Hermes Cpp Design

C++重构方案参考

Revision 1.0.0.1 revised by 吴塞 2022/04/13

*本文档观点仅供参考

1. 重构目标

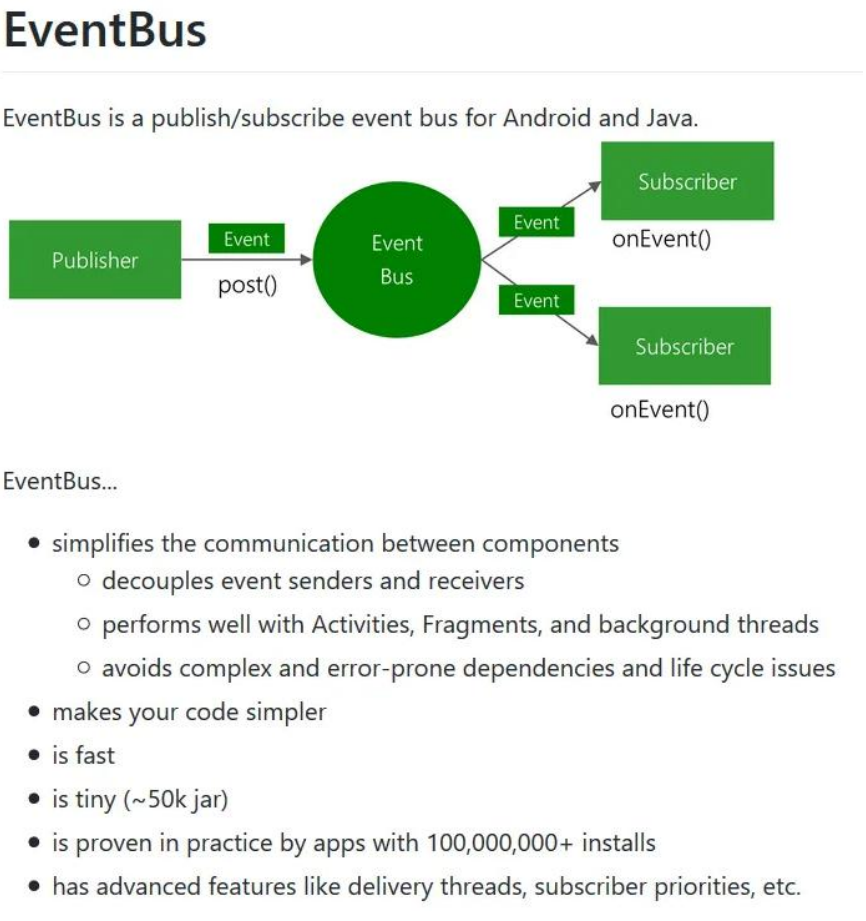
对代码进行重构划分模块，便于快速定位生产中的 bug，提高代码的可读性，对模块进行版本管理从而便于扩展与维护，避免出现“动一发而牵全身”的问题出现。

2. 常用的重构解耦的设计方法和模式

***基本设计原则**：既要解耦，也要根据实际情况，避免过度设计造成的维护困难，以下会介绍一些常用的设计思路和模型。

2.1 事件与回调的常用模型

(1) **EventCenter**(EventBus)与 **SubScribe**，这是一个 Android 的事件转发模型，模块之间彼此不可见，通过一个事件总线(事件中心)进行转发，Subscribers(订阅者)可以接收处理自己感兴趣的事件，详情参见 https://github.com/greenrobot/EventBus，具体实施我们可以使用 c/c++实现该模型，它的本质设计思路相当于汇编语言中的中断向量表，注意：实施过程中要注意线程安全问题。此模型相当于观察者模式的升级版，增加了集中式管理的概念，传统的观察者模式的缺点是使用不当会搞得回调成员变量遍布源码的各个角落和 virtual 函数定义数量过多，造成代码混乱。



(2) Route Document 模型，此模型和 **EvnetBus** 相类似，不过增加了文档描述更加规范化，该模型会将事件和处理过程的对应关系预先写在文件(**可以是 PlainText 也可以是 xml 或者 json 等，一般推荐 xml 描述**)里文档化，当主程序启动后，会动态解析文档并绑定(Bind)事件和处理过程(Callee or Procedure or Callback)，处理过程的实体可以是函数指针，模板闭包(Closure)，lambda 表达式等。此方法的优点是用文档表示事件和处理过程(Callee or Procedure or Callback)的对应关系，将业务处理逻辑进行更深一步的规范，代码逻辑不会混乱，所有开发人员遵循此框架编程开发，不会出现各种混乱的调用方式，缺点是过度设计造成的代码可读性变差简单问题复杂化，增加维护成本。[xml 使用得当会使应用程序更加规范，代码美观程度增加]

(3) 通用(Plugin)插件接口模型，此编程模型严格遵循插件规范，动态库的导出接口受到严格限制，模块间私有变量不可见，严格按照进程内通信规范传递参数，可以视为进程内通信， 也可以按照 Client 和 Server 的规则设计。轻量级的插件接口设计可以参考 libplugin，重量级规范程度比较高的可以参考例如比较古老的接口规范设计可以参考经典的微软 ActiveX 插件规范和 NPAPI 插件规范，或者大型软件诸如 3DMAX 或 MAYA 的插件接口设计。我们的情况简单一些，只需要定义一些简单的接口规范，不必像这些大型软件这么复杂，使用该模式，耦合度最低，查找定位 bug 最方便，方便组装和拆卸应用程序，缺点是可能过度设计，会造成重构的开发周期变得很长调用和数据共享方式和调用方式怪异等缺点，以下举例说明某 Plugin 动态库导出唯一函数为 exec，只用一个导出函数可以实现该模块内任意方法调用传参以及回调，参数 obj 为对象 id, cmd 为方法 id, arg 保存参数，callback 可以为类成员函数或者接口类指针，这种使用方法比较理想化将模块间的耦合度降至最低，但是却有使用时有很多不便利和语义不清的缺点，根据具体使用情况，可以设计一些更符合应用场景的接口来进行使用。

```
extern "C" int __declspec(dllexport) cdecl exec(
    void* obj, void* cmd, void* arg, void* callback)
```

(4) Qt 的信号与槽，信号与槽设计初衷和 EventBus 有些类似，为了解决传统观察者模式，包含类成员变量回调和互相包含类成员变量回调 (可以视为耦合的一种) 随着项目膨胀代码逐渐变得臃肿和阅读性差的缺点，用一种较为直观简洁的方式进行回调，该机制一定程度上简洁了代码和降低了耦合，但是缺少集中式管理和规范化使用的思想，connect 可以出现在源码的任意位置，使用不当也会造成代码混乱和可读性差的缺点。

(5) 使用代码结构及目录结构解耦，良好的源码子目录划分、源文件层次设计和头文件引用树的规划，可以自然使代码模块与结构变得清晰、可维护性高，(有些私有变量其实可以移到 cpp 文件里用其它方式保存，头文件过长变量过多也会增加代码的维护成本)目前 DataSvr 主要代码堆在一个目录，头文件引用树的结构混乱，这也是重构解耦需要解决的一个问题，良好的代码习惯不但自带解耦功能，还会使开发人员心情良好，提高工作效率。

*总结：以上列举了一些常用的并且可以简单实施的解耦方法，实际应用中其实并没有完美的方法只有根据实际情况使用较为合适的方法。目前 DataSvr 有很多直接调用全局变量的情况，其实也未必完全不可取，毕竟有时候代码简洁直观并且可以解决实际问题也是很好

的一种方法。在应用中有些不必要的情况应该避免追逐流行大厂框架等过度设计带来的一些不便。