工程配置协议拓展文档

1、文档说明

此文档用于指导开发人员进行 iiot 扩展工程配置的协议开发使用。
以 ATLAS 协议为例，后端示例代码如下附件 edge-config-atlas.zip：

edge-config-atlas.zip

前端示例代码如下 iiotEdgeConfigAtlas.zip

iiotEdgeConfigAtlas.zip

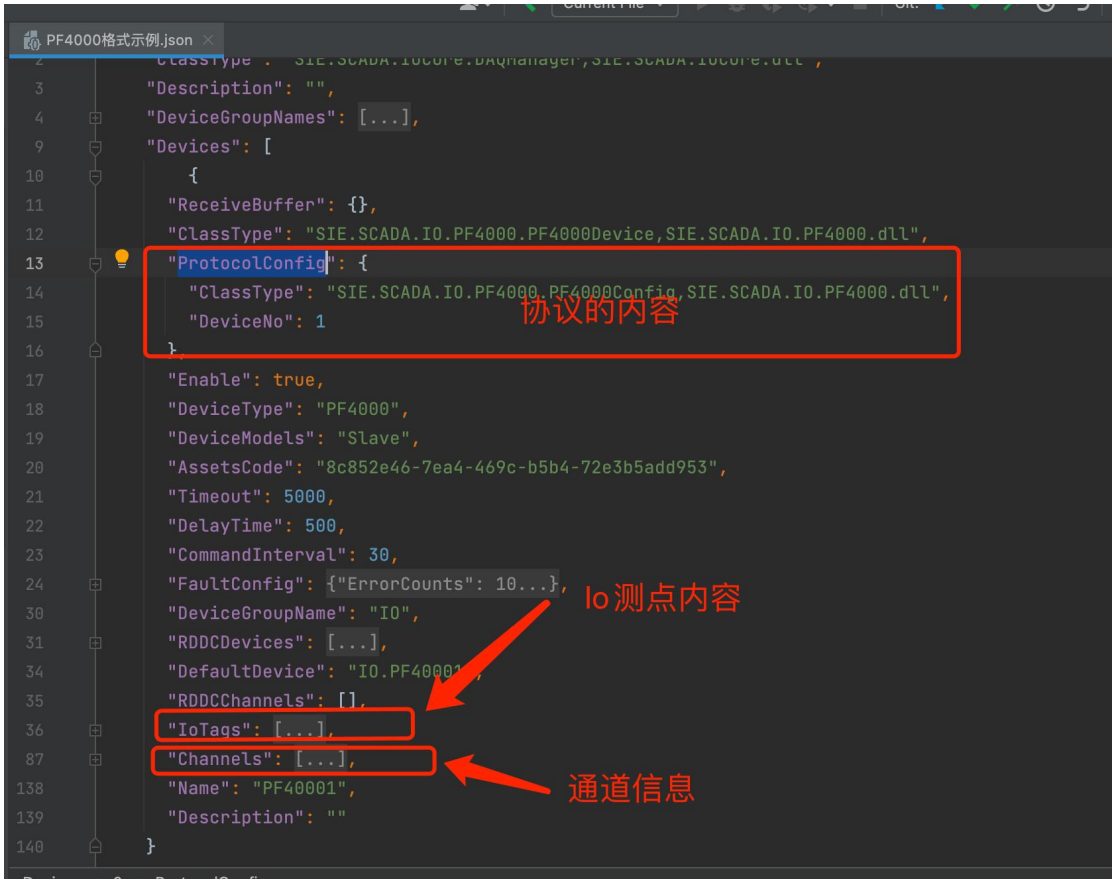
2、前提条件

1、IIOT 平台是基于 IIDP 技术平台开发的产品，其扩展开发依赖于 IIDP 的扩展方式，在 IIOT 扩展开发开始前，必须熟悉 IIDP 基于元模型驱动开发的知识，相关知识可通过“IIDP 开发文档”进行学习，网址：
http://iidp.chinasie.com:9999/iidpdoc/pages/0fd75a/

2、工程配置的拓展是基于 SMDC 已经支持该协议驱动，并提供对应的协议 IOLink.scfg 文件样例，根据 IOLink.scfg 文件的内容进行协议拓展。
3、IOLink.scfg 文件示例

IOLink.scfg

4、IOLink.scfg 文件中每个协议需要处理的 json 块主要是 ProtocolConfig、IoTags、Channels 需要处理。拓展的模型也是围绕协议、io 测点、通道进行处理



3、扩展方式

使用一个全新的 APP 进行扩展开发。

扩展步骤：

1) 创建一个全新的 APP
2) 如果是全新的功能，需要创建一个权限的模型
3) 如果是对原有功能的增强，需要创建一个原有模型的扩展模型
4) 在模型内创建属性、服务。如果是对原有功能的增强，创建服务时，必须与原有服务的签名一模一样
5) 注意：扩展的内容会完全覆盖原有功能，因此，如果需求非完全覆盖时，需要显式调用原有逻辑
6) 工程配置拓展协议的 app 需要依赖 iiot_edge_config 的 app

工程配置拓展说明：

每个设备协议包含设备信息、通道、协议三大块内容。其中设备信息所有协议相同，通道信息主要有 NetChannel、SerialChannel、VirtualChannel 这 3 种，如果新协议是这 3 种通道之一，可以不用拓展通道。协议块信息里，所有协议都不一样，所有协议都需要进行拓展。

4、后端拓展

4.1 拓展的 app 需要依赖 iiot_edge_config

app.json 文件示例：

```
{
  "name": "iiot_edge_config_atlas",
  "displayName": "工程配置 ATLAS 协议",
  "author": "iiot",
  "company": "sie",
  "category": "iiot",
  "product": "iiot",
  "description": "工程配置工程配置 ATLAS PF4000 协议",
  "summary": "工程配置 ATLAS 协议，支持在 IIOT 平台上在线编辑 SMDC 的采集通讯信息，并实时生成对应的配置文件，同步至 SMDC。",
  "type": "SDK",
  "tag": "master",
  "version": "v3.0.12.240101",
  "resolved": "com.sie.iiot.apps.config.model",
  "dependencies": [ "iiot_edge_config"],
  "icon": "",
```

```
  "license": "LGPL 3.0",
  "view": [

  ],
  "data": [
  ],
  "events": {
    "startUp": [
      "iiot_config_atlas_startup::startUp"
    ]
  }
}
```

4.2 配置设备类型的种子数据

备注：设备类型的种子数据用于“物实体”》”通讯配置”中的设备类型的级联下拉框选择用

1) 在 app 的 data 包下新增设备类型的种子数据 json 文件
(例如 iiot_config_atlas_device_type_data.json)。内容示例：

```
{
  "data": {
    "industry_atlas": {
      "model": "iiot_config_device_type",
      "properties": {
        "name": "ATLAS",
        "device_type": "industry_atlas",
        "industry_index": 6,
        "factory_index": 0,
        "device_index": 0,
        "description": "ATLAS 行业"
      }
    },
    "factory_atlas": {
      "model": "iiot_config_device_type",
      "properties": {
        "name": "ATLAS",
        "device_type": "factory_atlas",
        "industry_index": 6,
        "factory_index": 1,
        "device_index": 0,
        "description": "ATLAS 厂商",
        "parent_id": {
          "@ref": "industry_atlas"
        }
```

```
      }
    },
    "PF4000": {
      "model": "iiot_config_device_type",
      "properties": {
        "name": "PF4000",
        "device_type": "PF4000",
        "industry_index": 8,
        "factory_index": 5,
        "device_index": 7,
        "description": "ATLAS 的 PF4000 协议",
        "parent_id": {
          "@ref": "factory_atlas"
        }
      }
    }
  }
}
```

2) 参数含义

参数	含义	是否必填	备注
model	种子数据所属模型	是	固定，iiot_config_device_type
properties.name	设备类型的名称	是	
properties.device_type	smdc 的设备类型	是	
properties.industry_index	行业的排序	是	
properties.factory_index	厂商的排序	是	
properties.device_index	设备类型的排序	是	
properties.description	描述	否	
properties.parent_id	父目录的 id	否	不填则代表一级目录，行业不需要传 parent_id

4.3 Protocol 协议模型拓展

- 说明：协议模型主要是处理“通讯配置”中”协议”那部分表单的模型数据。对应 iolink 文件的 ProtocolConfig 部分的 json 数据

4.3.1 协议模型继承 iiot_config_base_protocol

1）协议需要继承 iiot_config_base_protocol， 注意 type 是 Data。不要用 Buss

2）需要拓展 getJson 方法，用于封装 iolink 文件中设备的协议部分的 json 内容。
拓展模型 iiot_config_atlas_pf4000_protocol 示例：

```
@SDK.Model(name = "iiot_config_atlas_pf4000_protocol",
            parent = "iiot_config_base_protocol",
            type = SDK.Model.ModelType.Data,
            displayName = "ATLAS PF4000 协议模型")
public class ConfigAtlasPF4000Protocol extends Model {
      static Property DeviceNo = Property.Integer().displayName("设备地址").tooltips("请输入设备地址").defaultValue(1).isRequired();

      /**
       * 协议的内容 json
       * @param rs
       * @param protocolData
       * @return
       */
       public Map<String, Object> getJson(RecordSet rs, Map<String, Object> protocolData) {
            protocolData.put("ClassType", "SIE.SCADA.IO.PF4000.PF4000Config,SIE.SCADA.IO.PF4000.dll");
            return protocolData;
      }
}
```

4.3.2 配置协议表单视图

1) 在 view 文件夹下新增协议的表单视图文件，比如 config_atlas_pf4000_protocol_form.json
2) config_atlas_pf4000_protocol_form.json 示例：

```
{
  "views": {
    "iiot_config_atlas_pf4000_protocol_form": {
      "name": "工程配置 ATLAS PF4000_表单",
      "model": "iiot_config_atlas_pf4000_protocol",
      "type": "form",
      "mode": "primary",
      "body": {
        "type": "form",
        "customView": [
          {
            "type": "row",
            "items": [
              {
                "name": "DeviceNo",
```

```
                    "text": "设备地址",
                    "span": 6,
                    "disabled": "(r, { model: t }) => {let detailVM =
tech_app.page.getNode('iiot_thing_entity_menu_table_detail');return
detailVM.$ds.clickType === 'preview' ? true : false}",
                    "onChange": "(state,v,{model:t})=>{let
formNote=tech_app.page.getNode('iiot_thing_entity_menu_form_main_detail_top_com
mon');formNote.$ds.isNotSave=true;formNote.$ds.isFormNotSave=true}"
                }
            ]
        }
    ]
  }
}
```

3）上面视图文件的视图表单的视图 key(iiot_config_atlas_pf4000_protocol_form) 和 model(iiot_config_atlas_pf4000_protocol) 需要给前端那边拓展使用

4.3.3 拓展的方法说明

4.3.3.1 getJson

1）getJson 方法用于处理跟 smdc 连接的 iolink 文件中 ProtocolConfig 块的 json 内容。注意 ClassType 需要从 smdc 的协议 json 文档拿
2）入参：protocolData，包含协议表单提交的内容
3）返回值：ProtocolConfig 块内容，包含 ClassType

4.4 设备模型 iiot_config_iolink_device 拓展

- 说明：iiot_config_iolink_device 模型是处理通讯配置相关的模型，负责封装协议、通道和工程配置保存和下发 smdc 的协议文件内容处理

4.4.1 拓展示例

```
@SDK.Model(name = "iiot_config_iolink_device",
           parent = "iiot_config_iolink_device",
           type = SDK.Model.ModelType.Buss,
           isAutoLog = Bool.True,
           displayName = "集中式配置 ATLAS 拓展配置")
```

```
public class ConfigIolinkDeviceAtlas extends Model {


    // region  拓展的

    //  获取设备的 classType

    public String getClassType(RecordSet rs, String deviceType) {
        if ("PF4000".equals(deviceType)) {
            return
"SIE.SCADA.IO.PF4000.PF4000Device,SIE.SCADA.IO.PF4000.dll";
        }
        return (String) rs.callSuper(ConfigIolinkDeviceAtlas.class, "getClassType",
deviceType);
    }
    //  根据设备类型获取对应的协议的模型

    public String getProtocolModel(RecordSet rs, String deviceType){
        if ("PF4000".equals(deviceType)) {
            return "iiot_config_atlas_pf4000_protocol";
        }
        return (String) rs.callSuper(ConfigIolinkDeviceAtlas.class,
"getProtocolModel", deviceType);
    }
    // endregion
}
```

4.4.2 拓展的方法说明

4.4.2.1 getClassType 方法

- 该方法返回值是设备的 iolink 文件里设备 Devices 的 ClassType 内容，需要从 smdc 的协议文件 IOLink.scfg 获取

4.4.2.2 getProtocolModel 方法

- 说明：getProtocolModel 根据不同的类型，返回不同的协议的模型名（比如上面的 iiot_config_atlas_pf4000_protocol）

4.5 IoTags 模型拓展

- 说明：主要处理工程配置 tab 页的功能，对应 iolink 文件里的 IoTags 部分

4.5.1 工程配置测点模型 iiot_config_base_iotags 拓展

拓展示例：

```
@SDK.Model(name = "iiot_config_base_iotags",
            parent = {"iiot_config_base_iotags"},
            isAutoLog = Bool.True,
            type = SDK.Model.ModelType.Buss,
            displayName = "集中式配置 ATLAS PF4000 测点模型")
public class ConfigAtlasPF4000IoTags extends Model {

      // AddressRegister 类型枚举不一样
      static Property AtlasPf4000AddressRegister = Property.Enum().displayName("地址
").tooltips("请输入地址").length(128).values(new LinkedHashMap<String,
                                 String>() {{
                        put("CellID", "CellID");
                        put("ChannelID", "ChannelID");
                        put("TorqueControllerName", "TorqueControllerName");
                        put("VINNumber", "VINNumber");
                        put("IdentifierData", "IdentifierData");
                        put("JobID", "JobID");
                        put("ParameterSetNumber", "ParameterSetNumber");
                        put("BatchSize","BatchSize");
                        put("BatchCounter","BatchCounter");
                        put("TighteningStatus" ,"TighteningStatus");
                        put("TorqueStatus" ,"TorqueStatus");
                        put("AngleStatus","AngleStatus");
                        put("TorqueMinLimit" ,"TorqueMinLimit");
                        put("TorqueMaxLimit" ,"TorqueMaxLimit");
                        put("TorqueFinalTarget","TorqueFinalTarget");
                        put("Torque" ,"Torque");
                        put("Angle","Angle");
                        put("AngleMin" ,"AngleMin");
                        put("AngleMax","AngleMax");
                        put("FinalAngleTarget" ,"FinalAngleTarget");
                        put("TimeStamp" ,"TimeStamp");
                        put("LastChangeParmeterSetTime","LastChangeParmeterSetTime");
                        put("BatchStatus" ,"BatchStatus");
                        put("TighteningID","TighteningID");
```

```
                }})
                .defaultValue("CellID").isStore(Bool.False);

    /**
     * 获取测点的表格视图的 key，可拓展
     * @param rs
     * @param deviceType
     * @return
     */
    public String getGridViewKey(RecordSet rs, String deviceType) {
        if ("PF4000".equals(deviceType)) {
            return "config_atlas_pf4000_iotags_grid";
        }
        return (String) rs.callSuper(ConfigAtlasPF4000IoTags.class, "getGridViewKey",
deviceType);
    }

    /**
     * 根据设备类型获取对应的处理器模型名称，可拓展
     * @param rs
     * @param deviceType
     * @return
     */
    public String getProcessModelName(RecordSet rs, String deviceType) {
        if ("PF4000".equals(deviceType)) {
            return "iiot_config_atlas_pf4000_iotags_processor";
        }
        return (String) rs.callSuper(ConfigAtlasPF4000IoTags.class,
"getProcessModelName", deviceType);
    }
}
```

4.5.1.1 拓展的字段（可选）

- 说明：当 iiot_config_base_iotags 的模型的字段不能满足新协议需求。需要定义新的字段时可按需添加字段。
- 拓展示例如上面的 AtlasPf4000AddressRegister。如是对地址（寄存器）的字段定义不满足（比如寄存器下拉框选项不一样），可以把字段定义为 isStore(Bool.False)， 同时在文档 8.2 章中的处理器模型里的 getModelFields 方法里，把 AtlasPf4000AddressRegister 的定义数据赋值 AddressRegister。

已有字段：

字段	名称	类型	备注
EntityId	物实体 id	String	
PropertyName	属性名称	String	
CustomPropertyId	属性 id	String	
DataType	数据类型	String	
Name	变量名	String	
ReadWriteMode	读写模式	Enum	
IsSaveInitValue	是否保存初值	Boolean	
HistoryStore	历史归档	String	
Description	描述	String	
MinValue	最小值	String	
MaxValue	最大值	String	
Digits	小数位	Integer	
LinkTagFullName	关联变量	String	
ExtendedDomain	拓展域	Integer	
AddressDataType	工程数据类型	Enum	
AddressRegNo	地址	String	
AddressRegister	寄存器	Enum	
AddressLength	长度	Integer	
AddressBitNo	位	String	
AddressCycle	扫描周期	String	
AddressExAddress	寄存器号	String	
ConverterMode	转换方式	Enum	
ConvertType	转换类型	Enum	
MaxEngineerValue	最大工程值	String	
MinEngineerValue	最小工程值	String	
ValueTable	值表	List	
EngineerToIOTagExpression	工程值转 IO 变量表达式	String	
IOTagToEngineerExpression	IO 变量转工程值表达式	String	
IsReport	是否上报	Boolean	
ReportModeType	上报类型	List	
ReportModeInterval	上报间隔	Integer	
ReportModeDeadband	死区	Integer	
TagType	测点类型	Integer	
AddressFilterString	过滤条件	String	
AddressRefreshModel	更新模式	String	
AddressTimeStampColumn	时间戳列	String	
AddressUnit	单位	String	
AddressRegisterString	寄存器	String	用于覆盖部分协议中的寄存器是输入框的

			问题

4.5.1.2 拓展的方法

4.5.1.2.1 getGridViewKey 获取工程配置视图 key

- 说明：4.5.2.1 中的表格视图配置完后，需要把表格的视图 key 的值在 getGridViewKey 里返回

4.5.1.2.2 getProcessModelName 获取处理器模型名

- 说明：不同协议的处理器模型不一样，4.5.2 章里的处理器模型写好好，需要在 getProcessModelName 里返回

4.5.2 工程配置处理器处理器

- 说明：主要是对工程配置中 json 数据解析，导入导出等功能处理的
- 处理器模型需要继承 iiot_config_base_iotags_processor， 类型需要是 Data

继承示例：

```
@SDK.Model(name = "iiot_config_atlas_pf4000_iotags_processor",
        parent = "iiot_config_base_iotags_processor",
        type = SDK.Model.ModelType.Data,
        displayName = "集中式配置 ATLAS PF4000 处理器测点模型")
public class ConfigAtlasPf4000IoTagsProcessor extends Model {

    /**
     * json 格式处理
     * @param rs
     * @param deviceData
     * @return
     */
    public Map<String, Object> getJson(RecordSet rs, Map<String, Object> tagData, Map<String, Object> deviceData) {
        tagData = (Map<String, Object>)rs.callSuper(ConfigAtlasPf4000IoTagsProcessor.class,"getJson", tagData, deviceData);
        // region Address 块
        Map<String, Object> addressPart = new HashMap<>();
        addressPart.put("Register", tagData.remove("AddressRegister"));
```

```
            String dataType = (String) tagData.remove("AddressDataType");
            addressPart.put("DataType", NumberUtils.parseInteger(dataType));
            addressPart.put("RwMode", tagData.get("ReadWriteMode"));
            addressPart.put("Cycle",
NumberUtils.parseInteger(tagData.remove("AddressCycle")));
            tagData.put("Address",addressPart);
            // endregion
            return tagData;
        }


        /**
         * 导出的属性列表
         * @return
         */
        public List<String> getExportProperty(RecordSet rs) {
            List<String> properties = new ArrayList<>();
            properties.add("PropertyName"); // 属性名称
            properties.add("CustomPropertyId"); // 属性 id
            properties.add("Name"); // 变量名
            properties.add("AddressDataType");// 数据类型
            properties.add("AddressRegister");// 地址
            properties.add("ReadWriteMode"); // 读写模式
            properties.add("AddressCycle");// 扫描周期
            properties.add("IsSaveInitValue");// 是否保存初值
            properties.add("HistoryStore");// 历史归档
            properties.add("Digits"); // 小数位
            properties.add("MinValue"); // 最小值
            properties.add("MaxValue"); // 最大值
            properties.add("ReportModeDeadband"); // 死区
            properties.add("LinkTagFullName"); // 关联变量
            properties.add("ExtendedDomain"); // 拓展域
            properties.add("ConverterMode"); // 转换方式
            properties.add("ConvertType"); // 转换类型
            properties.add("MinEngineerValue"); // 最小工程值
            properties.add("MaxEngineerValue"); // 最大工程值
            properties.add("EngineerToIOTagExpression"); // 工程值转 IO 变量表达式
            properties.add("IOTagToEngineerExpression"); // IO 变量转工程值表达式
            properties.add("ValueTable"); // 值表
            properties.add("Description"); // 描述
            return properties;
        }

        // 获取导出的表头名称
```

```
    public String getExportTagName(RecordSet rs, String property) {
        if ("AddressRegister".equals(property)) {
            return "地址";
        }
        return (String) rs.callSuper(ConfigAtlasPf4000IoTagsProcessor.class,
"getExportTagName", property);
    }
    // 获取导入的属性 property
    public String getImportProperty(RecordSet rs, String tagName) {
        if ("地址".equals(tagName)) {
            return "AddressRegister";
        }
        return (String) rs.callSuper(ConfigAtlasPf4000IoTagsProcessor.class,
"getImportProperty", tagName);
    }

    /**
     * 获取测点模型的字段定义，可拓展
     * @param rs
     * @return
     */
    public Map<String, Object> getModelFields(RecordSet rs) {
        Map<String, Object> fields = (Map<String,
Object>)rs.callSuper(ConfigAtlasPf4000IoTagsProcessor.class, "getModelFields");
        fields.put("AddressRegister", fields.get("AtlasPf4000AddressRegister"));
        return fields;
    }
}
```

4.5.2.1 工程配置表格视图配置

- 工程配置的表格视图里，model 固定为 iiot_config_base_iotags
新建视图文件（例如 config_atlas_pf4000_iotags_grid.json），内容如下：

```
{
  "views": {
    "config_atlas_pf4000_iotags_grid": {
      "name": "ATLAS_PF4000 协议测点_表格",
      "model": "iiot_config_base_iotags",
      "type": "grid",
      "mode": "primary",
```

```
    "body": {
      "type": "grid",
      "columns": [
        {
          "name": "PropertyName",
          "minWidth": "120",
          "displayName": "属性名称",
          "custom": "true"
        },
        {
          "name": "CustomPropertyId",
          "minWidth": "120",
          "displayName": "属性 ID",
          "custom": "true"
        },
        {
          "name": "DataType",
          "minWidth": "120",
          "displayName": "数据类型",
          "custom": "true"
        },
        {
          "name": "Name",
          "rowEditable": true,
          "editConfigs": {
            "editType": "input"
          },
          "minWidth": "120",
          "displayName": "变量名",
          "custom": "true"
        },
        {
          "name": "AddressDataType",
          "rowEditable": true,
          "editConfigs": {
            "editType": "select",
            "clearable": false,
            "options": [
              { "label": "Boolean", "value": "0" },
              { "label": "Long", "value": "1" },
              { "label": "ULong", "value": "2" },
              { "label": "Double", "value": "3" },
              { "label": "String", "value": "4" },
              { "label": "SByte", "value": "6" },
```

```
            { "label": "Byte", "value": "7" },
            { "label": "Int16", "value": "8" },
            { "label": "Uint16", "value": "9" },
            { "label": "Int32", "value": "10" },
            { "label": "Uint32", "value": "11" },
            { "label": "Float", "value": "12" },
            { "label": "Datetime", "value": "13" },
            { "label": "Dynamic", "value": "14" }
          ]
        },
        "minWidth": "120",
        "displayName": "工程数据类型",
        "custom": "true"
      },
      {
        "name": "AddressRegister",
        "minWidth": "120",
        "displayName": "地址",
        "custom": true,
        "rowEditable": true,
        "editConfigs": {
          "editType": "select"
        }
      },
      {
        "name": "ReadWriteMode",
        "minWidth": "120",
        "displayName": "读写类型",
        "custom": "true",
        "rowEditable": true,
        "editConfigs": {
          "editType": "select",
          "clearable": false,
          "options": [
            { "label": "ReadWrite", "value": "0" },
            { "label": "ReadOnly", "value": "1" },
            { "label": "WriteOnly", "value": "2" }
          ]
        }
      },
      {
        "name": "AddressCycle",
        "minWidth": "120",
        "displayName": "扫描周期",
```

```
      "custom": "true",
      "rowEditable": true,
      "editConfigs": {
        "editType": "input"
      }
    },
    {
      "name": "IsSaveInitValue",
      "minWidth": "120",
      "displayName": "是否保存初值",
      "custom": "true",
      "rowEditable": true,
      "editConfigs": {
        "editType": "select",
        "options": [
          { "label": "否", "value": false },
          { "label": "是", "value": true }
        ]
      }
    },
    {
      "name": "HistoryStore",
      "minWidth": "120",
      "displayName": "历史归档",
      "custom": "true"
    },
    {
      "name": "Digits",
      "minWidth": "120",
      "displayName": "小数位",
      "custom": "true",
      "rowEditable": true,
      "editConfigs": {
        "editType": "input"
      }
    },
    {
      "name": "MinValue",
      "minWidth": "120",
      "displayName": "最小值",
      "custom": "true",
      "rowEditable": true,
      "editConfigs": {
        "editType": "input"
```

```
    }
  },
  {
    "name": "MaxValue",
    "minWidth": "120",
    "displayName": "最大值",
    "custom": "true",
    "rowEditable": true,
    "editConfigs": {
      "editType": "input"
    }
  },
  {
    "name": "LinkTagFullName",
    "minWidth": "120",
    "displayName": "关联变量",
    "custom": "true"
  },
  {
    "name": "ExtendedDomain",
    "minWidth": "120",
    "displayName": "拓展域",
    "custom": "true"
  },
  {
    "name": "ConvertType",
    "displayName": "数据转换",
    "custom": "true",
    "type": "button",
    "options": {
      "type": "text"
    }
  },

  {
    "name": "Description",
    "minWidth": "120",
    "displayName": "描述",
    "custom": "true",
    "rowEditable": true,
    "editConfigs": {
      "editType": "input"
    }
  },
```

```
{
  "name": "IsReport",
  "hidden": true,
  "displayName": "是否上报",
  "custom": "true",
  "rowEditable": true,
  "editConfigs": {
    "editType": "input"
  }
},
{
  "name": "ReportModeType",
  "displayName": "上报设置",
  "custom": "true",
  "type": "button",
  "fixed": "right",
  "options": {
    "type": "text"
  }
},
{
  "name": "ReportModeDeadband",
  "hidden": true,
  "minWidth": "120",
  "displayName": "死区",
  "custom": "true",
  "rowEditable": true,
  "editConfigs": {
    "editType": "input"
  }
},
{
  "name": "ReportModeInterval",
  "hidden": true,
  "minWidth": "120",
  "displayName": "上报间隔",
  "custom": "true",
  "rowEditable": true,
  "editConfigs": {
    "editType": "input"
  }
},
{
```

```
      "name": "ConverterMode",
      "hidden": true,
      "minWidth": "120",
      "displayName": "转换方式",
      "custom": "true","rowEditable": true,
      "editConfigs": {
        "editType": "select",
        "options": [
          { "label": "无转换", "value": "None" },
          { "label": "线性转换", "value":"Linear" },
          { "label": "开方转换", "value":"Sqrt" },
          { "label": "值表转换", "value":"ValueTable" },
          { "label": "表达式", "value":"Expression" }
        ]
      }
    },
    {
      "name": "MaxEngineerValue",
      "hidden": true,
      "minWidth": "120",
      "displayName": "最大工程值",
      "custom": "true"
    },
    {
      "name": "MinEngineerValue",
      "hidden": true,
      "minWidth": "120",
      "displayName": "最小工程值",
      "custom": "true"
    },
    {
      "name": "ValueTable",
      "hidden": true,
      "minWidth": "120",
      "displayName": "值表",
      "custom": "true"
    },
    {
      "name": "EngineerToIOTagExpression",
      "hidden": true,
      "minWidth": "120",
      "displayName": "工程值转 IO 变量表达式",
      "custom": "true"
    },
```

```
            {
              "name": "IOTagToEngineerExpression",
              "hidden": true,
              "minWidth": "120",
              "displayName": "IO 变量转工程值表达式",
              "custom": "true"
            }
          ],
          "buttons": [
          ],
          "operationProperty": {
            "minWidth": "100"
          },
          "tbar": [
          ]
        }
      }
    }
}
```

4.5.2.2 处理器模型拓展方法说明

4.5.2.2.1 getJson 获取 json

该方法用于封装 iolink 文件中测点的 IoTags 里面的内容

4.5.2.2.2 getExportProperty 获取导出的属性列表

该方法用户工程配置里需要导出的模型属性列表

4.5.2.2.3 getExcelHeadDispalyName 获取导出表头名称(可选)

由于协议比较多，一个属性在不同协议中可能叫不同名称，比如地址。Address.register 在部份协议叫“寄存器”，部分协议叫“地址”。同一个属性的下拉框选项也可能不一样。因为如果有自定义的名称导出，需要拓展修改该方法

4.5.2.2.4 getExcelHeadValue 获取导入的表头的对应属性(可选)

由于协议比较多，一个属性在不同协议中可能叫不同名称，比如地址。Address.register 在部份协议叫“寄存器”，部分协议叫“地址”。因为如果有自定义的名称导入，需要拓展修

改该方法

4.5.2.2.5 getModelProperties 获取模型字段(可选)

- 说明：返回该工程配置模型的字段 Map
- 由于不同协议中，部分字段的枚举下拉框值不一样，前端又需要后端去定义枚举数据。比如 ATLAS PF4000 协议中，地址的下拉框枚举和其他协议不一样，因此在 iiot_config_base_iotags 的拓展里，用 AtlasPf4000AddressRegister 的 field 定义数据去覆盖 AddressRegister 数据。
- AtlasPf4000AddressRegister 的定义，在 4.5.1 章中 iiot_config_base_iotags 拓展增加

```
public Map<String, Object> getModelFields(RecordSet rs) {
        Map<String, Object> fields = (Map<String,
Object>)rs.callSuper(ConfigAtlasPf4000IoTagsProcessor.class, "getModelFields");
        fields.put("AddressRegister", fields.get("AtlasPf4000AddressRegister"));
        return fields;
}
```

4.6 Channel 通道拓展（可选）

备注：已支持 NetChannel、SerialChannel、VirtualChannel 这三种通道，这些通道不需要拓展。只需要前端那边直接使用即可

说明：通道相关模型主要是处理“通讯配置”中的通道部分的功能，对应 iolink 文件中的 Channels 部分

4.6.1 通道模型继承 iiot_config_base_channel

- 通道需要继承 iiot_config_base_channel 模型
- VirtualChannel 内容示例：

```
@SDK.Model(name = "iiot_config_virtual_channel",
            type = SDK.Model.ModelType.Data,
            parent = "iiot_config_base_channel",
            displayName = "集中式配置的虚拟通道")
public class ConfigVirtualChannel extends Model {
    static Property Name = Property.String().displayName("通道名").tooltips("请输入
```

```
通道名称").length(128).defaultValue("VirtualChannel1").isRequired();

        static Property Description = Property.String().displayName("描述").tooltips("请输

入描述").length(600);


        /**
         * json 格式处理
         * @param rs
         * @param channelData
         * @return
         */
        public Map<String, Object> getJson(RecordSet rs, Map<String, Object>
channelData) {
                if (channelData == null) {
                        return null;
                }
                channelData = (Map<String, Object>)rs.callSuper(ConfigNetChannel.class,
"getJson", channelData);
                // classType
                channelData.put("ClassType",
ChannelEnums.VIRTUAL_CHANNEL.getClassType());
                channelData.remove("EnableShared");
                return channelData;
        }
}
```

4.6.1.1 通道的表单视图

新建视图文件（比如 config_virtual_channel_form.json），示例如下：

```
{
    "views": {
        "iiot_config_virtual_channel_form": {
            "name": "工程配置虚拟通道_表单",
            "model": "iiot_config_virtual_channel",
            "type": "form",
            "mode": "primary",
            "body": {
```

```
            "type": "form",
            "customView": [
              {
                "type": "row",
                "items": [
                  {
                    "name": "Name",
                    "text": "通道名",
                    "span": 6,
                    "disabled": "(r, { model: t }) => {let detailVM =
tech_app.page.getNode('iiot_thing_entity_menu_table_detail');return
detailVM.$ds.clickType === 'preview' ? true : false}",
                    "onChange": "(state,v,{model:t})=>{let
formNote=tech_app.page.getNode('iiot_thing_entity_menu_form_main_detail_top_com
mon');formNote.$ds.isNotSave=true;formNote.$ds.isFormNotSave=true}"
                  },
                  {
                    "name": "Description",
                    "text": "描述",
                    "span": 6,
                    "disabled": "(r, { model: t }) => {let detailVM =
tech_app.page.getNode('iiot_thing_entity_menu_table_detail');return
detailVM.$ds.clickType === 'preview' ? true : false}",
                    "onChange": "(state,v,{model:t})=>{let
formNote=tech_app.page.getNode('iiot_thing_entity_menu_form_main_detail_top_com
mon');formNote.$ds.isNotSave=true;formNote.$ds.isFormNotSave=true}"
                  }
                ]
              }
            ]
          }
        }
      }
    }
```

4.6.1.2 拓展的方法

4.6.1.1.1 getJson

- 入参：channelData 通道的表单数据

- getJson 用于封装 Iolink 文件中的 Channels 块里的 json 内容。ClassType 是必填的，需要从 IOLink.scfg 里拿到该通道的 ClassType 内容

4.6.2 通道拓展 iiot_config_base_channel

4.6.2.1 拓展的方法

4.6.2.1.1 getChannelModel 获取通道模型

- 入参：channelType 通道类型，通道类型由前端传，需要跟前端约定
- 返回值：返回 5.6.1 章中定义的通道模型名

4.7 整理“工程配置模型关系表”表格

备注：
1、完成上述拓展后，可以开始前端拓展开发。
2、整理该协议的“工程配置模型关系表”关系，给前端拓展开发使用。示例如下，把该协议的设备类型、用到的通道和协议以及相关的模型，视图 key 都整理好，给前端拓展开发使用。

设备类型	通道类型（标红是默认）	通道模型	通道视图	协议模型	协议视图
PF4000	netChannel 、serialChannel	iiot_config_net_channel 、iiot_config_serial_channel	iiot_config_net_channel_form、iiot_config_serial_channel_form	iiot_config_atlas_pf4000_protocol	iiot_config_atlas_pf4000_protocol_form

4.8 下发文件内容查询（可选）

备注：和前端联调完，该协议能正常保存，回显后。可以用该接口调试发送 smdc 的文件内容。
- 背景说明：工程配置下发 SMDC 后，会把 IOLink.scfg、IOTService.scfg、TagInfo.scfg 这三个文件内容推送给 SMDC。
- 接口说明：用该接口查询生成的 IOLink.scfg、IOTService.scfg、TagInfo.scfg 内容，重点比对 IOLink.scfg 的结构，查看生成的 IOLink.scfg 内容是否正确。
- 需要先参考“IIOT 扩展开发指导文档”获取 token

4.7.1 请求语法

POST /api/root/rpc/service/ HTTP/2.0

4.7.2 请求参数

名称	类型	描述	示例	必填
params/app	String	APP 名称	"iiot_edge_config"	是
params/model	String	模型	"iiot_config_file_processor"	是
params/service	String	服务方法	"getAllJson"	是
params/args/gatewayId	String	smdc 的实体 id	"03i9zmgxa1la8"	是

4.7.3 返回参数

名称	类型	备注
result/data	Json	工程配置下发 smdc 的全部文件内容
result/data/iotService	Json	生成的 IOTService.scfg 内容
result/data/iolink	Json	生成的 IOLink.scfg 内容
result/data/tagInfo	Json	生成的 TagInfo.scfg 内容

4.7.4 请求示例

```
{
    "id": "guid",
    "jsonrpc": "2.0",
    "method": "service",
    "params": {
        "args": {
            "gatewayId":"03i9zmgxa1la8"
        },
        "context": {
            "uid": "",
            "lang": "zh_CN"
        },
        "model": "iiot_config_file_processor",
        "tag": "master",
        "service": "getAllJson",
        "app": "iiot_edge_config"
```

```
    }
}
```

4.7.5 返回示例


```
{
    "id": "guid",
    "jsonrpc": "2.0",
    "result": {
        "data": {
            "iotService": {
                "ServerConfig": {
                    "MqttAccount": "TO16Oljs",
                    "Description": "",
                    "UserName": null,
                    "Reporter": "SMDC",
                    "MqttServer": "127.0.0.1",
                    "ClassType":
"SIE.SCADA.IOTService.TargetServerConfig,SIE.SCADA.IOTService.dll",
                    "GatewaySN": "thing_24_01_10_1640904",
                    "IotPwd": null,
                    "MqttID": "03i9zmgxa1la8",
                    "IOTServerAddress": null,
                    "IotAccount": null,
                    "Name": null,
                    "MqttPassword": "XKXBZKe7",
                    "MqttPort": 30666,
                    "CycleTime": 1000,
                    "Enable": true,
                    "Password": null
                },
                "Description": "",
                "Entitys": [],
                "ClassType":
"SIE.SCADA.IOTService.IOTServiceManage,SIE.SCADA.IOTService.dll",
                "Name": null,
                "PropertyUpdataModel": []
            },
            "iolink": {
                "Description": "",
                "ClassType": "SIE.SCADA.IOCore.DAQManager,SIE.SCADA.IOCore.dll",
                "Devices": [
                    {
                        "DeviceType": "Simulator",
```

```
                    "Name": "Simulator1",
                    "Enable": true,
                    "AssetsCode": "364b25cd-0776-4c40-b5bd-455b955f4ca5",
                    "MesModel": null,
                    "Description": null,
                    "Timeout": 5000,
                    "DelayTime": 500,
                    "CommandInterval": 30,
                    "ReadWriteMode": 0,
                    "HandleType": 0,
                    "ProtocolConfig": {
                        "MinValue": 0,
                        "MaxValue": 1000,
                        "ClassType":
"SIE.SCADA.IO.Simulator.SimulatorConfig,SIE.SCADA.IO.Simulator.dll",
                        "Cycle": 100
                    },
                    "IoTags": [
                        {
                            "Name": "a",
                            "IsSaveInitValue": false,
                            "HistoryStore": null,
                            "Description": null,
                            "MinValue": -2147483648,
                            "MaxValue": 2147483647,
                            "Digits": null,
                            "LinkTagFullName": null,
                            "ExtendedDomain": null,
                            "AddressExAddress": null,
                            "ConverterMode": "None",
                            "ConvertType": "EngineerConvertToTagValue",
                            "MaxEngineerValue": null,
                            "MinEngineerValue": null,
                            "ValueTable": {},
                            "EngineerToIOTagExpression": null,
                            "IOTagToEngineerExpression": null,
                            "IsReport": true,
                            "TagType": 0,
                            "AddressFilterString": null,
                            "AddressRefreshModel": null,
                            "AddressTimeStampColumn": null,
                            "AddressUnit": null,
                            "ReadWriteMode": 0,
                            "ClassType": "Int32IOTag",
```

```
                                    "Address": {
                                        "Register": "Increase",
                                        "Length": 1,
                                        "RwMode": 0,
                                        "RegNo": "1",
                                        "DataType": 10,
                                        "BitNo": "0",
                                        "Cycle": 500
                                    }
                                }
                            ],
                            "FaultConfig": {
                                "CommandRetryCount": 3,
                                "Duration": 60,
                                "RetryInterval": 120,
                                "ErrorCounts": 10
                            },
                            "RDDCDevices": [
                                "IO.Simulator1"
                            ],
                            "DeviceGroupName": "IO",
                            "DefaultDevice": "IO.Simulator1",
                            "RDDCChannels": [],
                            "ClassType":
"SIE.SCADA.IO.Simulator.SimulatorDevice,SIE.SCADA.IO.Simulator.dll",
                            "Channels": [
                                {
                                    "Description": null,
                                    "ChannelConfig": {
                                        "WriteTimeOut": 1000,
                                        "ReadBufferSize": 102400,
                                        "WriteBufferSize": 1024,
                                        "ReadTimeout": 1000
                                    },
                                    "ClassType":
"SIE.SCADA.IOCore.Channel.VirtualChannel,SIE.SCADA.IOCore.dll",
                                    "IsSingleThread": true,
                                    "Name": "NetChannel1"
                                }
                            ]
                        }
                    ],
                    "DeviceGroupNames": [
                        {
```

```
                        "Name": "IO"
                    }
                ],
                "Name": "IO"
            },
            "tagInfo": {
                "Description": "",
                "TagGroups": [],
                "Tags": [],
                "Name": "Tag"
            }
        },
        "context": {
            "uid": "",
            "lang": "zh_CN",
            "token":
"cae26ebfc0a54a39b2e0da35633d19b6cd11cb19efe6457ce5c8305f32fe206a59763a24766
2f8ce53932676274d68774f8917fe4727ce9ce6683e2858afb96cb7e61a5d594f8456ea8c3b0c
e79e66db9fa09d230277c415a2fe0b9a6fac0d5fd97466f8e0d0306054173fba5fd7029f5ed2c2
e313d0998a857c1b7bf0fdcc5c79a76d177a3b73b8596a5cf0edafa6f3145b16b8f3a89f331e4
e31dd7bacf6aac6a4ea2067622c9aa120c90a7bf1e0b979e51b7c6b8dbbb1",
            "tenant": "root"
        }
    }
}
```


5、前端拓展

5.1 前提条件

1、首先需要学习如何新建前端扩展 app，熟悉如何新建扩展 app 后即可开始开发，新建 app 名称自拟。新建前端扩展 app 文档地址如下：
http://iidp.chinasie.com:9999/iidpdoc/pages/66a20f/#%E5%88%9B%E5%BB%BA%E6%89%A9%E5%B1%95%E5%BA%94%E7%94%A8%E7%9B%AE%E5%BD%95
2、后端已经扩展好新的设备类型，例如下面示例代码的‘pf4000’设备类型，页面‘设备类型’级联选择器里就会出现新的选项，接下来就可以添加修改以下的前端扩展使得新加的设备类型能够正常配置和使用。
3、前端扩展 app 文件示例：

iiotEdgeConfigAtlas.zip

4、当后端拓展开发完成后，需提供“工程配置模型关系表”关系，示例如下。前端拓展会根据设备类型(如 PF4000)动态加载对应的通道和协议的视图表单

设备类型	通道类型（标红是默认）	通道模型	通道视图	协议模型	协议视图
PF4000	netChannel 、serialChannel	iiot_config_net_channel 、iiot_config_serial_channel	iiot_config_net_channel_form、iiot_config_serial_channel_form	iiot_config_atlas_pf4000_protocol	iiot_config_atlas_pf4000_protocol_form

5.2 扩展代码示例

5.2.1、扩展通讯配置表单视图：

```
// 对接协议的扩展，扩展 openview 打开的视图（扩展名需大于’iiot_thing_entity_communicate_config_empty_openView_extend_demo’,可在后面加_xxx）
 'iiot_thing_entity_communicate_config_empty_openView_extend_demo_xxx': {
   type: 'merge',
   selector: {
     attr: 'id',
     value: 'iiot_thing_entity_menu_tab_2_content'
   },
   beforeOperate: (app, operateItem, options) => {
     return operateItem.view;
```

```
    },
    arrange: { //css 合并、reqAfter 合并
      css: 'concat',
      'view.created.openView.api.loadView.reqAfter': 'concat'
    },
    view: {
      type: 'container',
      items:[],
       //css 不同颜色对应的是不同地方 push 的配置，按照颜色进行对应的修改，如果有增加的就要做对应的增加样式，不同颜色对应的是此扩展里 push 配置的 preId。
      css: `
        #atlas_net_channel_table_detail #atlas_net_channel_form_main_detail_top_header{display: none}
        #atlas_serial_channel_table_detail #atlas_serial_channel_form_main_detail_top_header{display: none}
        #atlas_net_channel_container_form_main_wrap{padding: 0}
        #atlas_serial_channel_container_form_main_wrap{padding: 0}
        #atlas_pf4000_table_detail #atlas_pf4000_form_main_detail_top_header{display: none}
        #atlas_pf4000_container_form_main_wrap{padding: 0}
        #atlas_net_channel_form_main_detail_top_items_1{overflow-y: inherit !important;}
```

```
        #atlas_serial_channel_form_main_detail_top_items_1{overflow-y:
inherit !important;}
        #atlas_pf4000_form_main_detail_top_items_1{overflow-y: inherit !important;}
      `,
      view: {
        created: {
          openView: {
            showType: 'form',
            preId: 'iiot_thing_entity_communicate_config_',
            model: 'iiot_config_iolink_device',
            type: 'form',
            api: {
              loadView: {
                reqAfter: (vm, res) => {
                  let oldGetDefaultValueChannel;;
                  if (typeof
res.data.views.form.body.customView[0].items[0].commands.getDefaultValueChannel
=== 'string') {
                    oldGetDefaultValueChannel = new Function(
                      'return ' +
res.data.views.form.body.customView[0].items[0].commands.getDefaultValueChannel
                    )().bind(window);
```

```
                } else {
                  oldGetDefaultValueChannel =
res.data.views.form.body.customView[0].items[0].commands.getDefaultValueChannel
;
                }
                let newGetDefaultValueChannel = () => {
                  let formVM =
tech_app.page.getNode('iiot_thing_entity_communicate_config_form_main_detail_to
p_common');
                  let ChannelType = '';
                  let netChannelArr = ['PF4000']; //根据设备类型要求，按通道默认值
填写，当前例子 PF4000 设备类型默认的通道是网络通道，有新增的通道也可以自己新增，
下面的 if 逻辑添加上对应的逻辑就好
                  let serialChannelArr = [];
                  let virtualChannelArr = [];
                  if (netChannelArr.indexOf(formVM.$ds.form.DeviceType) !== -1) {
                    ChannelType = 'netChannel';
                  } else if
(serialChannelArr.indexOf(formVM.$ds.form.DeviceType) !== -1) {
                    ChannelType = 'serialChannel';
                  } else if
(virtualChannelArr.indexOf(formVM.$ds.form.DeviceType) !== -1) {
```

```
                    ChannelType = 'virtualChannel';
                  }
                  return ChannelType;
                };
                res.data.views.form.body.customView[0].items[0].commands.getDefa
ultValueChannel = () => {
                  let formVM =
tech_app.page.getNode('iiot_thing_entity_communicate_config_form_main_detail_to
p_common');
                  let DeviceType = formVM.$ds.form.DeviceType;
                  if (DeviceType === 'PF4000') {//填写此扩展所有的设备类型，多个则
用||分隔
                    return newGetDefaultValueChannel();
                  } else {
                    return oldGetDefaultValueChannel();
                  }
                };
                res.data.views.form.body.customView[1].items[1].items.push(
                  {
                    type: 'row',
                    display: (r, {model:t})=>{
                      return t.DeviceType === 'PF4000';//多个则用||分隔
```

```
                    },
                    items: [
                      {
                        type: 'radio-group',
                        name: 'ChannelType',
                        key: 'ChannelType_atlas_01',
                        radioStyle: 'button',
                        defaultValue: 'netChannel',
                        options: [{'text': '网络通道', 'value': 'netChannel'}, {'text': '串口通
道', 'value': 'serialChannel'}],
                        span: 6,
                        onChange: (state, v, {model:t})=>{
                          let
formNote=tech_app.page.getNode('iiot_thing_entity_menu_form_main_detail_top_co
mmon');
                            formNote.$ds.isNotSave=true;formNote.$ds.isFormNotSave
=true;
                        }
                      }
                    ]
                  }, //此块根据设备类型对应的通道进行配置，需要更改的有 display、key、
defaultValue 和 options，如若使用其他的渠道选项可以 push 其他的块。
```

```
                    {
                      type: 'container',
                      display: (r, {model:t})=>{
                        return t.ChannelType === 'netChannel' && t.DeviceType ===
'PF4000';//DeviceType 多个则用||分隔
                      },
                      created: (vm)=>{
                        let formVM =
vm.$select('iiot_thing_entity_menu_form_main_detail_top_common');
                        if (formVM.$ds.form.id){
                          vm.data.view.created.openView.api.form.search.params.args.f
ilter[0][2] = formVM.$ds.form.id;
                        }
                      },
                      items:[],
                      view: {
                        created: {
                          openView: {
                            showType: 'form',
                            preId: 'atlas_net_channel_',//此preId在上面css配置里有用到
（颜色相对应进行修改）
                            model: 'iiot_config_net_channel',
```

```
    type: 'form',
    api: {
      form: {
        search: {
          params: {
            args: {
              filter: [['EntityId', '=', '']],
              properties: ['Channel']
            },
            model: 'iiot_config_iolink_device',
            service: 'search'
          },
          reqAfter: (vm, res)=>{
            if (res.data.length>0&&res.data[0].Channel){
              res.data=[res.data[0].Channel];
            }
            return res;
          }
        }
      }
    }
  }
```

```
                    }
                  }
                }, ////此块根据设备类型对应的通道进行配置，需要更改的有 display、preId，preId 下面的 model 填入所选通道的视图别名（可询问后端），如若使用其他的渠道选项可以 push 其他的块。
                {
                  type: 'container',
                  display: (r, {model:t})=>{
                    return t.ChannelType === 'serialChannel' && t.DeviceType === 'PF4000';//DeviceType 多个则用||分隔
                  },
                  created: (vm)=>{
                    let formVM = vm.$select('iiot_thing_entity_menu_form_main_detail_top_common');
                    if (formVM.$ds.form.id){
                      vm.data.view.created.openView.api.form.search.params.args.filter[0][2] = formVM.$ds.form.id;
                    }
                  },
                  items:[],
                  view: {
                    created: {
```

```
openView: {
  showType: 'form',
  preId: 'atlas_serial_channel_',//此preId在上面css配置里有用到（颜色相对应进行修改）
  model: 'iiot_config_serial_channel',
  type: 'form',
  api: {
    form: {
      search: {
        params: {
          args: {
            filter: [['EntityId', '=', '']],
            properties: ['Channel']
          },
          model: 'iiot_config_iolink_device',
          service: 'search'
        },
        reqAfter: (vm, res)=>{
          if (res.data.length>0&&res.data[0].Channel){
            res.data=[res.data[0].Channel];
          }
          return res;
```

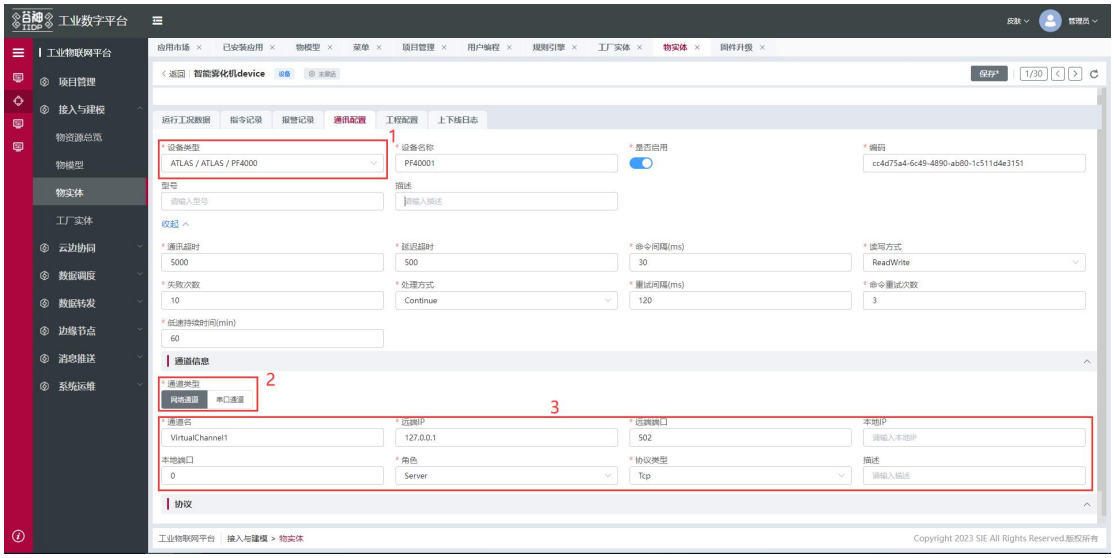
```
                    }
                  }
                }
              }
            }
          }
        }
      }
    );
    res.data.views.form.body.customView[1].items.push(
      {
        type: 'custompanel',
        panelStyle: 'custom',
        title: '协议',
        display: (r, {model:t})=>{
          return t.DeviceType === 'PF4000';//单个协议，扩展的那个协议就
填哪一个
        },
        items: [
          {
            type: 'container',
            created: (vm)=>{
```

```
                                let formVM =
vm.$select('iiot_thing_entity_menu_form_main_detail_top_common');
                                if (formVM.$ds.form.id){
                                  vm.data.view.created.openView.api.form.search.params.ar
gs.filter[0][2] = formVM.$ds.form.id;
                                  }
                               },
                               items:[],
                               view: {
                                 created: {
                                   openView: {
                                     showType: 'form',
                                     preId: 'atlas_pf4000_', //此 preId 在上面 css 配置里有用到
（颜色相对应进行修改）
                                     model: 'iiot_config_atlas_pf4000_protocol',
                                     type: 'form',
                                     api: {
                                       form: {
                                         search: {
                                           params: {
                                             args: {
                                               filter: [['EntityId', '=', '']],
```

```
                properties: ['ProtocolConfig']
              },
              model: 'iiot_config_iolink_device',
              service: 'search'
            },
            reqAfter: (vm, res)=>{
              if
(res.data.length>0&&res.data[0].ProtocolConfig){
                res.data=[res.data[0].ProtocolConfig];
              }
              return res;
            }
          }
        }
      }
    }
  }
]
```

```
                    }//此块根据设备类型协议进行配置，一种设备类型就要添加一块，需要更改的有 display、preId，preId 下面的 model 填入所选设备类型协议的视图别名（可询问后端），如若使用其他的协议可以 push 其他的块。
                    );
                    return res;
                }
            }
        }
    }
}
```

res.data.views.form.body.customView[1].items[1].items 需要 push 的内容规则说明：



例如此设备类型，在前提条件里的‘工程配置模型关系表’里对应的通道类型是两种，也就是上图红框 2 里的选项，此时就需要在表单视图里 push 一块视图。

以下字段是需要修改的：

display：判断哪些设备类型是使用此通道的，一一列上去。

key：保证唯一即可。

defaultValue：填入当前设备类型对应 5.1 章前提条件第 4 点里的‘工程配置模型关系表’里的默认通道。

options：通道选项，若通道类型是这两个的 netChannel、serialChannel，options 设置为[{'text': '网络通道', 'value': 'netChannel'}, {'text': '串口通道', 'value': 'serialChannel'}]，若通道类型是 virtualChannel，options 设置为[{'text': '虚拟通道', 'value': 'virtualChannel'}]，若通道类型为其他则按需修改。

如果需要用到多种不同的通道类型，则多 push 一块以上代码块并进行相对应的修改。

```
{
                type: 'row',
                display: (r, {model:t})=>{
                  return t.DeviceType === 'PF4000';
                },
                items: [
                  {
```

```
                        type: 'radio-group',
                        name: 'ChannelType',
                        key: 'ChannelType_atlas_01',
                        radioStyle: 'button',
                        defaultValue: 'netChannel',
                        options: [{'text': '网络通道', 'value': 'netChannel'}, {'text': '串口通
道', 'value': 'serialChannel'}],
                        span: 6,
                        onChange: (state, v, {model:t})=>{
                          let
formNote=tech_app.page.getNode('iiot_thing_entity_menu_form_main_detail_top_co
mmon');
                            formNote.$ds.isNotSave=true;formNote.$ds.isFormNotSave
=true;
                        }
                      }
                    ]
                  }
```

上图红框 3 里的内容也需要逐一对应 push 进去表单视图，例如此设备类型，在前提条件里的‘工程配置模型关系表’里对应的通道类型是两种，那就要加两块以下的代码块：
一个是‘设备类型为’PF4000，通道选择为 netChannel 时加载的视图
一个是‘设备类型为’PF4000，通道选择为 serialChannel 时加载的视图
以下字段是需要修改的：
display：判断选择‘设备类型为’xxxx，通道选择为 xxxx。
preId：保证唯一即可，会用到 css 配置里，可根据设备类型命名。
model：填入在前提条件里的‘工程配置模型关系表’对应的通道模型。

如果需要用到多种不同的通道类型，则对应多 push 对应数量的代码块进行修改。

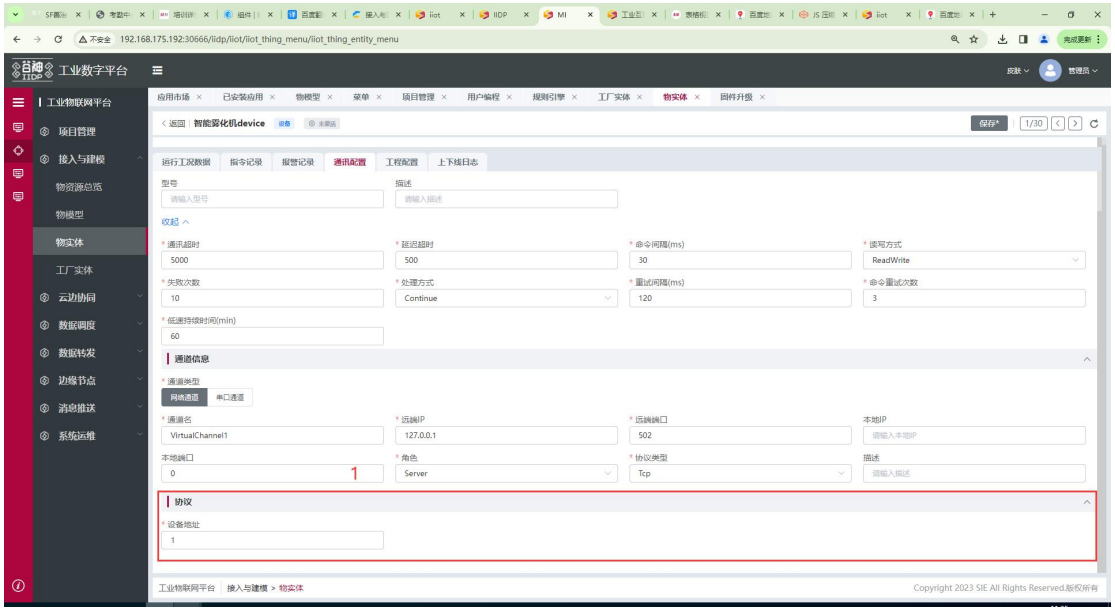
```
{
                        type: 'container',
                        display: (r, {model:t})=>{
                          return t.ChannelType === 'netChannel' && t.DeviceType ===
'PF4000';
                        },
                        created: (vm)=>{
                          let formVM =
vm.$select('iiot_thing_entity_menu_form_main_detail_top_common');
                          if (formVM.$ds.form.id){
                            vm.data.view.created.openView.api.form.search.params.args.f
ilter[0][2] = formVM.$ds.form.id;
                          }
                        },
                        items:[],
                        view: {
                          created: {
                            openView: {
                              showType: 'form',
                              preId: 'atlas_net_channel_',
                              model: 'iiot_config_net_channel',
                              type: 'form',
```

```
      api: {
        form: {
          search: {
            params: {
              args: {
                filter: [['EntityId', '=', '']],
                properties: ['Channel']
              },
              model: 'iiot_config_iolink_device',
              service: 'search'
            },
            reqAfter: (vm, res)=>{
              if (res.data.length>0&&res.data[0].Channel){
                res.data=[res.data[0].Channel];
              }
              return res;
            }
          }
        }
      }
    }
```

```
        }

    },
```

res.data.views.form.body.customView[1].items 需要 push 的内容规则说明：



每增加一个设备类型，就要 push 多一块代码块并进行相对应的修改。
以下字段是需要修改的：
display：判断选择的设备类型为 xxxx。
preId：保证唯一即可，会用到 css 配置里，可根据设备类型命名。
model：填入在前提条件里的‘工程配置模型关系表’对应的协议模型。

```
{

        type: 'custompanel',

        panelStyle: 'custom',

        title: '协议',

        display: (r, {model:t})=>{

          return t.DeviceType === 'PF4000';

        },
```

```
                    items: [
                      {
                        type: 'container',
                        created: (vm)=>{
                          let formVM =
vm.$select('iiot_thing_entity_menu_form_main_detail_top_common');
                          if (formVM.$ds.form.id){
                            vm.data.view.created.openView.api.form.search.params.ar
gs.filter[0][2] = formVM.$ds.form.id;
                          }
                        },
                        items:[],
                        view: {
                          created: {
                            openView: {
                              showType: 'form',
                              preId: 'atlas_pf4000_',
                              model: 'iiot_config_atlas_pf4000_protocol',
                              type: 'form',
                              api: {
                                form: {
                                  search: {
```

```
                                    params: {
                                      args: {
                                        filter: [['EntityId', '=', '']],
                                        properties: ['ProtocolConfig']
                                      },
                                      model: 'iiot_config_iolink_device',
                                      service: 'search'
                                    },
                                    reqAfter: (vm, res)=>{
                                      if
(res.data.length>0&&res.data[0].ProtocolConfig){
                                        res.data=[res.data[0].ProtocolConfig];
                                      }
                                      return res;
                                    }
                                  }
                                }
                              }
                            }
                          }
                        }
                      }
```

```
                    ]

                }
```

5.2.2、保存接口参数修改

```
// 保存入参 reqPrep 修改（扩展名需大于’iiot_thing_entity_extend_save_reqPrep_extend_demo’,可在后面加_xxx）


  'iiot_thing_entity_extend_save_reqPrep_extend_demo_xxx': {
    type: 'custom',
    selector: {
      attr: 'id',
      value: 'iiot_thing_entity_menu_form_main_detail_top_common'
    },
    beforeOperate: (app, operateItem, options) => {
      let tableDataDsConfig = options.element?.ds_config?.list.find(
        (item) => item.name === 'formUpdate'
      );
      let oldReqPrep = tableDataDsConfig.reqPrep;
      let newReqPrep = operateItem.view?.ds_config.list[0].reqPrep;
      operateItem.view.ds_config.list[0].reqPrep = async (vm, options, config) => {
        let detailVM = tech_app.page.getNode('iiot_thing_entity_menu_table_detail');
        if (detailVM.$ds.clickType === 'edit') {
```

```
            let formVM =
tech_app.page.getNode('iiot_thing_entity_communicate_config_form_main_detail_to
p_common');
            let DeviceType = formVM?.$ds.form.DeviceType;
            if (DeviceType === 'PF4000') { //列出此扩展所有的设备类型协议，多个则用||
分隔，作用是当选中您扩展的协议才进入你的方法。
              options = await newReqPrep(vm, options, config);
            } else {
              options = await oldReqPrep(vm, options, config);
            }
          }
          return options;
        };
        tableDataDsConfig.reqPrep = operateItem.view?.ds_config.list[0].reqPrep;
        return operateItem.view;
      },
      view: {
        ds_config: {
          list: [
            {
              name: 'formUpdate',
              reqPrep: async (vm, options, config) => {
```

```
                let extendResOne;
                let extendResTwo;
                let extendResThree;
                let edgeConfig;
                if (options.params.args.values.entity_type !== '3') {
                  try {
                    // 基础信息
                    if
(tech_app.page.getNode('iiot_thing_entity_communicate_config_form_main_detail_to
p_common')?.instance) {
                      extendResOne = await
tech_app.page.getNode('iiot_thing_entity_communicate_config_form_main_detail_to
p_common').instance.submit();
                    }
                    // 通道信息（需要把所有用到的通道列出来，不同颜色对应的是上一个扩
展里 push 配置的 preId）
                    if
(tech_app.page.getNode('atlas_net_channel_form_main_detail_top_common')?.instan
ce) { // 网络通道
                      extendResTwo = await
tech_app.page.getNode('atlas_net_channel_form_main_detail_top_common').instance
submit();
```

```
                    } else if
(tech_app.page.getNode('atlas_serial_channel_form_main_detail_top_common')?.inst
ance) { // 串口通道
                        extendResTwo = await
tech_app.page.getNode('atlas_serial_channel_form_main_detail_top_common').instan
ce.submit();
                    }
                    // 协议信息（需要把所有用到的协议列出来，不同颜色对应的是上一个扩
展里 push 配置的 preId）
                    if
(tech_app.page.getNode('atlas_pf4000_form_main_detail_top_common')?.instance)
{ // PF4000 协议
                        extendResThree = await
tech_app.page.getNode('atlas_pf4000_form_main_detail_top_common').instance.sub
mit(); //可以在后面写校验，校验不通过的 return ‘break’
                    }
                } catch {
                    window.ELEMENT.Message.error('请填写完整通讯配置');
                    let activeTabId = 'tab-iiot_thing_entity_menu_tab_2_content';
                    document.getElementById(activeTabId).click();
                    return 'break';
                }
```

```
                edgeConfig = {Channel: {...extendResTwo}, ProtocolConfig:
{...extendResThree}, ...extendResOne};
                options.params.args.values.edgeConfig = edgeConfig;
                let iotagsTableVM =
tech_app.page.getNode('iiot_thing_entity_menu_tab_io_tags_table_main_table');
                if (iotagsTableVM?.instance) {
                  options.params.args.values.io_tags =
iotagsTableVM.$ds.tableData.data;
                }
              } else {
                let variablesTableVM =
tech_app.page.getNode('iiot_thing_entity_menu_tab_variables_table_main_table');
                if (variablesTableVM?.instance) {
                  options.params.args.values.variables =
variablesTableVM.$ds.tableData.data;
                }
              }
              return options;
            }
          }
        ]
      }
```

```
  }

}
```

以上需要修改的通道和协议表单节点 id 均可在页面上可通过检阅元素找到。