游戏业务通信协议架构

- 通信端是否需要保存通信状态
  - 无状态的 (stateless)
    - 无连接的 (connectionless)
    - 持久连接 (persistent connection)
    - 非持久连接
  - 有状态的 (stateful)

游戏业务通信协议架构

- 可靠性
  - 至多一次 (at most once)
  - 至少一次 (at least once)
    - 幂等的 (idempotent)
    - 非幂等的
  - 不可靠的 (unreliable)
    - 无确认的
    - 有确认的

无状态协议

- 定义
  - 任意单一请求，效果不依赖其上下文
- 示例
  - HTTP
    - 面向连接的，可靠的，有确认的
  - DNS
    - 无连接的，不可靠的，有确认的

无状态协议

- 优点
  - 职责边界明确
  - 不需要状态回滚，容易实现断线重连
- 缺点
  - 所有参数必须在同一请求中传输
    - 消息大小增加
    - 可扩展性差

幂等协议

- 定义
  - 重复发送的相同请求，效果和只发送一次相同
- 幂等协议示例
  - 读取玩家名字
  - 设置玩家名字
- 非幂等协议示例
  - 给玩家加 100 金币

幂等协议

- 幂等变换
  - 非幂等协议
    - 给玩家加 100 金币
  - 等价的幂等协议
    - 生成订单“给玩家加 100 金币”
    - 提交订单

幂等协议

- 优点
  - 简化断线处理，增加可靠性
  - 对于只读请求，可缓存响应数据
- 缺点
  - 流程复杂化
    - 通信消息量增加
    - 延迟增加

协议分类

- 无状态的、幂等的
  - 获取玩家信息
- 无状态的、非幂等的
  - 购买道具
- 有状态的、幂等的
  - 进入副本
- ~~有状态的、非幂等的~~

游戏业务通信协议架构

- 消息分类
  - 客户端 >= 服务器：请求消息
    - 无状态的、非幂等的
  - 客户端 <= 服务器：响应消息
    - 和请求消息一一对应，所有流程公用
  - 客户端 <= 服务器：通知消息
    - 无状态的、非幂等的
  - 客户端 <= 服务器：数据消息
    - 有状态的、幂等的

游戏业务通信协议架构

- 玩家用例
  - 从点击某某按钮到看到某某为止
- 程序实现
  - 从发送某某请求到收到对应的响应为止
- 大干快上方针
  - 用尽可能少的代码完成需要的功能
  - 同时保证一定的可扩展性和可维护性

购买道具业务流程

- 某玩家要买 5 个飞镖
  - 每个飞镖 100 银币
  - 该玩家有 6 个飞镖和 3000 银币
- 客户端 >= 服务器
  - 请求消息: `Req_BuyItem { serial = 387, item = "dart", count = 5 }`
- 客户端 <= 服务器
  - 数据消息: `Data_Item { item = "dart", count = 11 }`
  - 数据消息: `Data_Item { item = "money", count = 2500 }`
  - 通知消息: `Ntfy_ItemChanged { item = "dart", delta_count = 5 }`
  - 通知消息: `Ntfy_ItemChanged { item = "money", delta_count = -500 }`
  - 响应消息: `Resp_BuyItem { serial = 387, err_code = 0 }`

消息顺序问题

- 服务器到客户端的数据消息是有状态的
  - 必须保证在客户端收到通知消息时数据可用
- 增加和更新数据
  - 先发增加和更新数据消息，再发通知消息
- 删除数据
  - 先发通知消息，再发删除数据消息

客户端多线程处理

- 网络线程
  - 把接收到的所有消息转发给数据线程
- 数据线程（Model/Controller）
  - 处理数据消息
  - 把通知消息和响应消息转发给 UI 线程
  - 触发 UI 更新事件
- UI 线程（View）