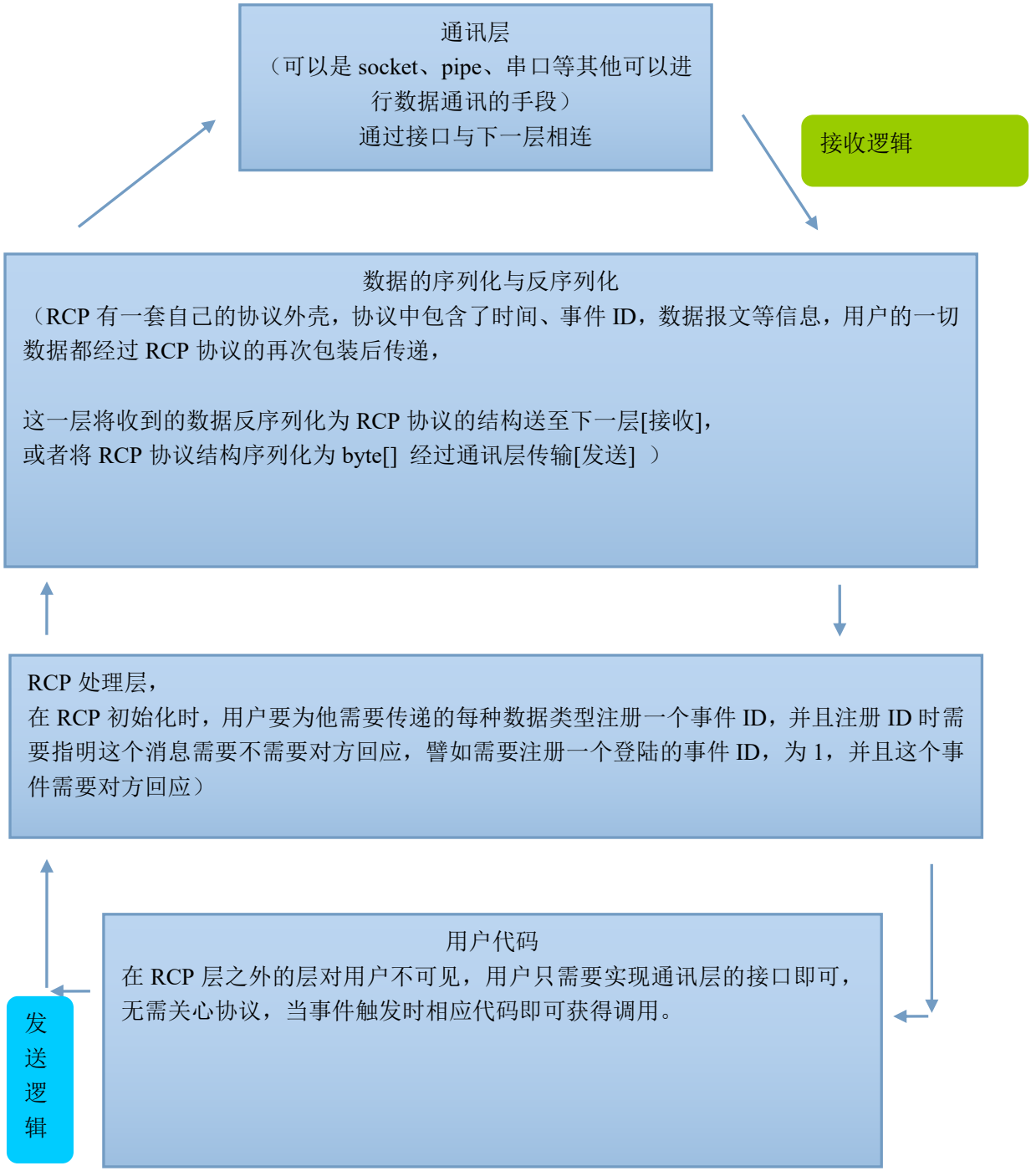
xlang 中 RCP 的设计说明

文档库地址 https://xlang.link/documents/index.html

xlang 中有一个 RCP（Remote Call Pipe）类，类似的现有主流设计都称为 RPC（Remote Procedure Call），之间略有不同。

RCP 的设计组成:

通讯层 -> 序列化与反序列化 -> 转换为逻辑事件 -> 激活用户代码



释义: 网络上的传输绝大多数情况下是为了在远程执行一个功能，

譬如 用户登录事件：

客户端代码：

```
bool login(String name, String pws){
   ///// 进行登录操作
}
```

服务器相应的代码:

```
void onLogin(String name, String pws){
   //// 去到数据库查表   然后发送给客户端结果
}
```

而在这两个函数之间的过程用户是不关心的，但是在没有现有可用资源的情况下，用户必须通过写 socket 或者其他手段，自己去实现这个传输过程。

RCP 的主要目的就是用来解决这两个函数之间的其他用户不用关心的过程，
以达到调用像调用本地代码一样的调用远程代码。

xlang 中的 RCP 类使用:

客户端:

```
RemoteCallPipe<IClient, IServer>   rcs   =   new RemoteCallPipe<IClient, IServer>();
```

RemoteCallPipe 类是一个特殊的模板类，这个特殊的模板类需要两个模板参数，远端和本地端，IClient 是本地端 IServer 是远端的接口(interface)，此代码实现了一个 RCP 实例，

IClient 的代码：

```
interface IClient{
   void onLoginConflict(long context);
};
```

IServer 的代码:

```
interface IServer{
      int login(long context, String name, String pwd);
};
```

服务端:

```
RemoteCallPipe<IServer,IClient>  rcs  =  new RemoteCallPipe<IServer,IClient>();
```

IClient、IServer 与客户端代码相同。

服务端的 RCP 模板参数与客户端的模板参数是相反的，RemoteCallPipe 类会自动实现第二个模板参数的所有方法，而第一个模板参数需要用户实现并手动设置，

所以需要在客户端实现 IClient 接口中的方法，在服务端代码中需要实现 IServer 中的方法。

客户端需要继承 IClient 类并实现：

```
class Client : IClient{
   void onLoginConflict(long context){
      // 提示：你的账户已经在其他地方登陆
   }
};
```

服务端需要继承 IServer 接口并实现

```
class Server : IServer{
      int login(long context, String name, String pwd){
            db.query(....)
            登陆成功 返回 0，不成功返回其他对应的代码.
      }
};
```

需要写的代码就只有这么多了， 剩下只需要给 RCP 设置一个通讯接口，

RCP 需要一个异步的数据传输接口 AsyncOutput:
客户端：只需要使用一个 StreamSocket 即可，然后把 socket 的数据收发与 RCP 对接在一起，用户甚至无需手动处理 send recv，全都丢给 RCP 就可以了。

```
//继承 AsyncOutput 接口并实现其方法
class XAsyncOutput : AsyncOutput{
    // new 一个 StreamSocket ， TCP 方式
    StreamSocket _socket = new StreamSocket();
    bool create(String host, int port){
        // 写一个 create 方法，来提供与服务器建立连接
        return _socket.connect(host, port);
    }
    void readloop(){
            new Thread(){
                void run()override{
                    byte [] buffer = new byte[1024];
                    int rd = 0;
                    // 开启一个线程读 socket,当有数据到达时直接扔给 RCP
                    while ((rd = _socket.read(buffer, 0, 1024)) > 0){
                        rcs.getAsyncDirectInput().dataArrives(this, 0, buffer, 0, rd);
                    }
                }
            }.start();
    }
    // AsyncOutput 接口的数据发送接口， 直接扔给 StreamSocket 去 write
    bool dataDeparture(AsyncInput input, long context, long bind, byte [] buffer ,int offset, int length){
        return (length == _socket.write(buffer, offset, length));
    }
    // AsyncOutput 的链接中断接口，无需处理
    void deleteInstance(AsyncInput input, long context){}
    // 新建链接功能 会返回一个 long 的 id 作为后续事件中的相关性识别
    long newInstance(AsyncInput input, String ip, int port, long bind){return 0;}
    long getInstanceId(long context){return 0;}
    long cloneContext(long context){return 0;}
    void releaseContext(long context){}
};


rcs.create(ao, ..., new Client, false);
// 使 RCP 管道创建
```

服务端：需要用同样的方法将 Unis（通用服务器接口服务与在服务端的 RCP 对接然后创建.）

接下来看看怎么使用：

现在有一个用户需要登录了，用户名为 xiaoming, 密码为 123456

客户端 代码:

```
int res = rcp.login(0, “xiaoming”, “123456”);
```

当程序执行到这一句的时候，服务端中的 login 函数将被调用,与此同时,客户端将挂起.

```
class Server : IServer{
    int login(long context, String name, String pwd){
        ...
        return 0;//登陆成功 返回 0
    }
};
```

现在服务端返回了代码 0， 客户端线程重新激活，并且拿到了返回值 res = 0。

整个调用、序列化、传输、以及事件通讯过程无需用户处理，用户就和在调用本地代码一样。

在 xlang 中 RCP 接口被做了更深一步的处理，直接将其变为模板语法，由 xlang 编译器对其接口进行实现，但不生成额外代码，用户无需显式注册事件 ID，由 xlang 编译器生成事件 ID 并绑定对应的方法，从而达到在事件被触发时，对应的方法将被准确调用，实现了 C/S 代码逻辑上的无缝连接。

详情查看 XStudio 向导模板: