Astro 轻应用

最佳实践

文档版本	01
发布日期	2024-04-25

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 用户与权限

1.1 什么是业务用户

什么是业务用户

AstroZero中存在两类用户：用户（User）和业务用户（PortalUser）。

- 用户（User）是指在AstroZero中添加的IAM用户或WeLink用户，是应用的开发者。
- 业务用户（PortalUser）是访问在AstroZero中开发的一个业务应用的用户账号。当一个项目使用了AstroZero提供的业务服务功能，而业务用户是用这个项目注册的账号来开通试用业务系统的，并不是直接注册AstroZero的账号，这种用户即是AstroZero的业务用户。

例如，在AstroZero中注册了账号“A”，并使用账号“A”在AstroZero中开发了一个应用“X”。账号“B”是通过应用“X”注册的账号，并登录使用应用“X”。这种情况下，账号“A”是管理员用户（User），账号“B”是业务用户（PortalUser）。关于用户和业务用户的更多介绍，请参见**图解AstroZero中用户那些事**。

业务用户拥有哪些能力

- 业务用户可以根据权限，访问应用、查看导航条菜单。
- 业务用户可以根据权限，查看应用的页面布局。
- 业务用户可以根据权限，查看、创建、编辑和删除应用中的对象。
- 业务用户可以根据权限，查看和编辑应用中的对象字段。
- 业务用户可以根据权限，执行应用中开发的服务编排、脚本和工作流。
- 业务用户可以根据权限，访问在应用中自定义的公共接口。

说明

在AstroZero中新添加的业务用户，若没有配置指定的权限，默认使用系统预置的Portal User Profile权限。若业务用户需要额外的权限，需要对业务用户进行权限的配置，具体操作请参见**如何给业务用户添加业务权限凭证**。

在开发环境新增业务用户

步骤1 使用华为账号，登录**华为云网站**，在顶部导航栏右侧单击“控制台”，进入华为云控制台。

步骤2 在左侧导航栏上方，单击，选择服务实例所在的区域项目。

步骤3 单击，在查找框中搜索“Astro轻应用（原AppCube）”，单击查找到的结果，进入AstroZero服务控制台。

步骤4 单击“进入首页”，进入“应用开发”页面。

步骤5 单击页面左上角的，选择“开发环境管理 > 业务配置中心”，进入AstroZero开发环境业务配置中心。

图 1-1 进入开发环境业务配置中心

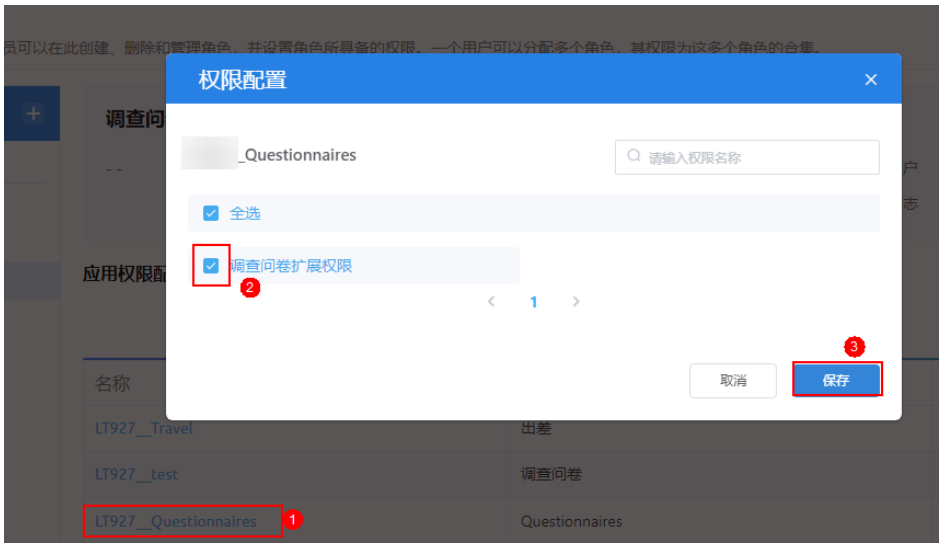
步骤6 （可选）创建角色权限，若还未规划给业务用户赋予权限，可跳过该步骤。

业务配置中心的角色权限都是基于权限“Portal User Profile”创建而来的。不同的业务用户权限可能有所不同，这时可通过配置扩展权限集给业务用户分配不同的权限。关于扩展权限集的更多介绍，请参见**设置扩展权限集**。

1. 在“业务配置中心”页面的主菜单中，选择“组织用户”。
2. 在左侧导航栏中，选择“角色权限”。
3. 单击“所有角色”后的“+”，设置添加的角色名称，例如“调查问卷权限”，并配置应用为可见和默认应用。

4. 给该权限添加扩展权限集，即在应用中导入扩展权限集。

图 1-2 添加扩展权限集



步骤7 创建业务用户，例如“testQuestionnaires”。

1. 在“业务配置中心”页面的主菜单中，选择“组织用户”。
2. 在左侧导航栏中，选择“用户管理”，单击“创建用户”。
3. 在“新建用户”页面，填写用户信息，用户名设置为“testQuestionnaires”。
4. （可选）为业务用户添加角色。
   - 若未创建角色权限，“角色”区域可暂不指定权限，后续再对业务用户进行权限配置。
   - 若已创建角色权限，则在“角色”区域中从左侧勾选已创建的角色，如“调查问卷权限”，再单击“>”赋予该用户相应的权限。
5. 设置完成后，单击“保存”。

----结束

在运行环境新增业务用户

步骤1 使用华为账号，登录**华为云网站**，在顶部导航栏右侧单击“控制台”，进入华为云控制台。

步骤2 在左侧导航栏上方，单击，选择服务实例所在的区域项目。

步骤3 单击，在查找框中搜索“Astro轻应用（原AppCube）”，单击查找到的结果，进入AstroZero服务控制台。

步骤4 单击“进入首页”，进入“应用开发”页面。

步骤5 在“工作台”页签中，单击页面左上角的，选择“业务配置中心”，进入运行环境业务配置中心。

图 1-3 进入运行环境业务配置中心

步骤6 （可选）创建角色权限，若还未规划给业务用户赋予权限，可跳过该步骤。

说明

购买AstroZero专业版实例时，才会显示“角色权限”菜单。

业务配置中心的角色权限都是基于权限“Portal User Profile”创建而来的。不同的业务用户权限可能有所不同，这时可通过配置扩展权限集给业务用户分配不同的权限。关于扩展权限集的更多介绍，请参见**设置扩展权限集**。

1. 在“业务配置中心”页面，选择“组织用户”。
2. 在左侧导航栏中，选择“角色权限”。
3. 单击“所有角色”后的“+”设置添加的角色名称，例如“调查问卷权限”，并配置应用为可见和默认应用。

4. 给该权限添加扩展权限集，该权限集是在应用中导入的扩展权限集。

图 1-4 添加扩展权限集

步骤7 创建业务用户，例如“testQuestionnaires”。

1. 在“业务配置中心”页面的主菜单中，选择“组织用户”。
2. 在左侧导航栏中，选择“用户管理”，单击“创建用户”。
3. 在“新建用户”页面，填写用户信息，用户名设置为“testQuestionnaires”。
4. （可选）为业务用户添加角色。
   - 若未创建角色权限，“角色”区域可暂不指定权限，后续再对业务用户进行权限配置。
   - 若已创建角色权限，则在“角色”区域中从左侧勾选已创建的角色，如“调查问卷权限”，再单击“>”赋予该用户相应的权限。
5. 设置完成后，单击“保存”。

----结束

1.2 如何添加业务权限凭证

为什么要添加业务权限凭证

通过业务权限凭证，可以控制AstroZero的用户对于自定义公共接口的访问权限。如果用户权限中包含了某个业务权限凭证，该用户将能够调用配置了相应业务权限凭证的自定义公共接口。

什么是业务权限凭证

业务权限凭证用于控制接口的访问权限，AstroZero提供了配置权限脚本和配置API接口两种方式来控制API接口的访问权限。建议优先通过权限脚本进行权限验证，根据脚本的返回值判断下一步的操作。

- 对于配置了业务权限凭证的接口，需要在权限的“业务权限凭证”页签中接入相应的业务权限凭证，才可调用API接口。
  - 若用户（User）没有配置业务权限凭证，则继续校验Profile权限中的数据权限（例如执行服务编排、执行脚本、对象的增删改查、API读、API写权限）。
  - 若业务用户（PortalUser）没有配置业务权限凭证，调用接口直接报错，不会继续校验Profile权限中的数据权限。
- 对于未配置业务权限凭证的接口，则需要查看内置系统参数“bingo.permission.customapi.check”取值，该参数控制公共接口未绑定业务权限凭证时的逻辑。
  - 若该参数取值为“是”，公共接口未绑定业务权限凭证时，业务用户（PortalUser）无法访问该接口；对于用户（User），则继续校验Profile权限中的数据权限。
  - 若该参数取值为“否”，公共接口未绑定业务权限凭证时，业务用户（PortalUser）可直接访问该接口，无需鉴权；对于用户（User），则继续校验Profile权限中的数据权限。

  例如：内置参数bingo.permission.customapi.check配置为“是”，接口未绑定业务权限凭证，即使在权限设置中勾选“API读”与“API写”权限，业务用户也无法操作对象数据。若业务用户需要操作对象数据，请将该内置参数配置为“否”。

  查看和配置该系统参数的方法为：在AstroZero管理中心选择“系统管理 > 系统参数”，在“内置系统参数”页签，查看和配置该参数。

图 1-5 内置系统参数

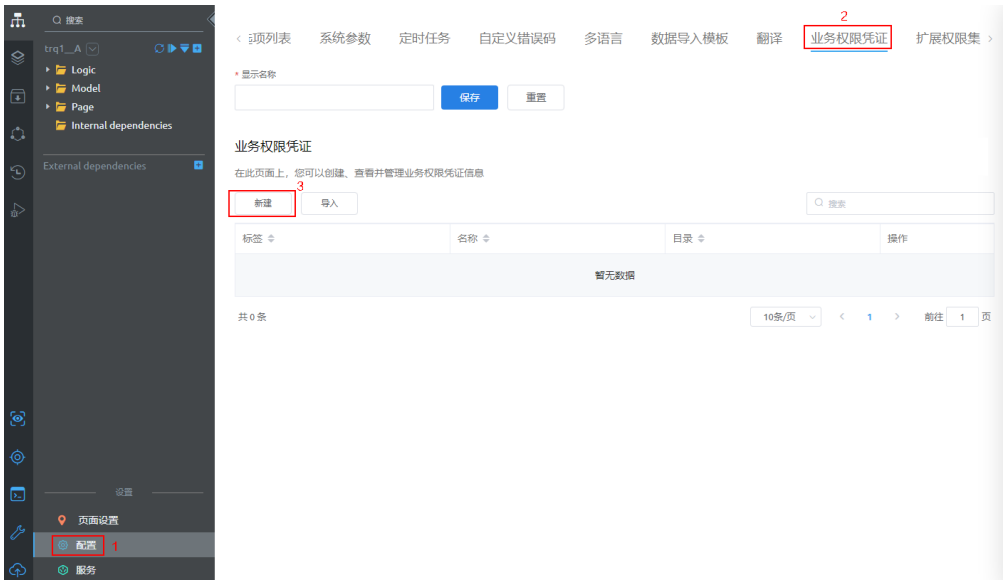
如何给业务用户添加业务权限凭证

步骤1 创建业务权限凭证。

以在“A”应用中添加业务用户业务权限凭证为例，创建1个业务凭证“cs”，即普通业务用户业务凭证。

1. 在AstroZero开发环境中，进入已创建的“A”应用中，单击“配置”。
2. 在“业务权限凭证”页签，单击“新建”。

图 1-6 在应用中创建业务权限凭证

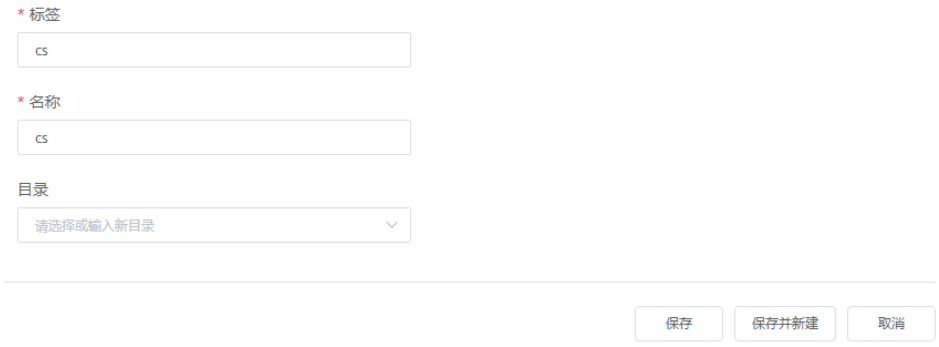


说明

如果您已经在AstroZero管理中心的“用户管理 > 业务权限凭证”中创建了业务凭证，可直接单击图1-6的“导入”，将已有的业务权限凭证导入当前应用中。

3. 在标签和名称输入框中，输入“cs”后，单击“保存并新建”，完成“cs”普通业务用户业务凭证的创建。

图 1-7 新建业务权限凭证



步骤2 创建权限配置。

业务用户的权限主要是在AstroZero预置的Portal User Profile权限基础上，进行自定义业务用户权限配置和拓展实现的。在AstroZero的权限配置功能中，基于某个权限配置

新建的Profile，将会继承原Profile的全部权限。在后续有新的业务用户注册时，只需要为新的业务用户配置对应的权限，即可获取该权限配置中的权限。

以在“A”应用中创建普通业务用户权限“csProfile”为例。

1. 登录AstroZero管理中心。
2. 在左侧导航栏中，选择“用户管理 > 权限配置”，单击“新建”。
3. “现有权限配置”选择“Portal User Profile”，克隆模式选择“普通克隆”，设置要新增的“权限配置名称”为“csProfile”，单击“保存”。

图 1-8 基于 Portal User Profile 新增权限配置

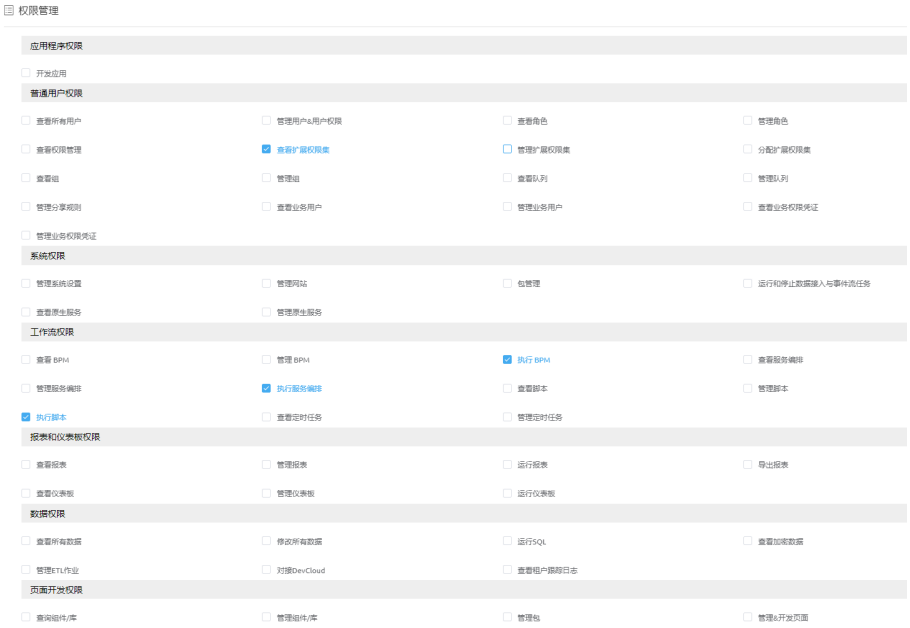
说明

- 克隆模式选择“继承克隆”时，除基本信息和业务权限凭证，其余权限屏蔽编辑按钮，如图1-9。
- 克隆模式选择“普通克隆”时，不会屏蔽编辑按钮。

图 1-9 继承克隆屏蔽编辑按钮

4. 在权限配置列表中，单击“csProfile”，进入权限配置详情。
5. 在“基本信息”页签，单击“基本信息”右侧编辑按钮，可进行权限配置。

图 1-10 权限配置



说明

这里的权限默认继承AstroZero预置的Portal User Profile权限基础，具体权限配置可以根据业务需要进行自由配置。

6. 在“业务权限凭证”页签，单击，勾选“cs”业务凭证，再单击，保存设置。

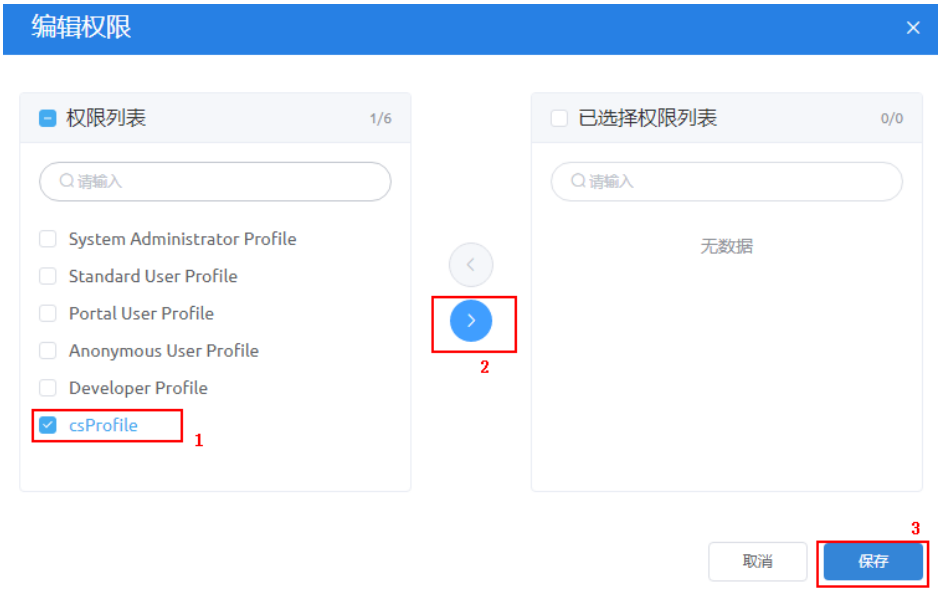
图 1-11 配置业务凭证权限

步骤3 配置业务用户权限。

在添加业务用户后，该业务用户是没有任何权限使用AstroZero提供的服务的。若需要该业务用户正常使用AstroZero，则需要为该用户配置相关的使用权限。以为“A”应用中的业务用户“test_cs”赋予csProfile权限为例。

1. 登录AstroZero管理中心。
2. 在左侧导航栏中，选择“用户管理 > 业务用户”。
3. 在业务用户列表中，单击需要配置权限的业务用户“test_cs”。
4. 在业务用户详情页面，单击“权限集”下的“编辑”。
5. 在编辑权限弹出框中，选中左侧列表的“csProfile”后，单击将选中的权限添加至右侧列表，单击“保存”完成权限的配置。

图 1-12 权限配置窗口



说明

- “csProfile”是基于“Portal User Profile”创建的扩展权限集。除了“csProfile”权限集，其他权限都是系统预置的权限集。对于系统预置的权限集，建议不进行修改，基于系统预置的权限集创建的权限集（如“csProfile”）继承了系统预置的所有权限。
- 勾选“覆盖业务用户权限”时，该业务用户所有权限读取“权限集”中的权限设置；不勾选时，该业务用户应用程序权限读取“Portal User Profile”设置，其余权限读取“权限集”中的权限设置。

图 1-13 覆盖业务用户权限配置

----结束

如何给接口添加业务权限凭证

若用户或业务用户需要通过权限访问接口，需要给该接口添加业务权限凭证。为了便于描述，下面以应用“A”为例，向您介绍如何给接口添加业务权限凭证。

步骤1 在“我的应用”中，单击“A”应用，进入“A”应用开发界面。

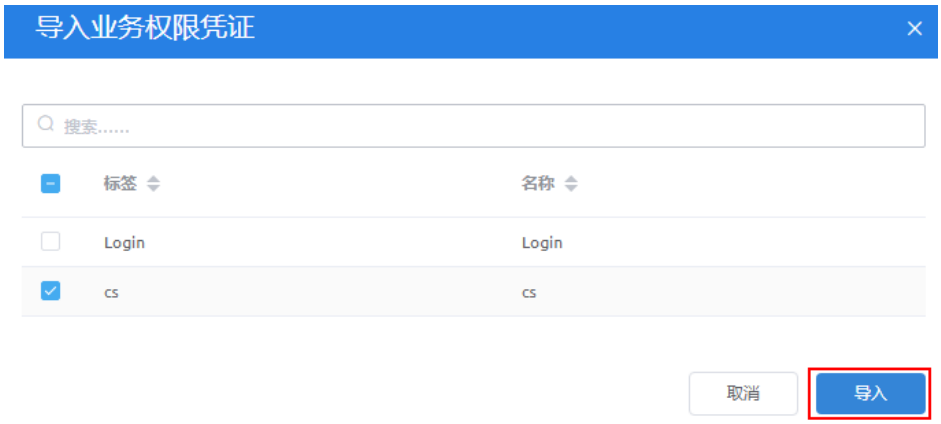
步骤2 在“A”应用开发页面的左下角，单击“配置”。

图 1-14 选择配置

步骤3 在右侧页面选择“业务权限凭证”，进入应用配置的业务权限凭证页面。

步骤4 单击“导入”，选择需要导入的业务权限凭证后，单击“导入”，完成业务权限凭证的导入。

图 1-15 导入业务权限凭证



说明

在“A”应用中选择导入业务权限凭证是因为在**步骤1**中已创建了需要的“cs”业务凭证。如果之前还未创建需要的业务凭证，则可以在本页面中单击“新建”，新建业务凭证，也可参考**步骤1**中操作创建业务凭证。

步骤5 在“A”应用开发页面的左下角，单击“服务”，进入自定义访问控制页面。

图 1-16 选择服务

步骤6 在自定义访问控制页面，单击需要配置业务凭证的接口名称，进入公共接口详情页面。

步骤7 单击业务权限凭证模块下的“编辑”，在“业务权限凭证列表”中选中需要添加的业务权限凭证，单击，将选择的业务权限凭证添加至右侧列表，单击“保存”，完成业务权限凭证的添加。

图 1-17 编辑业务权限凭证

步骤8 （可选）给权限绑定业务权限凭证。

须知

- 在以“A”应用为例的实例中权限绑定业务权限凭证的操作在**步骤2**中已执行过，此步骤可不执行。
- 若没有执行过权限绑定业务权限凭证的操作，此步骤必须执行。

1. 返回AstroZero开发环境首页，单击“管理”，进入AstroZero管理中心。
2. 在左侧导航栏中，选择“用户管理 > 权限配置”。
3. 在权限列表中，单击对应业务权限凭证的名称，进入该业务权限凭证的权限配置详情页面。
4. 单击“业务权限凭证”，单击  ，勾选“可接入”选项。

图 1-18 权限绑定业务权限凭证

步骤9 （可选）给用户或业务用户配置相应权限。

须知

- 在以“A”应用为例的实例中，配置“业务用户”权限的操作在**步骤3**中已执行过，此步骤可不执行。
- 若没有执行过配置“用户”或“业务用户”权限操作，此步骤必须执行。

1. 在左侧导航栏中，选择“用户管理 > 用户”或“用户管理 > 业务用户”。
2. 给用户或业务用户配置相应的权限，该用户或业务用户即可获取调用相应API接口的权限，否则将无法通过API接口的业务权限凭证校验。

----结束

1.3 如何实现业务用户登录

1.3.1 业务用户登录方式及机制

业务用户的登录方式

业务用户登录AstroZero有两种登录方式：后台登录和前台登录。

- 业务用户在后台登录时，是使用自定义的服务编排来调用“login”脚本，查询登录账号密码，判断当前登录的账号密码是否正确，来实现业务用户后台登录功能。
- 业务用户在前台登录时，需要先在线下开发一个登录组件，上传到高级页面，并在高级页面中配置组件桥接器中的数据。最后在页面中输入登录账号密码，通过调用“用户登录服务编排”，实现业务用户页面登录功能。

业务用户的登录机制

业务用户前台登录和后台登录，在登录过程中的服务逻辑实现过程如下：

1. 通过调用“账号密码校验”脚本，查询登录账号密码，判断当前登录的账号密码是否正确。
2. 如果判断账号密码错误，直接执行“账号密码错误”。账号密码正确，继续判断是否有验证码。
3. 如果判断当前登录没有验证码，则直接执行登录。当前有验证码，则继续判断验证码是否正确。
4. 如果判断验证码正确，则执行登录操作，验证码错误，则执行验证失败。

1.3.2 业务用户账号密码校验（登录）脚本

背景信息

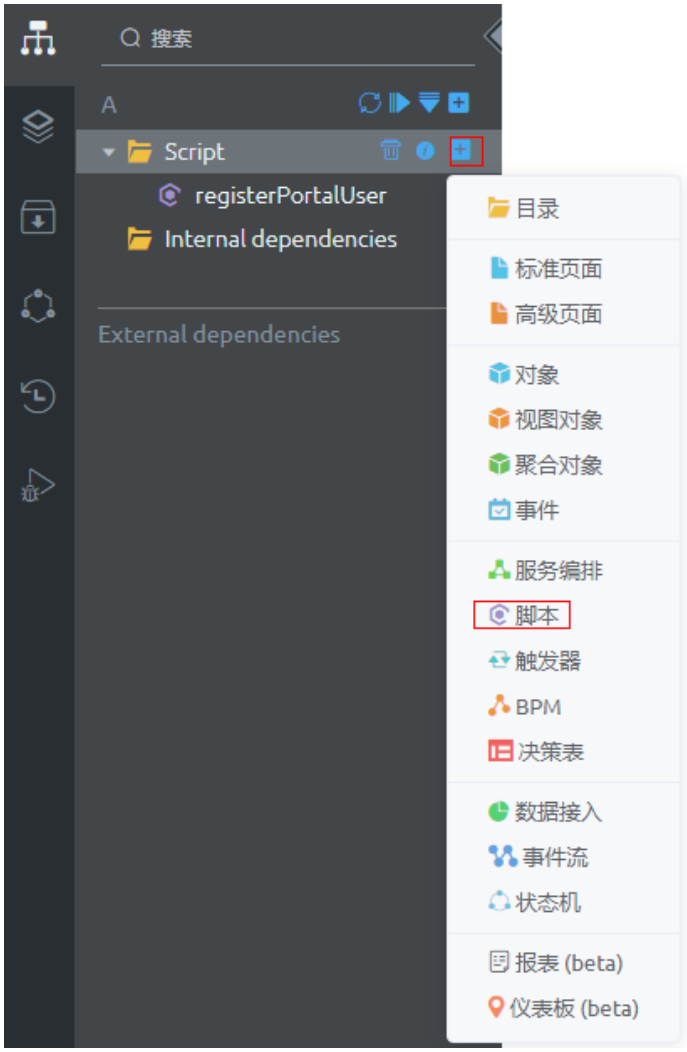
业务用户无论是使用“前台登录”还是“后台登录”，实现业务用户登录的基础都需要创建一个账号密码校验脚本。以“A”应用为例，介绍如何创建账号密码校验脚本。

操作步骤

步骤1 在“我的应用”中，单击“A”应用，进入应用开发页面。

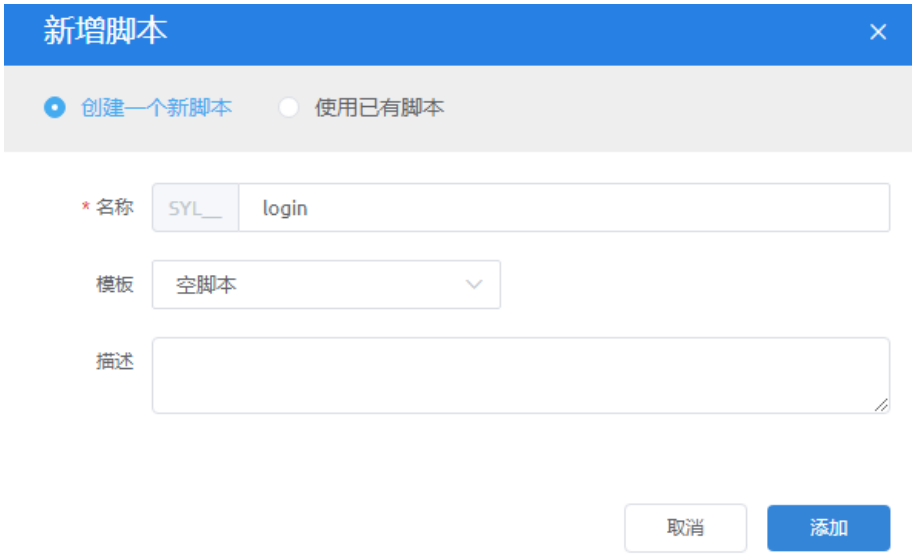
步骤2 将鼠标放在“Script”上，单击界面上出现的“+”，在弹出菜单中选择“脚本”。

图 1-19 脚本菜单



步骤3 选中“创建一个新脚本”，“名称”设置为“login”，单击“添加”。

图 1-20 新增脚本



步骤4 在代码编辑器中，插入如下脚本代码。

```
import * as buffer from"buffer";
import * as crypto from"crypto";
import * as db from"db";
//定义入参结构，账号的用户名、密码为必需字段。若根据业务需要，需其校验他字段（例如验证码），则根据账号密码字段的格式进行新增即可。
@action.object({type:"param"})
export class ActionInput{
    @action.param({type:'String', required:true, label:'string'})
    username:string;    //用户名
    @action.param({type:'String', required:true, label:'string'})
    password:string;    //密码
    @action.param({type:'String', required:true, label:'string'})
    captcha:string;    //验证码，本脚本只是为了校验账号密码，因此用不到验证码，验证码也不是必需字段。在实现业务用户后台登录的Flow中，Flow调用此脚本，需要判断验证码，所以在此脚本中添加了验证码字段。
}
//定义出参结构，结构中的字段可以根据业务需要按下方样例结构进行添加或减少。
@action.object({type:"param"})
export class ActionOutput{
    @action.param({type:'String'})
    msg:string;//登录信息
    @action.param({type:'String'})
    username:string;//用户名
    @action.param({type:'String'})
    userId:string;//用户ID
    @action.param({type:'String'})
    captcha:string;//验证码
}
//使用数据对象PortalUser
@useObject(['PortalUser'])
@action.object({type:"method"})
export class Login{
//定义接口类，接口的入参为ActionInput，出参为ActionOutput
    @action.method({ input:'ActionInput', output:'ActionOutput'})
    public login(input:ActionInput):ActionOutput{
        let out =new ActionOutput();
        //新建出参ActionOutput类型的实例，作为返回值
        let error =new Error();
        //新建错误类型的实例，用于在发生错误时保存错误信息
        try{
            out.captcha = input.captcha;
            let s = db.object('PortalUser');
            let condition ={
            "conjunction":"AND",
```

```
        "conditions":[{
            "field":"usrName",
            "operator":"eq",
            "value": input.username
            }]
        };
        let user = s.queryByCondition(condition);
        if(user && user.length ==1){
            if(validate(user[0].passwordSalt, user[0].userPassword, input.password)){
                out.msg ="登录成功！";
                out.username = user[0].usrName;
                out.userId = user[0].id;
            }else{
                out.msg ="账号或者密码错误！";
            }
        }else{
            out.msg ="账号或者密码错误！";
        }
    }catch(error){
        console.error(error.name, error.message);
        out.msg = error.message;
    }
    return out;
  }
}
function _salt(password:string, saltBuf: buffer.Buffer, encoding: buffer.Encoding=
buffer.Encoding.Base64):string{
  const passwordBuf = buffer.from(password)
  const crypt = crypto.pbkdf2(passwordBuf, saltBuf,1000,32, crypto.Hashs.SHA1)
  return crypt.toString(encoding)
}
function validate(salt:string, userSaltedPassword:string, password:string, encoding: buffer.Encoding=
buffer.Encoding.Base64):boolean{
  const saltBuf = buffer.from(salt, encoding);
  const saltedPassword = _salt(password, saltBuf, encoding);
  return saltedPassword === userSaltedPassword
}
```

步骤5 单击编辑器上方的，保存脚本。

步骤6 测试脚本能否正常执行。

1. 单击编辑器上方的，执行脚本。
2. 在界面底部的输入参数中，输入如下测试数据，单击。
   其中，“test_cs”、“*{XXXXXXXXX}*”为注册的业务用户账号和密码，“captcha”验证码非必填项为空。

   说明

   在业务配置中心创建的业务用户和使用脚本创建的业务用户不能通用，故此处设置的业务用户（如test_cs）不能为在业务配置中心创建的业务用户。业务配置中心创建的业务用户，只能在默认登录页中使用。

   ```
   {
   "username":"test_cs",
   "password":"{XXXXXXXXX}",
   "captcha":""
   }
   ```

   执行成功后，在“输出参数”中可参看结果。

图 1-21 返回结果

步骤7 测试成功后，单击编辑器上方的，启用该脚本。

----结束

1.3.3 如何实现业务用户后台登录

背景信息

业务用户在后台登录时，是使用自定义的服务编排，来调用“账号密码校验”脚本，查询登录账号密码，判断当前登录的账号密码是否正确来实现“业务用户”后台登录功能的。“业务用户登录”服务编排开发的大致过程为：先拖拽1个脚本图元，3个决策图元以及3个赋值图元，再分别配置各个图元属性，然后配置各个图元之间连线类型，最后保存启用。

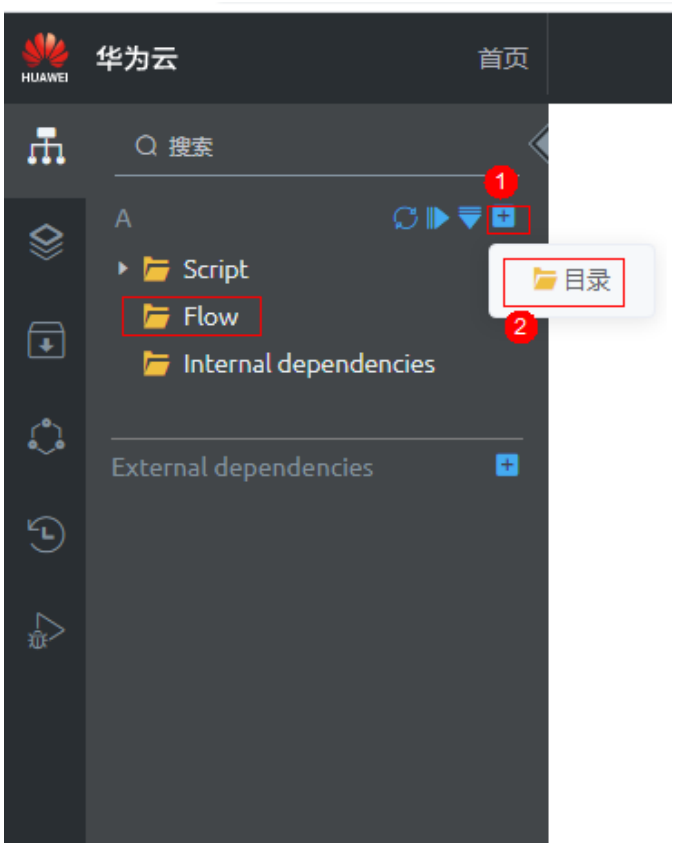
下面以“A”应用为例，向您介绍如何在后台实现业务用户登录。

操作步骤

步骤1 在“我的应用”中，单击“A”应用，进入应用开发页面。

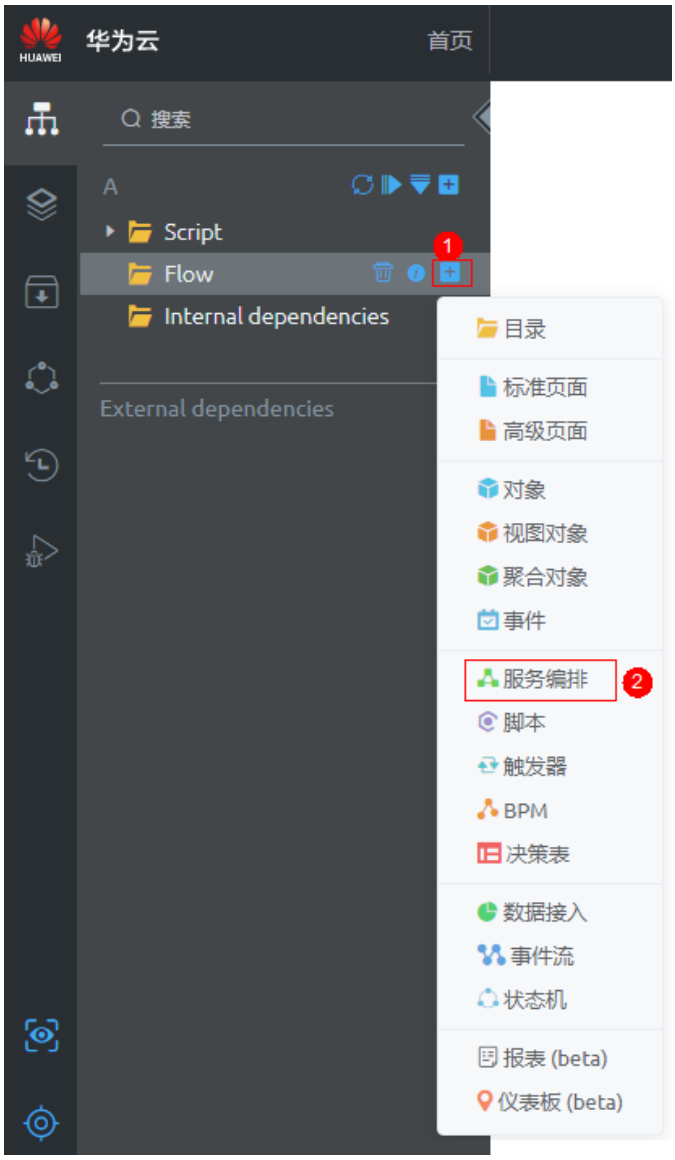
步骤2 在开发页面中，单击“目录”，创建一个Flow目录。

图 1-22 创建目录



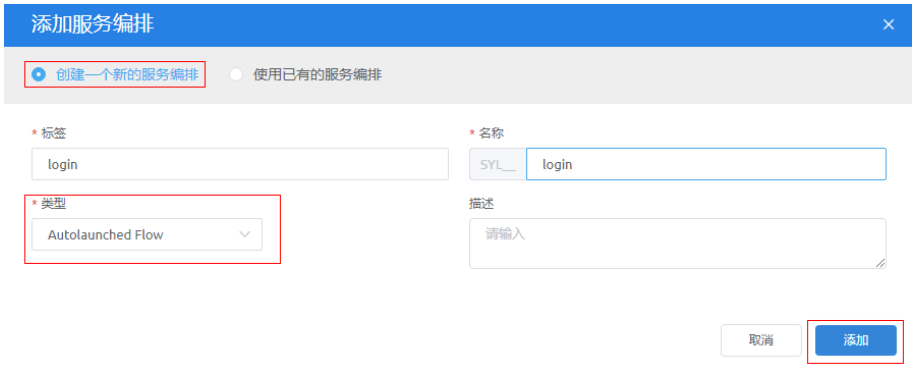
步骤3 将鼠标放在“Flow”上，单击界面上出现的“+”，在弹出菜单中选择“服务编排”。

图 1-23 选择服务编排



步骤4 选中“创建一个新的服务编排”，“标签”和“名称”设置为“login”，类型设置为“Autolaunched Flow”，单击“添加”。

图 1-24 创建服务编排



步骤5 定义服务编排用到的变量。

1. 单击，展开全局上下文，再单击“变量”后的，设置参数名称为“username”。

图 1-25 新增变量

2. 重复上一步，定义表1-1中其他变量。

表 1-1 服务编排变量说明

名称（变量名称，唯一标识）	数据类型
username	文本
password	文本
captcha	文本
msg	文本
userId	文本
loginMsg	文本

3. 单击“公式”后的，在左侧公式弹窗中，设置“名称”为“portalUserLogin”，“表达式”为“PORTALUSERLOGIN({!username})”，单击“保存”。

图 1-26 添加公式变量“portalUserLogin”

4. 参考上一步，创建表1-2中公式变量“verifyCode”。

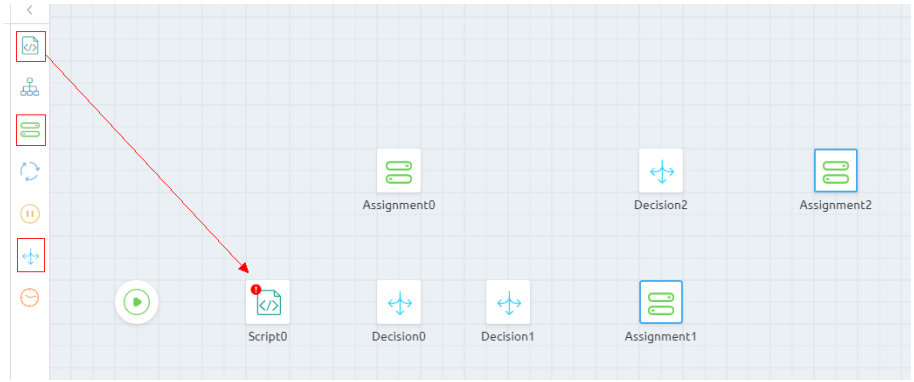
表 1-2 公式变量说明

名称	表达式
portalUserLogin	PORTALUSERLOGIN({!username})
verifyCode	VERIFYCODEWITHTYPE({!captcha},"login")

步骤6 拖拽图元到服务编排画布，并配置图元的基本属性。

1. 从图元区分别拖拽脚本（1个）、决策（3个）、赋值（3个）图元到画布中，图元排列如下图所示。

图 1-27 图元排列



2. 选中“Script0”图元，在右侧基本信息中，设置“标签”为“查询用户”。
3. 参考上一步，设置其他图元的“标签”属性，具体值如下表所示。

表 1-3 设置其他图元标签属性

名称（变量唯一标识，不需要修改）	标签
Decision0	判断账号密码
Decision1	判断是否包含验证码
Decision2	校验验证码
Assignment0	账号密码错误
Assignment1	执行登录
Assignment2	验证失败

图 1-28 修改后图元

步骤7 配置“查询用户”脚本图元。

1. 单击  ，指定图元对应的脚本名称（SYL__login），并配置脚本的输入输出参数。

图 1-29 指定脚本

2. 单击“全局上下文”，显示变量列表，从“变量”中，拖拽“username”、“password”和“captcha”到“输入参数”下对应的“源”输入框中，在“输出参数”下，单击4次“新增行”，依次添加下拉选项中的输出参数字段，并从“变量”中拖拽相应的字段到“目标”输入框下，字段与变量对应关系如下图所示。

   说明

   脚本图元中，输入参数、输出个数和指定脚本中需要的输入参数字段数是一致的。若自定义脚本的输入参数有额外字段，额外的字段也需要同样操作。

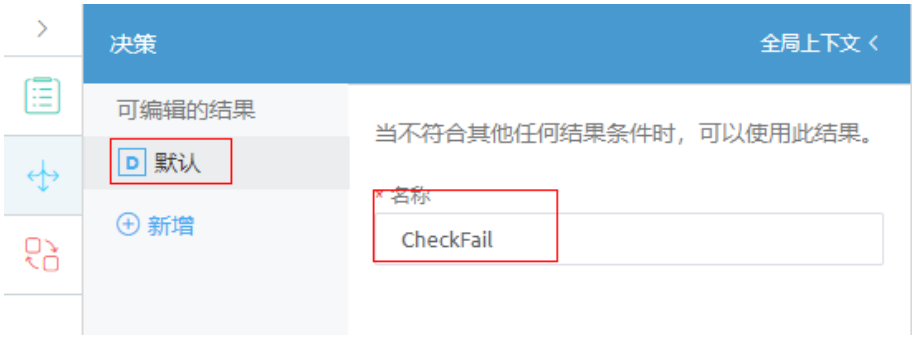
   请直接从全局上下文拖拽“变量”到对应的输入输出参数下，如果手动输入请确认输入参数与全局上下文中变量的值一致。

图 1-30 拖拽脚本的输入输出参数

步骤8 配置“判断账号密码”决策图元。

1. 选择“判断账号密码”图元，在右侧单击图标，修改“默认”的“名称”为“CheckFail”。

图 1-31 修改“默认”结果名称



2. 单击“新增”，增加一个可编辑的结果，修改结果为“CheckSuccess”，在“可视”下单击“新增行”，并拖拽变量中的“msg”到“资源”下，设置“比较符”为“==”，“值”为“"登录成功！"”。

须知

- 请直接从全局上下文拖拽变量“msg”到“资源”下，如果手动输入请确认输入参数与全局上下文中变量的值一致。
- “登录成功！”需要与“login”登录脚本中的输出参数一致。

图 1-32 修改可编辑的结果

步骤9 配置“判断是否包含验证码”决策图元。

1. 选择“判断是否包含验证码”图元，在右侧单击图标，修改“默认”的“名称”为“hasVerifyCode”。

图 1-33 修改默认结果名称

2. 单击“新增”，增加一个可编辑的结果，修改结果为“noVerifyCode”，在“可视”下单击“新增行”，并拖拽变量中的“captcha”到“资源”下，设置“比较符”为“==”，“值”为“""”。

图 1-34 修改可编辑的结果

步骤10 配置“校验验证码”决策图元。

1. 选择“校验验证码”图元，在右侧单击图标，修改“默认”的“名称”为“verifyCodeFail”。

图 1-35 修改“默认”名称

2. 单击“新增”，增加一个可编辑的结果，修改结果为“verifyCodeSuccess”，在右侧选择“公式”，并从全局上下文中，拖拽“verifyCode”到“公式”下。

图 1-36 修改可编辑的结果

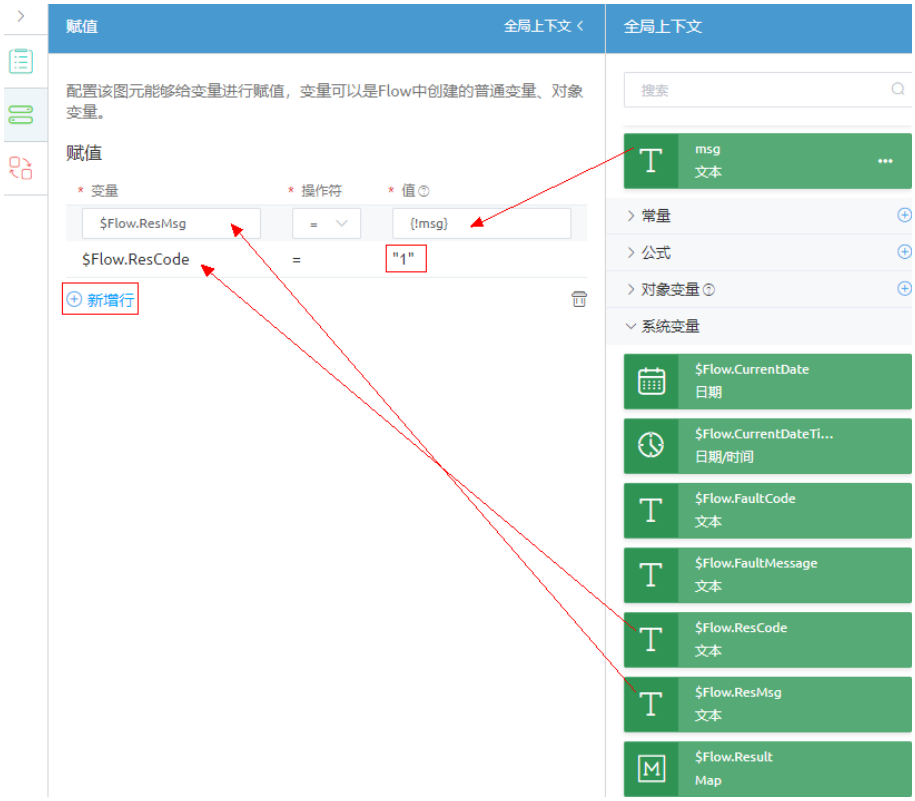
步骤11 配置“账号密码错误”赋值图元。

1. 选择“账号密码错误”图元，在右侧单击图标，单击“新增行”。
2. 从全局上下文的“系统变量”中，拖拽“$Flow.ResMsg”到“赋值”下，并设置“操作符”为“=”，拖拽“msg”到“值”。
3. 单击“新增行”，拖拽“系统变量”下的“$Flow.ResCode”到“赋值”的“变量”下，设置“操作符”为“=”，设置“值”为“"1"”。

说明

请直接从全局上下文拖拽变量到“值”下的对应位置，如果手动输入请确认输入参数与全局上下文中变量的值一致。

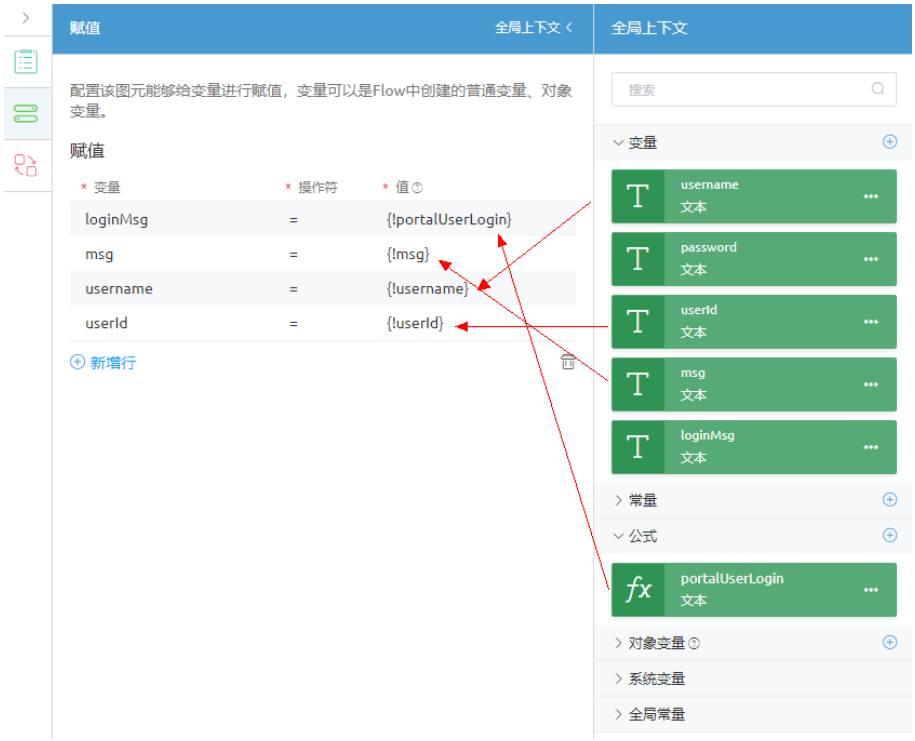
图 1-37 配置“账号密码错误”图元



步骤12 配置“执行登录”赋值图元。

1. 选择“执行登录”图元，在右侧单击图标，单击4次“新增行”。
2. 从全局上下文，拖拽“msg”等字段到“赋值”的“变量”下，并设置“操作符”为“=”，然后再拖拽“值”下的各个值，具体字段对应关系，如下图所示。

图 1-38 拖拽“执行登录”赋值的变量及值



注意

请直接从全局上下文拖拽变量到“值”下的对应位置，如果手动输入请确认输入参数与全局上下文中变量的值一致。

表 1-4 变量与值对应关系

变量	操作符	值
loginMsg	=	portalUserLogin
msg	=	msg
username	=	username
userId	=	userId

步骤13 配置“验证失败”赋值图元。

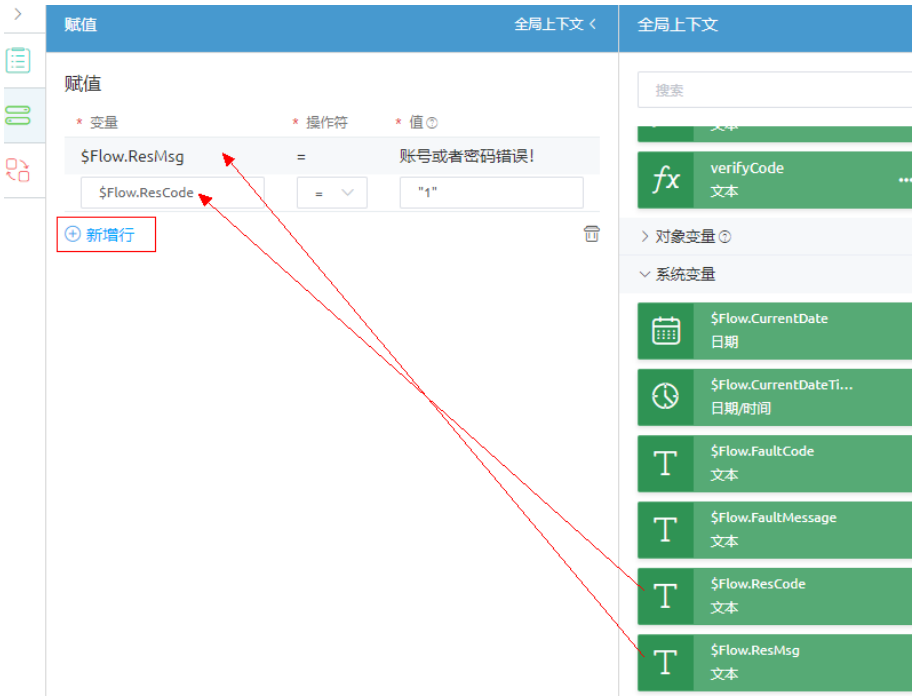
1. 选择“验证失败”图元，在右侧单击图标，单击“新增行”。

2. 从全局上下文“系统变量”，拖拽“$Flow.ResMsg”、“$Flow.ResCode”到“赋值”下，并设置操作符为“=”，分别设置“值”为“"账号或者密码错误！"”、“"1"”。

表 1-5 赋值

变量	操作符	值
$Flow.ResMsg	=	"账号或者密码错误！"
$Flow.ResCode	=	"1"

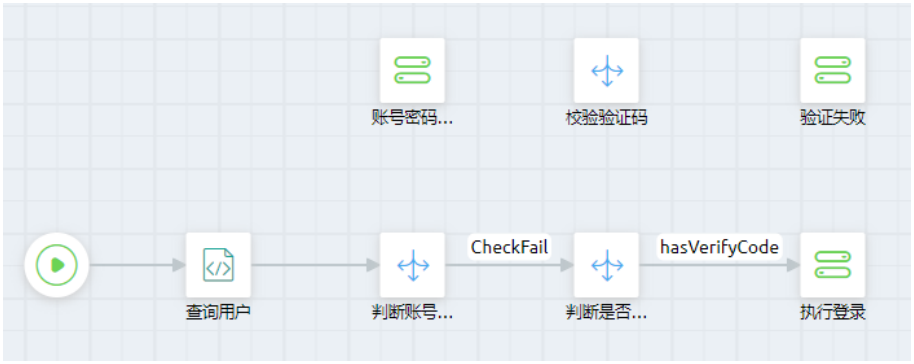
图 1-39 配置“验证失败”赋值图元



步骤14 拖拽图元连线，并配置连线属性。

1. 在画布上，把鼠标放在起点图元  图元上，从“+”拖动鼠标，在起点图元和“查询用户”图元间增加连线；即将当前脚本设置为服务编排的起始节点。
2. 依次在“查询用户”、“判断账号密码”、“判断是否包含验证码”、“执行登录”图元直接拖拽连线。

图 1-40 拖拽连线



3. 单击“判断账号密码”与“判断是否包含验证码”图元之间的连线，再右侧属性单击，在“连线”中修改“连线类型”为“CheckSuccess”。

图 1-41 选中连线

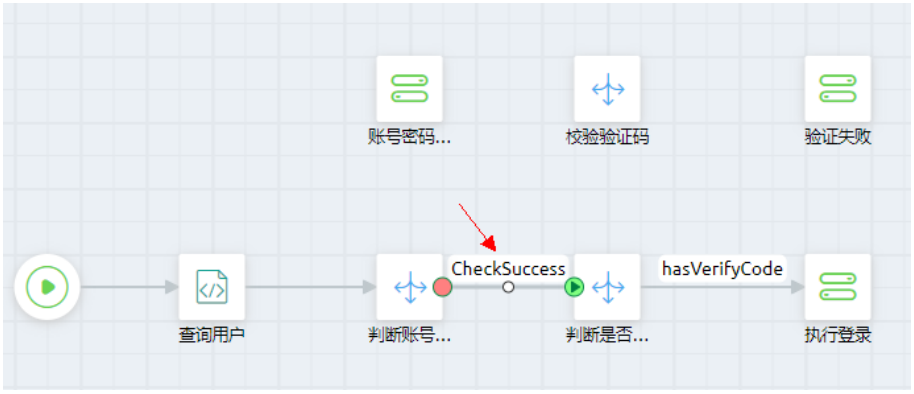
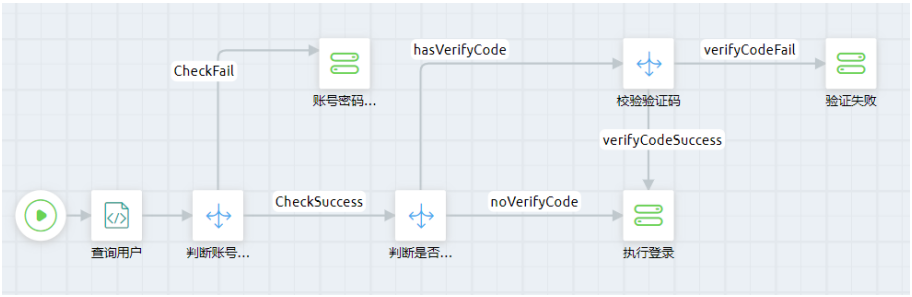


图 1-42 修改连线类型

4. 单击“判断是否包含验证码”与“执行登录”图元之间的连线，再右侧属性单击，在“连线”中修改“连线类型”为“noVerifyCode”。
5. 从“判断账号密码”图元上，拖拽一条连线到“账号密码错误”图元。
6. 从“判断是否包含验证码”图元上，拖拽一条连线到“校验验证码”图元。
7. 从“校验验证码”图元上，拖拽一条连线到“验证失败”图元。
8. 从“校验验证码”图元上，拖拽一条连线到“执行登录”图元，并设置该连线的“连线类型”为“verifyCodeSuccess”。

连线拖拽完成，如下图所示。

图 1-43 拖拽图元连线



步骤15 定义服务编排的输入、输出参数，并保存服务编排。

1. 鼠标在画布空白处点一下，单击右侧，设置服务编排的输入输出参数，如下图所示。

图 1-44 拖拽服务编排的输入输出参数

说明

- 服务编排的输入参数是用来执行服务编排时输入的参数，同时也是执行账号密码校验脚本时的输入参数。所以当账号密码校验脚本的有额外的输入参数字段，服务编排的输入参数也需要同步增加。
- 服务编排的输出参数是执行账号密码校验脚本时返回的参数，所以当账号密码校验脚本的有额外的输出参数字段，服务编排的输出参数也需要同步增加。

2. 单击服务编排页面上方的，保存服务编排。

步骤16 测试服务编排能否正常执行。

1. 单击服务编排页面上方的，进入服务编排测试页面。

图 1-45 服务编排测试页面

2. 在“Flow Run”界面中，输入测试数据，单击“运行”。
   其中，“test_cs”、“*{XXXXXXXX}*”为业务用户的账号和密码。

```
{
 "username": "test_cs",
 "password": "{XXXXXXXX}",
 "captcha": ""
 }
```

   执行成功后，界面上会返回设备对象中的全部信息，示例如下：

图 1-46 返回值示例

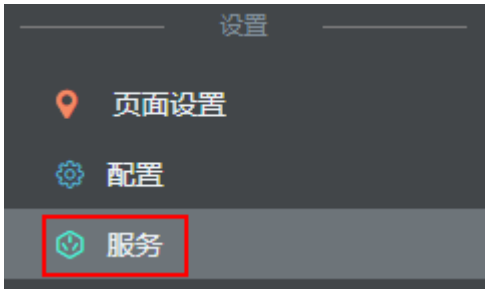
说明

返回值提示登录成功，完成业务用户的登录。业务用户登录成功后，返回AstroZero，刷新页面后在页面右上角可以看到当前登录的用户已变成在服务编排中输入的业务用户。

步骤17 登录测试成功后，再次使用租户账号登录AstroZero，进入“A”应用，选择刚刚测试成功的“login”服务编排进入编辑器，单击编辑器上方的  ，启用并发布服务编排。

步骤18 服务编排发布后，单击“A”应用开发页面左下角的“服务”，进入自定义访问控制（前后置处理）页面。

图 1-47 服务



步骤19 单击“新建”，新建公共接口。

图 1-48 新建公共接口

说明

在新建公共接口时，在资源中选择已发布的服务编排。URL填写的内容是定义新建的公共接口提供外部访问的URL。

步骤20 公共接口创建完成后，参考**如何给接口添加业务权限凭证**中操作，给测试成功的服务编排添加业务权限凭证。

图 1-49 给服务编排添加业务权限

----结束

1.3.4 如何实现业务用户页面登录

应用登录页面开发流程

业务用户在前台登录时，需要先在线下开发一个登录组件，上传到高级页面，并在高级页面中配置组件桥接器中的数据。最后在页面中，输入登录账号密码，通过来调用“用户登录服务编排”，实现“业务用户”页面登录功能。

以“A”应用为例，介绍如何开发登录页面，具体流程如图1-50所示。

图 1-50 应用登录页面开发流程

步骤 1：自定义开发组件

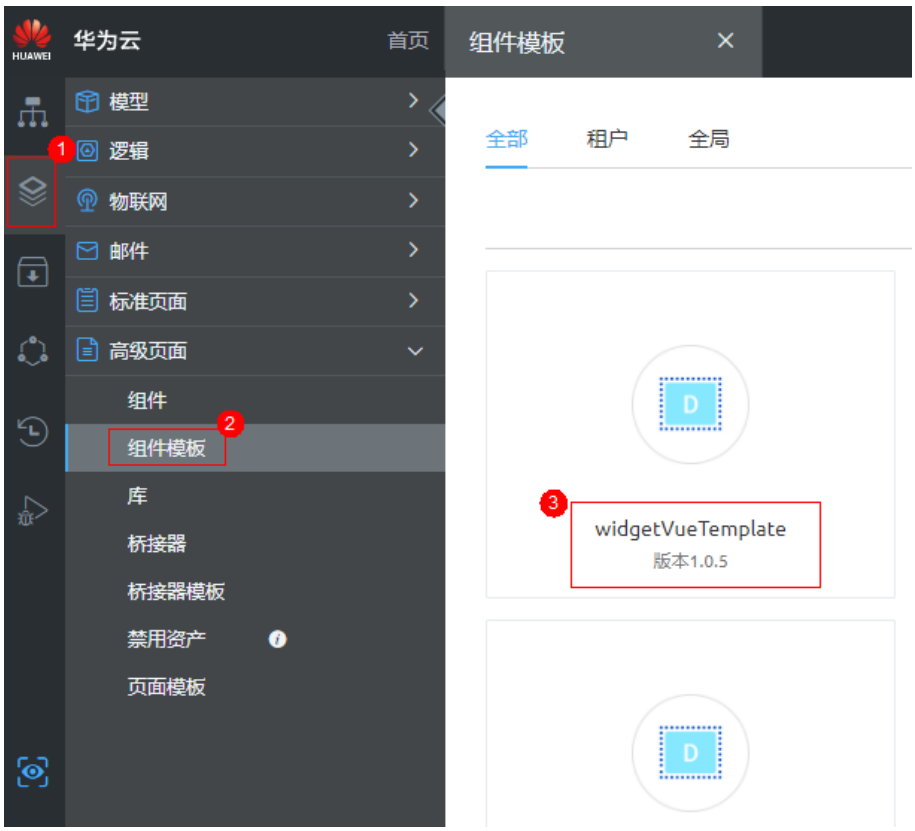
步骤1 开发自定义组件。

本示例中，提供了一个已开发好的自定义登录组件userLogin，单击**userLogin.zip**可直接下载使用。若想要了解自定义登录组件的开发方法，请参考**步骤2**。

步骤2 （可选）线下开发自定义登录组件。

1. 在“资产 > 高级页面 > 组件模板”中，单击“widgetVueTemplate”，再单击“下载”。

图 1-51 下载组件模板

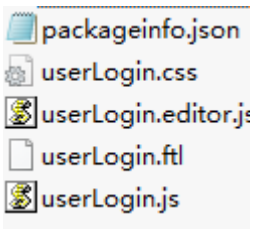


2. 输入组件名称，单击“保存”，将组件模板保存到本地，并解压。

图 1-52 输入组件名称

3. 查看解压后的组件目录。

图 1-53 解压后目录



- userLogin.js：存放vue业务逻辑的代码，请根据业务需求自行开发。
- userLogin.ftl：存放html代码，请根据业务需求自行开发。
- userLogin.css：存放样式代码，请根据业务需求自行开发。
- userLogin.editor.js、packageinfo.json：配置文件，请参考**步骤2.4**和**步骤2.5**修改。

4. 修改userLogin.editor.js文件中的config代码，用于配置桥接器。

代码示例如下：

```
config: [
        {
            type: 'connectorV2',
            name: 'FlowConnector',
            label: 'Flow Connector',
            model: 'ViewModel',
        },
        {
            type: 'connectorV2',
            name: 'common.GetConnector',
            label: 'View API Get Connector',
            model: 'ViewModel',
        },
        {
            type: 'connectorV2',
            name: 'common.PostConnector',
            label: 'View API Post Connector',
            model: 'ViewModel',
        },
        {
            type: 'connectorV2',
            name: 'common.PutConnector',
            label: 'View API Put Connector',
```



```
            model: 'ViewModel',
        },
        {
            type: 'connectorV2',
            name: 'common.DeleteConnector',
            label: 'View API Delete Connector',
            model: 'ViewModel',
        },
    ]
```

5. 修改packageinfo.json文件中的“requires” 。

   代码示例如下加粗内容：

```
{
  "widgetApi": [
    {
      "name": "userLogin"
    }
  ],
  "widgetDescription": "",
  "authorName": "",
  "localFileBasePath": "",
  "requires": [
    {
      "name": "global_Vue",
      "version": "100.8.3"
    },
    {
      "name": "global_Element",
      "version": "101.0.3"
    },
    {
      "name": "global_VueI18n",
      "version": "100.7.3"
    },
    {
      "name": "global_VueRouter",
      "version": "1.0.1"
    }
  ]
}
```

6. 将修改后的配置文件和自定义开发的文件，压缩成一个zip包。

   说明

   用户开发登录组件代码，可参考已开发好的**userLogin.zip**中代码进行开发。

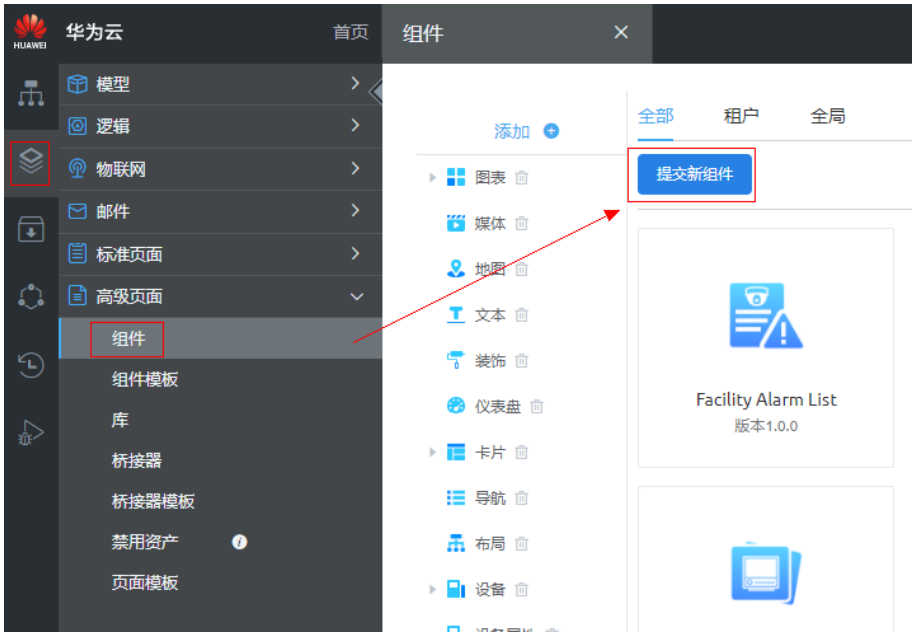
----结束

步骤 2：上传自定义登录组件

步骤1 在AstroZero开发环境首页“项目 > 我的应用”中，单击“A”应用。

步骤2 在左侧列表中，单击  ，选择“高级页面 > 组件”，进入组件页面。

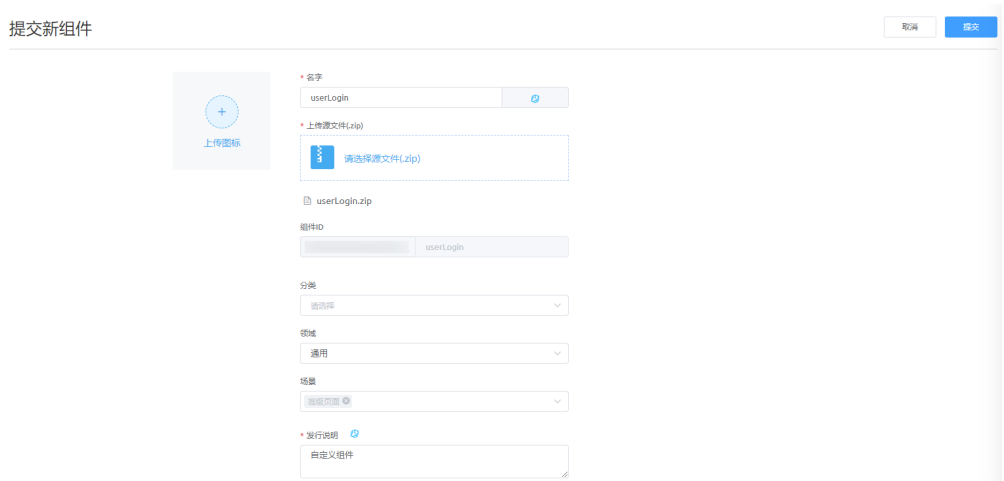
图 1-54 组件列表页面



步骤3 单击“提交新组件”，进入提交新组件页面。

步骤4 单击“上传”，选择自定义组件包“userLogin.zip”，设置“发行说明”后，单击“提交”，上传组件。

图 1-55 上传组件



上传成功后，在后续创建的高级页面中，即可使用该自定义组件。

----结束

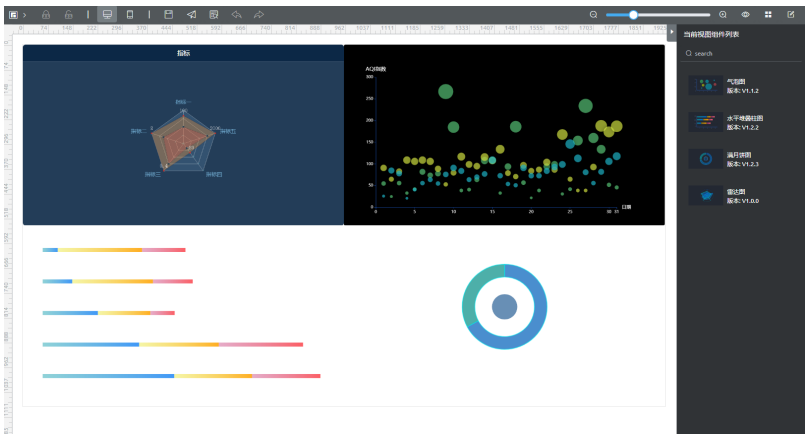
步骤 3：创建高级页面

“业务用户登录”页面是一个高级页面，主要是通过引用上传的自定义登录组件，再配置相关参数，来实现登录功能。

在组装业务用户登录页面时，需要配置从登录页面登录成功后跳转的页面，所以需要提前创建一个高级页面。在本示例中，假设已创建了一个Home页面。Home页面中没

有实际数据，在Home页面中拖拽了几个图表组件，然后保存发布成样例页面。在实际开发过程中，可以根据业务需要，选择跳转到需要的页面，或者执行自定义的业务逻辑。

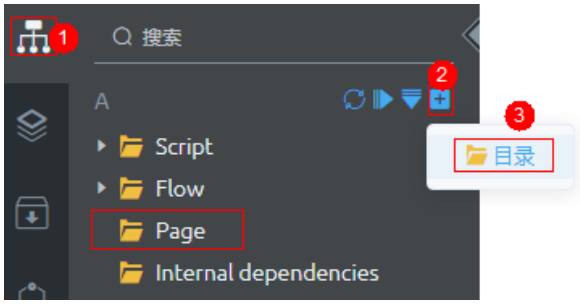
图 1-56 Home 页面



步骤1 进入AstroZero开发环境首页，在“项目 > 我的应用”中，单击“A”应用。

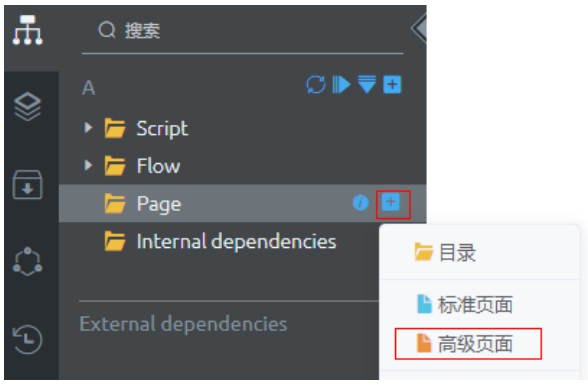
步骤2 单击  ，进入工作目录，并创建一个Page目录。

图 1-57 创建 Page 目录



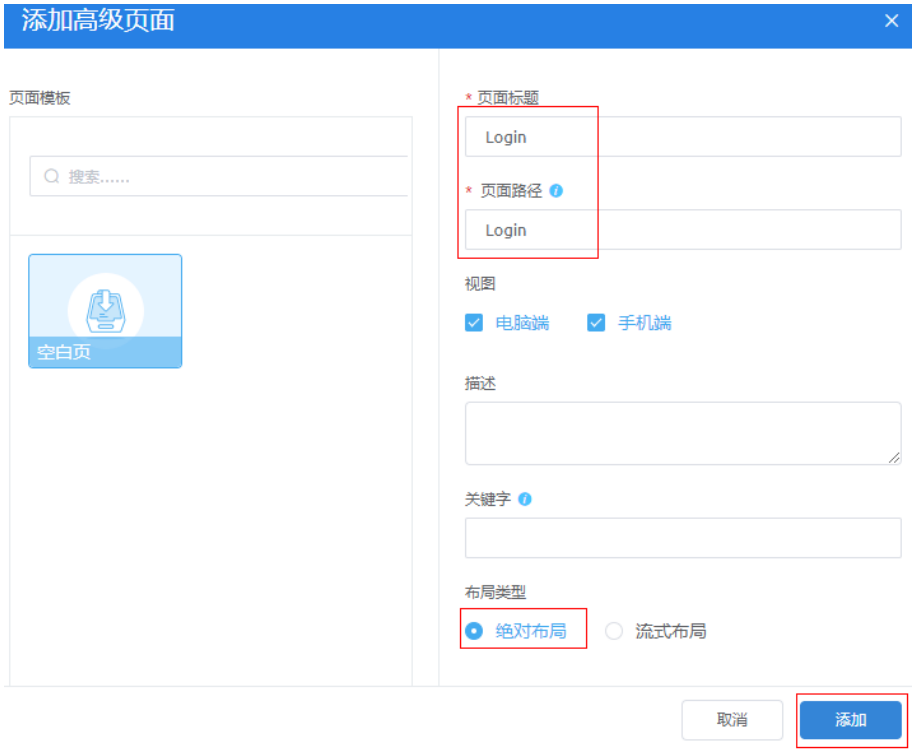
步骤3 将鼠标放在“Page”上，单击界面上出现的“+”，在弹出菜单中选择“高级页面”。

图 1-58 创建高级页面



步骤4 设置“页面标题”和“页面路径”为“Login”，并选择“绝对布局”，单击“添加”。

图 1-59 添加高级页面



----结束

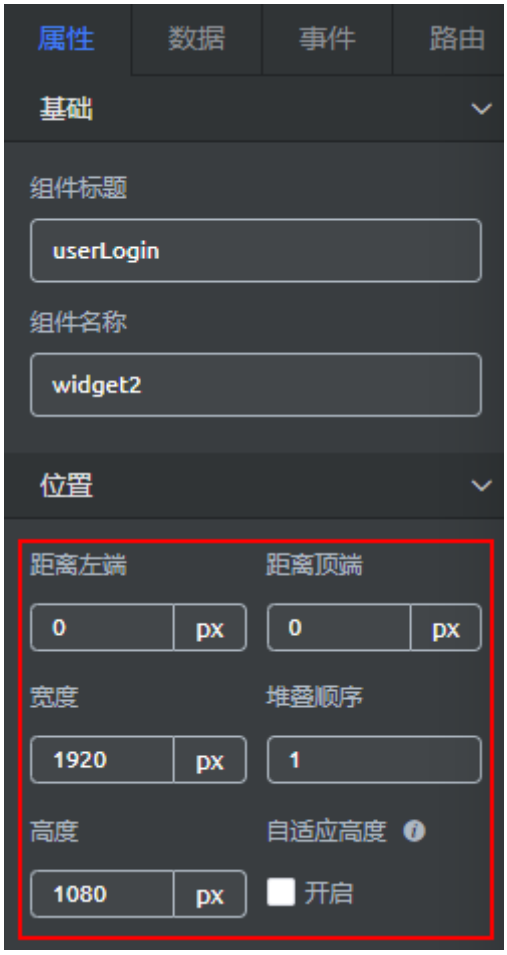
步骤 4：拖拽组件开发登录页面

步骤1 在高级页面开发界面，单击左上角的，选择“全部”，在“自定义”页签，拖拽“userLogin”组件到页面编辑区。

步骤2 设置自定义组件“userLogin”的位置属性。

1. 将“userLogin”组件拖拽到页面编辑区后，会在右侧显示该组件的属性配置面板。
2. 在“位置”中，设置“距离左端”、“距离顶端”为“0”，“宽度”为“1920”，“高度”为“1080”。

图 1-60 设置组件位置



3. 组件位置、样式等属性修改完成后，单击页面上方的，保存页面修改。
4. 在目录中，将鼠标放在“Login”上，单击“...”，然后选择“设置”，在弹窗中设置页面的拉伸属性，再单击“保存”。

图 1-61 页面设置

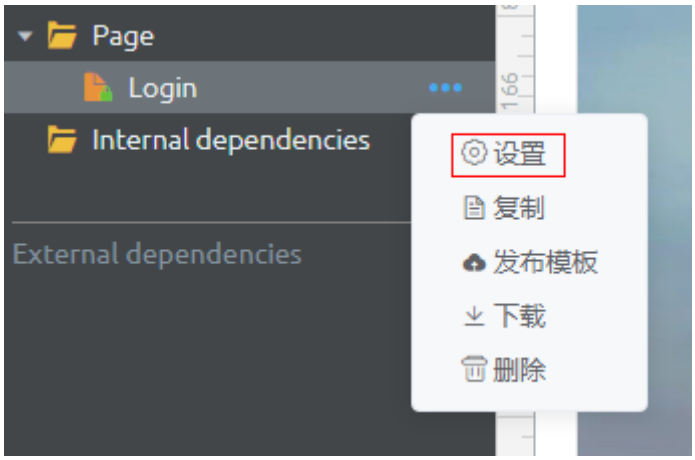
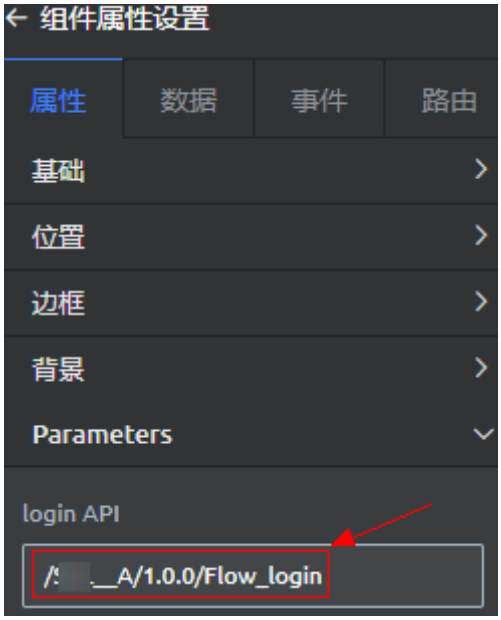


图 1-62 设置页面拉伸

步骤3 设置自定义组件的桥接器。

1. 选中“userLogin”组件，在“属性 > Parameters”中，设置“login API”为“/*命名空间*__A/1.0.0/Flow_login”。

图 1-63 添加登录服务 URL

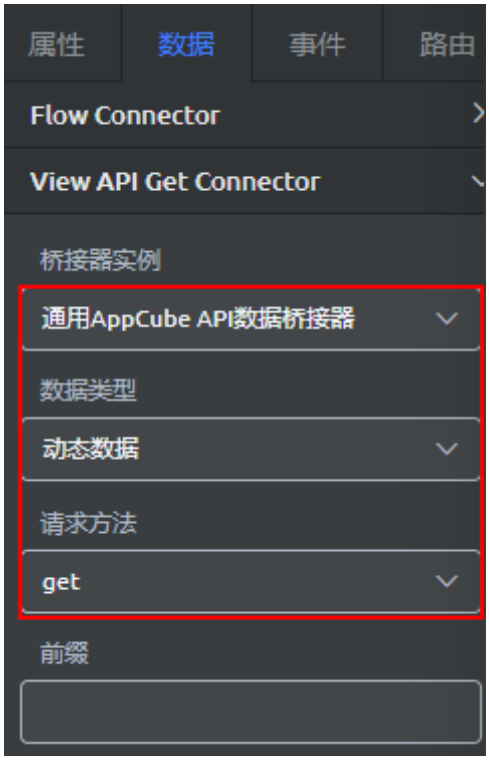


“login API”为登录接口的URL后半段，“命名空间”请根据实际情况配置，“A”为应用名，登录接口是在**如何实现业务用户后台登录**中创建的。

图 1-64 loginAPI 地址

2. 在“数据”页签，单击“View API Get Connector”，设置“桥接器实例”为“通用AppCube API数据桥接器”，“数据类型”为“动态数据”，“请求方法”为“get”，如下图所示。

图 1-65 设置桥接器



- 桥接器实例：调用的桥接器名称。
- 请求方法：调用的方法名，如get（查询）、put（修改）、post（增加）和delete（删除）。
- 调用周期：此处不用配置。调用周期是每隔多少秒调用一次后台接口或者获取静态数据，默认配置为“0”，表示只调用一次或者只获取一次静态数据。

3. 参考上一步，分别设置下图中框选的桥接器实例。

图 1-66 设置其他桥接器

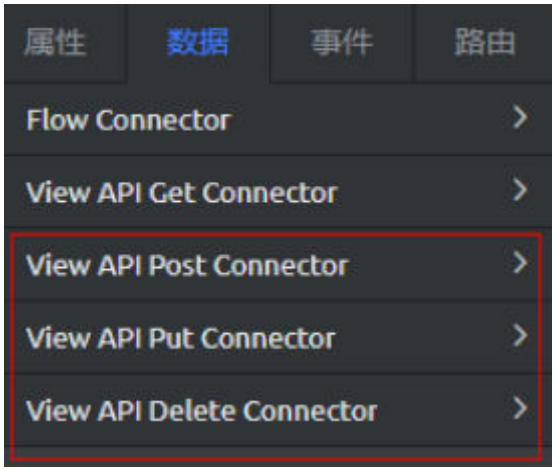


表 1-6 桥接器实例配置

数据名	桥接器实例	数据类型	Request Method
View API Post Connector	通用AppCube API数据桥接器	动态数据	Post
View API Put Connector	通用AppCube API数据桥接器	动态数据	Put
View API Delete Connector	通用AppCube API数据桥接器	动态数据	Delete

步骤4 设置自定义组件“userLogin”的“事件”，使自定义组件与其他页面关联。

图 1-67 需要配置的事件

1. 单击“go Homepage”后的齿轮图标，选择新建动作的类型。

图 1-68 新建动作

2. 选择“新建动作 > 默认 > 页面跳转”，进入跳转编辑页面。

图 1-69 编辑页面跳转

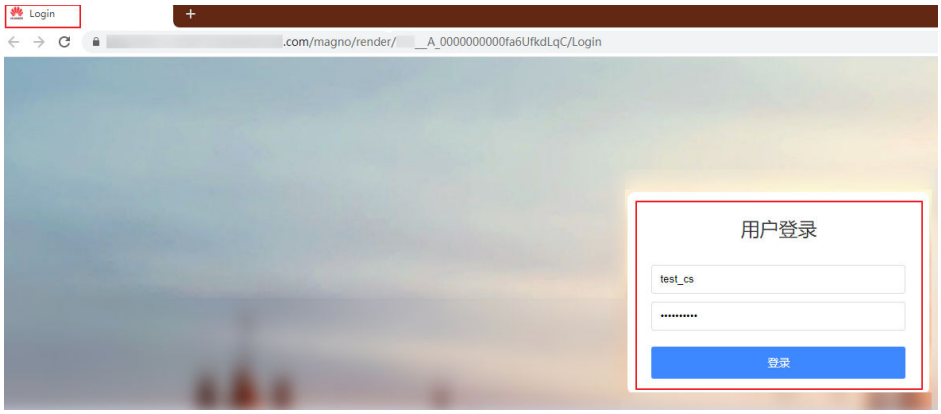
说明

这里选择的高级页面是之前创建并发布的“Home”，在实际开发过程中，可以根据业务需要，选择跳转到需要的页面，或者执行自定义的业务逻辑。

步骤5 在上述步骤完成后，单击Login页面上方的，保存页面，再单击，发布页面。

步骤6 页面发布成功后，单击，即可预览发布的登录页面。

图 1-70 预览的登录页面

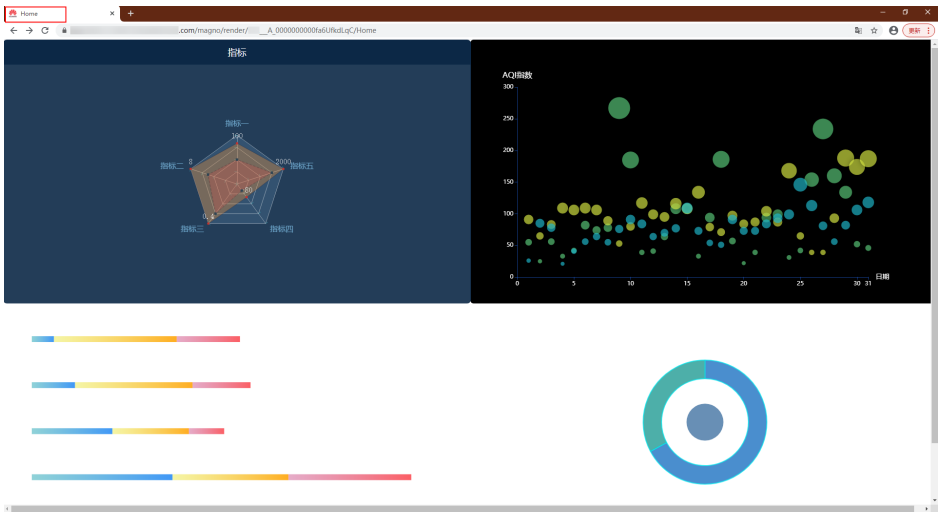


说明

当前登录页中，输入业务用户账号及密码，单击“登录”的登录逻辑是通过“自定义登录”组件，调用用户登录服务编排完成的。

步骤7 在预览的登录页面中，输入配置了权限的业务用户的账号密码，单击登录后，若页面跳转到“Home”页面，则业务用户登录成功。

图 1-71 登录成功后的 Home 页面



----结束

1.4 平台权限认证机制

权限认证机制介绍

权限认证是指当用户登录AstroZero时，对用户拥有的权限进行认证的机制。AstroZero提供的所有接口都需要通过认证后，才可以访问。而AstroZero中存在如下几类用户，不同类别的用户访问接口的权限会有所不同：

- 用户 (User)：也可以称为平台用户，用于管理开发应用。

  使用认证机制为：“账号密码登录认证”、“单点登录认证”和“OAuth 2.0鉴权登录认证”。
- 业务用户（Portal User）：是访问AstroZero提供的业务服务的用户。

  使用的认证机制为：“账号密码登录认证”、“单点登录认证”和“OAuth 2.0鉴权登录认证”。
- 匿名用户（Guest）：对于该类用户，平台接口都需要进行校验，未登录前使用Guest用户身份来访问平台接口，一般会赋予最小的权限。

  使用的认证机制为：“OAuth 2.0鉴权登录认证”。

权限认证方式

AstroZero中使用的认证方式是：在请求消息头上，设置“access-token”进行认证。

在请求消息头上，设置“access-token”是通过不同的认证接口，来获取到Token的。Token在计算机系统中代表令牌（临时）的意思，拥有Token代表拥有某种权限。Token认证是在调用API时，将Token加到请求消息头，从而通过身份认证，获得操作API的权限。

如何通过 Token 进行认证

通过Token进行接口认证的方法说明，如表1-7所示。

表 1-7 接口认证说明

认证类别	认证方式	说明	使用的用户	是否可以直接访问接口
账号密码认证类别	账号密码登录	通过账号密码直接访问登录接口，成功校验则返回“access-token”，写到Cookies上。	用户	是
单点登录认证类别	单点登录（“CAS”协议）	支持CAS单点登录方式，通过CAS server登录后携带ticket重定向到AstroZero。AstroZero后台使用ticket到CAS Server，校验ticket合法性，获取ticket对应的租户（可选）、用户信息，然后校验该用户是否存在，在指定了租户的情况下，如果该用户不存在，则创建对应用户。在没有指定租户的情况下，该用户不存在，则直接报错，并返回到CAS Server。 成功验证用户身份后，则生成Session写到Cookies上， 重定向到指定页面。例如，通过红标华为WeLink单点登录AstroZero，AstroZero返回access-token给WeLink，供WeLink访问AstroZero接口。	用户、业务用户	是
OAuth 2.0认证登录类别	使用OAuth 2.0的client credentials鉴权模式	根据鉴权ID和鉴权密钥，调用获取用户Token接口，来获取access_token。匿名用户一般通过该方式，来获取Token。	用户	是
匿名登录认证类别	使用OAuth 2.0的client credentials鉴权模式	匿名用户一般根据鉴权ID和鉴权密钥，调用获取用户Token接口，来获取access_token。	匿名用户	是
业务用户登录使用服务编排中公式登录	业务用户登录（服务编排中PORTALUSERLOGIN公式）	指定业务用户名，使用服务编排的“PORTALUSERLOGIN”公式，来登录。	业务用户	否（只能在服务编排中访问）

RefreshToken 特性

通过配置内置系统参数“bingo.service.refreshtoken.enable”为“是”的方式，来选择开启RefreshToken特性。开启后，业务用户通过服务编排的“PORTALUSERLOGIN”公式登录AstroZero时，会在Cookies中多返回一个“refresh-token”参数。可再通过发起请求“POST https://*AstroZero域名*/baas/auth/v1.0/

refreshtoken”来获取一个新的Token（调该接口时需要在请求消息头上设置参数“Content-Type”值为“application/json”，请求消息体中设置参数“grant_type”值为“refresh_token”，“refresh_token”值为Cookies中返回的“refresh-token”参数值）。原有的Token不管有没有失效，都会被置成失效状态。一般情况下，“refresh-token”的有效期会比较长，在用户的权限配置中可通过配置“刷新凭证时长”来控制其有效期。

步骤1 开启RefreshToken特性。

1. 登录AstroZero管理中心。
2. 在左侧导航栏中，选择“系统管理 > 系统参数”。
3. 在“内置系统参数”页签，查找“bingo.service.refreshtoken.enable”，单击搜索结果中该参数名称。
4. 在参数详情页，单击“值”后的，设置参数值为“是”，单击“保存”。

图 1-72 查找参数

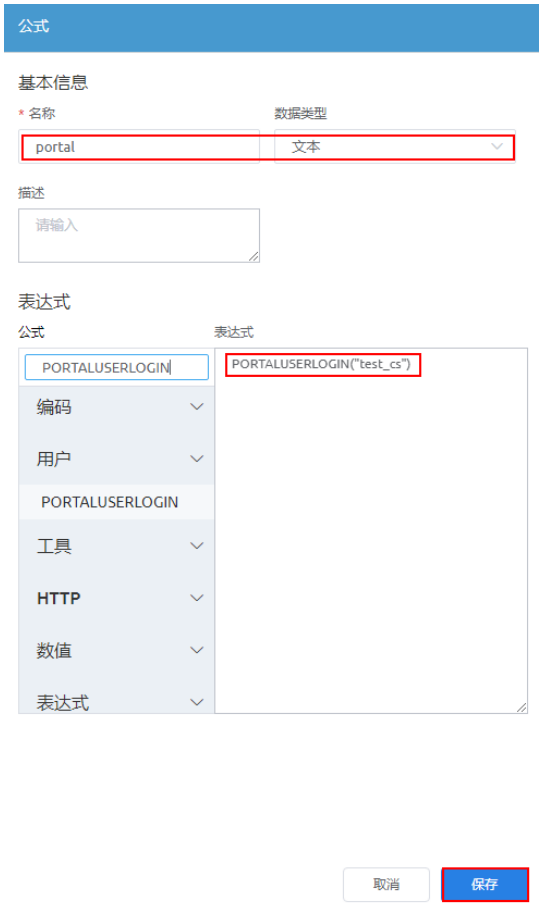
图 1-73 设置参数值为“是”

步骤2 业务用户（例如“test_cs”）通过服务编排的“PORTALUSERLOGIN”公式，登录AstroZero。

1. 在“我的应用”中，单击某个应用，进入应用开发页面。
2. 将鼠标放在左侧某个文件夹上，单击界面上出现的“+”，在弹出菜单中选择“服务编排”。
3. 设置标签为“testPortalUserLogin”，单击“名称”的输入框，系统会自动填充，单击“添加”。
4. 单击，在全局上下文页面，单击“公式”后的加号。

5. 在公式页面，设置“名称”为“portal”，“数据类型”设置为“文本”，表达式为“PORTALUSERLOGIN("test_cs")”，单击“保存”，创建公式变量“portal”，表达式中的“test_cs”为要登录系统的业务用户名。

图 1-74 公式



6. 单击，在全局上下文页面，单击“变量”后的加号，默认创建名为“variable0”的文本类型变量。
7. 单击，按照下图从全局上下文页面中拖拽参数到出参区域，设置服务编排的出参。

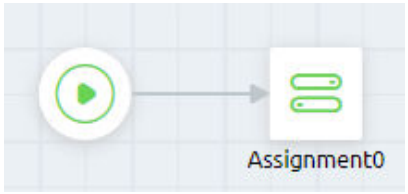
图 1-75 设置服务编排的出参

8. 在左侧拖拽“逻辑”下“赋值”图标至画布中，按照下图设置赋值图元。

图 1-76 设置赋值图元

9. 连接所有元素。

图 1-77 连接所有元素



10. 单击，保存服务编排。
11. 单击，不用输入入参直接单击“运行”，运行服务编排。
    输出如下信息，表明业务用户已登录AstroZero。

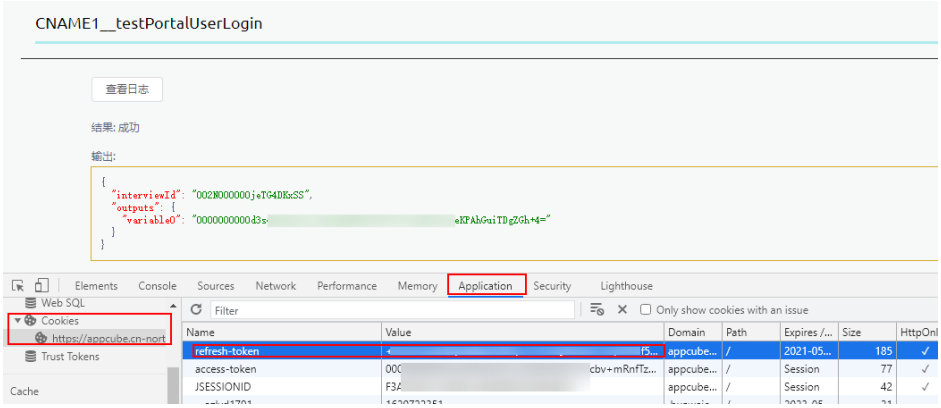
```
{
 "interviewId": "002N000000jeTG4DKxSS",
 "outputs": {
  "variable0": "XXX"
```

```
 }
}
```

其中，“variable0”的值即为业务用户的access-token值。

12. 在输出的页面，按“F12”或者“Ctrl + Shift + I”，可开启调试工具，在Cookies中获取“refresh-token”值。

图 1-78 获取“refresh-token”值



步骤3 使用Postman发送请求，获取新Token。

1. 在本地PC上，下载并安装**Postman**工具，该工具仅用于测试使用。
2. 使用Postman进行POST请求，URL配置为“https://*AstroZero域名*/baas/auth/v1.0/refreshtoken”，在请求消息头上设置参数“Content-Type”值为“application/json”，请求消息体中设置参数“grant_type”值为“refresh_token”。

   其中，“refresh_token”值为**步骤2.12**中获取的“refresh-token”参数值。输出的result值即为新的Token。原有的Token不管有没有失效，都会被置成失效状态。

图 1-79 Postman 发送请求

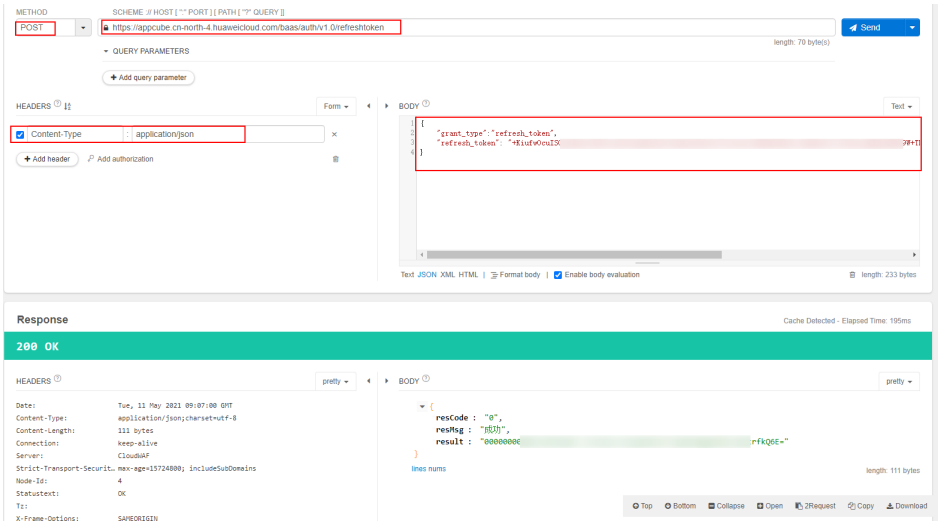


表 1-8 请求消息头

消息头名称	描述	是否必选
Content-Type	HTTP协议中设定的一个参数，用于标识返回的内容用什么格式去解析，此处必须配置为“application/json”，表示浏览器将返回内容解析为json对象。	是

表 1-9 消息体说明

参数名	描述	是否必选
grant_type	授权类型，配置为“refresh_token”。	是
refresh_token	配置为步骤2.12中，获取的“refresh-token”参数值。	是

----结束

1.5 轻应用建议的权限机制

什么是应用的权限机制

应用的权限机制是指应用是以何种方式配置应用权限来进行权限判断的，而配置应用的权限有两种方式：

- 当应用使用的元数据不多时，直接在元数据对象上配置权限。
- 当应用使用的元数据过多时，使用业务凭证配置公共接口权限。

建议的权限机制

轻应用下的应用模板一般为轻量级应用，不涉及复杂化的代码，用户零代码（如拖拽组件，简单配置）或者低代码就能轻松完成应用的搭建。并且一般轻应用使用的元数据数量不多，所以在轻应用中，建议使用的权限机制是直接在元数据对象上配置权限。

如何在元数据对象上配置权限

在元数据对象上配置权限，需要自定义对象模型，然后在自定义对象的自定义字段页签中新建字段，并给这个字段配置权限 。

以新建一个“调查问卷”模板轻应用“B”为例，向您介绍如何在元数据对象上配置权限。

步骤1 进入AstroZero经典开发环境，在“首页 > 项目 > 轻应用”中，将鼠标放在调查问卷模板上，单击“使用模板”，创建轻应用“B”。

图 1-80 调查问卷

步骤2 在应用“B”开发页面，选中Model目录下的元数据对象“*命名空间*__Questionnaires_qR__CST”，单击“自定义字段”。

说明

若创建的是空白轻应用，需要手动创建数据对象。单击Model目录后的，单击“对象”，进行创建。对象创建完成后，在“自定义字段”页签，单击“新建”，添加字段。

步骤3 单击“新增”，字段类型选择“文本”，单击“下一步”。

步骤4 设置字段标签和名称为“label”，数据长度设置为255，单击“下一步”。

步骤5 勾选所需的读取和编辑权限，单击“下一步”后，单击“保存”。

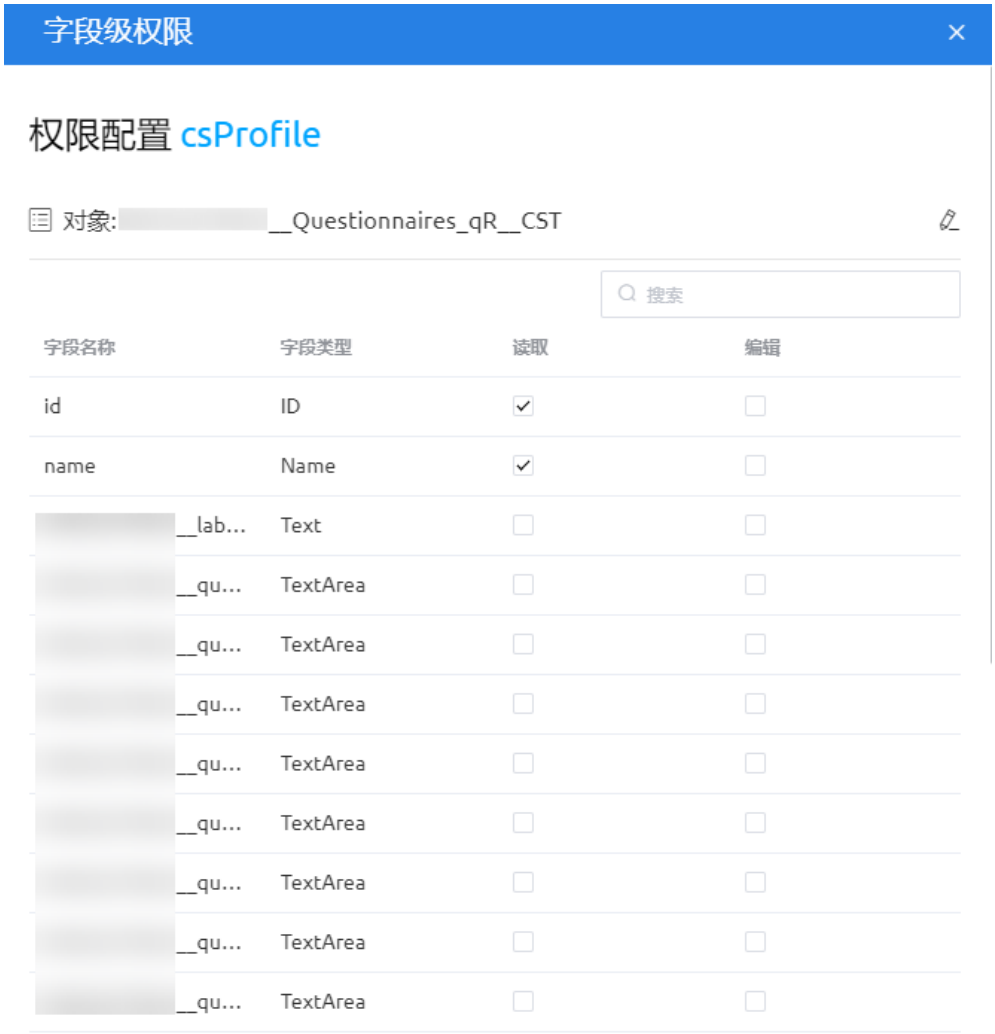
步骤6 配置元数据权限。

1. 在AstroZero管理中心的左侧导航栏中，选择“用户管理 > 权限配置”，进入权限配置列表。

图 1-81 权限配置

2. 单击元数据对象需要配置的权限名称（如元数据对象“*命名空间*__Questionnaires_qR__CST”需要配置的权限为“csProfile”），进入权限配置详情页面。
3. 在“自定义对象权限”，单击需要配置权限的自定义对象名称（本示例为*命名空间*__Questionnaires_qR__CST），在弹出框中单击，进行权限配置。

图 1-82 权限配置



----结束

如何给接口添加业务权限凭证

若用户或业务用户需要通过权限访问接口，需要给该接口添加业务权限凭证，具体操作请参见**如何给接口添加业务权限凭证**。

1.6 行业应用建议的权限机制

什么是应用的权限机制

应用的权限机制是指应用是以何种方式配置应用权限来进行权限判断的，配置应用的权限有两种方式：

- 当应用使用的元数据不多时，直接在元数据对象上配置权限。
- 当应用使用的元数据过多时，用业务凭证配置公共接口权限。

建议的权限机制

行业应用较轻应用而言，功能更加丰富，您可使用AstroZero提供的各种能力组件，采用低码或多码化模式开发应用。并且一般行业应用使用的元数据数量较多，按轻应用中直接配置元数据权限会比较麻烦，所以建议使用业务凭证配置公共接口权限。

如何使用业务凭证配置公共接口权限

在行业应用中，当需要给创建的公共接口添加业务凭证控制权限时，首先需要将接口发布到外部网关，然后给接口添加业务凭证。

以“A”应用为例，介绍如何给公共接口添加业务权限凭证。

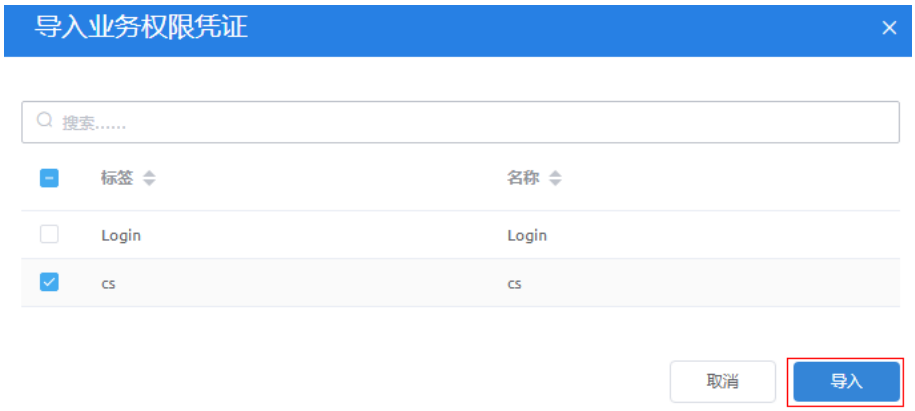
步骤1 在“我的应用”中，单击“A”应用，进入“A”应用开发页面。

步骤2 在“A”应用开发页面，单击页面左下方的“配置”，进入应用配置页面。

步骤3 在“业务权限凭证”，单击“导入”。

步骤4 选择需要导入的业务权限凭证，单击“导入”，完成业务权限凭证的导入。

图 1-83 导入业务权限凭证



说明

在“A”应用中选择导入业务权限凭证是因为在**如何给业务用户添加业务权限凭证**中已创建了需要的“cs”业务凭证。如果之前还未创建需要的业务凭证，则可以在本页面中单击“新建”，新建业务凭证，也可以参考**如何给业务用户添加业务权限凭证**中操作创建业务凭证。

步骤5 创建公共接口。

1. 在“A”应用的开发页面，单击页面左下方的“服务”，进入公共接口创建页面。
2. 在公共接口中，单击“新建”，创建“用户注册”脚本对应的公共接口，接口详细信息如**表1-10**所示。

说明

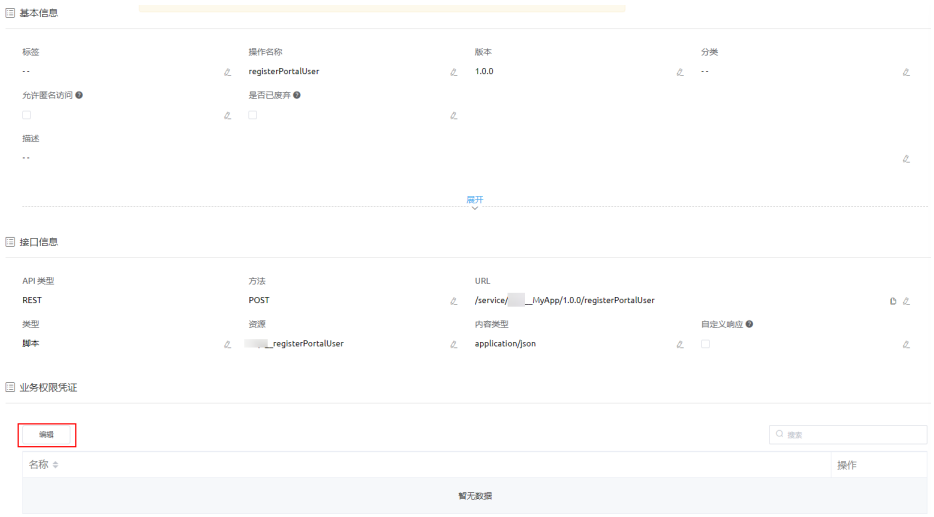
脚本registerPortalUser在**什么是业务用户**中已创建，如果在“资源”下拉框中，未找到需要关联的脚本，请检查相关脚本是否已启用。

表 1-10 公共接口

操作名称	版本	URL	类型	资源	方法
registerPortalUser	1.0.0	/registerPortalUser	脚本	*命名空间*__registerPortalUser	POST

3. 在公共接口详情页，单击业务权限凭证下的“编辑”。

图 1-84 公共接口详情



4. 在左侧列表中，选中需要添加的业务权限凭证，单击，将选择的业务权限凭证添加至右侧列表，单击“保存”，完成业务权限凭证的添加。

图 1-85 编辑业务权限凭证

5. （可选）给权限绑定业务权限凭证。
如何给权限绑定业务权限凭证，请参见**如何给业务用户添加业务权限凭证**。

6. （可选）给用户或业务用户配置相应权限。

   如何给用户或业务用户配置相应权限，请参见**如何给业务用户添加业务权限凭证**。

----结束

1.7 如何集成第三方账号登录

第三方账号认证方式

第三方账号集成登录AstroZero时，需要在AstroZero上认证，AstroZero认证成功后返回“access-token”，第三方账号通过“access-token”访问AstroZero。第三方账号获取“access-token”的方式有如下两种：

- OAuth 2.0协议接入认证

  根据鉴权ID和鉴权密钥，调用获取用户Token接口，来获取access_token。
- 单点登录（“CAS”协议）认证

  第三方账号通过CAS server鉴权，鉴权成功后发送ticket到AstroZero，AstroZero去CAS server验证ticket合法性并登录，成功后返回access-token。

OAuth2.0 协议接入认证分类

AstroZero采用OAuth 2.0协议进行接入认证时，第三方系统在调用AstroZero业务接口前，需要在AstroZero上进行鉴权注册，获取接入客户端ID、密钥等鉴权信息，才能实现调用AstroZero业务接口。

AstroZero对第三方系统进行OAuth 2.0接入认证的授权类型分为客户端模式和授权码模式两种，具体操作请参见**配置OAuth管理**。

单点登录（“CAS”协议）认证

单点登录认证是让AstroZero作为客户端，第三方进行cas鉴权作为服务端的认证方式。当AstroZero作为客户端时，第三方账号通过cas鉴权获得ticket，第三方账号携带ticket重定向到AstroZero，AstroZero后台获取ticket后到CAS Server校验ticket合法性，ticket合法性验证成功后AstroZero返回access-token给第三方账号供第三方账号访问AstroZero。

AstroZero作为客户端获取到ticket后，提供了sso快捷校验ticket合法性方法。使用sso对ticket进行验证样例代码如下：

```
import * as user from 'user';
import * as sys from 'sys';
import * as sso from 'sso'; //使用sso单点登录需要引用

//输入参数
@action.object({type: "param"})
export class MyObject {
    @action.param({type: 'String'})
    ticket: string;  //第三方进行cas鉴权时得到的ticket
    @action.param({type: 'string'})
    service: string; //第三方提供的service
}

//输出参数
@action.object({type: "param"})
export class Output {
```

```
    @action.param({type: 'String'})
    username: string; //返回的用户名
    @action.param({type: 'String'})
    token: string;    //返回的Access-token
}

//sso登录逻辑
@action.object({type: "method"})
export class ActionDemo {
@action.method({ label: 'greeting something', description: 'greeting something.', input: 'MyObject', output:
'Output' })
    public greet(inarg: MyObject): Output {
        console.log(inarg.ticket);
        let ticket = inarg.ticket;
        let service = inarg.service;
    //cas服务的域名
        let casUrl = "http://10.70.67.246:8899/cas"
    //根据传入的service和ticket使用sso去cas校验ticket的合法性并返回用户信息
        let cli = sso.newClientWithCas(service, ticket, casUrl);
        let a = cli.validateTicket();
        console.log(a);
        let out = new Output();
        out.username = a;
        //检测user是否存在
        createUser(out.username);
    //sso进行登录验证并返回 access-token
        let token = cli.login();
        out.token = token;
        return out;
    }
}

//检测user是否存在逻辑
function createUser(username: string) {
    try{
        let u = {
        "userName": username,
        "name": username,
        "email": username,
        "languageLocaleKey": "zh_CN",
        "timeZoneSidKey": "10",
        "profile": "000T0000000000000002",
    }
    let id = user.createInnerUser(u)
    console.log(id)
    }catch(e) {
        console.log(e)
    }
}
```

2 开发规范

2.1 应用工程构建规范

2.1.1 工程命名和目录结构

什么是应用工程

应用工程是指使用AstroZero开发的App和BO，应用工程规范包括App、BO的命名规范、目录结构规范，工程配置规范和工程服务规范。

- **App**是一个可复用的独立业务应用，如设备维修管理系统应用。

  App的交付件包括：

  - 逻辑接口Interface：对外暴露的接口服务，Restful API接口可以直接被前台或第三方系统调用。
  - 事件Event：对外暴露、供外部订阅的事件。
  - 菜单Menu：面向租户业务使用人员，提供配置操作能力。
- **BO**是一个可复用的领域服务，是最小粒度的复用单元。例如，为了使业务应用开发人员高效的实现与物联设备的交互，对物联设备进行业务抽象，封装为设备BO。
  - BO是资产封装的规范，可以基于平台元数据能力实现，也可以用其他方式实现。
  - BO是有边界的，BO与BO之间不能共享数据。
  - BO对外开放数据视图、接口服务、配置机制和事件等来进行交互，内部实现对外不可见。
  - BO的数据视图，以聚合根的形式开放供外部查询。BO可以同时开放多个聚合根。
  - BO与BO之间允许有接口调用的引用关系，但设计时应尽量避免BO与BO之间的直接交互，并完全禁止双向依赖。
  - BO中可以包含UI内容，如Layout、Widget，但仅作为参考实现。
  - BO应尽量利用配置能力，来适配不同的业务场景。配置无法满足时，才考虑对BO做Addon扩展。扩展不能影响BO的向前兼容性。

- BO被打包在App或Addon中，租户通过安装App或Addon，来获取BO。

说明

- 本文中字符数，均以英文字符作为计数单位，一个汉字算3个字符。
- 开发态的名称定义使用英文，展现给租户使用的配置参数显示则使用中文。

BO的交付件包括：

- 接口Interface：BO对外暴露的接口服务，一类是可以直接被前台或第三方系统调用的Restful API接口，一类是只能在开发平台上通过编排来调用的SDK API。
- 事件Event：对外暴露、供外部订阅的事件。
- 配置Configuration：面向开发人员或IT管理人员的，提供的配置能力。其中，配置菜单是BO提供的配置界面，选项列表是BO内部使用的字典项，系统参数是BO内部需要读取的控制参数。

工程命名规范

APP和BO采用统一的命名规范，如表2-1所示。

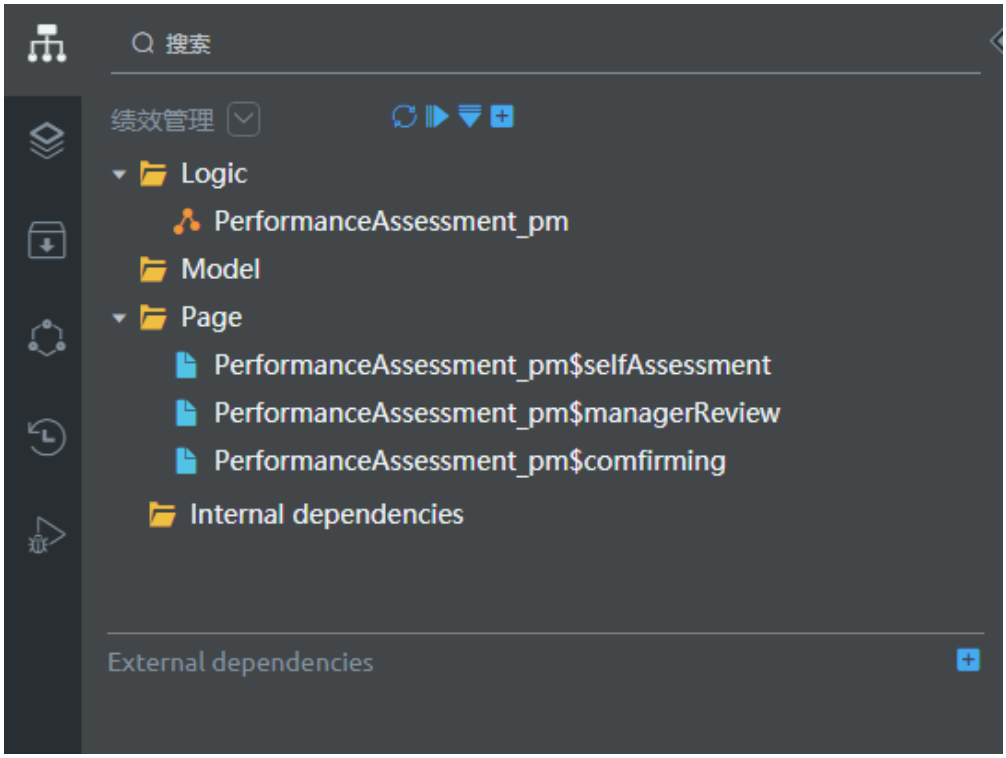
表 2-1 命名规范

参数	命名要求
标签	• 最多输入80个字符。 • 输入有意义的英文，采用大驼峰式命名，单词间以单个空格分割，不要含有制表符或者特殊字符。 • 相关联的工程可以采用相同的前缀进行标识。 • 工程命名可以参考业务架构的2层视图。
名称	• 最多输入64个字符。 • 必须以字母开头，且只能包含字母、数字和下划线。 平台会自动将租户的命名空间添加为应用的名称前缀。
描述	最多输入255个字符，介绍当前工程的主要功能。

工程目录结构

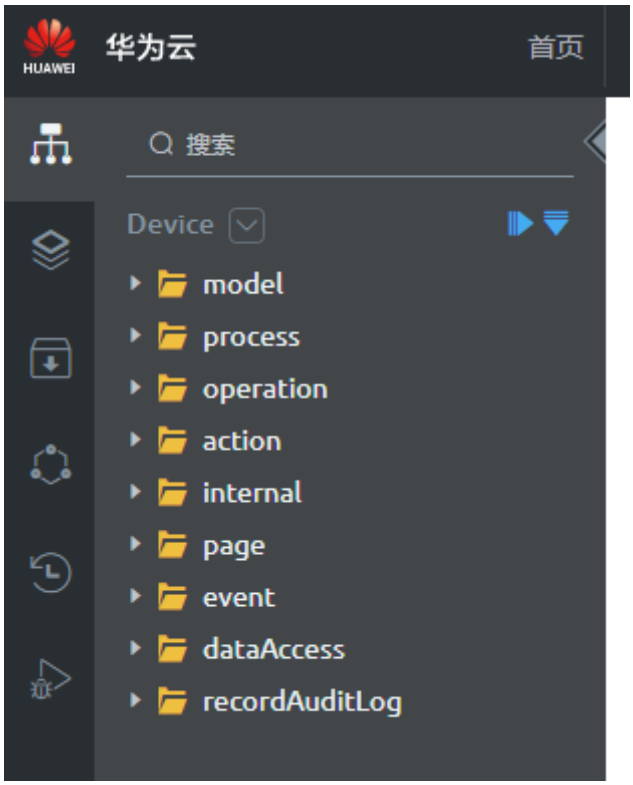
- **APP目录总体结构**
  - Logic：用于存放后台逻辑。
  - Model：用于存放数据模型（即数据对象）。
  - Page：用于存放前端页面。
  - Internal dependencies：内部依赖文件。

图 2-1 APP 工程目录示例



- **BO目录总体结构**
  - Model：存放数据模型，其下包含聚合根Aggregrator和对象Object两个子目录。
  - Process：流程级API，可单向依赖其它资产。
  - Operation：操作级API，提供完整的业务操作。
  - Action：原子级的API，可重用逻辑块。
  - Internal：内部实现。
  - Page：页面，按照具体的业务功能分目录聚合。
  - Event：BO对外抛出的事件。
  - DataAccess：数据接入。

图 2-2 BO 工程目录示例



- **目录树命名**

  APP、BO中目录树的所有目录都采用全英文大驼峰命名，不要存在空格或者特殊字符。每个目录下的资产，按照功能建立子目录以方便区分、管理和查找。

- **对象脚本页面命名**
  - Logic下的脚本和服务编排命名：脚本采用小驼峰命名，例如createDeviceInstance。服务编排采用驼峰命名，首字母小写，不要携带下划线。
  - Model下的Object命名：采用全英文的大驼峰命名。
  - Page下的页面文件命名：大驼峰英文字母组成，使用连字符“-”来提高可读性，而不是使用下划线“_”。
- **调用约束**
  - BO内：Process –> Operation –> Action –> Internal，下层不允许调用上层逻辑。
  - BO间：只有Process API允许调用其他BO的API。BO间的调用关系，只允许单向调用，不允许循环调用。
  - APP：只能调用其他APP或者BO提供的API。

2.1.2 工程配置

选项列表

如果对象的字段允许在多个已定义的值中选择一个，使用选项列表Picklist。选项列表的定义示例，如图2-3所示。

在基线包中导入的Picklist，默认为基线资产，不允许在界面上修改和删除，仅支持在界面上新增、编辑和删除非基线的PickList。

图 2-3 选项列表定义示例

- 显示名称

  BO的选项列表需要定义显示名称，用于显示在BO配置菜单中，应避免与其他可选配置项以及菜单名重复，建议以“工程名+选项列表”的固定方式拼接而成。使用中文字符，最长不超过32个字符。

- 选项列表定义
  - 名称：最多输入64个字符，必须以字母开头且只能包含字母、数字和下划线，采用大驼峰式命名，例如DeviceStatus。
  - 标签：最多输入255个字符，应该输入有意义的英文，采用大驼峰式命名，单词间以单个空格分割，不要含有制表符或者特殊字符。
  - 描述：最多输入255个字符，介绍当前PickList的使用场景。
- 选项列表的值定义
  - PickList取值选项不允许重复
  - 每个取值最多输入64个字符
  - 如果取值是字符类型，采用大驼峰形式命名，不要含有空格或者制表符等占位符，例如Active。

系统参数

对于业务逻辑中存在可变性的参数（例如对接第三方接口的IP地址），无论是在页面还是服务编排和脚本中，都建议使用系统参数，而不能声明为内部变量。系统参数的定义示例，如图2-4所示。

在基线包中导入的Picklist，默认为基线资产，不允许在界面上修改和删除。仅支持在界面上新增、编辑和删除，非基线的PickList。

图 2-4 系统参数定义示例

- 显示名称

  BO的系统参数需要定义显示名称，用于显示在BO配置菜单中，应避免与其他可选配置项以及菜单名重复，建议以“工程名+系统参数”的固定方式拼接而成。使用中文字符，最长不超过32个字符。

- 系统参数
  - 名称：最多输入64个字符，必须以字母开头且只能包含字母、数字和下划线，采用大驼峰式命名，例如ThirdPartyAddress。
  - 值：文本类型的参数，取值最多输入255个字符。对于敏感信息（例如第三方系统用户名、密码），需要加密保存。数字类型参数，取值允许输入数字、负号和小数点。
  - 描述：最多输入255个字符，用于介绍当前参数的用途。

App 导航菜单

App菜单需要按照功能聚集，并按照功能关系形成菜单层级。App的菜单定义示例，如图2-5所示，运行显示效果，如图2-6所示。

图 2-5 App 导航菜单定义示例

图 2-6 App 导航菜单运行显示示例

每个App均需要设置默认主页；菜单层级最大支持三级，层次过深会导致菜单栏占用过多页面空间，且不利于菜单名称展示（英文菜单尤其明显）。

- **标签**
  - 标签应直观体现菜单功能，起到功能导航作用。
  - 最多输入60个字符，采用中文菜单时，不超过10个汉字
  - 不能包含空格、制表符以及特殊字符。
- **图标**

  菜单图标不能重复。
- **描述**

  最多输入255个字符，介绍当前菜单的具体功能。

BO 配置导航

BO的配置导航菜单需要按照功能聚集，并按照功能关系形成菜单层级。BO的菜单定义示例，如图2-7所示。

图 2-7 BO 配置导航的定义示例

BO配置导航的要求与App菜单一致。

运营配置

运营配置用于租户配置BO的可变项。

- **显示名称**:
  BO的运营配置定义显示名称，用于在“应用管理 > BO配置”菜单下显示的BO名称。应避免与其他可选配置项以及菜单名重复，建议此名称和工程名保持一致以提高区分度。使用中文字符，不要含有制表符或者特殊字符，最长不超过32个字符。
- **布局**
  只有把配置项从“可选配置”中拖入到“已选配置”后，才可以在BO配置菜单中展现。

2.1.3 工程服务

本章节提供API API和SDK的定义要求。

Rest API

REST中定义工程对外，提供的服务。

- **HTTP动词**
  常用的HTTP动词如下（括号里是对应的SQL命令）：
  - GET（SELECT）：从服务器取出一项或多项资源。
  - POST（CREATE）：在服务器新建一个资源。
  - PUT（UPDATE）：客户端提供改变后的完整资源，在服务器更新资源。

  - PATCH（UPDATE）：客户端提供改变的属性，在服务器更新资源。
  - DELETE（DELETE）：从服务器删除资源。

- **资源路径**
  - URL中不能有动词

    在Restful架构中，每个网址代表的是一种资源，所以网址中不能有动词，只能有名词，动词由HTTP的动作来表示。
  - URL结尾不应该包含斜杠“/”

    正斜杠（/）不会增加语义值，且可能导致混淆。
  - 正斜杠分隔符“/”必须用来指示层级关系

    路径中的正斜杠“/”字符，用于指示资源之间的层次关系。
  - 应该使用连字符“-”来提高URL的可读性，而不是使用下划线“_”

    一些文本查看器为了区分强调URL，经常会在URL下加上下划线。这样下划线“_”字符可能被文本查看器中，默认的下划线部分的遮蔽或完全隐藏。为避免这种混淆，请使用连字符而不是下划线。
  - URL路径中首选小写字母

    URL是对大小写敏感的，所以为了避免歧义，尽量用小写字符。

    补充说明：主机名（Host）和scheme（协议名称:http/ftp/...）对大小写是不敏感的。
  - 应该将API的版本号放入URL

    另一种做法是，将版本号放在HTTP头信息中，但不如放入URL方便和直观。
  - URL路径名词均为复数

    为了保证URL格式的一致性，建议使用复数形式。

- **响应状态码**
  - 200 | OK ： 用于一般性的成功返回。
  - 201 | Created [POST/PUT/PATCH]：用户新建或修改数据成功。
  - 202 | Accepted：用于Controller控制类资源异步处理的返回，仅表示请求已经收到。对于耗时比较久的处理，一般用异步处理来完成。
  - 204 | No Content：此状态可能会出现在PUT、POST、DELETE的请求中，一般表示资源存在，但消息体中不会返回任何资源相关的状态或信息。
  - 301 | Moved Permanently：资源的URI被转移，需要使用新的URI访问。
  - 302 | Found：不推荐使用，此代码在HTTP 1.1协议中被303/307替代。目前对302的使用和最初HTTP 1.0定义的语意是有出入的，应该只有在GET/HEAD方法下，客户端才能根据Location执行自动跳转，而目前的客户端基本上是不会判断原请求方法的，无条件的执行临时重定向。
  - 303 | See Other： 返回一个资源地址URI的引用，但不强制要求客户端获取该地址的状态（访问该地址）。
  - 304 | Not Modified：有一些类似于204状态，服务器端的资源与客户端最近访问的资源版本一致，并无修改，不返回资源消息体。可以用来降低服务端的压力。
  - 307 | Temporary Redirect： 目前URI不能提供当前请求的服务，临时性重定向到另外一个URI。在HTTP 1.1中307是用来替代早期HTTP 1.0中使用不当的302。
  - 400 | Bad Request：用于客户端一般性错误返回，在其它4xx错误以外的错误，也可以使用400，具体错误信息可以放在body中。

- 401 | Unauthorized： 在访问一个需要验证的资源时，验证错误。
- 403 | Forbidden：一般用于非验证性资源访问被禁止，例如对于某些客户端只开放部分API的访问权限，而另外一些API可能无法访问时，可以给予403状态。
- 404 | Not Found：找不到URI对应的资源。
- 405 | Method Not Allowed：HTTP的方法不支持，例如某些只读资源，可能不支持POST/DELETE。但405的响应header中必须声明该URI所支持的方法。
- 406 | Not Acceptable：客户端所请求的资源数据格式类型不被支持，例如客户端请求数据格式为application/xml，但服务器端只支持application/json。
- 409 | Conflict： 资源状态冲突，例如客户端尝试删除一个非空的Store资源。
- 412 | Precondition Failed：用于有条件的操作不被满足时。
- 415 | Unsupported Media Type：客户所支持的数据类型，服务端无法满足。
- 500 | Internal Server Error：服务器端的接口错误，此错误于客户端无关。

SDK

SDK表示工程对应用编排提供的API，SDK没有暴露REST接口。BO之间调用，只能使用SDK。

2.2 脚本开发规范

2.2.1 命名

脚本（简称Script）用于开发原子功能API。脚本不能被外部直接通过Rest接口调用，只能用于业务编排。对脚本要进行合理的抽象和封装，以便在不同的服务编排中复用。本章节介绍脚本中函数、结构体、参数等的命名要求。

所有的名称定义要能体现其作用，避免使用缩写（专有名词除外）。

- 脚本采用小驼峰命名，例如createDeviceInstance。
- 结构体（struct）采用大驼峰命名，例如QueryPaymentResult。
- 结构体内的字段采用小驼峰命名，例如customerName 。
- 类、枚举值和接口采用大驼峰命名。
  ```
  class Comments {...}
  interface Console {...}
  enum Direction { Up = 1, Down, Left, Right}
  ```
- 函数采用小驼峰命名。
  ```
  addOfferingAssociatePriceAndStock(input: Input): Output {...}
  ```
- 属性或变量采用小驼峰命名。
  ```
  class Comments {
          userId: String;
          content: String;
      }
  let oneComments = newComments();
  ```
  - 变量为单数时，命名包含对象名称，如level1Catalog、level2Catalog。
  - 变量为复数时，命名包含集合名称，如attributeRelationRuleList。

2.2.2 注释

本章节介绍脚本注释的原则和写法。

总体原则

- 无用的代码不能以注释形式存在。
- 能用代码说明的尽量不要添加注释，脚本注释尽可能简洁。
- 建议注释统一用英文。
- 出入参不必写注释说明。

注释方式

- 在函数方法和结构体的元数据描述上，添加注释。

```
/**
* 根据产品ID查询产品详情信息
*/
@action.method({ input: "Input", output: "Output", label: 'queryProductDetailForCart' })
```

- 方法内关键业务语句前，必须添加注释。
  - 方法内的单行注释以“//”开头，应放在相关代码的上方或右方，不可放于下方。若放于上方，注释需与前面的代码间用空行隔开。
  - 注释与代码的比例没有量化标准。在删掉所有的代码内容、仅保留代码层级结构和注释，如果通过注释，可以很容易理解方法内都做了哪些事情（类似于伪代码），则认为注释比例合理。

```
// 校验密码是否正确
let password = input.password;
if (accountRecord["Password"] != password) {
error.name = "CM-001003";
error.message = "Invalid loginId or password.";
throw error;
}
```

2.2.3 引用

本章节介绍脚本间相互引用的原则。

- 脚本中不要包含没用到的标准库或对象的引用。

  例如，如果实际没用到sys模块的任何方法，则不要包含如下语句：

```
import * as sys from 'sys';
```

  如果实际上只用到了CA_PC_Offering对象，则示例语句中其他对象需要删除：

```
@useObject(['CA_PC_FeeItemDef', 'CA_PC_Offering', 'CA_PC_CatalogOffering', 'CA_PC_Catalog',
'CA_PC_OfferingAttribute'])
```

- 只引用使用到的对象，而不用import *。

  【推荐】：

```
import { OfferingObjectQuery } from './ca_pc__getOfferingObject';
import { setI18nError } from 'context';
import { sql } from 'db';
```

  【不推荐】：

```
import * as queryOfferingObjectAction from './ca_pc__getOfferingObject';
import * as context from 'context';
import * as db from 'db';
```

- @action.method属于必须的方法注解，写在方法上面。

```
@action.method({ input: "Input", output: "Output", label: 'queryClassification' })
queryClassification(input: Input): Output {
```

```
...
}
```

- 为方便页面直接调用脚本，仅在脚本最后统一导出需要的对象，而不是导出所有对象。如果不需要，可以不用导出。

  例如，getOfferingObject脚本中最后一行导出OfferingObjectQuery对象：

```
export let theAction = new OfferingObjectQuery();
```

- 可以把公共的对象Object定义，按分类放在一些公共的脚本里（例如pc_XXX.ts、cm_XXX.ts），其他需要的脚本直接引用即可。

2.2.4 定义

本章节介绍字段、对象、结构体等的定义规则。

- 每个字段的定义，均需要定义type、label、description、required和isCollection，有默认值的非必填。
- 当字段为集合类型时，需要定义成“[]”。

```
@action.param({
    type: "Attribute",
    label: "Attribute",
    description: "attributeList",
    required:false
    isCollection: true
})
attributeList: Attribute[];
```

- @action.object时，需要在脚本中详细定义清楚Object，不要引用其他脚本的Object。
- 如果是嵌套结构体，则从下到上粒度依次变小。

```
export class ProductObject {
      //ignore
}

export class StockObject {
      //ignore
}

export class ProductStock {
    @action.param({ type: 'Struct', label: "ProductObject", isCollection: false })
    productInfomation: ProductObject;

    @action.param({ type: 'Struct', label: "StockObject", isCollection: false })
    stock: StockObject;
}

@action.object({ type: "param" })
export class Input {
    @action.param({ type: 'String', label: "productId", isCollection: false })
    productId: String;
}
```

- 不需要多定义Output对象，可以直接在方法使用定义的对象出参。

  【推荐】：

```
@action.object({ type: "param" })
export class Input {
    @action.param({ type: 'String', label: "productId", isCollection: false })
    productId: String;
}

@action.object({ type: "param" })
export class getProductStock {
@action.method({ input: "Input", output: "ProductStock", label: 'getProductStock' })
getProductStock(input: Input): ProductStock {
```

【不推荐】：如果出参对象包含从外部引入的对象，还是要按该方式定义。

```
@action.object({ type: "param" })
export class Input {
    @action.param({ type: 'String', label: "productId", isCollection: false })
    productId: String;
}

@action.object({ type: "param" })
export class Output {
    @action.param({ type: 'Object', isCollection: false })
    ProductStock: ProductStock;
}

@action.object({ type: "param" })
export class getProductStock {
@action.method({ input: "Input", output: "Output", label: 'getProductStock' })
getProductStock(input: Input): Output {
```

- 除非业务有特殊要求，新增修改脚本不返回结果码和结果信息。

2.2.5 调试

本章节介绍调试、日志代码的规则。

- try、catch

  不需要做统一异常处理时，都不用写try、catch语句，只需抛出统一的错误码。

  【推荐】：

```
if (input.customerId == "") {
      context.throwError("STO-001003");
};
```

  【不推荐】：

```
if (input.customerId == "") {
      context.setError('STO-001003', 'You have not login or the session has expired, please
login again.');
      return;
};
```

- 打印日志（Console.log()）

  根据业务场景需要，来控制打印日志语句，建议一个脚本最多不要超过三个打印日志语句。

- 最终归档时，脚本不要有调试代码。

2.2.6 语法

本章节介绍常见语句示例，以给出语法规则。

- 对象判空

  单个对象示例：

```
if(object){
    //ignore
}
```

  集合对象示例：

```
if(collection&&collection.length!=0){
    //ignore
}
```

- 数字类型统一定义成“Number”、日期类型定义成“Date”。

```
@action.param({
    type: "Date",
    label: "effectiveTime",
```



```
    description: "Date."
})
effectiveTime: Date;

@action.param({
    type: "Number",
    label: "salePrice",
    description: "salePrice."
})
salePrice: number;
```

- 变量或数组定义时，要说明类型，集合需要加上泛型。

```
let isDone:Boolean = false;
let decLiteral : number =6;
let productList = new Array<productObject>();
```

- 遍历循环推荐用“forEach”，不推荐用“for…in”。

```
testArray.forEach((value, index, array )=>{
      //ignore
});
```

- 推荐用“let”变量声明，不推荐用“var”。

  【推荐】：

```
let offeringId : String = “aaa”;
```

  【不推荐】：

```
var offeringId = “aaa”;
```

- 前段代码有对象声明时，推荐使用“.fieldName”获取对象的字段，而不是用“[‘fieldName’]”。

  【推荐】：

```
offeringStruct.id = result[i].base.offeringId;
```

  【不推荐】：

```
offeringStruct.id = result[i][‘base’][‘offeringId’];
```

- 在需要默认值的情况下，使用“||”代替“if”判断。

```
function(value){
      //使用||对value进行判空
      value = value || “hello”
}
```

- 创建一个已知对象时，推荐用“new”方法。

  【推荐】：

```
let offeringIdRequest = new OfferingIdRequest();
offeringIdRequest.catalogList = catalogList;
offeringIdRequest.classificationList:=classificationList;
let getOfferingIdByConditionInput = {
      "OfferingIdRequest": offeringIdRequest
}
```

  【不推荐】：

```
let getOfferingIdByConditionInput = {
      "OfferingIdRequest": {
            "catalogList": catalogList,
            "classificationList": classificationList,
            "status": "",
            "stockCondition": "",
            "keyWord": "",
            "offset": "",
            "limit": ""
      }
}
```

- 函数定义

```
let traitRec = function(xxxx,xxx) {
   //ignore
 }
```

只能在函数定义后的语句中，使用该函数。

- 没有初始化值的变量申明，使用“undefined”，不要使用“null”。
```
let object = undefined;
```
- 局部变量需要在“class”内定义，不要在全局命名空间内定义类型/值（即不要在“class”外定义变量），常量可以定义成全局。
```
let identityIdList = [];
```
- 使用“lambda”表达式代替匿名函数。

  只有需要时，才把“arrow”函数的参数括起来。正确使用“arrow”的示例如下：
```
x => x + x
(x,y) => x + y
<T>(x: T, y: T) => x === y
```

2.2.7 SQL

本章节介绍脚本中SQL的规则。

- 不推荐用拼接SQL方法，避免注入风险。
```
let sql = "select id,name,ExternalCode,FeeType,FeeItemName,FeeItemDescription,Remark,Status from CA_PC_FeeItemDef where 1=1";
if (!InputParams.id && !InputParams.name && !InputParams.feeType && !InputParams.feeItemName && !InputParams.status) {
    context.setI18nError("ca_pc__001013");
    return;
}
if (InputParams.id) {
    sql += "and id ='" + InputParams.id + "'"
}
```
- 多表复杂查询建议用“sql.exec()”或“sql.excute()”方法，“excute()”方法比“exec()”多返回字段集和操作成功数。
```
let result = execsql.exec("select id,name,OfferingId,ParentId,SkuCode,ChannelId,Status,ProductLabel,PriceCode,DefaultChoose,PaymentType from CA_PC_Product where id in (" + str + ")",
{
    params: productId
});
```

  多表复杂查询只能使用拼接SQL方法，但是有限制，例如示例中的“str”要求如下：

  - str如果来源于入参，则入参在拼接SQL之前需要进行校验，以免引入SQL注入攻击。如果来源于内部数据，可以不进行校验。
  - 不推荐直接在SQL中拼接入参，应该采用在SQL中拼接占位符，然后把入参放入参数的数组中，例如：
```
attriSql = "select AttrDef from DE_DeviceDefAttri where DeviceDef=? and ExternalCode=? and AttrType='DYNAMIC' and ValueType='1'";
attriRecords = db.sql().exec(attriSql, {
    params: [deviceDefId, defExternalCode]
  });
```
- 对于单表查询和增删改SQL，推荐使用Orm接口方法。

  Options选项可以选择返回字段、排序、聚合运算和分页等功能。当value不存在时，默认为null类型。
```
let CA_PC_Stock = db.object('CA_PC_Stock');
let amountCount = 0;
if (UpdateStock.amount) {
    let record = { Amount: UpdateStock.amount };
    amountCount = CA_PC_Stock.updateByCondition({
        "conjunction": "AND",
        "conditions": [{
            "field": "SkuCode",
```

```
                "operator": "eq",
                "value": UpdateStock.skuCode
            }]
    }, record);
}
```

- 避免在循环中调用方法和操作数据库，可以用“in”来查询在集合中的结果。

```
let productIdList = input.productId;
let str = "";
let arr = [];
for (let i = 0; i < productIdList.length; i++) {
    arr[i] = "?";
}
str = arr.toString();
let result = execsql.exec("select
id,name,OfferingId,ParentId,SkuCode,ChannelId,Status,ProductLabel,PriceCode,DefaultChoose,Payment
Type from CA_PC_Product where id in (" + str + ")",
{
    params: productIdList
});
```

- 对sql进行优化时，尽量使用有索引的字段，避免使用没有索引的字段。
- 批量操作数据库时，尽量使用已封装好的批量操作接口。

  例如，Orm.batchInsert、Orm.batchUpdate接口和Orm.deleteByCondition批量删除接口。如果批量创建父子对象记录，且批量创建的记录在一个完整的事务中，全部成功或全部失败，建议使用Orm.compositeInsert接口。

```
var s = db.object('Customer__CST');
var records = [];
var record = {
   "name": "hello",
   "count__CST": 123,
   "Contacts": {
      "records": [
         {
            "name": "hello_contact1"
         },
         {
            "name": "hello_contact2"
         },
      ]
   }
};
records.push(record);
var ids = s.compositeInsert(records);
console.log("id list = ", ids);

count = s.count();
console.log("record count = ", count);
```

- 匹配查询推荐用like，日期比较推荐用“<”、“>”。

  【推荐】：

```
select id from t where name like ‘abc%’
select id from t where createdate>=’2005-11-30’ and createdate<’2005-12-1’
```

  【不推荐】：

```
select id from t where substring(name,1,3)=’abc’
select id from t where datediff(day,createdate, ‘2005-11-30’)
```

- 使用exists替代in，使用not exists替代not in。

  【推荐】：

```
SELECT * FROM EMP (基础表) WHERE EMPNO > 0 AND EXISTS (SELECT ‘X’ FROM DEPT WHERE
DEPT.DEPTNO = EMP.DEPTNO AND LOC = ‘MELB’)
```

  【不推荐】：

```
SELECT * FROM EMP (基础表) WHERE EMPNO > 0 AND DEPTNO IN(SELECT DEPTNO FROM DEPT
WHERE LOC =’MELB’)
```

- 避免在索引列上使用is null和is not null，会造成索引失效。

  【推荐】：

  ```
  SELECT … FROM DEPARTMENT WHERE DEPT_CODE >=0
  ```

  【不推荐】：

  ```
  SELECT … FROM DEPARTMENT WHERE DEPT_CODE IS NOT NULL
  ```

- 注意事项
  - 尽量避免Select字句中，使用“*”。
  - 尽量避免在where子句中，使用!=或<>操作符。
  - 尽量避免在where子句中，对字段进行函数操作。
  - 尽量避免在SQL中，使用“!=”、“||”、“+”符号。
  - 尽量避免在where条件中，做数据筛选。
  - 尽量避免查询中，按字段排序。
  - 尽量避免多表关联查询和嵌套查询，不要使用超过2表的关联查询。
  - 不要在循环内重复使用同一条件查询，应该在循环外处理。例如，公共数据仅在循环外查询一次。
  - 不要在同一个脚本的多个方法内，使用同一条件多次查询，可以定义类的成员变量。
  - 避免关联查询

    关联的条件很多情况下都是唯一的，可以提前做单独查询。例如，使用公共数据作为后续条件，避免关联查询。
  - 避免频繁的数据库交互

    例如，查询5000条数据，查询补充数据的时候，不要在循环内多次交互数据库，把可以合并的条件在循环外拼接并进行一次性查询，在循环内只需要从结果集中获取数据，可以极大提升查询性能。
  - 尽量利用对象做临时缓存

    例如查询到DeviceDef后，按照“id:Object”的方式存起来，后续查询时先判断缓存对象中是否已经存在，如果存在则直接获取不再查询。

2.2.8 代码风格

本章节介绍代码风格建议。

- 每句话后面加分号，脚本写完右键选择“Format Document”统一格式。
- string类型赋值统一使用双引号，获取字段统一使用单引号。
- 相关代码写在一起，不相关逻辑最好以空行隔开。
- 总是使用“{}”把循环体和条件语句括起来。
- 开始的“{”总是在同一行。
- 小括号里开始不要有空白逗号，冒号、分号后要有一个空格。

  ```
  for (var i = 0, n = str.length; i < 10; i++) { }
  if (x < 10) { }
  function f(x: number, y: string): void { }
  ```

- 每个变量声明语句只声明一个变量。

  例如，使用如下方式

  ```
  var x = 1; var y = 2;
  ```

  而不是下面的方式


```
var x = 1, y = 2;
```

- else要在结束的“}”后，另起一行。
- 一个函数仅完成一件功能，即使是简单功能也应该编写单独的方法实现。
- 单个方法的方法体不要太长，建议控制在150行以内，保证代码可读性，也方便维护、测试。

2.2.9 安全编码

本章节介绍编码的安全要求。

输入校验

不能依赖客户端校验，必须使用服务端代码对输入数据进行最终校验。对于在客户端已经做了输入校验，在服务器端再次以相同的规则进行校验时，一旦数据不合法，必须使会话失效，并记录告警日志。

必须假定所有用户产生的输入都是不可信的，并对它们进行合法性校验和值域校验，一旦数据不合法，应该告知用户输入非法并建议用户纠正输入。

- 如果输入为数字参数，必须进行数字型判断。
- 如果输入只允许包含某些特定的字符或字符的组合，使用白名单（推荐使用正则表达式）进行输入校验。
- 如果输入为字符串参数，必须进行字符型合法性判断。
- 如果明确输入数据的长度限制，必须校验输入数据的长度。
- 如果输入数据为数值且明确范围，必须检验数据的范围。
- 禁止通过字符串串联，直接使用用户输入构造可执行SQL语句，降低SQL注入攻击的风险。
- 用于重定向的输入参数不能包含回车和换行字符，以防止HTTP响应拆分攻击。

  说明

  “回车”字符有多种表示方式（CR = %0d = \r ），“换行”字符有多种表示方式（LF = %0a= \n ）。

上传下载

- 必须在服务器端采用白名单方式，对上传或下载的文件类型、大小进行严格的限制。
- 禁止以用户提交的数据，作为读、写、上传、下载文件的路径或文件名，以防止目录跨越和不安全直接对象引用攻击。
- 禁止将敏感文件（如日志文件、配置文件、数据库文件等），存放在Web内容目录下。

  Web内容目录指的是：通过Web可以直接浏览、访问的目录，存放在Web内容目录下的文件容易被攻击者直接下载。

会话及权限管理

对于每一个授权访问的接口内，都必须核实用户的会话标识是否合法、用户是否被授权执行这个操作。授权和用户角色数据必须存放在服务器端，不能存放在客户端，鉴权处理也必须在服务器端完成。

敏感数据保护

敏感数据包括但不限于口令、密钥、证书、会话标识、License、隐私数据（如短消息的内容）、授权凭据和个人数据（如姓名、住址、电话等）等。在程序文件、配置文件、日志文件、备份文件及数据库中，都有可能包含敏感数据。

- 禁止在代码和日志中，存储敏感数据。
- 禁止密钥或账号的口令，以明文形式存储在数据库或文件中。
- 禁止使用自己开发的加密算法，必须使用公开、安全的标准加密算法。

2.2.10 调用约束

本章节介绍脚本的调用约束。

- 不允许在应用项目的Script中，调用BO的Script。
  例如， 设备管理应用的Script，不可调用设备BO内的任何Script。
- 不允许跨BO，调用Script。
  例如，人员BO的Script，不允许调用设备BO内的任何Script。

2.2.11 代码规范案例汇总

- 脚本中的class和脚本名应该一致。

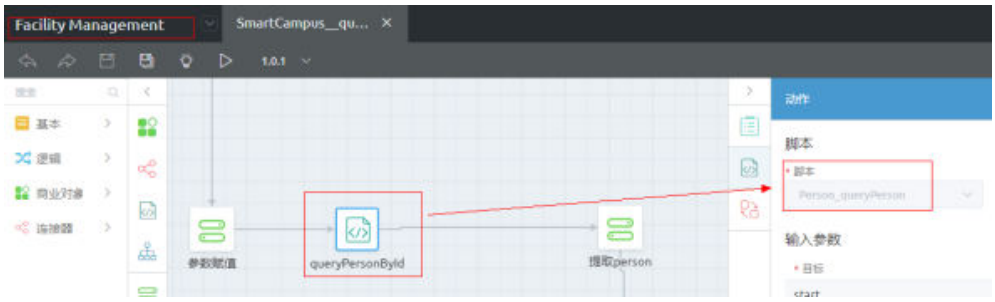
- 不要把整个处理逻辑使用try catch包起来，这样会导致报错时，无法定位原始错误行，需要去掉try catch。

  不推荐以下写法:

```
public run(input: ParamsInput): ParamsOutput {
    let out = new ParamsOutput();
    let error = new Error();
    try {…
    } catch (error) {
        context.setError(error.name, error.message);
    }
    return out;
```

- APP、BO中，需要使用rest或sdk，来直接调用其它BO的内部服务编排。

- APP、BO中，需要使用rest或者sdk，来直接调用其它BO的内部脚本。



- 需要先定义错误码，才能正确的抛出错误码信息，否则会随意抛出错误。

```
let childrenCount = queryChildrenCount(input.id);
if (childrenCount > 0) {
    context.setError('error', 'children organizations should be deleted first!');
    return out;
}
```

```
let childrenCount = queryChildrenCount(input.id);
if (childrenCount > 0) {
    context.throwError("Organization.ChildOrgExist");
}
```

名称	格式	类别	语言
Organization.ChildOrgExist	当前组织存在下层组织，不允许删除！		中文
Organization.ChildOrgExist	The current organization has a lower organization and is not allowed to delete!		英语

- 不允许跨BO、APP直接操作其它BO、APP的对象，应该调用rest或者sdk。

  不推荐以下写法：

```
@useObject(['SmartCampus__Video_Patrol_Task__CST', 'SmartCampus__Video_Patrol_Task_Camera__CST', 'DE_Devices', 'SPA_Basic'])
//method
ate doConditionSql(input: ParamsInput) {
let queryCondition = input.condition || {};
let sql = 'select a.id as id, a.SmartCampus__start_time__CST as startTime, a.SmartCampus__end_time__CST as endTime, a.SmartCampus__plan_id__CST as plan
        + ' a.SmartCampus__task_name__CST as taskName, a.SmartCampus__actual_play_duration__CST as actualPlayDuration, a.SmartCampus__space_id__CST as
        + ' a.SmartCampus__processor_id__CST as processorId, a.SmartCampus__processor_name__CST as processorName, a.SmartCampus__camera_count__CST as c
        + ' from SmartCampus__Video_Patrol_Task__CST a left join SPA_Basic b on a.SmartCampus__space_id__CST = b.id ';

let countSql = 'select count(1) as totalNum from SmartCampus__Video_Patrol_Task__CST a left join SPA_Basic b on a.SmartCampus__space_id__CST = b.id ';
let paramArray = [];
```

- 对于确实需要查询所有数据的场景，需要考虑平台的查询记录数限制，并自行控制游标查询。

```
    //针对查询数据超过5000条的sql，多次查询实现查询所有数据。要求：入参中的selectSql中不能包含limit
    protected queryAllMatchedDatas(selectSql, param) {
        let isFinished = false;
        let execSql = selectSql;
        let res = [];
        let limit = 0;
        while (!isFinished) {
            let tmpRes = db.sql().exec(execSql, param);
```

2.3 服务编排开发规范

2.3.1 元素命名

本章节介绍服务编排及其变量、图元、连线的命名要求。

服务编排命名

服务编排本身的名称，参照Java定义方法的命名规则：

- 驼峰命名，首字母小写，不要携带下划线。
- 动宾结构，例如，notifyOrderCompletion，错误样例OrderCompletionNotification。

- 尽量不用缩写，除非是专有名词，例如，invokeCRM，错误样例paymentCbk。
- 应简单明了，表示业务意义，而不是内部实现。例如，createPayment，错误样例InsertAndUpdatePayData。

变量命名

在服务编排流程中，除了系统的输入变量和输出变量，在服务编排流程使用的内部变量：

- 输入变量和输出变量：命名遵循接口设计文档的要求。
- 内部变量：元素命名遵循驼峰命名原则。
- 变量为单数时，命名包含对象名称（如Level1Catalog、Level2Catalog），变量为复数时命名包含集合名称（如Level1CatalogList或者Level2CatalogArray）。
- 数组型变量的下标命名要符合规范，当使用I、J、K等字母来命名时，需要明确变量含义，勿重复使用。

图元命名

服务编排图元的名称，采用动名短语形式，每个单词首字母大写，单词之间有空格。名称一般不要超过五个单词，短语尽可能简明，描述该图元的主要作用。例如Valid Input、Call Device Service。

连接线命名

在服务编排的图元中，除了Decision连接线外，其他图元的连接线采用系统的固定命名。

Decision连接线名称使用英文，采用单词首字母大写，一般不要超过三个单词。采用动名短语，尽可能简明、准确的描述该条件的判断逻辑。

2.3.2 图元编排

本章节介绍服务编排中图元布局、摆放的要求。

为了服务编排画布排版美观以及方便后续的服务编排检视，服务编排的图元编排遵循以下原则：

- 在配置服务编排前，需明确具体步骤和子流程的分解，合理编排图元。
- 业务逻辑采用自上而下、从左往右的页面布局方式，业务逻辑展示清晰，同一任务多个步骤，横向排列，不同任务之间竖向排列。利用横向和竖向，做到层次缩进。
- Decision图元里的Default改为表示主流程，类似于If Error Else MainProcess这种结构，把异常处理优先标出。
- 在同一个版面中，图元之间的间隔大小相同。
- 尽量避免连接线相互交叉。
- 创建或查询较复杂的父子对象结构场景（如创建订单和相关对象），应将父对象和典型子对象的创建和查询编排到子流程中，供多个流程共享。
- 对于服务编排中常用到的业务功能（如获取Offer实例），可以考虑编排到子流程或用Script实现，供多个流程共享。

- 编排每个服务编排时，首先都应该设置入参校验步骤，图元类型为“Decision”，分支优先考虑异常场景。服务编排流程中，其他的“Decision”图元也要首先考虑异常场景。
- 避免使用循环套循环。
- 不允许在子流程中结束，所有结束出口应在最外层服务编排中体现。子流程中，应始终返回出参。

2.3.3 参数定义

本章节介绍服务编排中入参、出参的定义要求。

- 服务编排的入参和出参，需要根据设计文档做必填参数校验。
- 出入参数不满足设计文档要求时，需要返回错误提示。
- 出入参数需要定义清楚每个字段。如果存在结构体，结构体中也应定义清楚每个字段。除了本身是预留的定制扩展的结构体之外，不允许有空结构的结构体。
- 服务编排中，暴露给用户由用户输入的参数，必须放到服务编排的“入参”中。服务编排中，不能存在用户能够使用但不在“入参”及“出参”的参数。

2.3.4 异常处理

本章节介绍服务编排逻辑中对异常判断、错误码的定义要求。

处理图元的异常

支持异常处理的服务编排图元包括子服务编排图元、脚本图元、记录创建图元、记录删除图元、记录查询图元、记录删除图元、连接器图元和BO图元。当服务编排中使用这些图元时，对可能引入错误的每个图元都需要做异常处理判断。

步骤1 在定义服务编排时，如图2-8所示，从图元中拖出一条异常处理的连线，连线类型选择“出错时”。

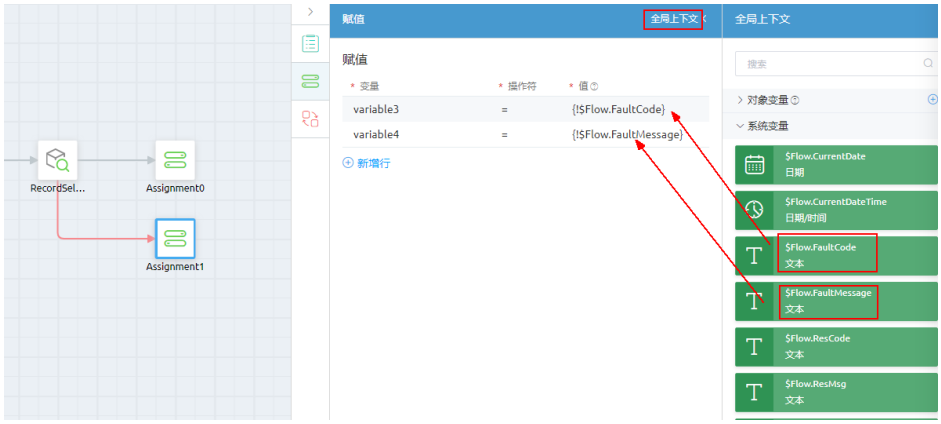
当流程走到对象记录查询图元出现异常时，就会走连线类型为“出错时”的分支，执行图元“Assignment1” 。

图 2-8 处理图元异常

步骤2 （可选）如果需要关注出错后的具体错误码和错误信息，则执行该步骤。

如图2-9所示，配置图元“Assignment1”，通过系统变量 “$Flow.**Fault**Code” 和 “$Flow.**Fault**Message”获取这些图元出错后的错误码和错误信息，这些错误信息是系统返回的。

图 2-9 输出错误信息



----结束

流程结束时返回错误

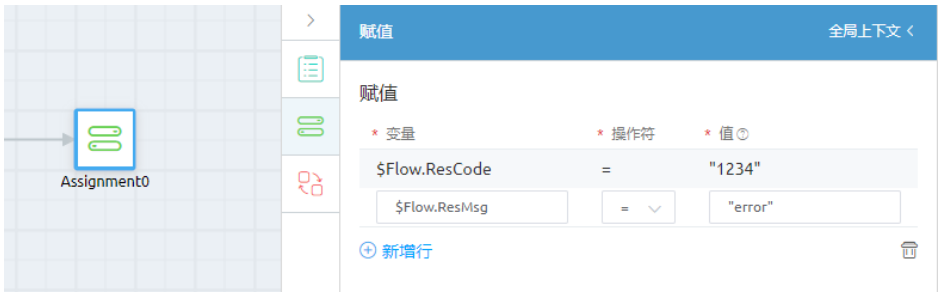
如图2-10所示，将赋值图元作为流程结束的节点，通过系统变量 “$Flow.ResCode” 和 “$Flow.ResMsg”自定义流程出错后的错误码和错误信息。

说明

服务编排中，关于错误码的系统变量有两类，区别如下：

- “$Flow.FaultCode” 和 “$Flow.FaultMessage”为系统返回的错误码和错误信息。
- “$Flow.ResCode” 和 “$Flow.ResMsg”为自定义输出的错误码和错误信息。

图 2-10 流程结束时返回错误



引入国际化错误码

参考自定义错误码，先创建错误码，也可以引入账号下已有的错误码，然后在服务编排中使用错误码。具体方法为在“全局上下文”新建公式变量，使用表达式SETI18NERROR(*ErrorCodeName*,'*变量1*','*变量2*')方式引入错误码。

图 2-11 服务编排里使用错误码

2.3.5 调用约束

本章节介绍服务编排中，对脚本的调用约束。

- 不允许在应用项目的服务编排中，调用BO的内部服务编排。
- 不允许在应用项目的服务编排中，编排调用BO的脚本。
- 不允许跨BO调用脚本和内部服务编排。
  例如，人员BO的服务编排不允许调用空间BO的内部服务编排和脚本。

2.4 前端开发规范

2.4.1 标准页面公共组件样式规范

本章节介绍标准页面前端组件的颜色、图标、文字、按钮、提示框、输入框和选择器等样式规范。

应用开发主要包括标准页面和高级页面两种页面类型：

- 高级页面：由于自定义开发Widget，代码DOM结构不可能保持唯一，因此无法统一样式类，可参考规范自行调整。
- 标准页面：针对多页面重复使用的组件，基线已预置公共组件样式并统一管理。例如，业务侧开发引入主题库ThemeCSS4IOC后，根据需要选择填入相应样式类，即可实现组件效果，如图2-12所示。

图 2-12 引入样式类

BackgroundColor 背景底色

背景底色的样式规范，如图2-13所示。

图 2-13 背景底色

表 2-2 背景底色的样式类

颜色	色值	引用类
主色	#ff8b00	std-bg-color main-01
主色-提示	#3f75fc	std-bg-color main-02
辅助色-严重	#ff0000	std-bg-color assit-error
辅助色-严重	#ff0000	std-bg-color assit-error
辅助色-重要	#ffc600	std-bg-color assit-important
辅助色-一般	#5af0ff	std-bg-color assit-normal
辅助色-提示/不确定	#b8bbcc	std-bg-color assit-prompt
辅助色-成功	#00ffc6	std-bg-color assit-success
中性色-主要文字	#ffffff	std-bg-color neutral-main
中性色-常规文字	#cfd2e5	std-bg-color neutral-normal
中性色-次要文字	#9297b6	std-bg-color neutral-minor
中性色-中性色01	#1b1a33	std-bg-color neutral-bgc01
中性色-中性色02	#1f2033	std-bg-color neutral-bgc02
中性色-中性色03	#4b4d73	std-bg-color neutral-bgc03
中性色-中性色04	#595b80	std-bg-color neutral-bgc04

文字

文字的颜色使用主题色橙色（#FF8B00）和中性色（#FFFFFF、#CFD2E5和#9297B6）。

图 2-14 文字规范

Button 按钮

Button按钮的样式规范，如图2-15所示。

图 2-15 Button 按钮样式规范

表 2-3 Button 按钮的风格样式类

组件风格	引用类
积极操作（完成/确定/保存）	std-positive
辅助操作（查询/上一步/返回）	std-support
消极操作（重置/取消）	std-negative
带图标按钮	std-icon
文字按钮	std-text

Input 输入框/Search 搜索框

Input输入框/Search搜索框的样式规范，如图2-16所示。

图 2-16 Input 输入框/Search 搜索框样式规范

表 2-4 Input 输入框/Search 搜索框的风格样式类

组件风格	引用类
初始态	无
带边框（常用于弹窗中的输入框）	std-border
搜索框01（带底色方框）	std-serach__01
搜索框02	std-serach__02

Select 选择器（单选/级联选择）

Select选择器（单选/级联选择）的样式规范，如图2-17所示。

图 2-17 Select 选择器（单选/级联选择）样式规范

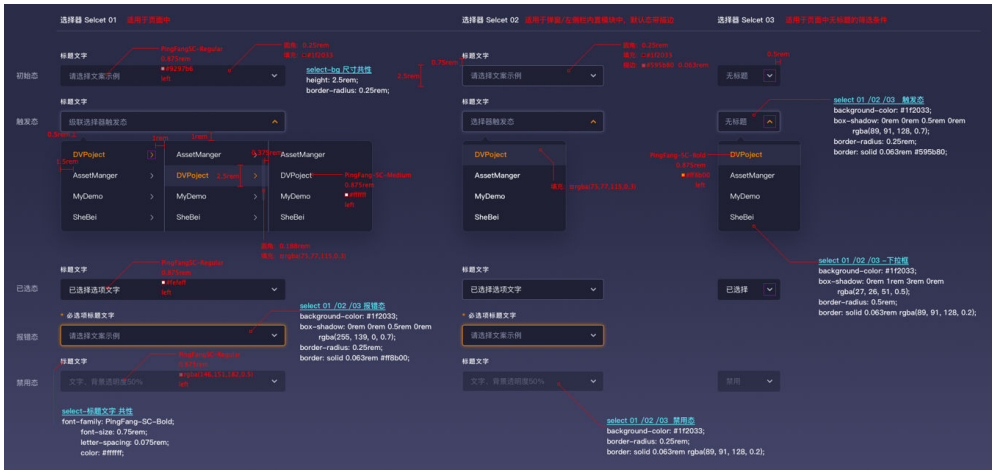


表 2-5 Select 选择器（单选/级联选择）的风格样式类

组件风格	引用类
默认状态	无
带边框（上图选择器Select 02）	std-border

DateTimePicker 日期时间选择器

DateTimePicker日期时间选择器的样式规范，如图2-18所示。

图 2-18 DateTimePicker 日期时间选择器样式规范

表 2-6 DateTimePicker 日期时间选择器的风格样式类

组件风格	引用类
初始状态	无
带有背景颜色	ivu-time-picker
input带边框	time-border

Radio 单选框/ CheckBox 复选框

Radio单选框、CheckBox复选框的样式规范，如图2-19所示。

图 2-19 Radio 单选框 、 CheckBox 复选框样式规范

Slider 滑块

Slider滑块的样式规范，如图2-20所示。

图 2-20 Slider 滑块样式规范

Upload 上传

Upload上传的样式规范，如图2-21所示。

图 2-21 Upload 上传样式规范

Tree 树形控件

Tree树形控件的样式规范，如图2-22所示。

图 2-22 Tree 树形控件样式规范

表 2-7 Tree 树形控件的风格样式类

组件风格	引用类
初始状态	无
带有选中的背景颜色	ivu-tree-view

Pagination 分页

Pagination分页的样式规范，如图2-23所示。

图 2-23 Pagination 分页样式规范

表 2-8 Pagination 分页的风格样式类

组件风格	引用类
非modal普通页面下	stc-pageDivide-simple
modal下的分页组件	stc-pageDivide-modal

Message 弹窗

Message弹窗的样式规范，如图2-24所示。

图 2-24 Message 弹窗样式规范

Tab 标签页

- 标签页样式1
  适用于普通页面的标签页样式，如图2-25所示。

  图 2-25 普通页面的标签页样式规范

- 标签页样式2
  适用于弹窗中的标签页样式，如图2-26所示。

图 2-26 弹窗中的标签页样式规范

- 标签页样式3
  适用于弹窗中或三级的标签页样式，如图2-27所示。

图 2-27 弹窗中或三级的标签页样式规范

表 2-9 Tab 标签页的风格样式类

组件风格	引用类
标签页样式1（适用于普通页面）	dream-tabs-simple-page
标签页样式2（适用于弹窗中）	dream-tabs-simple-modal1
标签页样式3（适用于弹窗中或三级）	dream-tabs-simple-modal2

Step 步骤条（横向/竖向）

Step步骤条（横向/竖向）的样式规范，如图2-28所示。通过该组件的“方向”属性，可控制风格为横向或者竖向。

图 2-28 Step 步骤条（横向/竖向）样式规范

表 2-10 Step 步骤条（横向/竖向）的风格样式类

组件风格	引用类
Step步骤条样式	dream-steps-simple

TimeLine 时间线

TimeLine时间线的样式规范，如图2-29所示。

图 2-29 TimeLine 时间线样式规范

表 2-11 TimeLine 时间线的风格样式类

组件风格	引用类
TimeLine时间线样式	dream-timeline-simple

Table 表格

- 普通页面表格样式，如图2-30所示。

图 2-30 普通页面表格样式规范

- modal下表格样式，如图2-31所示。

图 2-31 modal 下表格样式规范

在表格容器的表格组件上，添加表2-12中的样式类。

表 2-12 modal 下表格的风格样式类

组件风格	引用类
modal下的表格	stc-table-modal

组件风格	引用类
非modal普通页面下(有操作栏)	stc-table-simple
非modal普通页面下(无操作栏)	stc-table-simple-nooperation

弹窗模态框

弹窗模态框的样式规范，如图2-32所示。

图 2-32 弹窗模态框样式规范

Title 标题

Title标题样式参考规范，如图2-33所示。

图 2-33 Title 标题样式规范

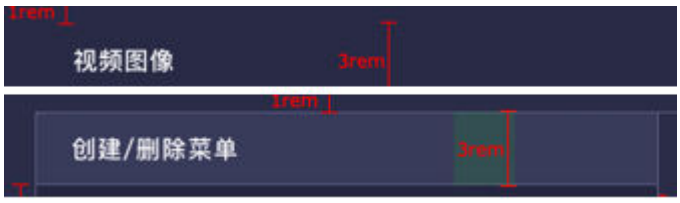


表 2-13 Title 标题的风格样式类

组件风格	引用类
无背景图	无
卡片中带背景标题	std-card-title

Border 边框

Border边框的样式规范，如表2-14所示。

表 2-14 Border 边框的风格样式类

组件风格	引用类
全边框	std-border full
左边框	std-border left
右边框	std-border right
上边框	std-border top
底部边框	std-border bottom
加宽(width:2px)	strong
定制边框颜色(color: #ff8b00)	highlight

BreadCrumb 面包屑

BreadCrumb面包屑的样式规范，如图2-34所示。

图 2-34 BreadCrumb 面包屑样式规范

2.4.2 Widget 开发规范

本章节介绍前端页面的设计原则和Widget的开发规范。

设计原则

高级页面开发模式下，页面是由N个widget构成（N>=1）。Widget可以理解为页面中的小组件，组件的拆分可根据具体的业务和组件的复用情况进行。如果某个页面不可拆分或无复用场景，则可以只有一个Widget构成。

组件存在的最主要的意义是“可在不同页面中被复用”。理论上拆分越细，可被复用的概率就越大。但是带来的副作用是页面开发的复杂度上升，所以拆分的力度需要进行综合的考虑。

Widget 命名

Widget命名采用三段式命名：命名空间_业务名称_widget（固定字段），业务名称采用大驼峰命名。

函数/方法注释

函数/方法注释采用典型的JSDoc的注释方式。

- 函数注释入参说明：@param + 1个空格+ {参数类型} + 1个空格 + 参数名+ 1个空格 + 参数说明。
- 函数注释出参说明：@return + 1个空格 + {参数类型} + 1个空格 + 出参说明（若没有出参，则出参说明可不写）。

有关键逻辑或者重大变更时，增加改动者及改动描述。

```
/**
 * 在地图上创建标志
 *
 * @param {String} latLng 经纬度信息
 * @param {Object} mapInfo 地图描点信息
 * @return {Boolean} 在地图上创建点选标志的结果
 * @version 20180310  modify by xxx shangsan 修改地图上描点的样式
 * @version 20180314  modify by xxx wangwu 在地图上描点增加事件发送，供外部订阅
 */
var placeMarkerAndPanTo=function(latLng, mapInfo) {
        //返回创建点选标志的结果
    return true;
 }
```

方法内注释

方法内使用单行注释，以“//”开头。对代码的注释，应放在其上方或右方（对单条语句的注释）相邻位置。

```
var placeMarkerAndPanTo=function(latLng, mapInfo)
{
  //返回创建点选标志的结果
  return true;
}
```

FTL 注释

- 在一些主要节点标签结束后，加上注释。

```
<div class="success-operate">
    <a href="javascript:;" class="btn-md-red"
        @click="trackOrder">Track Order</a> <!-- btn-md-red end 跳转到订单详情页查看订单-->
    <a href="javascript:;" class="btn-md-primary"
        @click="continueShopping">Continue Shopping</a> <!-- btn-md- primary end 跳转到商城首页继续购物-->
</div>
```

- 在一些循环结束后，加上注释。

```
<li class="li-tab" v-for="(item,index) in g_viewOrderData.tabsParam">
    <div class="item-wrap">
      <div class="img-wrap">
            <img :src="item.url"></img>
      </div>
     <div class="wid-tab-text">{{item.value}}</div>
  </div>
</li><!--loop end 遍历订单上的状态页签，获取名称和链接-->
```

样式文件

- 样式文件中，图片及字体文件的引用，需使用相对路径。
- 字体文件需要放置在与样式文件同级的目录中，目录名称为font、fonts（两个任选其一即可），可设置子层目录，子层目录命名无要求。
- 目录及图片、文件的命名仅支持数字、字母（大小写均可）、下划线和中划线。
- 支持的图片格式，包含png、jpg、jpeg、gif、bmp和webp。
- 支持的字体文件格式，包含tiff、woff、ttf、otf、eot和svg。
- 多个Widget样式可以抽取为单独库文件，独立库文件可以方便进行整体样式主题的切换。缺点是Widget的运行，需要对这个库文件产生依赖。

  如果Widget比较独立，也可以将样式文件定义到Widget作用域的css中。

响应式设计

Widget需要支持响应式设计，页面需要在多屏下，进行充分测试验证。

浏览器兼容性

组件开发需要考虑浏览器的兼容性，对多浏览进行适配验证。

桥接器 Bridge

- Widget数据访问需要封装到Bridge中，不允许在Widget中，直接通过ajax访问业务数据服务。

  通过Bridge的封装，可以保持Widget的稳定性。未来数据源的切换不需要升级Widget，只需要切换Bridge即可。
- 对于数据消费类的Bridge，mock目录不允许为空。

  Mock数据会在编辑状态加载，如果直接调用API，可能会因为API的不稳定而引起UI渲染异常。
- 一个查询API一个Bridge，更新API当前可以共用同一个Bridge。
- 前端对象需要在Bridge中，进行明确定义，字段清晰。

开放性

- Widget自身是开放的，直接引用业界优秀的开源组件可以提高开发效率。

  例如，使用MVVM框架Vue（Vue相比AngularJS更轻量，性能更优）以及使用基于Vue的前端控件库Quasar，Element-UI。
- 不允许引入JQuery，因为框架已默认提供，引入后会引起运行冲突异常。
- 非特殊场景，组件的尺寸大小不建议写成固定值，可以指定最大/最小值。
- 非特殊场景，不允许直接对DOM元素添加CSS。DOM元素的样式通过显式的class引用，保证样式只作用在当前组件。

2.5 错误码定义规范

错误码使用原则

- 原则上，系统里面所有的报错或提示信息，都禁止直接在代码里面写成固定的提示语，必须使用错误码的形式。

- 错误码的命名，必须遵循错误码命名规范。
- 错误码要配置在APP或BO内。

错误码命名规范

- BO名称/APP名称+“.”+错误码名称。
- 错误码名称采用大驼峰的方式，通常采用简洁的英文单词，不要使用汉语拼音。
- 状态码不能返回4xx，否则页面会跳转到登录页。
- 错误码的中英文错误信息都要定义。
- 错误参数信息不能使用 \ ，应该使用{0}。

错误码定义范例

所有的错误码，都需要在AstroZero上查看。

表 2-15 错误码定义范例

APP	错误码名称	英文错误信息	状态码	中文错误信息
AlarmBO	Alarm.ServiceUnAvailable	The request has failed due to a temporary failure of the server.	500	服务不可用。
	Alarm.InvalidOperator	The input operator is invalid.	500	非法操作符类型。
	Alarm.illegalParameter	The input parameter is illegal.	500	非法参数类型。

2.6 高性能编码规范

可以通过优化表结构、业务代码逻辑等提高性能。

- 减少复杂接口设计
  - 单个服务不要实现太多的功能，服务越复杂可重用性越差。
  - 单个服务不要返回太多的数据，包括记录数以及关联数据。
- 表结构优化
  - 常用的查询条件字段创建索引，创建索引时区分度大的字段放在前面。
  - 不要对所有字段建立索引，太多的索引会导致增删改变慢，且维护成本升高。
- 业务逻辑优化
  - 多表关联查询：不要使用超过两个表的关联查询。

    错误示例：

```
function updateAttrValue(externalCode, deviceCode, channelCode, attrValue) {
    let sql = db.sql();
    let selectPreSql = ""select b.id as id, a.id as deviceid, b.AttrDef as attrDef, c.name as code""
    let fromSql = "" from DE_Devices a, DE_DeviceAttr b, DE_AttrDef c, DE_ExternalChannel d ""
    let whereSql = ""where a.id=b.Device and b.AttrDef=c.id and b.ExternalCode=? and a.ExternalCode=? and a.Chann
    let selectAllSql = selectPreSql + fromSql + whereSql;
    console.log(selectAllSql);
```


- 重复查询：不要在循环内重复使用同一条件查询，应该在循环外处理。不要在同一个脚本的多个方法内使用同一条件多次查询，可以定义类的成员变量。

  错误示例：

```
if (params && typeof (params) == 'object') {
    for (let key in params) {
        this.updateAttrValue(key, this.getDeviceData(input.deviceCode), params[key]);
    }
}
```

- 非必要的关联查询：关联的条件很多情况下都是唯一的，可以提前做单独查询。

  错误示例（如下两个关联查询，均是为了获取Device ID）：

```
let selectPreSql = ""select b.id as id""
let fromSql = "" from DE_Devices a, DE_DeviceAttr b, DE_ExternalChannel c ""
let whereSql = ""where a.id=b.Device and b.ExternalCode=? and a.ExternalCode=? and a.Channel=c.id and c.name=?"";
let selectAllSql = selectPreSql + fromSql + whereSql;
console.log(selectAllSql);
let records = sql.exec(selectAllSql, { params: [externalCode, deviceCode, channelCode] });
console.log(""records"", records)

if (records.length && records.length > 0) {
    let count = db.object('DE_DeviceAttr').update(records[0]['id'], { AttrValue: attrValue });
} else {
    console.info(""not find external code "" + externalCode + "" in DE_DeviceAttr"");
    //查询定义级外部编码
    let attriSql = ""select b.id as device, a.AttrDef as attrDef from DE_DeviceDefAttri a, DE_Devices b, DE_ExternalChannel c where a.DeviceDef=b.DeviceDef and a.ExternalCode=?""
        + "" and b.ExternalCode=? and b.Channel=c.id and c.name=?"";
    console.log(""attriSql"", attriSql);
```

- 不要在循环内，每条记录都去查询一次数据库，频繁的数据库交互，会严重影响性能。

  例如，查询5000条数据，查询补充数据时，不要在循环内多次交互数据库，把可以合并的条件在循环外拼接并进行一次性查询，在循环内只需要从结果集中获取数据，可以极大提升查询性能。

  错误示例：

```
//4、补充其它数据
let deviceDefSql = "select id as id,name as code,DeviceName as
let deviceProdSql = "select id as id,name as code,ProductName
for (let device of devices) {
    // 设备规格定义
    if (device["deviceDef"]) {…
    }
    else {
        device["deviceDef"] = {};
    }

    //设备实例所属的设备产品
    if (device["deviceProduct"]) {…
    }
    else {
        device["deviceProduct"] = null;
    }
```

- 利用对象做临时缓存

  例如，查询到DeviceDef后，按照id:Object的方式存起来，后续查询时先判断缓存对象中是否已存在，如果存在则直接获取不再查询。

```
203     //规格定义属性数据
204     private deviceDefAttrMap = {};
205
374         if (deviceDefIds.length > 0) {
375             deviceDefAttrSql = deviceDefAttrSql.substr(0, deviceDefAttrSql.length - 1);
376             deviceDefAttrSql += ") ";
377             let deviceDefAttrs = this.queryAllMatchedDatas(deviceDefAttrSql, { params: deviceDefIds })
378             for (let defAttr of (deviceDefAttrs || [])) {
379                 if (this.deviceDefAttrMap[defAttr['deviceDef']]) {
380                     this.deviceDefAttrMap[defAttr['deviceDef']][defAttr['attrDef']] = defAttr;
381                 }
382                 else {
383                     let tmp = {};
384                     tmp[defAttr['attrDef']] = defAttr;
385                     this.deviceDefAttrMap[defAttr['deviceDef']] = tmp;
```

- SQL拼接应该尽量避免可能导致使用不到索引的情况。

  如下操作，可能会导致索引无效：

  - 在索引字段上，使用like进行查询匹配。

    错误示例：

    ```
    explain for select name from de_devices where name like 'bbb';
    ```

  - 使用or语句做SQL拼接。

    错误示例：

    ```
    explain for select name from de_devices where name = 'bbb' or id =
    '050C000000SDVhsWIpsn';
    ```

  - 多个字段建立组合索引，但仅使用了部分字段作为查询条件，索引失效。

    错误示例：

    ```
    explain for select name from de_devices where externalcode = 'bbb';
    ```

  - 数据类型出现隐式转化，例如数字转型为文本，使索引无效，产生全表扫描。

    错误示例：

    ```
    explain for select name from de_devices where name  = 123;
    ```

  - 在索引列上，使用IS NULL或IS NOT NULL操作。索引是不索引空值的，所以这样的操作不能使用索引，可以用其他的办法处理。

    错误示例：

    ```
    explain for select name from de_devices where name is not null;
    ```

    数字类型，判断大于0，字符串类型设置一个默认值，判断是否等于默认值即可。

  - 在索引字段上，使用“<>”，它的处理只会产生全表扫描。

    优化方法： key<>0改为key>0 or key<0

    错误示例：

    ```
    explain for select name from de_devices where name <> 'bb';
    ```

  - 对索引字段进行计算操作、字段上使用函数。

    错误示例：

    ```
    explain for select name from de_devices where lower(name) =  'bbb';
    ```

2.7 系统参数命名规范

当前支持在AstroZero管理中心、BO和APP界面，创建系统参数。

BO 系统参数

BO中的系统参数，统一命名规则为： BO名称 + 下划线 + 系统参数名称。

```
<BOName>_<SystemParameterName>
```

系统参数名采用首字母大写的驼峰样式，例如SampleBO_DefaultValue 、AlarmBO_SecurityLevel 和GISBO_MapServerHost。

创建系统参数时，勾选“使用命名空间”这个选项，确保跨租户全局唯一。另外，系统参数名需要尽可能的表明参数的业务含义，避免与其它参数混淆或者完全看不出业务含义。

图 2-35 创建 BO 系统参数

APP 系统参数

APP中的系统参数，统一命名规则为： APP名称 + 下划线 + 系统参数名称。

```
<APPName>_<SystemParameterName>
```

系统参数名采用首字母大写的驼峰样式，例如SampleMgmt_MaxValue 、SecurityMgmt_DefaultSecurityLevel 和IOCMgmt_DataSourceType。

创建系统参数时，勾选“使用命名空间”这个选项，确保跨租户全局唯一。另外，系统参数名需要尽可能的表明参数的业务含义，避免与其它参数混淆或者完全看不出业务含义。

管理中心系统参数

原则上，所有的系统参数都应该放到各BO及APP中，但是如果确实有些系统参数是共性的，也可以在管理中心创建。

管理中心创建的系统参数，统一命名规则为 ： Common + 下划线 + 系统参数名称。

```
Common_<SystemParameterName>
```

系统参数名采用首字母大写的驼峰样式，例如Common_CurrentEnvDomain 、Common_ClientTimezone。

创建系统参数时，勾选“使用命名空间”这个选项，确保跨租户全局唯一。另外，系统参数名需要尽可能的表明参数的业务含义，避免与其它参数混淆或者完全看不出业务含义。

2.8 业务权限配置规范

在创建FUNCTION类型的权限资源时，只能选择已有的业务权限，而不是根据名称自动创建业务权限。

AstroZero的业务权限，仅能在定义接口时由开发者在AstroZero管理中心（用户管理 > 业务权限凭证）创建。权限BO仅关联已有业务权限，不包含自动创建、删除平台业务权限的逻辑。

配置规范

业务权限配置的粒度，应符合实际业务使用场景，使实际业务中既能灵活配置权限，又不产生越权的问题。

如果一个权限中需要调用多个接口，其权限资源应配置上多个接口对应的业务权限（即创建多个FUNCTION类型的权限资源），然后分别给这些接口配置业务权限。

通常一个接口对应一个业务权限，而不是一个业务权限（一把钥匙）可以调通（打开）多个自定义接口API（N个锁），即尽量不要多个接口配置同一个业务权限（特殊场景除外，例如一个接口既用于普通业务又作为机机接口），要按实际业务场景配置。具体配置原则如下：

- 区分使用者权限

  例如，SystemManagement_Common分配给匿名用户，SystemManagement_Operator_Update分配给系统管理员。
- 区分读写权限

  例如，SystemManagement_Operator_QueryAll只读，SystemManagement_Operator_Create可写。
- 区分单个/批量权限

  例如，SystemManagement_Operator_Delete单个删除，SystemManagement_Operator_BatchDelete批量删除。
- 区分同一操作结果不同权限

  例如，SystemManagement_Operator_QueryAll查询全部信息，SystemManagement_Operator_QueryBasic查询基本信息。
- 区分可操作对象不同权限

  例如，SystemManagement_Operator_QueryBasic查询返回多个结果，SystemManagement_Operator_QueryCurrent查询仅返回当前对象结果。

业务权限命名规则

- 模块名称（APP、BO名称）[必选]_特性[必选]_操作[非必选]，大驼峰格式。
  例如，SystemManagement_Common、SystemManagement_Operator_Create。
- 业务权限分类（目录）原则。
  - 以APP、BO为模块进行分类，命名与APP、BO一致。
  - 目录必选。
  - 定义业务权限时，目录名称可直接输入。

3 多人协作开发

3.1 经典版设计器

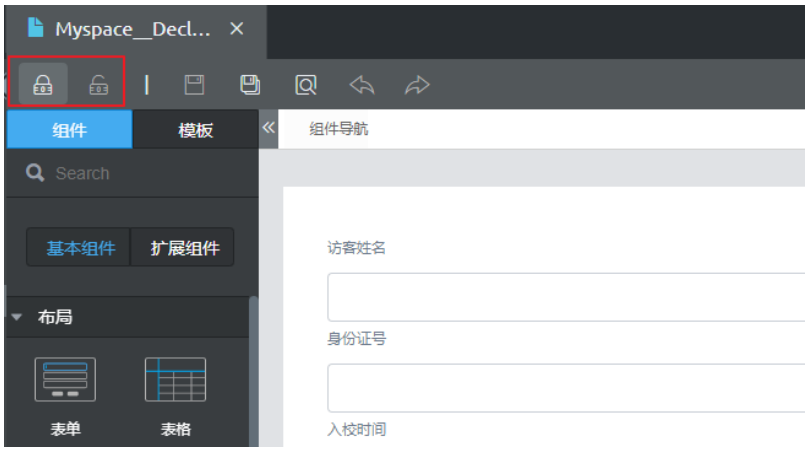
如何实现多人协作开发

通过在账号下，创建用户（子账号），可实现多人协作开发。

为了方便同一个开发团队成员之间，能够更好的配合开发项目。在AstroZero中，同一个账号下的所有子账号开发的内容都可以互相查看，配合开发。

多人登录后，在开发过程中是用锁的机制进行隔离的。每个人开发时，获取锁，在完成开发后释放锁，否则别的账号会无法操作该页面。

图 3-1 获取锁及释放锁



须知

当前AstroZero未提供版本回退的相关能力，所以在做删除资源操作时，需确定该资源无人使用。

如何添加子账号

AstroZero提供了免费版、标准版、专业版和专享版四种规格套餐。不同的规格套餐，添加的子账号个数也不同，详情请参见**产品规格差异**。在AstroZero中如何添加子账号，请参见**如何添加开发者账号（IAM账号）**和**如何为WeLink用户添加开发者权限**。

3.2 新版设计器

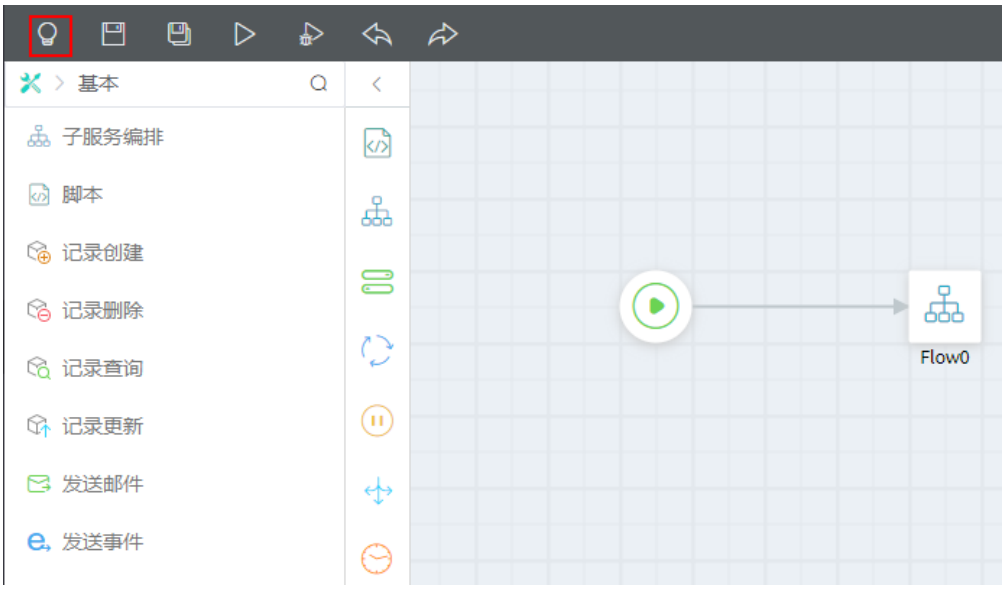
如何实现多人协作开发

通过在账号下，创建用户（子账号），可实现多人协作开发。为了方便同一个开发团队成员之间，能够更好的配合开发项目。在AstroZero中，同一个账号下的所有子账号开发的内容都可以互相查看，配合开发。多人登录后，在开发过程中是用锁的机制进行隔离的。

以服务编排为例，当多人编辑已有服务编排时，为防止多人篡改，低代码平台提供了一套保护机制（上锁机制），即同一时间只有一个用户能编辑元素。假设，A用户打开X服务编排，并进行编辑，此时B用户也打开X服务编排，B用户的服务编排编辑器将自动进入锁定模式。

- 保持A用户对X流程的编辑设计权限。A用户编辑完成后，关闭元素的编辑界面时，将自动解锁或单击启用  按钮，进行自动解锁。

  图 3-2 单击启动按钮

  说明

  关闭元素的编辑界面包括多种场景，如退出元素的编辑界面、退出应用设计器和关闭应用设计器浏览器页面等。

- 当A用户锁定X流程后，系统会只读模式锁定B用户对X流程的操作，B用户的服务编排编辑器页面会有提示“该*资源*已经由*A用户*锁定，点击此处强制获得编辑权”。B用户如果按照提示强制获得编辑权，A用户的X流程编辑器画面会自动锁定。

如何手动释放锁

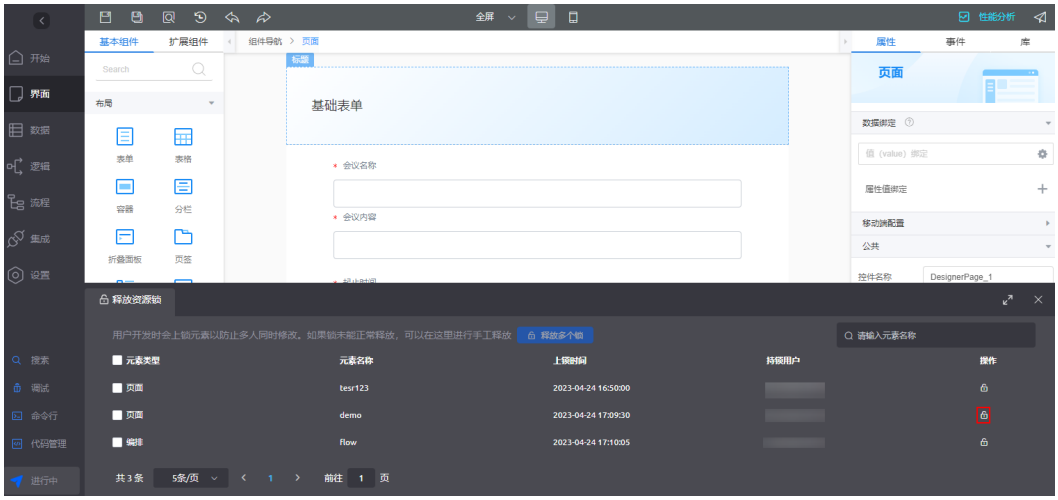
如果元素被上锁，但开发者暂时无法联系到上锁人，又不愿意等待定时释放，开发者可以通过界面强行释放锁。

步骤1 参考**如何登录新版应用设计器**中操作，登录新版应用设计器。

步骤2 在左侧导航栏中，选择“代码管理 > 释放资源锁”。

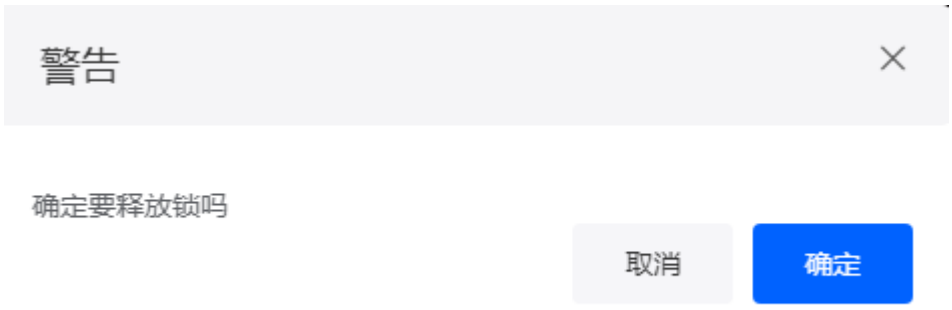
步骤3 在上锁元素列表中，选择需要手工释放锁的元素，单击该元素后的  。

图 3-3 强制获取锁



步骤4 在弹出的警告页面，单击“确定”，释放锁。

图 3-4 确认是否释放锁



----结束

如何添加子账号

AstroZero提供了免费版、标准版、专业版和专享版四种规格套餐。不同的规格套餐，添加的子账号个数也不同，详情请参见**产品规格差异**。在AstroZero中如何添加子账号，请参见**如何添加开发者账号（IAM账号）**和**如何为WeLink用户添加开发者权限**。

4 专享版

监控远程调用

AstroZero会记录您通过自定义连接器、服务脚本等方式访问远程接口的调用日志(含通过VPN方式访问内网)，强烈建议您开通云日志服务LTS并授权华为运维人员把上述日志自动上传LTS以进行访问性能、频率等方面的统计和告警，以便可以及时发现和定位问题。

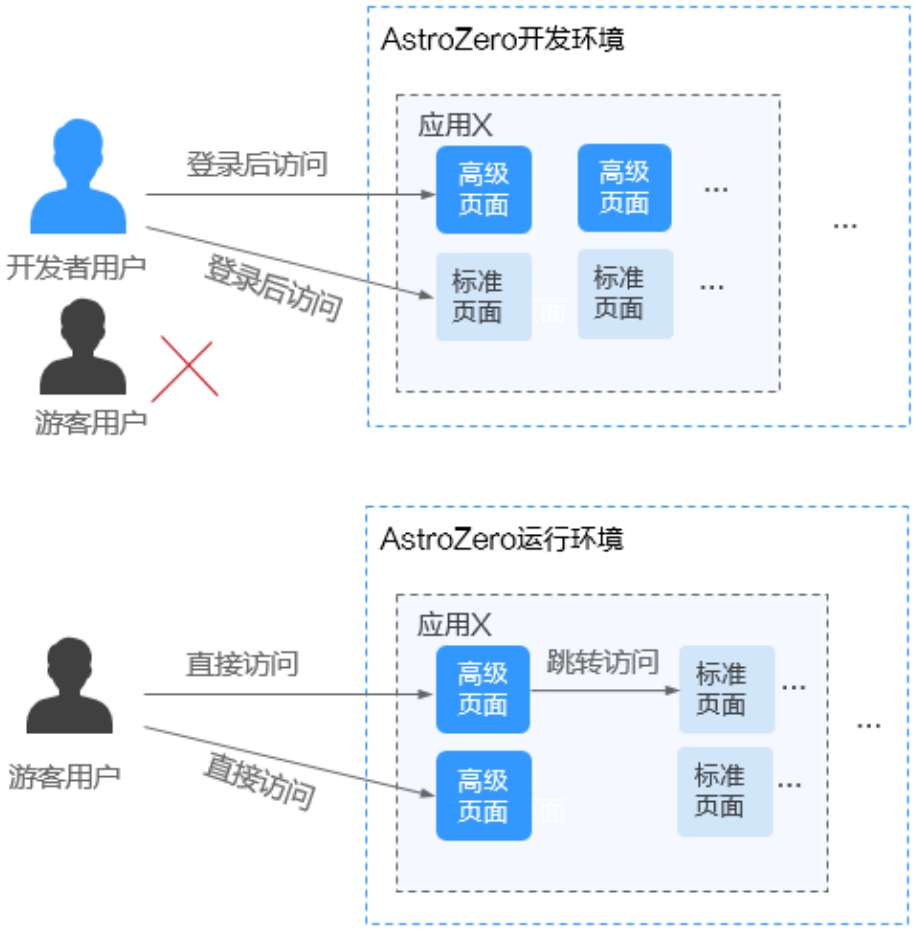
预留逃生通道

- 当您需要在专享版中使用VPN打通线上线下网络时，强烈建议您采用双链路模式进行主备容灾，避免单条链路发送网络故障导致业务不可用。
- 对于依赖远程服务的重要功能，比如登录认证依赖通过VPN通道访问内网接口，建议您增加对接口调用的容错处理，及时检测发现和向用户反馈网络故障等错误，并同时增加不依赖远程服务但可能能力降级的云上实现作为逃生手段。

5 如何在运行环境中，实现游客访问标准页面

在AstroZero开发环境中，应用开发完成后，**仅开发者账号**可预览高级页面和标准页面。当应用发布到**运行环境**后，游客可直接访问高级页面；标准页面本身属于管理后台的表单类页面，**AstroZero不支持游客直接访问和预览**标准页面。

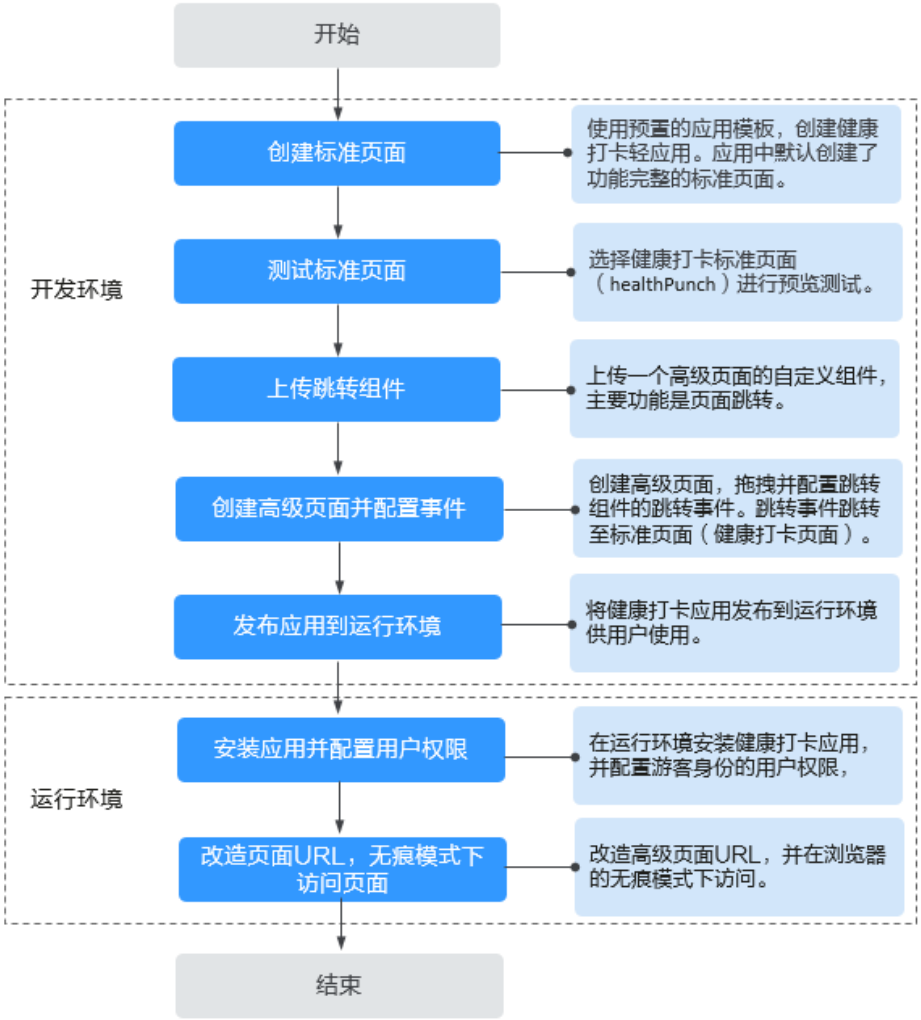
在运行环境中，若游客需要提交一个表单，该如何实现？本节将为您详细介绍实现方法。



实现流程

在开发环境中，用标准页面实现提交表单，在高级页面中引入自定义的跳转组件，该组件用于自动跳转到标准页面。发布到运行环境，在运行环境中，当游客访问该高级页面时，会自动跳转访问标准页面。

图 5-1 实现流程



操作步骤

步骤1 使用应用模板快速创建一个应用及标准页面。

说明

使用应用模板创建的应用中，默认包含有完整功能的标准页面。示例步骤中不单独创建标准页面，仅使用应用模板中的标准页面。该标准页面对应您实际开发过程中需要提供给游客访问的标准页面。

1. 登录AstroZero开发环境，在首页“项目”下，单击“轻应用”。

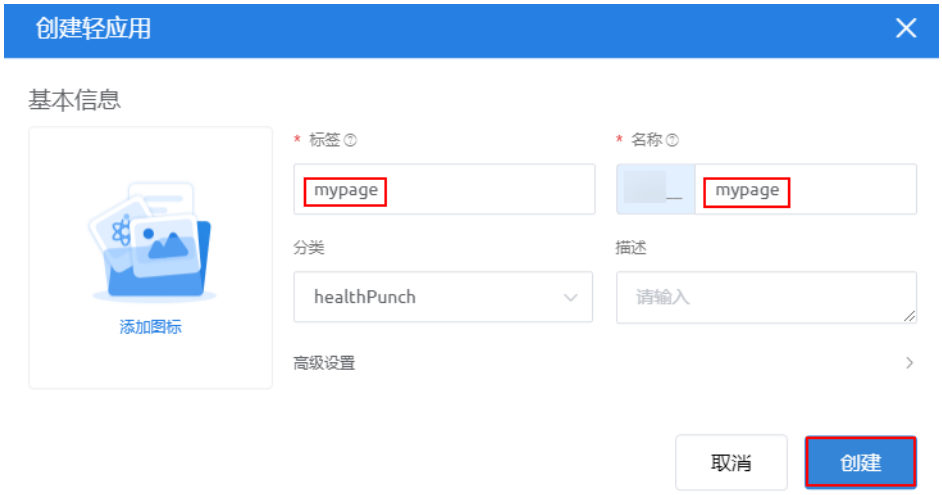
图 5-2 单击轻应用

2. 鼠标放在“健康打卡”应用模板上，会出现“查看模板”和“使用模板”图标，单击“使用模板”。

图 5-3 健康打卡

3. 在弹窗中输入应用标签及名称“mypage”，单击“创建”。

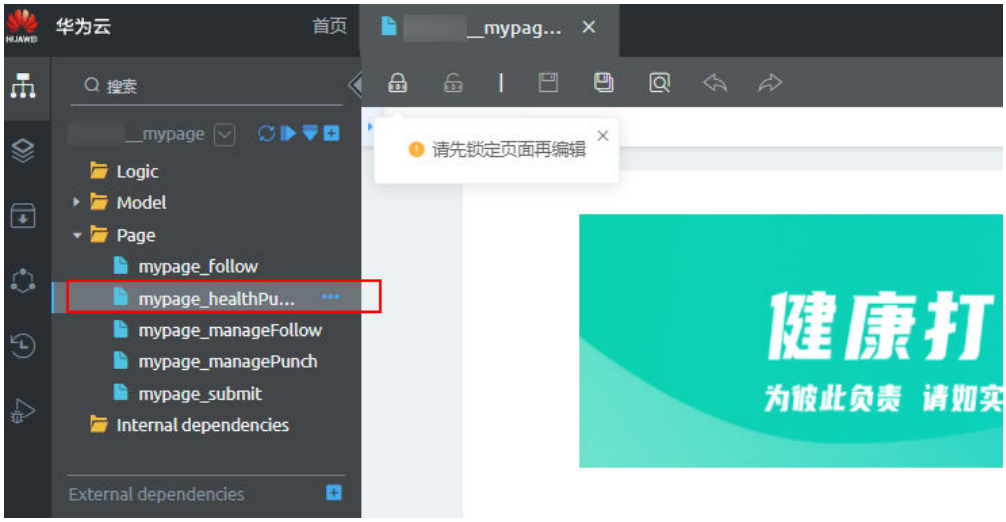
图 5-4 设置应用标签及名称



步骤2 测试标准页面功能。

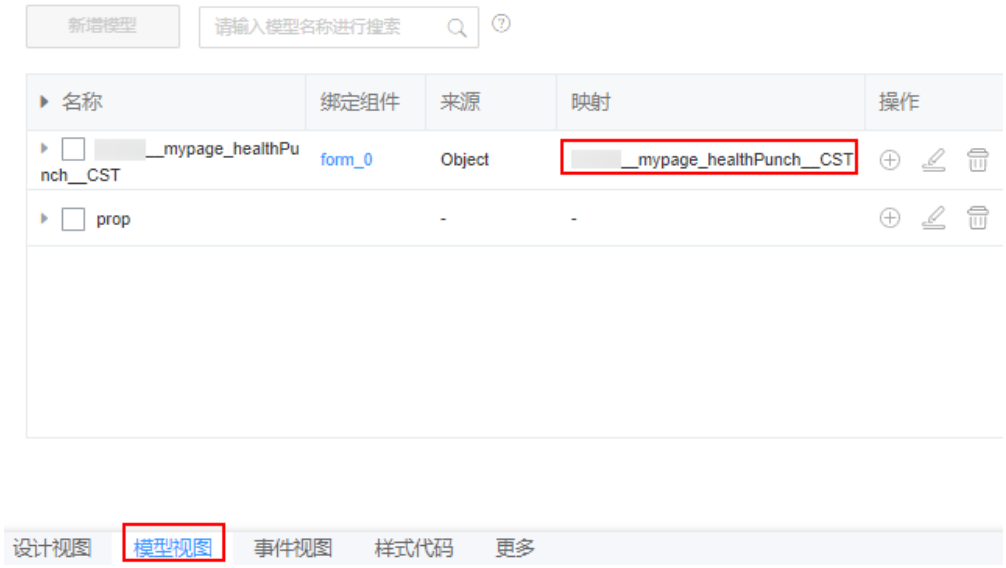
1. 在“Page”目录中，单击“mypage_healthPunch”，查看此打卡页面。
   其中，“mypage”为“应用名”，请根据实际情况查看。

图 5-5 查看打卡页面



2. 在页面左下方单击“模型视图”，查看页面中使用到的模型。
   图中“hwtest”为租户的命名空间，请根据实际情况查看。

图 5-6 查看页面中使用到的模型



3. 在页面上方单击，预览页面，并在页面中输入打卡相关信息后，单击“提交”，测试页面功能是否正常。

图 5-7 测试页面功能

步骤3 上传一个跳转组件。

1. 单击**下载链接**获取跳转组件包“Entrance4Std.zip”，在应用开发工作台页面单击 按钮，选择“高级页面 > 组件”，单击“提交新组件”，提交该组件到组件库中。

图 5-8 进入组件提交页面

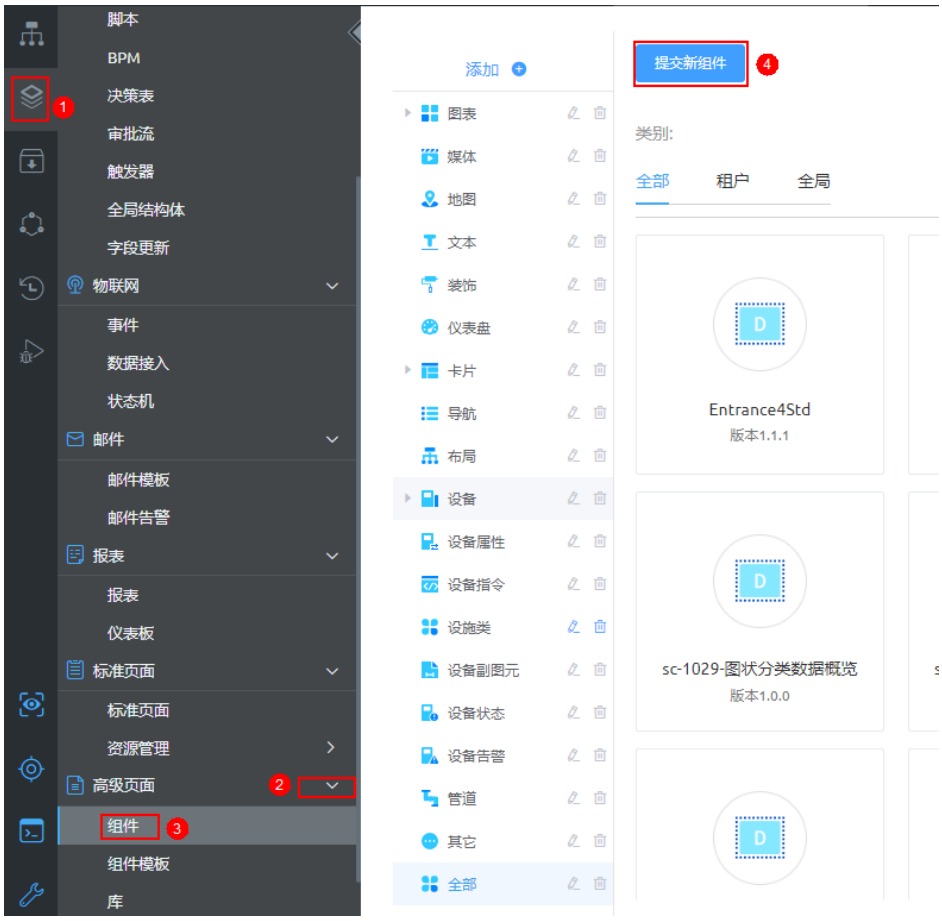
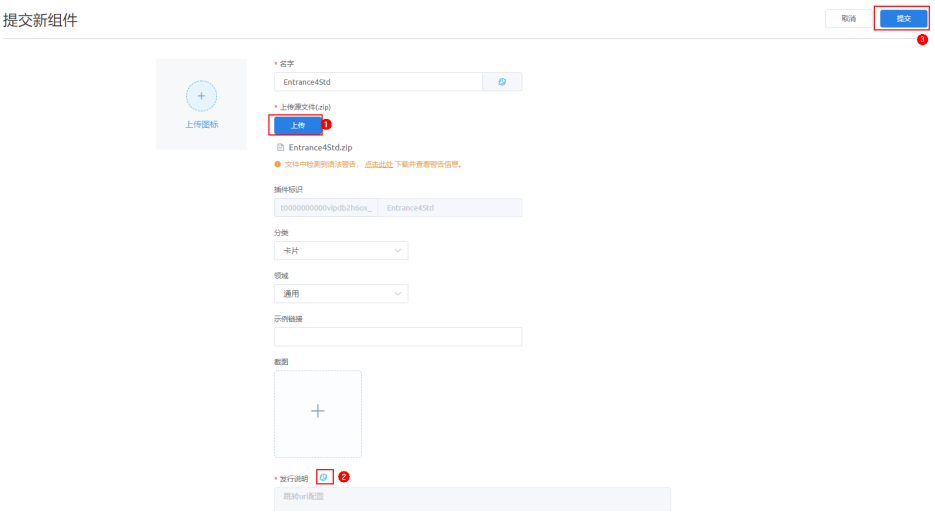


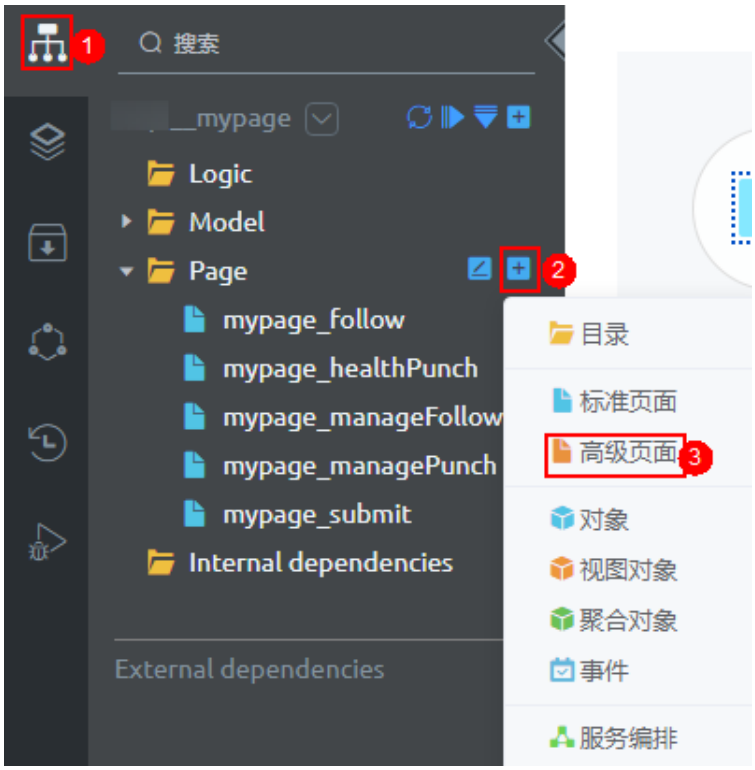
图 5-9 上传提交界面



步骤4 创建一个高级页面并配置标准页面的跳转事件。

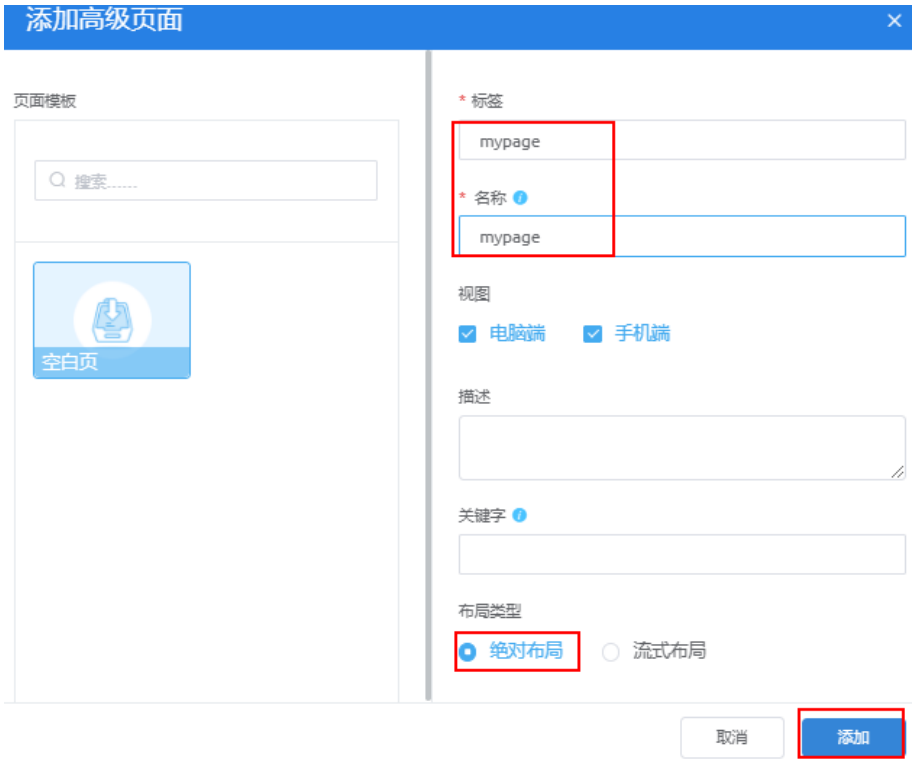
1. 鼠标放在“Page”下，单击界面上出现的“+”，在弹出菜单中选择“高级页面”。

图 5-10 选择高级页面



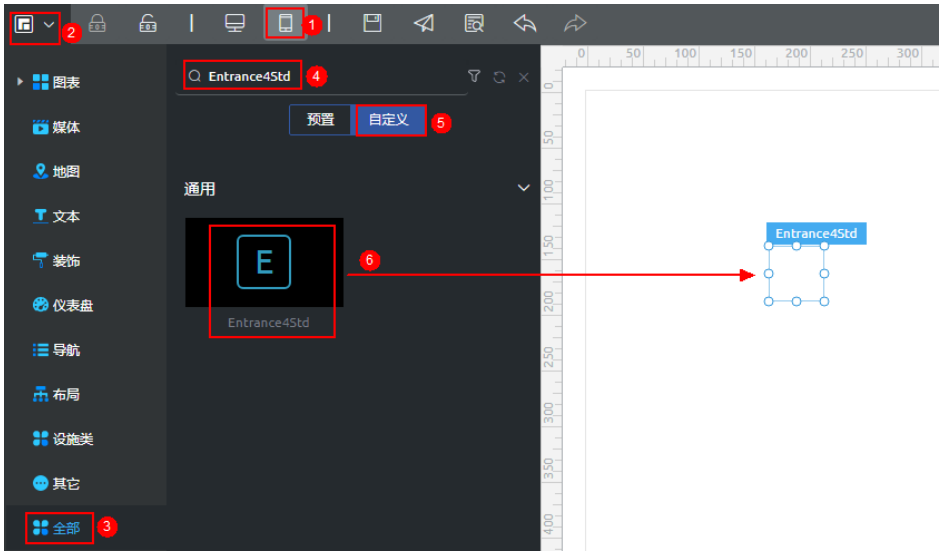
2. 页面模板选择“空白页”，设置“标签”和“名称”为“mypage”，并选择“绝对布局”，单击“添加”。

图 5-11 添加高级页面



3. 单击 “mypage”高级页面，选择“手机”视图，单击图标区域最左边“组件列表” 图标，搜索自定义组件Entrance4Std，拖拽至画布。

图 5-12 拖拽组件 Entrance4Std 到画布



4. 设置组件Entrance4Std的goToPageX事件。

   在“事件”页签下，单击“goToPageX”后的齿轮图标，在弹窗中，单击“新建动作”，再单击“自定义 > 自定义动作”，在事件编辑下，输入如下代码，单击“确定”，再单击“确定”。

```
location.href="/besBaas/baas/abc/foundation/index.html#/hwtest__mypage_healthPunch";
```


说明

- 脚本中加粗的“hwtest”为当前租户的命名空间，“mypage”为标准页面所在的应用名，请根据实际情况替换。
- 若想跳转到其他标准页面，请替换“hwtest__mypage_healthPunch”为实际要跳转的标准页面名称。

图 5-13 设置组件 Entrance4Std 的事件

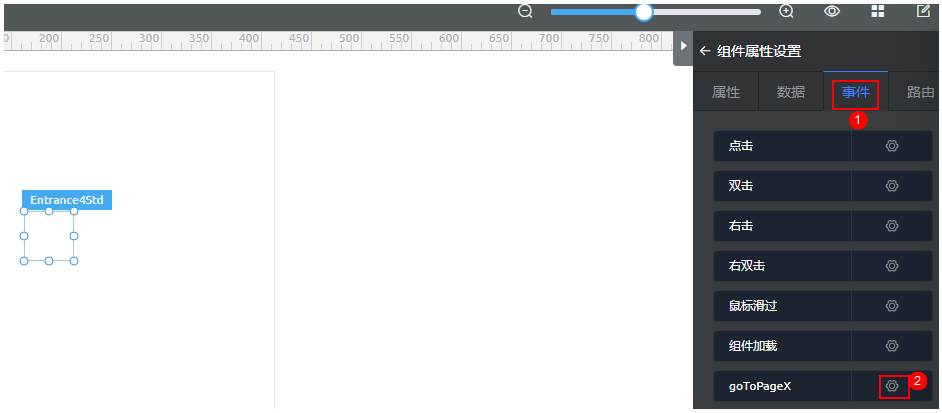


图 5-14 设置自定义动作

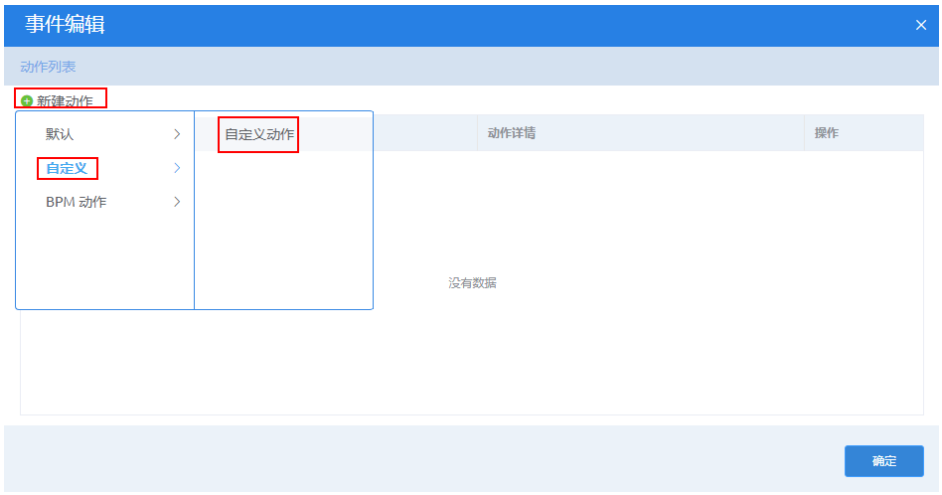
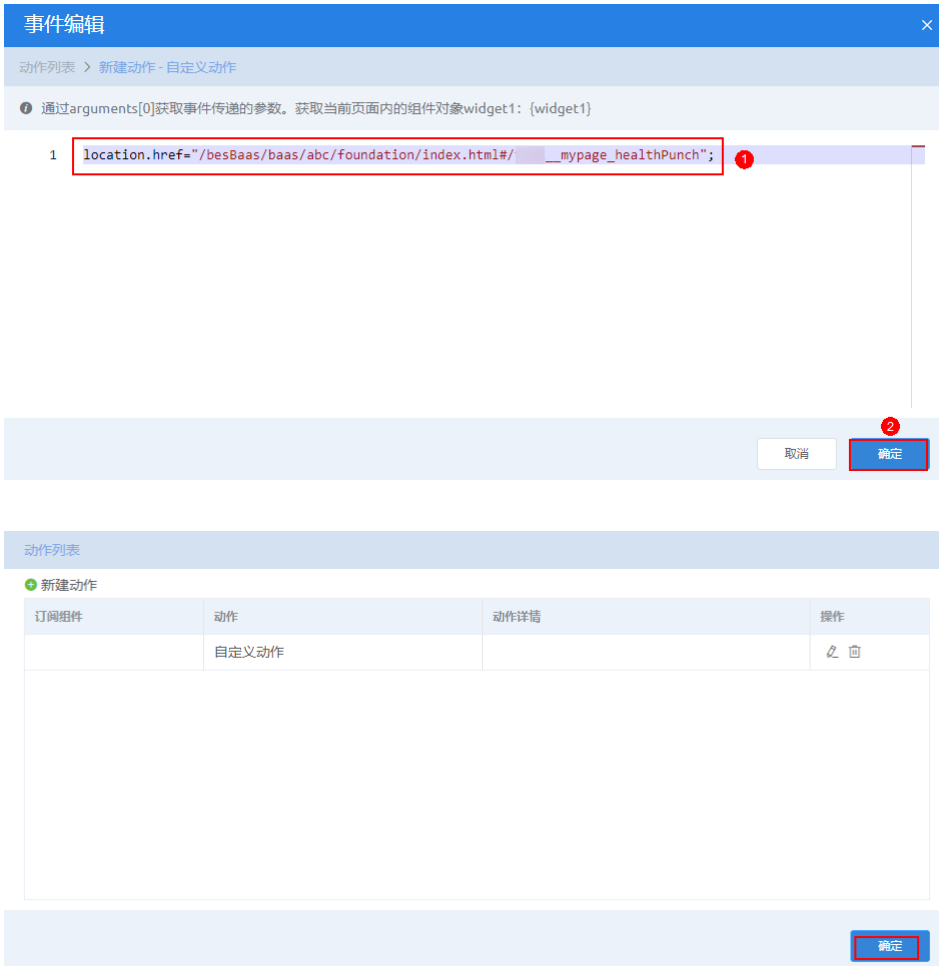


图 5-15 自定义动作



5. 单击页面上方图标，单击发布页面，单击进行页面预览。
6. 获取页面网址。

   单击，即可生成页面网址，用鼠标复制记录此地址。后续在运行环境中，需要改造此地址，改造后是游客访问的地址。

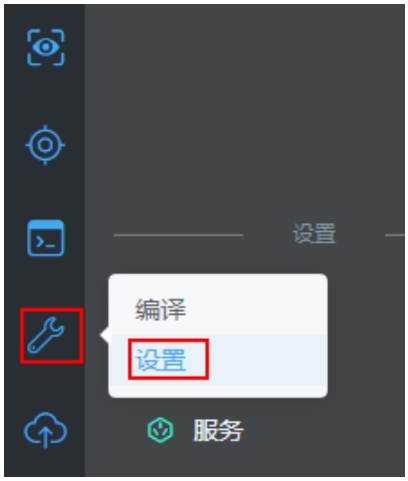
图 5-16 发布

> **⚠ 注意**
>
> 页面网址是弹出页面上“页面成功发布，网址是”后边的内容，不是浏览器的链接地址，格式如：“https://*AstroZero域名*/magno/render/hwtest__mypage_*0000000000XXXXXXX*/view-mobile/mypage”。

步骤5 编译发布应用到运行环境。

1. 单击左下角  ，选择“设置”，在“编译设置”页面，确认包类型已选择“资产包”，并单击“保存”。

   图 5-17 编译设置入口

2. 单击  ，选择“编译”进行编译。
3. 编译完成后，单击左下角  ，选择“我的仓库”，将应用程序安装包发布到当前租户的私仓。
4. 填写版本信息，单击“发布”。

   发布成功后，页面显示“程序包已经被成功上传到我的仓库。”。

步骤6 在运行环境中安装应用，并配置标准页面中使用到的对象、接口、脚本等权限。

1. 使用当前租户开发者账号访问并登录AstroZero运行环境，或者直接在右上角单击账号名，在下拉菜单中选择“运行环境”。

2. 在运行环境首页，单击“我的仓库”，在我的仓库找到**步骤5**发布的“mypage”应用，单击“安装”。

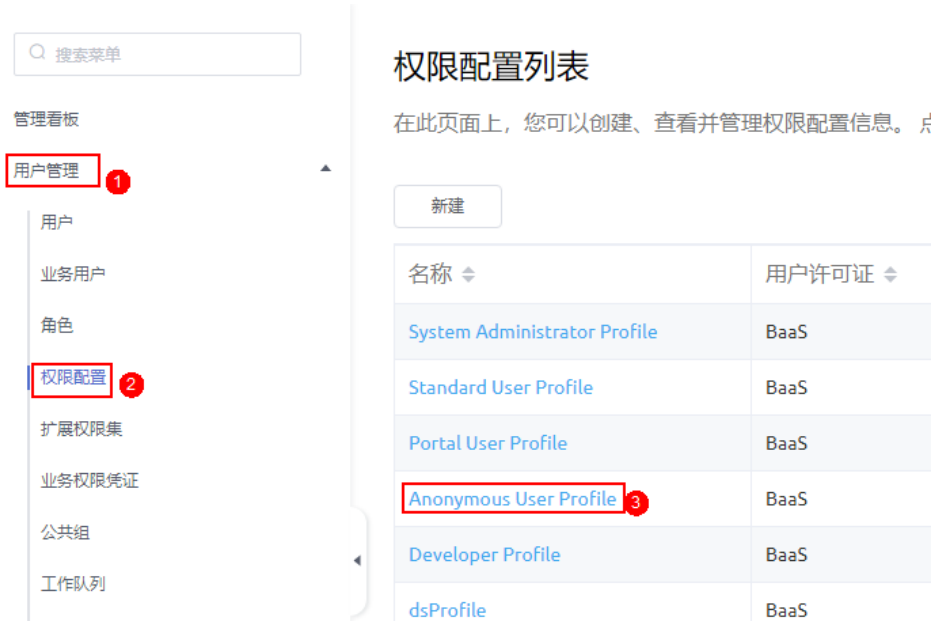
说明

如果“我的仓库”下没有待安装的“mypage”应用，您需要参考**步骤5**中的操作方式，编译一个“资产包”应用，并选择发布到“我的仓库”。

图 5-18 在我的仓库安装应用

3. 配置匿名访问权限。
   a. 在运行环境管理中心选择“用户管理 > 权限配置”，在权限配置列表单击“Anonymous User Profile”。
      该权限为匿名用户访问AstroZero时用的权限，需要将用户使用到的资源进行匿名访问授权。

图 5-19 单击 Anonymous User Profile



b. 选择“自定义对象权限”页签，搜索“mypage”，单击右上角的，勾选对象“*命名空间*__mypage_follow__CST”、“*命名空间*__mypage_healthPunch__CST”的全部权限，然后单击，保存设置。

图 5-20 授权自定义对象权限的“读取与创建”权限

c. 在右侧选择“系统参数”，然后在“内置系统参数”下，搜索“bingo.guest.api.route.whitelist”，单击参数名进入编辑，设置默认值为“否”。

图 5-21 搜索内置系统参数

图 5-22 修改“值”

d. 参考上一步，修改内置系统参数“bingo.permission.resource.default.switch”值为“是”。

图 5-23 修改内置系统参数

步骤7 制作游客访问的页面地址。

说明

为什么需要改造地址，不能直接使用高级页面的预览地址，这是因为高级页面中使用的是一个跳转组件，高级页面预览地址会自动跳转到标准页面，标准页面是不支持游客直接访问的，因此需要根据开发环境中发布的地址，改造成高级页面发布在运行环境的预览地址。

1. **复制**开发环境中高级页面发布成功后的页面网址，此地址是在**步骤4.6**中获取的。
   https://appcube.cn-north-4.huaweicloud.com/magno/render/hwtest__mypage_*00000000000XXXXXXX*/view-mobile/mypage

图 5-24 开发环境中的页面地址

2. 将开发环境域名“https://appcube.cn-north-4.huaweicloud.com/”，替换为运行环境域名 “https://appcuberun.cn-north-4.huaweicloud.com/”。
https://appcuberun.cn-north-4.huaweicloud.com/magno/render/hwtest__mypage_*0000000000XXXXXX*/view-mobile/mypage
此地址即为下一步需要访问的高级页面地址。

步骤8 在无痕模式下，访问高级页面（即上一步改造后的地址）。

因高级页面中的跳转事件，访问高级页面时，将直接跳转至标准页面，在页面中提交打卡信息，验证页面功能。

图 5-25 访问高级页面，验证页面功能

----结束

扩展知识

标准页面预置了二维码组件，用于生成二维码。使用该组件可将游客访问的打卡页面地址做成二维码，后续可将此二维码直接分享给其他游客（匿名用户）扫码体检。

制作二维码方法如下：

步骤1 使用开发者账号登录AstroZero开发环境，进入一个应用，例如**步骤1**创建的“mypage”应用。

步骤2 鼠标放在“Page”下，单击界面上出现的“+”，在弹出菜单中选择“标准页面”。

步骤3 设置页面标签和名称，例如“testcode”，单击“添加”。

步骤4 参考**图5-26**，向标准页面中拖入一个二维码组件，在右侧属性面板中设置“内容或链接”为**步骤7.2**改造后的地址。

https://appcuberun.cn-north-4.huaweicloud.com/magno/render/hwtest__mypage_*00000000XXXXXXX*/view-mobile/mypage

图 5-26 配置二维码组件

表 5-1 二维码组件属性说明

属性	说明
内容或链接	设置扫描二维码后，要展示的文本内容或者跳转的链接地址。
宽度	二维码的宽度，单位像素。 默认值：150
容错级别	二维码被遮挡或残破时依然能被识别的几率，容错级别越高抗残破或遮挡的能力就越强。
空白间距	四周空白间距，单位像素。
前景色	二维码的颜色。
背景色	背景颜色。
中间Logo	二维码中间Logo图片，可不用设置。 设置Logo后，如果扫描二维码识别失败，可以调高容错级别或调大二维码

属性	说明
Logo大小	Logo大小，单位像素。

步骤5 单击页面上方保存页面，单击预览页面。

图 5-27 生成二维码

步骤6 将上一步生成的二维码分享给游客，游客通过微信或者支付宝扫描二维码，可扫码体验，进行健康打卡。

----结束

6 如何配置应用主题

背景信息

参考**自定义标准页面的主题样式**和**高级页面中如何引入第三方库**中操作，可设置应用中标准页面和高级页面的主题样式。当应用较多或者应用内的标准页面和高级页面较多时，手动逐个设置应用页面主题样式就比较繁琐。此时您可参考本章节操作，通过安装应用主题配置资产包，统一管理各应用的主题，快速配置主题样式。

说明

- 该操作只适用于AstroZero开发环境。
- 配置应用主题将直接修改应用的元数据配置，重新安装应用将覆盖已有的主题库相关信息，需要重新进行主题配置操作。

前提条件

配置应用主题前，需要先自定义主题样式库，并上传到资产库中，具体操作请参考**如何引入第三方库**中“操作步骤”的前两步。

操作步骤

步骤1 安装并预览“应用主题配置”。

1. 单击**下载链接**，获取应用包“AppThemeConfig.zip”。
2. 参考**如何登录经典版环境配置**中操作，登录经典版**开发环境**的环境配置。
3. 在左侧导航栏中，选择“应用管理 > 软件包管理 > 软件包安装”，单击“新建”，将应用包拖入安装。
4. 在弹出的页面，单击“继续”，完成应用包的安装。
5. 安装完成后，返回到开发环境首页，选择“库”，单击已安装的“应用主题配置”。

图 6-1 在库中选中应用

6. 在应用左侧菜单栏中，单击  ，预览该应用。

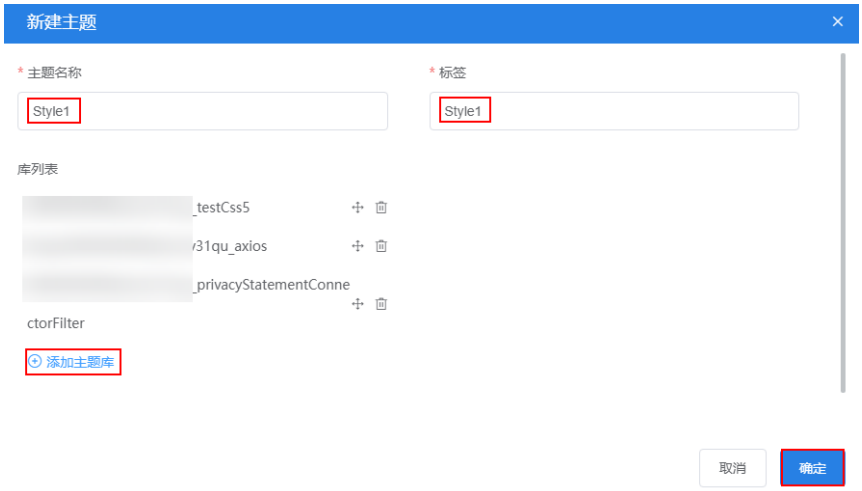
步骤2 定义主题。

1. 在“主题定义”页面，单击“新建主题”。

图 6-2 新建主题

2. 在“新建主题”页面，设置主题名称和标签，单击“添加主题库”，为该主题添加主题样式库，单击“确定”。

图 6-3 为主题添加主题样式库



3. 按照上述操作，依次新建其他主题。

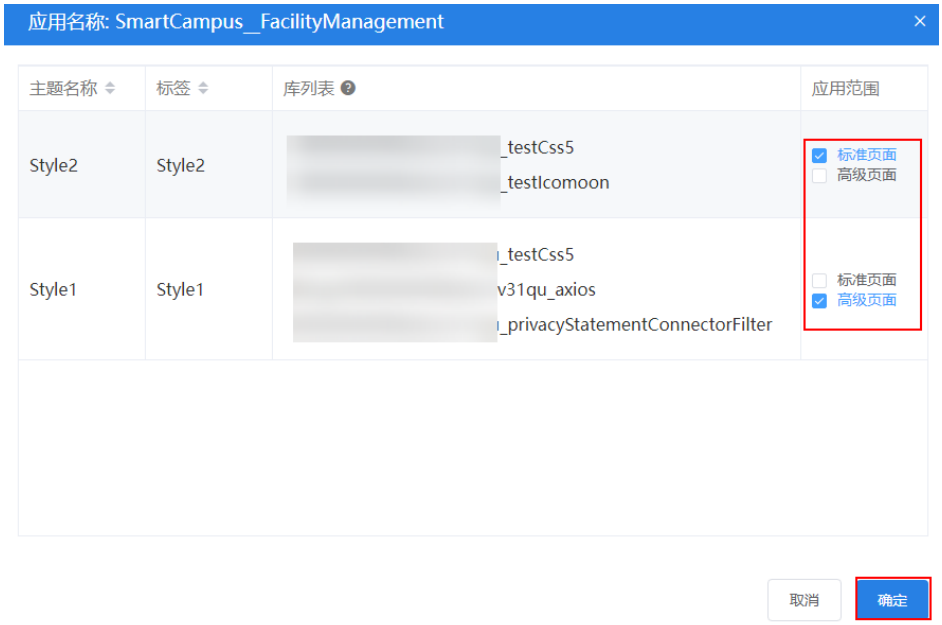
步骤3 配置应用的主题。

1. 在“应用主题配置”页面，单击需要配置主题应用后的  。
   在应用主题配置页面，会显示当前环境中已安装的所有应用。
2. 在需要配置主题的应用后，单击齿轮图标。

图 6-4 单击齿轮图标

3. 为应用配置主题，可针对标准页面和高级页面配置不同的主题。

图 6-5 配置应用的主题



4. 配置完成后，在“主题定义”页面单击“刷新”，可查看到主题关联的应用，即使用该主题的应用。

图 6-6 查看主题关联的应用

步骤4 （可选）迁移主题配置数据。

1. 在“主题定义”页面，单击“导出”，可将当前主题配置的数据和资源包导出。
2. 在目标环境上安装“应用主题配置”，预览应用后，在“主题定义”页面单击“导入”，实现数据的迁移。

----结束