飞书北邮课程实践文稿 副本

#0 课前准备

在开始上课之前，需要同学们准备如下工作，方便课堂上与讲师一块同步操作。

1. 注册并下载飞书，包括 PC 版和手机客户端；（后续登录 IDE 、开放平台和小程序调试都需要用到）
2. 下载飞书小程序 IDE（通过飞书客户端扫码登录）；
3. 注册登录轻服务，通过手机号注册，需要实名认证。需要提前提交实名认证信息，轻服务工作人员统一审核。
4. 对前端开发的技术 HTML / CSS / ES6 有初步的了解

#1 项目介绍

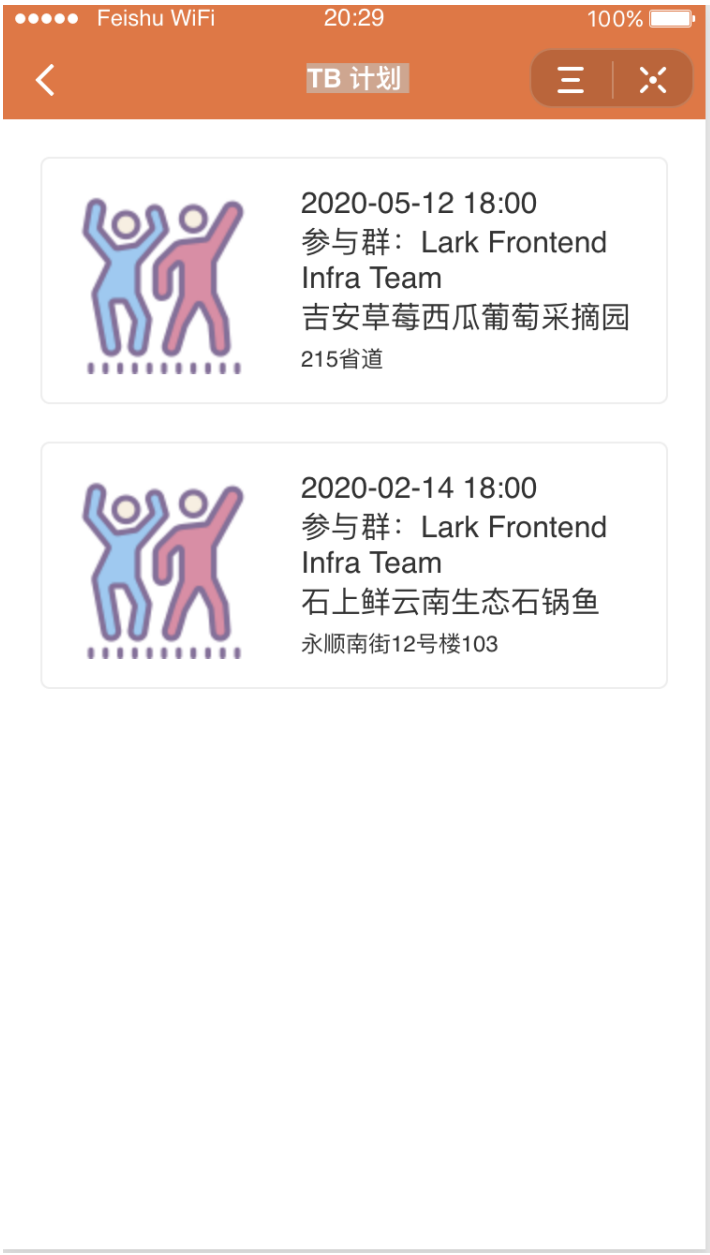
通过讲解团建小程序的整体开发流程，掌握 飞书小程序 + 轻服务 应用开发。团建小程序主要有四个功能模块，下面详细介绍下。

#1.1 团建地点列表

即小程序的首页，展示推荐的团建地点列表，长按地点可以进入创建日程页面。点击左下角按钮可以查看尚未进行的团建活动，点击右下角按钮可以查看已经完成的团建。点击中间的按钮可以按条件搜索团建。

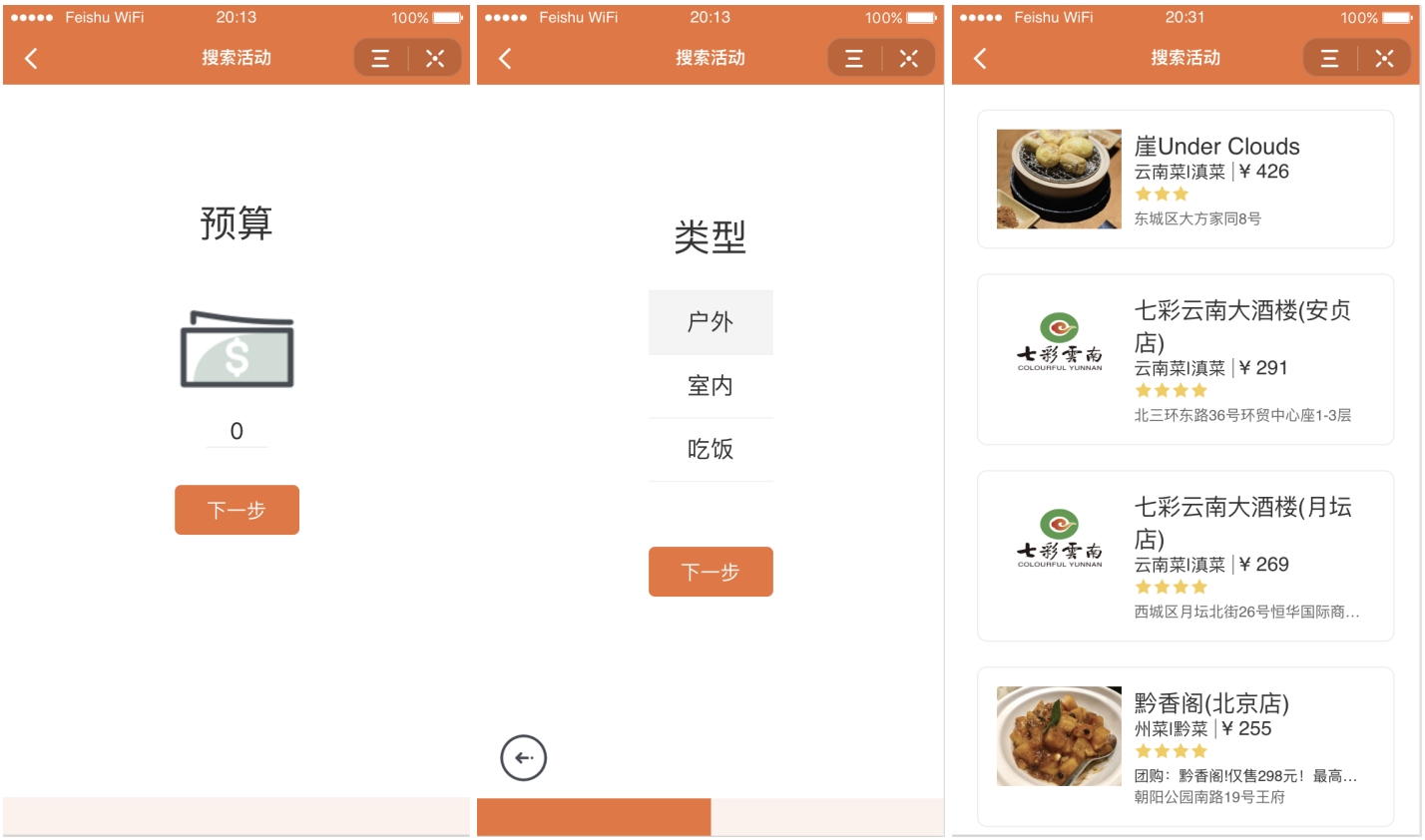
#1.2 团建计划与团建历史

因为团建计划与历史团建界面一样，只是根据时间来划分，所以归为一个模块。团建计划中展示的是尚未进行的团建活动，团建历史展示的是已经完成的团建活动。



#1.3 搜索团建

点击首页的加号按钮，可以进入搜索团建页面。通过填入预算金额和团建类型，能够筛选出符合条件的团建列表。



#1.4 创建日程

输入群名称，可以对你拥有的飞书群进行模糊搜索，选择对应的群以及团建日期。点击创建日程即可在飞书中创建对应的日程安排。

#2 概要设计

要实现上述功能，我们从技术角度去拆分，划分出相应的功能模块。我们将从客户端和服务端两个角度去考虑如何划分功能模块。

#2.1 客户端功能模块

客户端负责界面展示层的功能，我们可以采用目前主流的小程序技术去实现，这里我们以飞书小程序技术去实现，其他技术相似。我们按页面去划分功能，可以划分出如下功能模块。

#2.1.1 团建列表推荐页

团建列表推荐页（也是小程序的首页），需要展示推荐的团建地点列表。长按对应的团建地点，可以跳转到创建日程页。在下面的导航栏中，提供跳转到团建计划、团建历史和搜索页面的按钮。

#2.1.2 团建计划和团建历史页

团建计划页和团建历史页除了内部填充的数据不一样，界面是完全一样的，所以在此处归为一个页面。该页面只需要从服务端拉取对应的数据，将团建计划展示出来即可。

#2.1.3 搜索活动页

搜索活动页分两个步骤，第一步需要用户输入团建预算金额，第二部需要选择团建类型。在完成两个步骤的选择之后，将选择发送给服务端，搜索出对应的团建结果，并跳转到搜索结果页。

#2.1.4 搜索结果页

搜索结果页负责展示搜索结果，点击团建地点，可以进入创建日程页。

#2.1.5 创建日程页

创建日程页需要用户选择飞书群和日期，才能创建日程。用户可以在输入框中输入群名称，会进行模糊搜索，展示下拉列表。

#2.2 服务端功能模块

服务端则负责数据存储和一定的业务逻辑功能，为了简化开发，我们使用字节跳动提供的轻服务去完成服务端。我们将在 3.2 节介绍轻服务。服务端主要实现团建小程序数据的增删改查，对外表现是一系列的功能接口。

#2.2.1 团建推荐列表

团建地点数据源：我们通过爬虫从大众点评抓取团建地点列表，存储在轻服务数据库中。

在返回团建推荐列表时，为了使每次的结果都不一样，可以借助智能推荐技术，实现推荐。此处为了简化，用随机函数随机了一个推荐列表。

#2.2.2 团建计划列表

从创建的团建计划中，选择团建时间在今天之后（包括今天）的团建，返回给前端。

#2.2.3 团建历史列表

从创建的团建计划中，选择团建时间在今天之前的团建，返回给前端。

#2.2.4 搜索团建

根据团建预算金额和团建类型，搜索整个团建列表，将结果返回。

#2.2.5 搜索群名称

搜索群名称会根据输入的关键字，将请求转发给飞书开放平台的接口，对当前登录用户的所有群按关键字进行过滤。

#2.2.6 创建日程

根据用户选择的群和日期，调用飞书开放平台的接口，为对应群创建日程，并发送给群成员。

#3 技术介绍

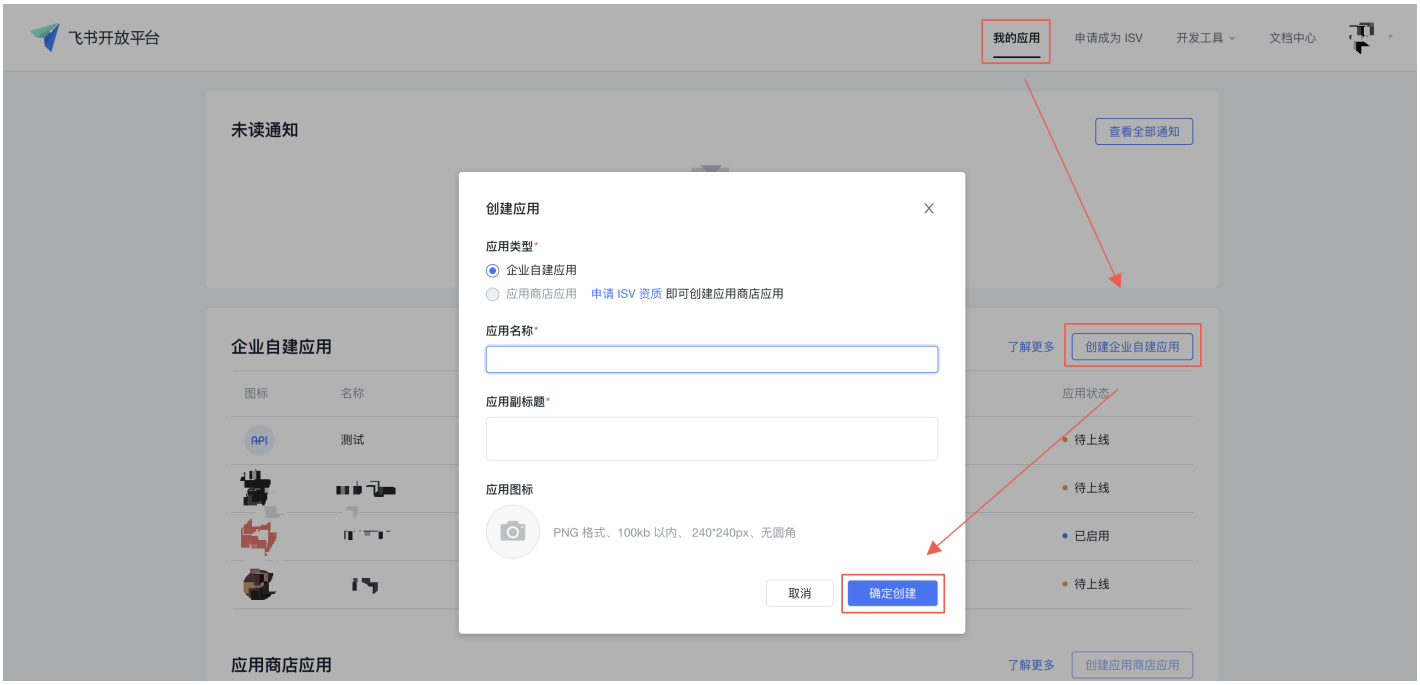
#3.1 如何开发飞书小程序

参考飞书开发平台的开发流程，教学生如何开发一个小程序版本的 Hello World。主要内容包括如下几个步骤。

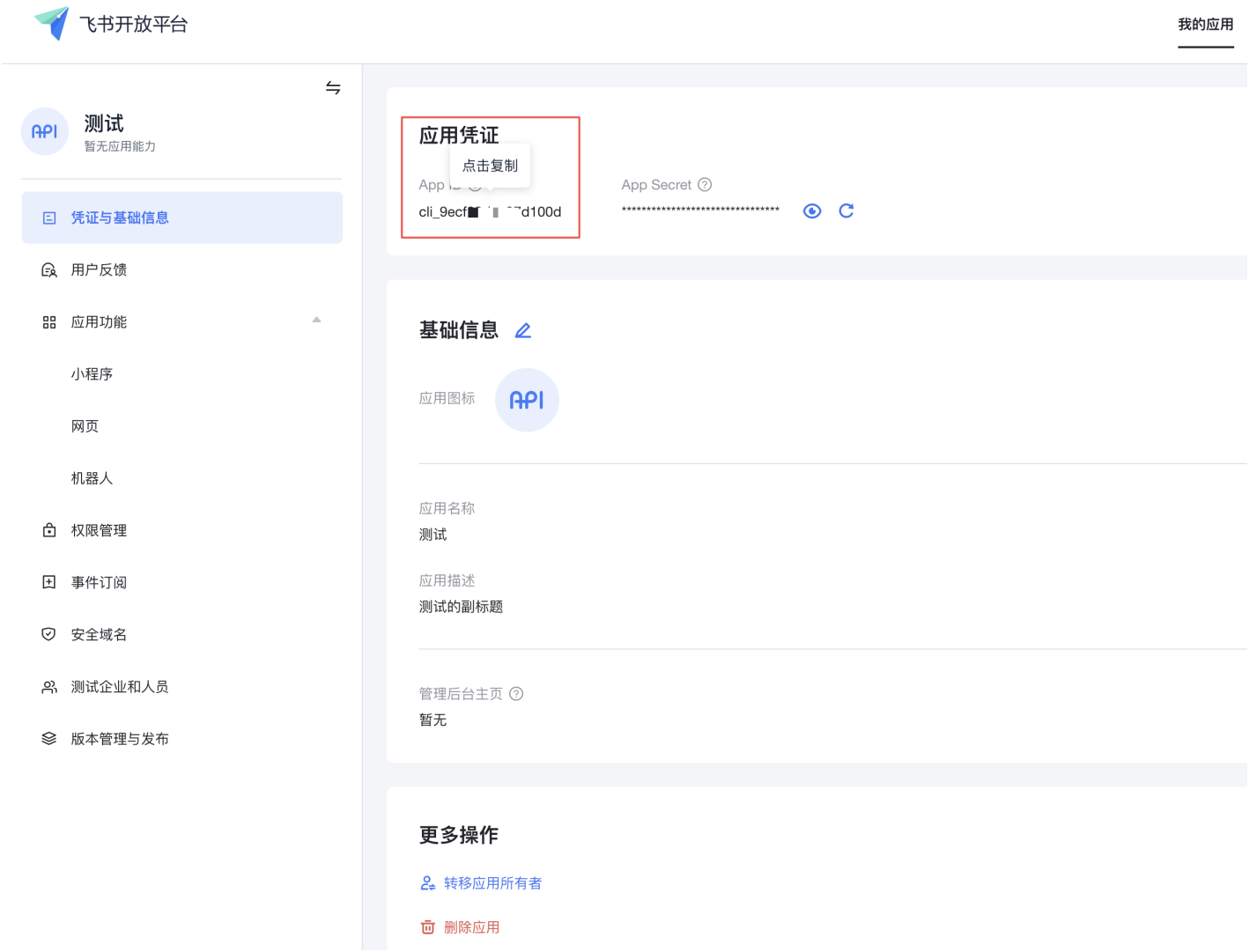
#3.1.1 飞书开放平台注册与应用创建

在创建飞书小程序时，需要先在开发者平台创建小程序。

进入 open.feishu.cn ，使用飞书账号登录即可。登录之后，点击导航栏 “**我的应用**” 标签页，进入我的应用列表页。然后点击“**创建企业自建应用**”，在弹窗中输入应用名称和副标题，点击创建即可。（无需指定应用图标，会有默认的图标）



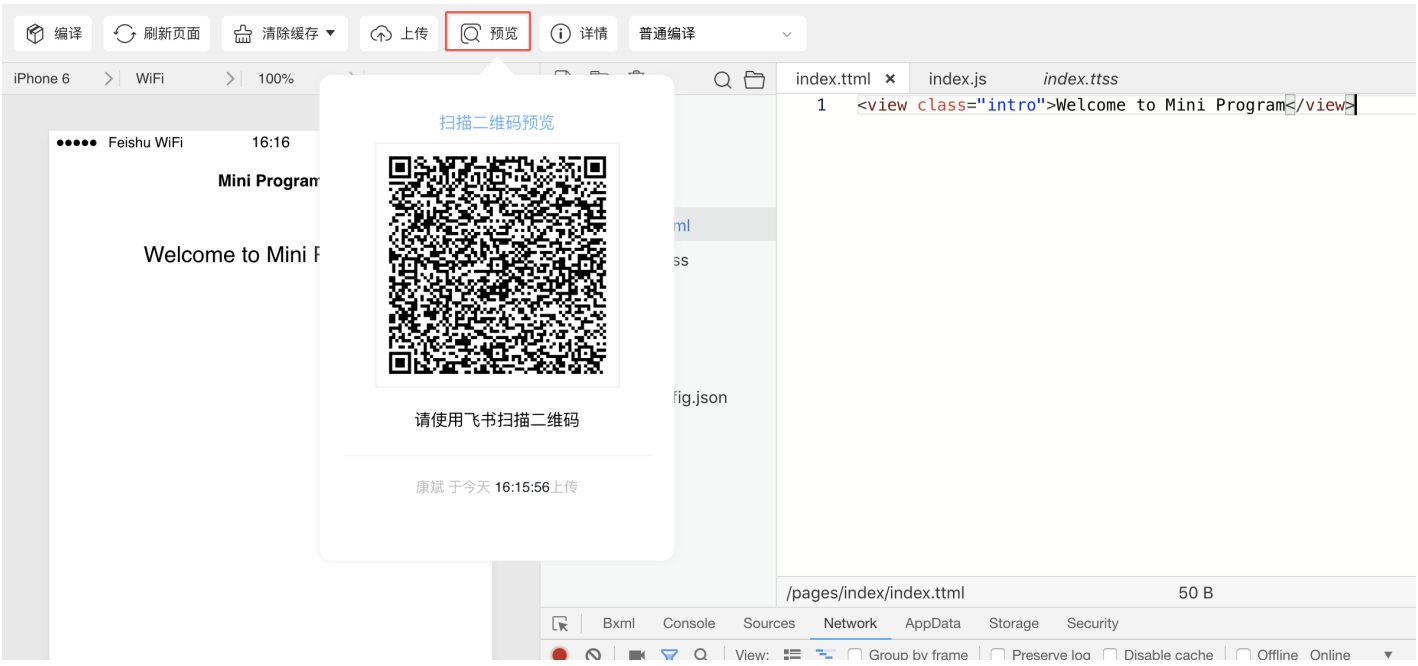
点击刚刚创建的应用，进入应用配置页，点击复制 App ID 即可。



#3.1.2 创建飞书小程序版本的 Hello World

下载并打开飞书小程序 IDE，点击新建按钮，选择一个空的目录作为工程目录，复制刚刚创建的应用的 AppID，点击确认即可创建一个飞书小程序。

至此完成一个小程序的创建，我们可以点击 IDE 预览按钮，通过飞书手机客户端扫描二维码在飞书上预览我们刚刚创建的小程序。



#3.1.3 飞书小程序项目结构介绍

我们需要对飞书小程序的项目结构做个简单的介绍，方便同学们了解特定文件的作用，完整的信息可以查看文档。

小程序的基础目录结构如下：

```
|____app.ttss
|____app.json
|____project.config.json
|____pages
|       |____index
|       |        |____index.js
|       |        |____index.json
|       |        |____index.ttml
|       |        |____index.ttss
|____app.js
```

以上目录结构中的 pages/可以根据实际情况进行灵活配置。小程序的主体由下面单个文件组成：

文件	必填	作用
app.js	是	小程序入口逻辑
app.json	是	小程序公共设置，例如：所有页面路径等
app.ttss	是	小程序公共样式

一个小程序页面由四个文件组成，如下：

文件	必填	作用
js	是	页面逻辑
json	否	页面配置
ttss	否	页面样式表
ttml	是	页面结构

此处我们介绍一下 `app.json` 的一些配置。

`app.json`是一个用来对飞书小程序进行全局配置的文件，用来配置页面的路径，窗口样式表现等。`app.json`的配置选项如下：

```
{
  "pages": ["pages/index/index", "pages/logs/index"],
  "window": {
    "navigationBarTitleText": "Demo"
  },
  "tabBar": {
    "list": [
      {
        "pagePath": "pages/index/index",
        "text": "首页"
      },
      {
        "pagePath": "pages/logs/logs",
        "text": "日志"
      }
    ]
  },
  "permission": {
    "scope.userLocation": {
      "desc": "你的位置信息将用于小程序位置接口的效果展示"
```

```
21      }
22    }
23  }
```

配置项说明：

属性	类型	必填	描述	最低版本
pages	String Array	是	配置页面路径	
window	Object	否	配置默认页面的窗口表现	
tabBar	Object	否	配置底部 tab 的表现	
navigateToMiniProgramAppIdList	Array	否	需要跳转的小程序列表，相关 api 见tt.navigateToMiniProgram	1.15.0
permission	Object	否	配置部分授权弹窗的副标题	

完整的配置说明，可以参考官方文档：全局配置。

#3.1.4 逻辑层介绍

#3.1.4.1 启动程序 App()

App(params) 是框架启动小程序的入口函数，需要开发者可以通过 App(params)的参数指定小程序的生命周期函数和其他一些自定义参数。

属性	类型	描述	触发时机
onLaunch	Function	生命周期函数--监听小程序初始化	当小程序初始化完成时，会触发 onLaunch（全局只触发一次）
onShow	Function	生命周期函数--监听小程序显示	当小程序启动，或从后台进入前台显示，会触发 onShow
onHide	Function	生命周期函数--监听小程序隐藏	当小程序从前台进入后台，会触发 onHide
onError	Function	错误监听函数	当小程序发生脚本错误，或者 api 调用失败时，会触发 onError 并带上错误信息
其他	Any		开发者可以添加任意的函数或数据到 Object 参数中，用 this 可以访问

#3.1.4.2 启动页面 Page()

`Page(params)` 是进入某个页面的时候会执行的页面入口函数，params 是一个 object 类型的参数，定义了页面初始数据，生命周期钩子函数，事件处理函数等。

属性	类型	描述
data	Object	页面的初始数据
onLoad	Function	生命周期函数--监听页面加载
onReady	Function	生命周期函数--监听页面初次渲染完成
onShow	Function	生命周期函数--监听页面显示
onHide	Function	生命周期函数--监听页面隐藏
onUnload	Function	生命周期函数--监听页面卸载
onPullDownRefresh	Function	页面相关事件处理函数--监听用户下拉动作
onReachBottom	Function	页面上拉触底事件的处理函数
onShareAppMessage	Function	用户点击右上角转发
onPageScroll	Function	页面滚动触发事件的处理函数
其他	Any	开发者可以添加任意的函数或数据到 object 参数中，在页面的函数中用 this 可以访问

启动参数

onLoad 生命周期函数会接受到当前页面打开时设置的 **query** 参数。比如：

```
// 页面url是'page/index/index?a=1&key=value'
onLoad: functions (options) {
        console.log(options) // {a: '1', key: 'value'}
}
```

页面首次渲染

初始化数据将作为页面的第一次渲染。data 将会以 JSON 的形式由逻辑层传至渲染层，所以其数据必须是可以转成 JSON 的格式：字符串，数字，布尔值，对象，数组。渲染层可以通过 TTML 对数据进行绑定。示例代码：

```
<!-- index.ttml -->
<view>{{text}}</view>
<view>{{array[0].msg}}</view>
```

```
// index.js
```

```
Page({
  data: {
    text: "init data",
    array: [{ msg: "1" }, { msg: "2" }]
  }
});
```

页面事件处理

除了初始化数据和生命周期函数，Page 中还可以定义一些特殊的函数：事件处理函数。在渲染层可以在组件中加入事件绑定，当达到触发事件时，就会执行 Page 中定义的事件处理函数。示例代码：

```
<!-- index.ttml -->
<view bindtap="viewTap"> click me </view>
```

```
// index.js
Page({
  viewTap: function() {
    console.log("view tap");
  }
});
```

更新页面渲染 setData()

字段	类型	必填	描述
data	Object	是	这次要改变的数据
callback	Function	否	回调函数

参数 data 以 key，value 的形式表示将 this.data 中的 key 对应的值改变成 value。 callback 是一个回调函数，在这次 setData 对界面渲染完毕后调用。示例代码：

```
<!--index.ttml-->
<view>{{text}}</view>
<button bindtap="changeText">Change normal data</button>
<view>{{num}}</view>
<button bindtap="changeNum">Change normal num</button>
```

```
<view>{{array[0].text}}</view>
<button bindtap="changeItemInArray">Change Array data</button>
<view>{{object.text}}</view>
<button bindtap="changeItemInObject">Change Object data</button>
<view>{{newField.text}}</view>
<button bindtap="addNewField">Add new data</button>
```

```
//index.js
Page({
  data: {
    text: "init data",
    num: 0,
    array: [{ text: "init data" }],
    object: {
      text: "init data"
    }
  },
  changeText: function() {
    // this.data.text = 'changed data'  // 这样无法更新UI
    this.setData({
      text: "changed data"
    });
  },
  changeNum: function() {
    this.data.num = 1;
    this.setData({
      num: this.data.num
    });
  },
  changeItemInArray: function() {
    this.setData({
      "array[0].text": "changed data"
    });
  },
```

```
  changeItemInObject: function() {
    this.setData({
      "object.text": "changed data"
    });
  },
  addNewField: function() {
    this.setData({
      "newField.text": "new data"
    });
  }
});
```

#3.1.5 视图层介绍

#3.1.5.1 TTML

TTML 是用来编写页面结构用的标签语言。主要包括下面一些特性：

数据绑定

```
<!--ttml-->
<view> {{message}} </view>
```

```
// page.js
Page({
  data: {
    message: "Hello World!"
  }
});
```

列表渲染

```
<!--ttml-->
<view tt:for="{{array}}"> {{item}} </view>
```

```
// page.js
Page({
  data: {
```

```
    array: [1, 2, 3, 4, 5]
  }
});
```

条件渲染

```
<!--ttml-->
<view tt:if="{{view == 'A'}}"> A </view>
<view tt:elif="{{view == 'B'}}"> B </view>
<view tt:else="{{view == 'C'}}"> C </view>
```

```
// page.js
Page({
  data: {
    view: "A"
  }
});
```

事件

```
<!--ttml-->
<view bindtap="add"> {{count}} </view>
```

```
// page.js
Page({
  data: {
    count: 1
  },
  add: function(e) {
    this.setData({
      count: this.data.count + 1
    });
  }
});
```

#3.1.5.2 TTSS

TTSS 是一套样式语言，用于描述 TTML 的组件样式。为了适应广大的前端开发者，TTSS 具有 CSS 大部分特性。同时 TTSS 对 CSS 进行了扩充以及修改。与 CSS 相比，TTSS 扩展的特性有：**尺寸单位**和**样式导入**。

尺寸单位

rpx（responsive pixel）: 可以根据屏幕宽度进行自适应。规定屏幕宽为 750rpx。如在 iPhone6 上，屏幕宽度为 375px，共有 750 个物理像素，则 750rpx = 375px = 750 物理像素，1rpx = 0.5px = 1 物理像素。

设备	rpx 换算 px (屏幕宽度/750)	px 换算 rpx (750/屏幕宽度)
iPhone5	1rpx = 0.42px	1px = 2.34rpx
iPhone6	1rpx = 0.5px	1px = 2rpx
iPhone6 Plus	1rpx = 0.552px	1px = 1.81rpx

> **建议**：设计师可以用 iPhone6 作为视觉稿的标准。

样式导入

使用`@import`语句可以导入外联样式表，`@import`后跟需要导入的外联样式表的相对路径，用`;`表示语句结束。示例代码：

```
/** common.ttss **/
.small-p {
  padding: 5px;
}
```

```
/** app.ttss **/
@import "common.ttss";
.middle-p {
  padding: 15px;
}
```

#3.1.5.3 组件

小程序框架为开发者提供了一系列基础组件，开发者可以通过组合这些基础组件进行业务开发。

什么是组件：

- 在 ttml 中使用组件，是视图层的基本组成单元。
- 一个组件通常包括开始标签和结束标签，属性用来修饰这个组件，内容在两个标签之内。

```
<text prop-name="propValue">你好，小程序</text>
```

> 注意：所有组件与属性都是小写，以连字符-连接

更多的组件类型，可以参考官方文档：组件。

#3.2 如何使用轻服务

3.2.1轻服务的介绍与优势

轻服务是面向未来的云服务产品，提供开箱即用的开发体验。开发者无需考虑服务器和数据库等基础设施的搭建，更不用操心测试环境配置、数据备份和线上运维等一系列繁琐之事，只需专注于产品开发本身。

在轻服务登录页可以通过手机号码进行短信登录，需要通过实名认证才能使用轻服务。

3.2.2 云函数入门

云函数是组成轻服务应用的基本单元，也是开发者编写主要业务和功能的地方。

#3.2.2.1 创建云函数

一个最简单的云函数：

```
module.exports = async function(params, context) {
  console.log(params);
  return {
    test: "Hello World!",
  };}

```

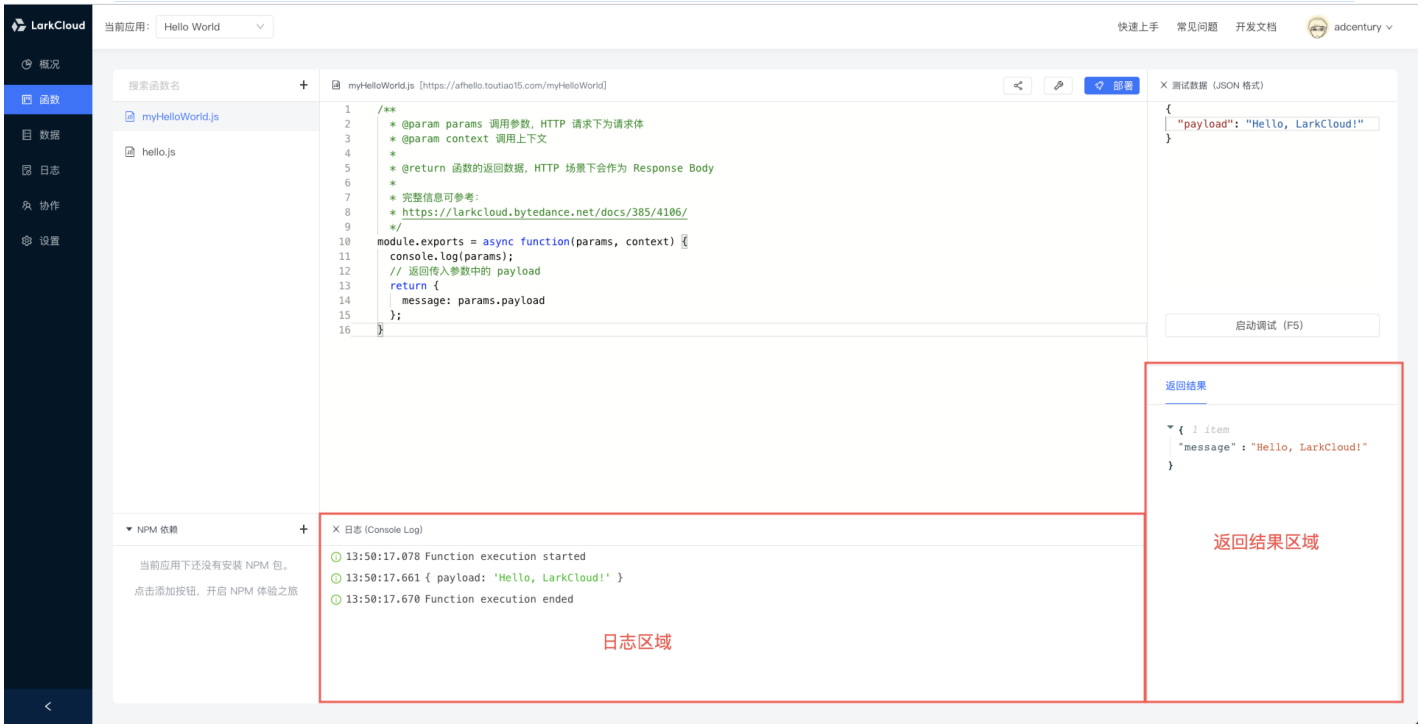
- @param params：通过 HTTP 访问函数时，params 为请求体 body 内容。
- @param context: 代表请求的上下文信息，拥有以下对象属性：
  - context.method：HTTP 请求的 Method，可能的值为 `GET`、`POST`、`PUT`、`DELETE`；
  - context.query：HTTP 请求的 Query String，?a=1&b=ok 会被转换为 `{ a: '1', b: 'ok' }`；
  - context.headers：HTTP 请求的 Headers，需注意 key 的字母均为**小写**，如 `content-type`；

- context.set(key, value)：设置返回头的方法，如：`context.set('x-my-header', 'hello')`；
- context.status(code)：设置返回的 Status Code，如: `context.status(301)`。

- @return: 函数的返回数据，HTTP 场景下会作为 Response Body

#3.2.2.2 调试云函数

调试是开发中必不可少的部分，轻服务提供了一套方便快捷的在线调试和修改函数方案。



具体操作办法：

- 调试入口位于「函数」页面右侧。任何时候，点击「启动调试」按钮，即会按照本地已修改的最新函数版本运行调试。
- 调试时，通过右上角的「测试数据」区域，可以携带调试数据，这些数据会被体现在函数的 params 参数中。（测试数据必须是 `JSON` 格式。）
- 完成调试后，可以在页面中的右下角「返回结果」区域，查看函数的执行结果。在下方弹起的「日志」区域可以查看函数执行过程中产生的日志（函数中可通过 `console.log` 产生日志）。

#3.2.2.3 部署云函数

轻服务中的云函数需要部署后才能被外部访问，若对云函数内容进行了修改，也需要部署后才能在线上生效。

具体操作：

- 在「函数」页面中，点击编辑器右上方的「上线」按钮，在弹出的对话框中确认即可开始部署函数。
- 默认情况下，每次部署只针对当前选中的函数，若希望一次性部署该应用下的所有函数，只需打开「上线所有函数」选项即可。

#3.2.2.4 调用云函数

轻服务的云函数支持通过 HTTP 请求进行调用。具体操作办法：

- 上线完成后，在编辑器左上方的函数名称之后，会出现函数的调用地址，点击时将会复制该地址。
- 通过向该地址发送 HTTP 请求（`GET`、`POST` 均可），即可实现对函数的调用。
- 另外，在小程序中，还有另外一种通过 SDK 调用云函数的方法，详见3.3.

3.2.3 如何通过轻服务 SDK 对数据库进行增删改查

轻服务提供了一套简单易用的数据库功能，在云函数中，只需通过 `larkcloud.db` 即可进行操作，例如：

#3.2.3.1 插入一条记录

```
module.exports = async function(params, context) {
  // 使用 larkcloud.db.table 获取数据表
  const GoodsTable = larkcloud.db.table('goods');
  // 使用 save 方法新增一条记录
  const result = await GoodsTable.save({name: 'iPhone XS Max', price: 8000});

  return {
    result,
  };
}
```

#3.2.3.2 查询一条记录

```
module.exports = async function(params, context) {
  // 使用 larkcloud.db.table 获取数据表
  const GoodsTable = larkcloud.db.table('goods');
  // 调用 where 按 name 查找，调用 findOne 返回一条结果
  const result = await GoodsTable.where({name: 'iPhone XS Max'}).findOne();

  return {
    result,
  };
}
```

#3.2.3.3 修改一条记录

```
module.exports = async function(params, context) {
  // 使用 larkcloud.db.table 获取数据表
  const GoodsTable = larkcloud.db.table('goods');
  // 调用 where 按 name 查找，调用 findOne 返回一条结果
  const doc = await GoodsTable.where({name: 'iPhone XS Max'}).findOne();
  // 将价格字段增加500
  doc.price += 500;
  // 调用 save 将修改的数据记录存回数据库
  const result = await GoodsTable.save(doc);

  return {
    result,
  };
}
```

#3.2.3.4 删除一条记录

```
module.exports = async function(params, context) {
  // 使用 larkcloud.db.table 获取数据表
  const GoodsTable = larkcloud.db.table('goods');
  // 调用 where 按 name 查找，调用 findOne 返回一条结果
  const doc = await GoodsTable.where({name: 'iPhone XS Max'}).findOne();
  // 调用 delete 将数据从数据库中删除
  const result = await GoodsTable.delete(doc);

  return {
    result,
  };
}
```

#3.3 如何将飞书小程序与轻服务结合使用

飞书小程序本质上是个前端应用，需要借助轻服务去实现后端的数据存储。我们通过轻服务的 SDK，调用云函数去实现后端的业务逻辑。

#3.3.1 集成轻服务的 SDK

点击打开 larkcloud-mp.min.js 并下载 JS 文件，移动到小程序 `libs` 目录。

#3.3.2 如何在飞书小程序中调用云函数

轻服务支持在**微信小程序**和**头条小程序**中调用云函数，使用 JS SDK 的 `larkcloud.run` 方法即可。

代码示例：

```
// 在小程序中，需要先下载 SDK，并移动至 libs 目录
const LarkCloud = require('./libs/larkcloud-mp-alpha.min.js');
const serviceId = 'xxx'; // 替换成你的 serviceId，可在轻服务后台「设置」页面获取
// 初始化
const larkcloud = new LarkCloud({ serviceId });
// 云函数名称
const fnName = 'hello';
// 通过 larkcloud.run 调用云函数
larkcloud.run(fnName, { message: 'hello' })
  .then(data => {
    // 处理正常结果
  })
  .catch(error => {
    // 处理异常结果
  });

```

#4 团建小程序开发讲解

#4.1 团建地点列表

#4.1.1 飞书小程序代码讲解（TTML、TTSS、JS）

本功能位于页面 recommand, 为本小程序的首页。
本页面的加载时机是在 App.onLaunch 中，即当小程序初始化完成时会直接 tt.redirectTo 到本页面。
本页面相关代码目录为 pages/recommand。下面重点关注下该目录下的 recommand.ttml 和 recommand.ttss

#1 recommand.ttml

定义 recommand 页面的页面结构。重点代码：

```
<view
    class="recommand-item"
    tt:for="{{recommandList}}"
    id="item-{{index}}"
    ...
>
    <view class="content">
        <image class="img" src="{{item.img}}" />
        <text class="name">{{item.name}}</text>
        ...
    </view>
  </view>

```

使用 tt:for 进行列表渲染：

- tt:for 指定用于渲染的数组数据，这里即为 recommandList。（关于这个数据 recommandList 是从哪里来的，会在 recommand.js 中讲到）
- 默认数组的当前项的下标变量名默认为 index，数组当前项的变量名默认为 item.

#2 recommand.js

定义本页面的页面入口函数。在进入本页面时即会执行。重点代码：

```
Page({
  data: {
    recommandList: [],
    ...
  },
  onLoad: function (options) {
    tt.showLoading({
        title: '光速加载中...',
    });
    larkCloud.run('getRecommands').then(data => {
      tt.hideToast({});
      this.setData({
        recommandList: data.map((item) => ({
          ...item,
          starList: Array(Math.floor(+item.star)).fill()
```

```
        }))
      });
    });
  },
  ...
})
```

关键技术点：

Page() 参数data 与 this.setData 方法

- Page（）的参数 data 用于定义页面初始数据。这里我们定义了初始数据 recommandList 为 [] , 即初始时这个推荐数据数组为空。
- 在执行一些操作（包括生命周期、事件处理函数等）的时候，可以通过 this.setData() 方法更新 data 数据。
- data 中的数据可以直接在对应的 ttml 中使用，例如 recommand.ttml 中就直接使用了 recommandList。当 data 通过 this.setData() 更新时，recommand.ttml 中的对应数据会更新页面渲染。

larkCloud.run 调用云函数

- 这里通过 larkClud.run 调用 'getRecommands', 即调用了轻服务中对应的云函数，本质上是发起了一次 http 请求。
- 该云函数的返回值可以通过 then 获取并进行处理。这里调用 getRecommands 得到的返回值 data 即为我们需要的 recommandList 数据。
- getRecommands 云函数具体细节参见 4.1.2

tt.showLoading() 与 tt.hideToast

- 调用 tt.showLoading 可以显示 loading 提示框。在 Page 一开始执行 onLoad (页面加载生命周期）时，这时候还没有请求得到数据，可以先调用 方法让页面展示一个加载中的过渡提示。
- 调用 tt.hideToast 可以隐藏 loading 提示框。当执行 larkCloud.run 调用云函数获取到数据之后，执行该方法隐藏提示框，展示数据列表。

#4.1.2 轻服务代码讲解

小程序 recommands 页面涉及调用的云函数只有 getRecommands 一个。

#1 getRecommands 云函数

重点代码：

```
module.exports = async function() {
```

```
  const totalResult = await larkcloud.db.table('shops').where().find(); // 从 shops 表中获取所有数据

  const typeArr = Object.keys(totalResult.reduce((acc, cur) => {
      const type = cur.type;
      pre[type] = true;
      return acc;
    }, {})); // 将所有数据的 type 字段提取出来，得到一个 type 数组，这里为  [ 'indoor', 'food', 'outdoor', '文艺', '温泉', '火锅', '烧烤烤串', '自助餐' ]


  const randomType = ['', '', ''].map(item => typeArr[Math.floor(Math.random() * 3)]); // 只使用 typeArr 的前三种 （indoor、food、outdoor），并将其随机打乱顺序，得到例如 [ 'outdoor', 'indoor', 'food' ]

  const randomSkip = Math.floor(Math.random() * 200); // 得到一个 0～199 的随机整数

  const result = await Promise.all(randomType.map(type => {
      return larkcloud.db.table('shops').where({ type }).sort({ star: -1 }).skip(randomSkip).limit(30).find();// 在 shop 表中分别查找三种 type 的数据。针对查找到的每种 type 的数据按照 star 降序排序、然后跳过前 randomSkip 项、获取 30 个。 最终反回的是 3*30 的二维数组
    }));

  const recommandList = _.range(10).map(i => {
    const res = result[i % 3 ];
    const length = res.length;
    const index = Math.floor(Math.random() * length);
    return res[index];
  }); // 按照代码所示随机策略从 result 这个 3*30 的数组中抽取 10 项组成 recommandList 数组

  return _.filter(recommandList, (v) => { // 去掉 recommandList 中没有价格的项目
      return !!v.price;
  });
}
```

轻服务数据库的数据查询操作

- larkcloud.db.table 获取数据表：此处 larkcloud.db.table('shops') 获取到 shops 数据表，该表里存储有商家数据信息
- table.where({字段名:值}).find() 查询字段值等于某个值的数据：此处 .where({ type }) 即获取指定 type 的数据
- skip(n) 指定跳过 n 项
- limit(n) 指定返回 n 项

推荐随机策略

- 将三种 type 顺序随机打乱
- 每种 type 的数据按照 star 降序排序，然后跳过前若干随机数项，获取 30 个。
- 最后的10个数据按顺序轮流为三种 type 的数据。每种 type 中被选中的数据再从这个 type 的所有数据中随机抽取得到。

> 这个策略不是最好的，有修改空间。可以思考下这个逻辑有什么问题。

#4.2 获取飞书群数据

获取飞书群数据使用了飞书开放平台的 API ，允许开发者对直接获取飞书的群数据信息。

飞书开放平台的接口在请求时需要在请求头传入 `Authorization: Bearer ${userAccessToken}`，所以首先要通过一系列 登录、验证逻辑获取 userAccessToken 。

#4.2.1 飞书开放平台简介

飞书开放平台提供了丰富的 API，允许开发者开发第三方应用，充分扩展飞书的能力。

#4.2.2 获取飞书群数据的小程序代码

#1 app.js

重点代码：

```
App({
  onLaunch: function () {
    tt.login({
        success (res) {
          larkCloud.run('getCustomSession', {
                code: res.code
```

```
            }).then(customSession => {
                tt.setStorage({
                    key: 'customSession', // 缓存数据的key
                    data: customSession, // 要缓存的数据
                });
        }
    })
  }

```

小程序 api : tt.login()

- 获取临时登录凭证，得到的 res.code 就是临时登录凭证， 有效期 3 分钟。参见文档：login。
- tt.login 返回的 code 通过调用云函数 getCustomSesssion 发送给服务端，之后服务端会进行一系列处理得到 customSession （详见 4.2.3 ）, 通过 tt.setStorage 存储在本地。

#2 page/create/create.js

重点代码：

```
handleGroupConfirm(e) {
    larkCloud.run('getGroup', {
      keywords: e.detail.value,
      sessionKey: this.data.customSession // 将存在本地的 customSession 作为 body 数据发回服务端
    }).then(data => {
      this.setData({
        groupList: data.data.groups
      })
    });
  },
```

调用云函数 getGroup 的时候要将 customSession 发送回服务端。

#4.2.3 获取飞书群数据的轻服务代码讲解

#1 getAcessToken 云函数

通过 APP_ID 和 APP_SECRET 获取 app_access_token ，即获取调用飞书开发平台的接口获取应用资源时的凭证。重点代码：

```
const ACCESS_URL = 'https://open.feishu.cn/open-apis/auth/v3/app_access_token/internal/';

async function getAccessTokenFromLark() {
  const { data } = await axios({
    url: ACCESS_URL,
    method: 'POST',
    data: {
      app_id: APP_ID,
      app_secret: APP_SECRET,
    },
  });
    return data.app_access_token;
}

function saveToken(key, value, expire) {
  return larkcloud.redis.setex(key, expire, value);
}

module.exports = async function getAccessToken() {
  const existsToken = await larkcloud.redis.get(ACCESS_TOKEN_KEY); // 先去 redis 获取这个值
  if (!existsToken) { // redis 获取不到再执行 getAccessTokenFromLark 获取 accessToken
    const token = await getAccessTokenFromLark();
    saveToken(ACCESS_TOKEN_KEY, token, ACCESS_TOKEN_EXPIRE).catch(e => console.error(e.message));
    return token;
  }
  return existsToken;
}
```

APP_ID 和 APP_SECRET

- 来自飞书开放平台 > 我的应用 （这里是我们创建的这个 TB 小程序，从这里拷贝下来，只有创建这个小程序的人可以看到）
- 这两个值需要存入对应本轻服务项目的环境变量。使用的时候可以通过 larkCloud.env.APP_ID / APP_SECRET 获取

调用飞书开放平台接口获取 app_access_token

- 企业自建应用（这里就是我们的小程序）需要通过此接口获取 app_access_token。因为后续调用飞书开发平台的接口获取应用资源时，需要使用 app_access_token 作为授权凭证。参见文档：获取 app_access_token（企业自建应用）
- 调用此接口时需要将 APP_ID 和 APP_SECRET 作为 body 数据

larkcloud.redis

- redis (高速缓存)是一个 key-value 的数据存储系统。参见文档：高速缓存 Redis
- 这里获取到 app_access_token 后会通过 larkcloud.redis.setex(key, expire, value) 将该值存储下来，之后获取 app_access_token 会先从 redis 里访问

#2 getCustomSession 云函数

通过code （小程序前端通过 login 接口获取到 ）、app_access_token (服务端通过 getAccessToken 云函数得到） 获取 sessionKey （会话密钥）等信息。把这些信息存储在数据表和 redis， 返回一个 customSession 用于查找 sessionKey 等信息。重点代码：

```
module.exports = async function(params) {
    const { code } = params; // 小程序传递过来的，参见 4.2.2 1.
    const accessToken = await getAccessToken(); // 获取企业自建应用（就是我们的小程序）的 app_access_token，参见 4.2.3 1.

    const { data } = await axios({ // 请求飞书开放平台接口 https://open.feishu.cn/open-apis/mina/v2/tokenLoginValidate
        url: LOGIN_URL,
        method: 'POST',
        headers: {
          Authorization: `Bearer ${accessToken}`,
        },
        data: {
          code,
        },
    });
    const user = {
        uid: data.data.uid,
        openId: data.data.open_id,
        unionId: data.data.union_id,
```

```
        sessionKey: data.data.session_key, // 会话密钥
         ...
      };
    const userData = await userUpsert({ // 存入 user 数据表，然后 userData 就生成了唯一的 _id        .
        ..user,
        accessToken: data.data.access_token,
        refreshToken: data.data.refresh_token,
    });

  const customSession = await saveSession(userData._id.toString()); // 以 user._id 为值、8位随机数为 key 存入 redis。
  return customSession; // 返回这个 8 位随机数
}

async function userUpsert(user) {
  await larkcloud.db.table('user').where({ unionId: user.unionId }).upsert().set(user)
    .save(); // 根据 unionId 插入/更新 user 数据到 user 表，然后返回这个表的本条数据，这样这条数据多了唯一标志符 _id
  return larkcloud.db.table('user').where({ unionId: user.unionId }).findOne();
}

async function saveSession(objectId) {
  // @param objectId 是 user 数据表中每条数据的唯一标志符 _id，它这个是通过进行数据库插入操作产生的唯一值
  const sessionKey = createSessionKey();
  await larkcloud.redis.setex(sessionKey, CUSTOM_SESSION_EXPIRE, objectId); // 将这个 user._id 为值、8位随机数为 key 存入 redis
  return sessionKey;
}

function createSessionKey() {
  return randomBytes(8).toString('hex');
}

```

飞书开放平台接口

- 当小程序前端通过 login 获取到 code、服务端得到的 app_access_token 后，服务端调用飞书开放平台接口 可以获取 sessionKey 、openId 等信息。 详见文档：code2session
- 调用此接口时，code 作为 body，app_access_token 作为请求头 Authorization的一部分

轻服务的数据库存储操作

- 参见文档：数据库入门
- 将调用飞书开放平台接口得到的一堆 sessionKey、openId 等数据，存入 user 数据表，存入的时候会生成一个唯一的_id 。

给前端返回 customSession

- 以 user 数据中的唯一 _id 为值、一个随机数 customSession 为 key 存到 redis 里面。
- 最终给前端返回的是这个 customSession 。之后前端请求的时候带上这个 customSession, 就可以去 redis 里面查到 user 数据、进而查到 user.accessToken 数据。

#3 getUserInfo 云函数

服务端拿到 customSession 后去 redis 里面查到对应值 _id ，然后再去 user 数据表里面根据 _id 拿到这个 user 的数据 。重点代码：

```
function findSession(sessionKey) {
  return larkcloud.redis.get(sessionKey);
}

module.exports = async function(params) {
  const { sessionKey } = params; // 前端请求 body 里的 customSession
  const objectId = await findSession(sessionKey); // 去 redis 里面查到对应值 _id

  if (!objectId) {
    console.log('error');
  }

  const user = await larkcloud.db.table('user').where({ // 去 user 表里拿到所有数据
    _id: new (larkcloud.db.ObjectId)(objectId)
  }).findOne()

  if (!user) {
    throw new Error('未找到用户');
```

```
  }
  return user; // 之后使用 user.acessToken
}

```

#4 getGroup 云函数

拿到 user.accessToken 之后再去请求飞书开放平台的一些搜索群的接口拿业务数据。重点代码：

```
const SEARCH_GROUP_URL = 'https://open.feishu.cn/open-apis/chat/v4/search';

async function searchLarkGroup(keywords, userAccessToken, pageSize) {
  const { data } = await axios({
    url: SEARCH_GROUP_URL, // 请求飞书开放平台的接口，获取 lark 群数据
    method: 'GET',
    headers: {
      Authorization: `Bearer ${userAccessToken}`,
    },
    params: {
      page_size: pageSize,
      query: keywords,
      // page_token: pageToken,
    },
  }).catch(e => {
    if (e.response && e.response.data) {
        console.error(JSON.stringify(e.response.data));
      } else {
        console.error(e.message);
      }
    });

    return data;
  }
```

飞书开放平台搜索群的接口

- 接口说明参见：搜索用户所在的群列表；

- 请求此接口的时候： userAccessToken 作为请求头的 Authorization 的一部分；query（搜索文本)、page_size(获取数量）作为 body

#5 相关资料

将相关资料可以打成压缩包或者放在 Github 上。

- 团建小程序客户端代码
- 团建小程序轻服务代码
- 轻服务的小程序版 SDK