表情服务

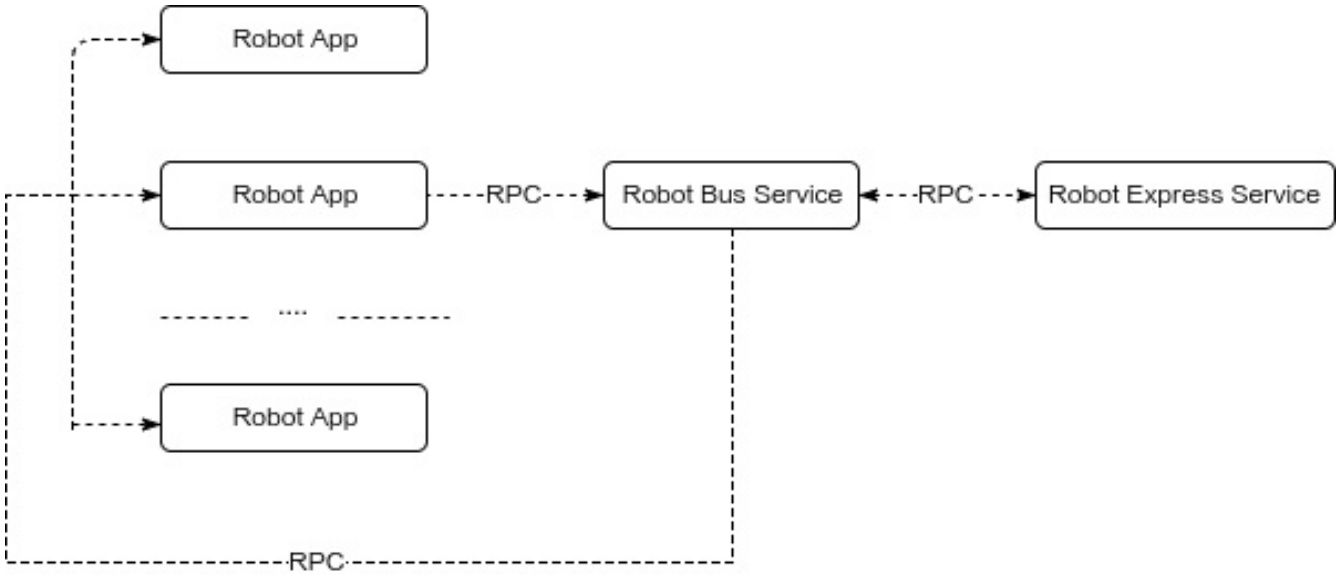
修改历史

Version	Contributor	Date	Change Log
v1.0	彭钉	17/10/06	初版

目录

表情服务结构设计

1、表情服务\总线\应用关系示意图



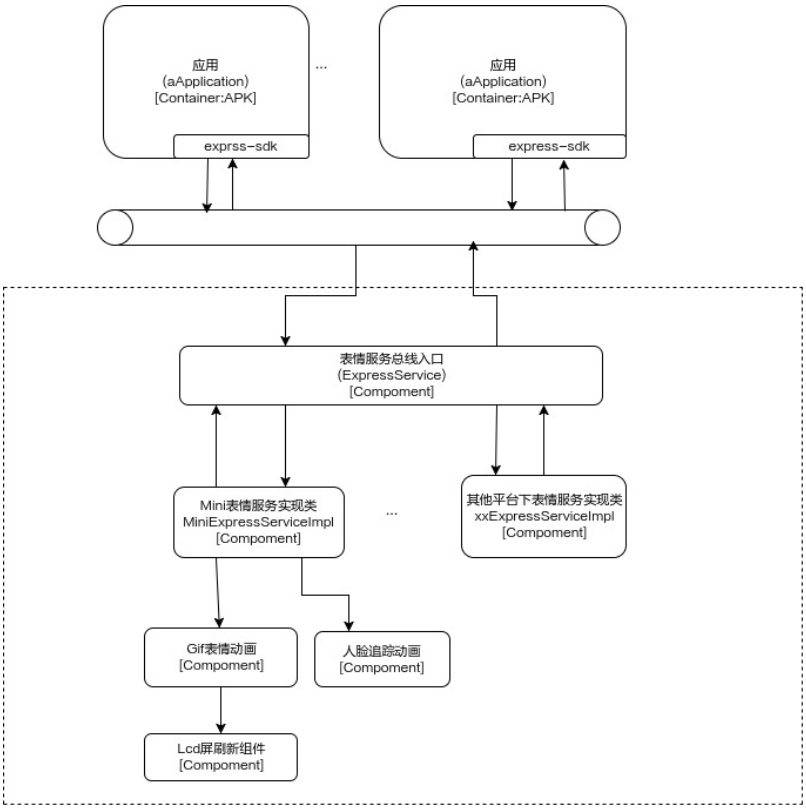
说明：

- Robot Bus Service --总线服务进程
- Robot App -- 机器人端应用进程

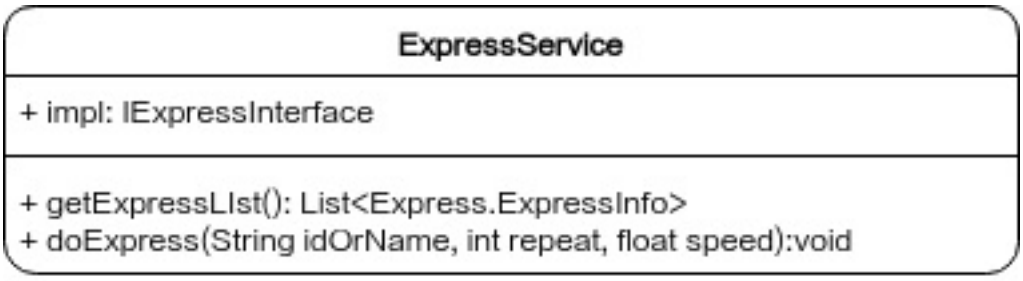
- Robot Express Service -- 表情服务进程

2、表情服务设计

- 内部组件图



- sdk对外接口定义



3、表情服务接入总线

- 声明表情服务

```
<application>
    <service android:name=".ExpressService">
```

```
        <!-- name：服务名称，应用调用哪个服务的标识 -->
        <meta-data android:name="name"
android:value="MotionService" />
        <!-- action 用于启动总线服务，label & description 描述服务的信息
-->
        <meta-data android:name="label" android:value="@string/label"
/>
        <meta-data android:name="description"
android:value="@string/description" />
        <intent-filter>
            <!-- host & scheme 值统一固定，path 表示某个调用 -->
            <action android:name="ubtrobot.service_bus.action.SERVICE"
/>
            <data android:host="express.ubtrobot.com"
android:scheme="call" android:path="/express/doExpress"/>
            <data android:host="express.ubtrobot.com"
android:scheme="call" android:path="/express/getExpressList"/>
            ...
      </service>
      <meta-data
        android:name="mst-conn-mod"
        android:value="MST_SERVICE" />
</application>
```

定义表情服务消息格式

```
syntax = "proto3";
option java_package = "com.ubtrobot.express.protos" ;

 //响应参数
message ExpressListRes{
    repeated ExpressInfo express = 1;
 }

message ExpressInfo {
```

```
    int32 id = 1;//id
    string name = 2;//名称
    int32 duration = 3;//时长
}

//请求参数
message DoExpressReq {
    string name = 1;
    float speed = 2;
    int32 repeat = 3;
}
```

- 实现ExpressService

```
// 总线服务实现
@BusService(name = "express")
public final class ExpressService extends AbstractCallable {
    private IExpressInterface impl;
    //...
    @DebugLog
    @Call(path = "/getExpressList")
    public void onGetExpressList(Request request, Responder
responder){
        try {
            List<Express.ExpressInfo> expressInfos  = impl.getExpressList();
            if (expressInfos == null) {

responder.respondFailure(CallGlobalCode.INTERNAL_ERROR, "call
fail: 表情解析失败");
            }else {
                responder.respondSuccess(ProtoParam.pack(
                        Express.ExpressListRes.newBuilder()
                                .addAllExpress(expressInfos)
                                .build()));
            }
        }catch (Exception e){
            e.printStackTrace();
```

```
            responder.respondFailure(CallGlobalCode.INTERNAL_ERROR,
"call fail : " + e.getMessage());
        }
    }

    @DebugLog
    @Call(path = "/doExpress")
    public void onDoExpress(Request request, Responder responder){
        try {
            String value =
ProtoParam.from(request.getParam()).unpack(StringValue.class).getVal
ue();
            int code = impl.doExpress(value);
            if (code == 0) {
                responder.respondSuccess();
            }else {
                responder.respondFailure(CallGlobalCode.BAD_REQUEST,
"表情不支持");
            }
        } catch (ProtoParam.InvalidProtoParamException e) {
            e.printStackTrace();
            responder.respondFailure(CallGlobalCode.BAD_REQUEST, "call
fail : " + e.getMessage());
        }catch (Exception e){
            e.printStackTrace();
            responder.respondFailure(CallGlobalCode.INTERNAL_ERROR,
"call fail : " + e.getMessage());
        }
    }
}
```