服务追踪状态信息传递

由于服务化之后，一个请求将通过多个子系统协同处理完成。任何一个子系统出现故障，都将会影响服务的正常运行。因此，如何从多个子系统中快速定位问题，将会是服务化中非常重要的一项功能。

业界的普遍做法，采用类似trace\_id或log\_id的标注法，在请求的来源为每个请求打个标记，然后该标记将会传递到每个子系统中。在具体追查问题时，只要通过trace\_id或log\_id把流向各个子系统的所有请求日志聚合分析，即可方便的进行问题定位。这就引出了这些追踪信息（trace\_id或log\_id）如何传递的问题？

鉴于我们采用thrift作为开发框架，而thrift本身不支持类似trace\_id或log\_id的机制。最直观的解决方案，就是定制thrift的transport，增加自定义头部。调用方调用的时候，把状态信息通过自定义头部传递到服务后端。

那自定义头部如何定义呢？通常的做法就是增加一个结构体：

type XmHeader struct {

RpcId int32 `"rpcId,1" json:"rpcId"`

LogId int32 `"logId,2" json:"logId"`

AppId string `"appId,3" json:"appId"`

BodyLen int32 `"bodyLen,4" json:"bodyLen"`

}

如果每次都发送一个固定长度的头部结构体，就会存在可扩展问题。比如,以后头部要增加字段时，客户端和服务端必须同时升级，否则将会出现无法服务的情况。考虑到thrift本身支持多版本的平滑升级，因此，我们的自定义头部通过thrift的binary protocol进行序列化，然后发送给后端，如下图所示：

包体，由配置的具体的协议序列化

采用binary协议

序列化头部

当自定义头部需要新增字段时，只需要在xmheader.thrift定义的xmheader结构体，增加一个字段。然后采用thrift生成序列化的代码ttypes.go覆盖xmheader.go即可。

注意：编译的时候可能会出现一些错误: undefined: thrift，因为我们生成的代码和thrift是同一个包里。所以把thrift包引用删除即可。