比如渠道的新建阶梯费率，矩阵费率，新增矩阵阶梯费率，弹出一个阶梯费率，这种感觉使用云凤蝶实现起来比金蝉更困难)。云凤蝶应用的用户体验要比金蝉好，页面的流畅度、组件的交互性都高于金蝉。

说到交互复杂、数据维护复杂的页面，既然金蝉和云凤蝶实现起来都比较难受，那么使用bigfig可以很好的解决这种问题。

Bigfish: bigfish的主要核心就是一个react项目。集成了antd pro(UI库),dva(数据的集中管理插件),router(用于处理页面的路由),exlint(进行代码的规范化),jext(用来进行前端的自动化测试),request(用来与后端进行数据请求交互)等功能，bigfish几乎包含了所有web前端需要的技术。也就是说只要安装了bigfish，上述的所有功能插件都可以直接使用，当然不仅限于这些核心插件，完全支持第三方插件的引用与使用，Bigfish 还可以打通 Basement、LinkE 等发布流程，可将应用发布到线下及线上环境。不管对内还是对外应用、中台还是运营、pc 还是 h5，Bigfish都可以完全胜任。个人感觉，对于新人来说，bigfish是这3种框架中，最容易上手的框架，因为他的项目创建，以及用到的MVVM设计模式，都跟常规的react脚手架搭建的项目的风格一致。同时新版的bigfish使用的语言是typescript, 也就是可以定义类型的js语言，这样，在多人开发时，只需要知道使用的变量类型就可以很明确的知道该变量中有什么属性、方法等。新人通过简单的项目创建文档创建项目后，通过引入AntD组件，可以简单的做出一些基本的页面布局，但是其他的页面交互，如：路由，发送请求，自定义mock数据等，通过bigfish文档的学习是可以快速上手的。对于新人怎样快速上手bigfish, 接下来会讲到。那么以上三种前端框架，我个人还是更喜欢bigfish，因为bigfish更接近于传统的react项目开发，只要使用过react脚手架项目，都可以很快的上手bigfish，同时react社区庞大，资源丰富。开发的灵活性也很强。

拿三个框架的服务群比较：

云凤蝶：1049人

Bigfish:：2150人

金蝉： 已解散

从灵活上来讲，bigfish更灵活，可扩展性更强一些。

推荐bigfish

从新人上手的角度介绍一下bigfish的使用：

之前提到了Bigfish 打通了 Basement、LinkE 等发布流程，那么新人首先应该了解通过Basement来创建项目(做到熟练创建)。

开发项目之前，熟悉react技术的新人需要掌握以下技术：

* 1.git的熟练使用，因为项目最终要放在代码仓库中，开发者需要通过git进行代码的提交，拉取，合并，分支切换等操作。如果开发者是从金蝉或云凤蝶过来的，那么他对git的操作已经生疏了，所以还是要重温一下git的相关操作。
* 2.typescript语法的熟练掌握，因为从现在开始入手bigfish的话，肯定是使用最新版本的bigfish
* 3.熟悉bigfish中使用的UI框架中的组件，比如antD中的，techUI中的，要做到通过文档快速找到符合PRD需求的组件，并对组件进行属性、事件的使用做到心中有数。
* 4.Css预处理器，less，sass语法的使用。bigfish集成了对Css预处理器的实现，使用动态的css语言可以是编写样式更加灵活。
* 5.无状态组件的理解与使用，由于今年来reacthock的流行，使得大多数的react组件都是用到了效率较高的无状态组。
* 6.使用bigfish中的Request插件发送http请求。
* 7.D.VA框架的使用，dav是一个数据层统一管理框架，它是bigfish对redux的进一步封装。我们将项目中用到的数据全部交由dva进行管理是一个很安全的选择。
* 8.自己写mock数据的能力，因大部分情况下前端页面的搭建要比后端接口的开发更快，所以，在页面开发完成以后，在正式接口联调之前，前端可以根据后端提供的接口来自己mock数据进行接口的模拟调用(bigfish中自带了node服务，可以让开发者来请求)，这样可以方便提前发现潜在问题。Mock数据的编写要基于bigfish的规范，这样才能实现。
* 9.独立部署前端代码的能力
* 10.代码封装与复用的能力，涉及到多个项目并行开发的时候，自己用过的逻辑无需重新编写，将通用的代码进行高复用性的封装，可以提高开发效率。

有了以上8点技术储备以及2点基本能力就可以进行高效率的项目开发了。

后续根据开发经验的不断成熟，可以多培养自己架构层面的能力

* 项目的分层能力，如共用组件与页面的分层，页面中小组件的管理拆分。UI组件与容器组件的合理拆分，等等。
* 自定义复杂业务组件的能力
* 功能文档的编写能力，将自己开发的功能形成文字，方便日后维护。页面的业务逻辑与交互效果要做到有据可查。

代码复用：

对于前端来讲，页面上往往会使用到大量重复的组件，如输入框，下拉 列表/按钮等，如果我们每使用一个控件都手写一遍的话，代码中就会产 生很多重复代码，如果涉及到事件的处理，就更为繁琐了，那么前端就要对组件进行封装，为后面的再次使用做准备。而对于项目中常用的方法，也要进行封装与精简处理，比如http请求等必备逻辑。这样既提高了开发效率，又降低了维护的风险，以后只对封装的逻辑进行优化即可。

在UI层面：

封装自定义UI组件，使得复杂的组件可以在其他页面

上做到拿来即用。当其他项目需要这个组件时，也可以通过导入 组件的形式直接使用，无需再次开发。

在渠道项目中，将费率的表格组件进行了封装，有阶梯费率组件/ 固定费率组件/矩阵费率/矩阵阶梯费率。然后根据是表格否可编 辑，又做了一些样式上的调整。

在通用方法层面，做了方法的封装，如

表单校验的逻辑

TR接口调用的逻辑

表单初始值设置的逻辑

筛选下拉列表的逻辑

根据枚举填充下拉列表的逻辑

将表格生成excle文件并导出的逻辑

代码扩展

我理解的，就是在网络课堂上，有的课程是章节式的，有的课程是一字排开、不非章的。那么，章节式编排就是可扩展的。无章编排则是不可扩展的。 比如，将来要在课程里插入新的一节： - 对于连续编排的课程，如果要在里面插入一节，则要修改后面几十节的标题 - 对于章节编排的课程，影响最多一章，对后面的章节没有影响。这便是可扩展的设计。在软件设计中，可扩展性强的代码是最受欢迎的代码。当然我们渠道前端项目中在很多地方也封装了很多可扩展性的代码：

在渠道的签约功能的费率部分，我做了代码扩展的处理，因为后面接到返佣的项目，在进行价格配置的时候，需要用到渠道中固定费率/矩阵费率、阶梯费率、矩阵阶梯费的新增功能，但是两者的展示与功能有细微的差别，所以不能直接copy代码，比如二者的表头展示不同，渠道展示的是要求用户填写整数或者小数，而反佣则需要展示币种单位，同时还有国际化的要求。这就要求在源代码的基础上进行扩展，我将表头的title作为参数进行传递，这样使用方需要什么title就传什么title就可以了。另外返佣要求对阶梯费率的table做强制性的数字校验，要求一旦数字输入不符合要求，就将Input做标红处理，于是我在源代码的基础上做了动态参数处理，每个参数是一个函数，一旦存在该函数，则自动执行这些函数。这样实现了在源代码不变的情况下，通过使用方增加业务函数，即可实现复用+扩展。

交互标准：

1. 尽量不要使用is\_xxx来命名类型为boolean的入参，否则后端javaBean可能会接收不到这样的参数。比如渠道项目中新增费率过程中保底金额/封顶金额是数字还是百分比的一个判断，之前使用的是isPersent，后面重命名了。

2.请求响应传输数据格式：JSON，JSON数据尽量简单轻量，避免多级JSON的出现；

3.接口响应的数据格式：尽量做到格式的固定，目前请求成功的接口格式都很统一，错误的接口返回的格式有时候不固定，特别对于后端新人来讲返回的格式都不规范，这样造成前端页面报错，或无法给用户展示正确的错误原因。

4.如果使用bigfish来开发项目的话，那么就更要求后端接口的规范化了，因为这时候不仅后端需要明确每个出入参的类型，前端也要明确这些类型，前端也要定义class接受后端返回的实例，而一旦有类型发生错误，也会导致项目报错。如果后面用到bigfish开发前端项目，那么前后端涉及到交互的数据就要及时进行沟通，后端字段类型发生变化也要告知及时告知前端。

5.前端尽量避免做一些业务逻辑的操作，比如金额的计算，因为js在计算本来就是不够准确，特别是涉及到浮点型的时候，会精度不够精确。建议涉及到数字的时候，前端以字符串的形式提交给后端，由后端转为数字在进行计算。比如汇率部分可以保留10位小数，那么前端js肯定无法对这个长的小数做精确计算。

容易出bug的的功能或组件：

在项目的开发过程中，由于每次的迭代后，一个地方有所改动，那么会影响到其他地方，以为js不像java语言那样存在横切代码逻辑，所以，无法做到使用切面的思想来做代码增加。那么当一旦手动插入js代码就很有可能会影响到其他逻辑，甚至页面报错。比如，新签约页面中，一旦有汇率表格中字段或验证规则的改变，就会影响到签约的修改页面/详情页面/审核页面/还有可能影响到解约的审核页面。所以我会将公共展示的代码抽离出来，新增与修改页面公用一套table展示逻辑，详情及审核页面公用一套展示逻辑，一旦上述table组件有所变化，仅需做出少部分修改即可，无需每个页面都改一次。

编码规范

出于项目长久维护的角度来考虑，代码的编写就必须要求按照一定的编码规范来编写，首先要求代码的一定要符合标准的格式，比如缩进的空格个数，单引号还是双引号的统一，尽可能少的减少http请求，对于跟新较少的内容

web安全相关的问题

1.文件上传漏洞

不要允许用户上传任意文件可能会让攻击者注入危险内容或恶意代码，并在服务器上运行。应该在上传前检查文件类型及后缀。定义上传文件类型白名单，即只允许白名单里面类型的文件上传。文件上传目录禁止执行脚本解析，避免攻击者进行二次攻击。

2. XSS 跨站脚本攻击

用户可以随意的通过输入框来输入html格式的字符串或者脚本，轻则影响页面样式，重则危害网站的安全，如果别有用心的人插入非法脚本，则页面有可能出现广告。输入过滤。永远不要相信用户的输入，对用户输入的数据做一定的过滤。如输入的数据是否符合预期的格式，比如日期格式，Email格式，电话号码格式等等。这样可以初步对XSS漏洞进行防御。

因此，后台服务器需要在接收到用户输入的数据后，对特殊危险字符进行过滤或者转义处理，然后再存储到数据库中。

3.尽量使用post进行敏感数据的请求，而不要以GET的形式将数据放在url中进行传递

4.跨站请求伪造

比如一用户在页面上输入了个人信息，在本地有保存服务器生成的cookie，那么他一但点击了一个恶意链接访问到了恶意网站，那么他cookie中的信息也会被带过去，那么就造成了个人敏感信息的泄露。

所以应用程序和用户进行交互过程中，特别是账户交易这种核心步骤，强制用户输入验证码，才能完成最终请求。在通常情况下，验证码够很好地遏制这种情况的出现。

5.尽量避免SQL注入

对数据表的访问权限进行严格控制，尽量限制用户不必要的访问权限  
  
  
  
  
渠道手续费/返佣 迁移到bigfish框架，从技术方面和功能层面，迁移的成本

首先渠道手续费/返佣 来迁移到bigfish是可以实现的，准确的说是用bigfish来重新开发这两个项目。那么重构起来需要重新将项目的整体业务逻辑进行梳理，对于复杂的前端交互逻辑，我这边也做了详细的记录，每个细节都有记录。但是由于项目的交互逻辑还是较为复杂的，同时以页面为单位进行开发，需要测试同学介入进行功能模块的配合测试。

从技术方面来讲：

bigfish可以完全支持现有项目的逻辑交互，表单校验，接口请求，以及页面风格，交互风格等都与金蝉项目保持一致。重构项目使用的核心技术就是react这样的视图模型框架以及dva这样的数据集中管理框架,利用react的由数据驱动组件的特性，可以很好的开发一个中后台应用。另外使用bigfish可以通过自定义组件的技术来实现业务中复杂的组件，比如渠道中较为复杂的各种费率表格，以及表格输入框中内容的校验。

从功能上来讲：

比如项目中的签约功能，因为是根据计费模板来动态展示表格，以及表格上的title,这里也是实现起来难度最大的部分，不过可以借助金蝉项目中现有的业务逻辑作为指导，在bigfish上完成其功能代码的实现。在

对于前端的工作来讲需要进行以下几点：

1.路由的重新配置，保证与原始项目路由一致；

2.出入参类型的定义，与后端接口定义的一致；

3.权限的控制，bigfish提供了鉴权的解决方案，通过传递booean类型 来实现权限的分配

4.通用组件的封装(自定义组件，如各种费率的table，以及动态table, 汇率的table)

5.通用方法的封装（http请求基于bigfish，鉴权逻辑的封装，组件填充值与初始赋值逻辑的封装）

6.所有表单校验需要重新处理(校验机制与金蝉不同)

7.所有的excle导出逻辑需要重写

8.前端全局变量以及金蝉store中维护的数据，需全部转为交由dav 进行处理

9.

但是涉及到权限的部分，可能需要重新采取鉴权策略，因为之前的项目是通过金蟾提供的权限处理机制，只需要将申请下来的权限码发在金蟾的权限码管理中即可实现鉴权，但是bigfish的话，就需要前端调用后端提供的接口，同时后端需要返回当前用户所拥有的权限码，前端根据当前用户是否存在某个权限码来控制页面部分的显示或隐藏。

涉及到枚举的部分，金蟾会对枚举做一个很好的使用

未知问题：

金蝉中后端用的是TR服务，bigfish中是否可以使用TR服务，还需要后端进行协商。还有就是新服务对于现有后端代码的出入参的要求有没有什么影响。

金蟾涉及到文件上传功能，上传的形式与bigfish有所不同，需要重构上传逻辑。

迁移可能需要人力成本与时间成本的投入，如果时间不急的话，可以慢慢迁移，做到稳中求精。

项目开发中也会遇到很多阻碍项目正常开发的困难，导致项目停滞不前，这种情况下在自己临时没有开发思路的时候，会与其他同学进行问题的探讨，或者咨询开发过类似同样需求的同学，比如，返佣中的国际化问题，在分页的组件中，无法使用内部提供的国际化技术来实现其他语言的展示，经过探讨分析，最终大家都使用的是修改DOM节点来实现国际化。项目中遇到问题也是好事，因为通过不断的总结经验，不仅可以让自己在技术层面得到了经验的提升，同时在其他同学遇到相同问题的时候也可以及时做出帮助解决。也可以让整个前端社区更加壮大。