高可用架构设计与实践

讲师：孙玄@58

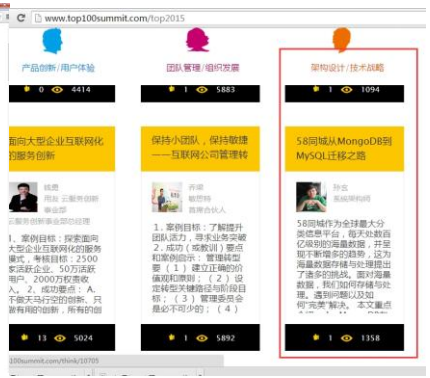
法律声明

关于我

- 58同城高级系统架构师
- 公司技术委员主席
- 即时通讯、转转、C2C技术负责人
- 前百度高级工程师
- 代表58同城对外嘉宾分享
  - QCon
  - SDCC
  - DTCC
  - Top100
  - 程序员
  - UPYUN
  - TINGYUN
  - ……

代表58对外交流

- Qcon(全球软件开发大会)
- SDCC(中国开发者大会)
- Top100(全球案例研究峰会)
- DTCC(中国数据库技术大会)
- 《程序员》撰稿2次
- 58技术发展这10年[计划中]



炼数成金课程

课程

- 《MongoDB实战》
  - 已开课
  - 欢迎大家报名学习
- 《大规模高性能分布式存储系统设计与实现》
  - 已开课
  - 欢迎大家报名学习

上节课回顾

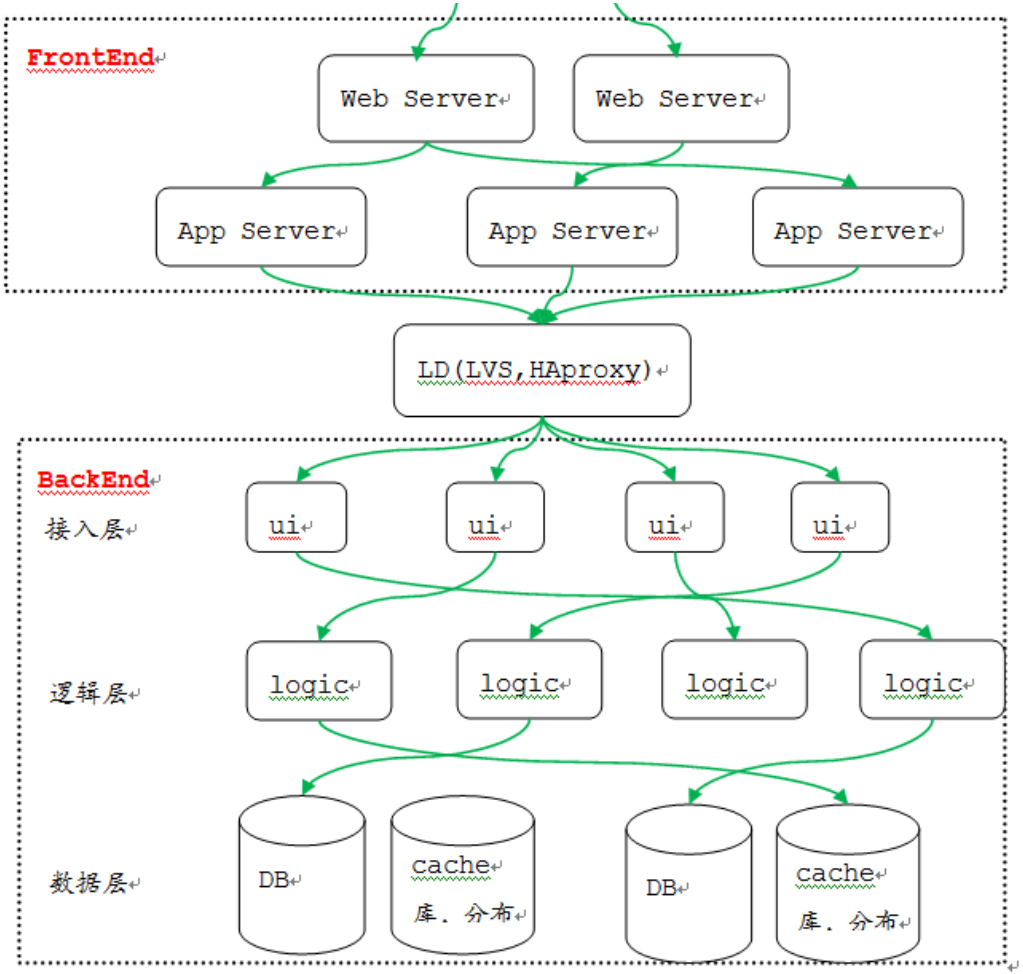
- 互联网产品通用技术架构
- 接入层的作用是什么
- 接入层Session如何设计?
  - Session复制、Session绑定、Session记录方式、Session高可用等
- 接入层数据安全如何保证?
  - 对称加密、非对称加密、多种方法使用等
- 接入层数据正确性如何保证?
- 接入层高可用设计方案?
  - 无状态、动态扩展
- 接入层高可用设计最佳实践是什么?
- 我们的实践案例;

OutLine

- 逻辑层都做什么?
- 逻辑层整体架构设计
- 无状态业务逻辑层如何设计?
- 业务逻辑层如何纯异步调用?
- 业务逻辑层如何分级管理?
- 业务逻辑层如何设置合理的超时?
- 业务逻辑层服务降级如何设计?
- 业务逻辑层如何做到幂等设计?
- 业务逻辑层高可用设计最佳实践是什么?
- 我们的实践案例;

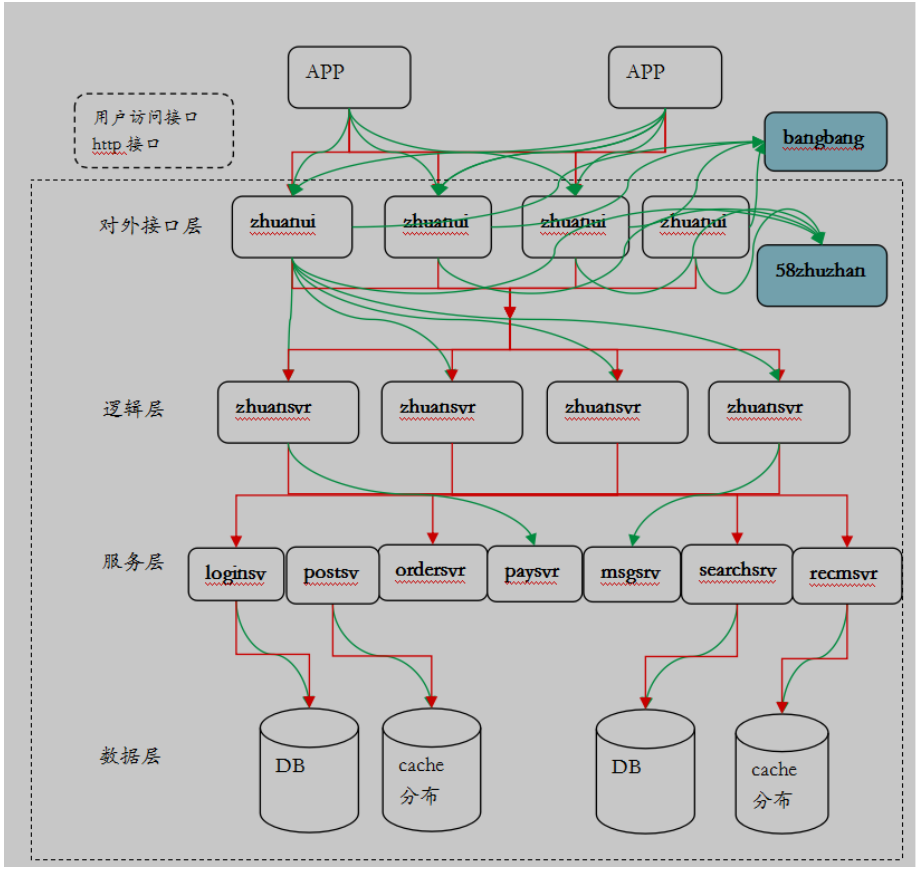
互联网产品通用技术架构

- 接入层
- 逻辑层
- 数据层



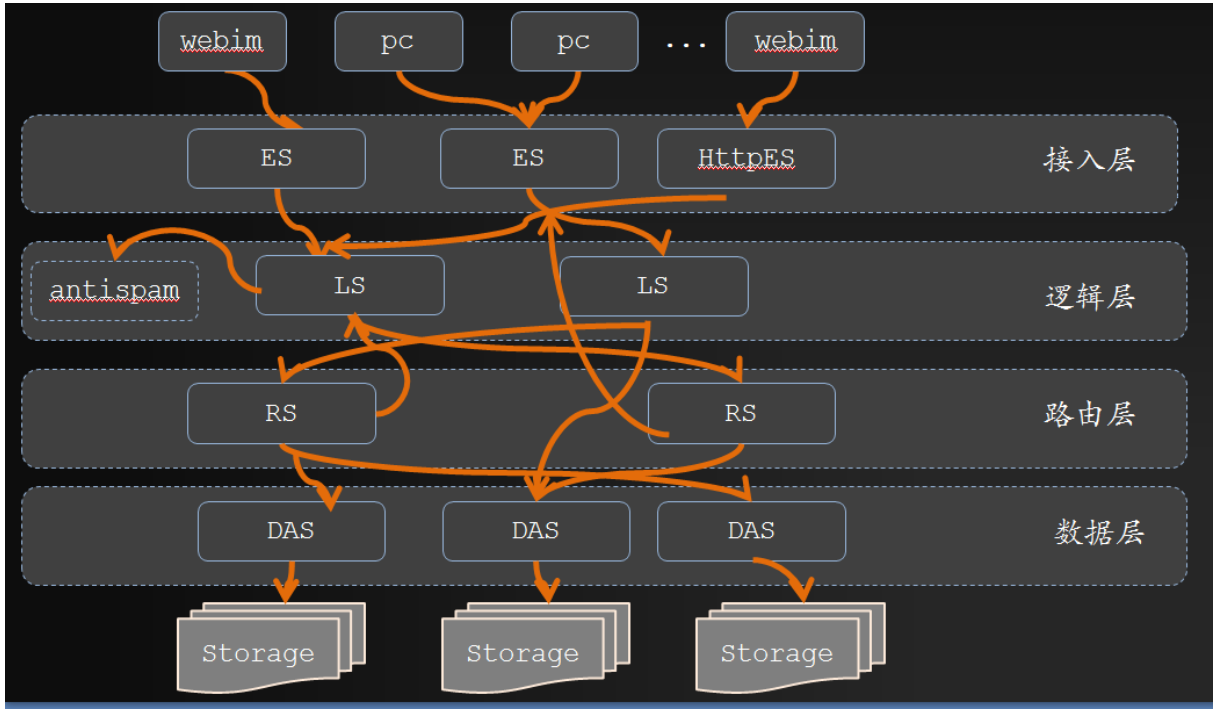
互联网产品通用技术架构

- 接入层
- 逻辑层
- 数据层



互联网产品通用技术架构

- 接入层
- 逻辑层
- 数据层



逻辑层都做什么

逻辑层职责

- 整个系统中业务逻辑的处理
- 58帮帮为例
  - 用户相关（用户登录登出、用户信息设置查询）
  - 好友相关（添加好友、获取好友、删除好友、修改好友信息等）
  - 消息相关（收发好友消息、收发陌生人消息、消息确认、通用消息处理、离线消息等）

逻辑层整体架构

- 逻辑层业务多
- 业务层逻辑复杂
- 逻辑层如何设计?
  - ALL IN ONE方式
    - 所有业务一个整体
    - 一个文件里
    - 一个类里
    - ……

逻辑层整体架构

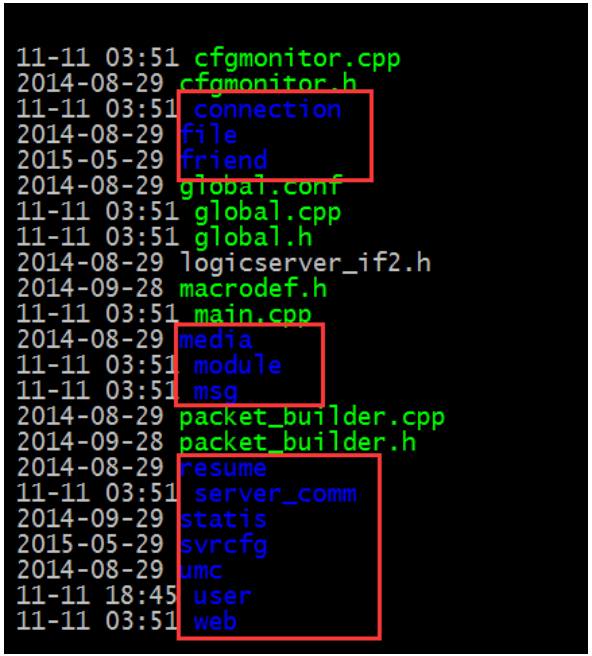
逻辑层整体架构

- 逻辑层如何设计?
  - ALL IN ONE方式
    - 有什么问题?
      - 文件太复杂
      - 耦合性严重
      - 开发代价高
      - 维护代价高
      - 牵一发动全身
    - 适合场景
      - 创业期
      - 业务不复杂

逻辑层整体架构

逻辑层整体架构

- 逻辑层如何设计?
  - ALL IN ONE（业务垂直划分）方式
    - 业务垂直划分
      - » 一个单独业务一个组件（目录和文件）



逻辑层整体架构

- 逻辑层如何设计?
  - ALL IN ONE（业务垂直划分）方式
    - 优点
      - 业务独立
      - 耦合性降低
      - 业务之间开发互不影响
      - 开发效率高
      - 运维相对简单

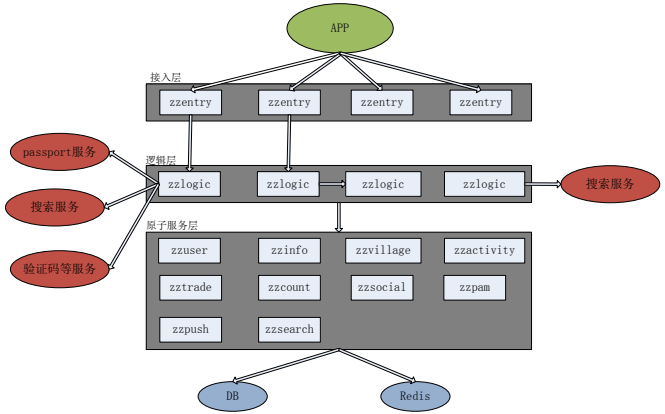
逻辑层整体架构

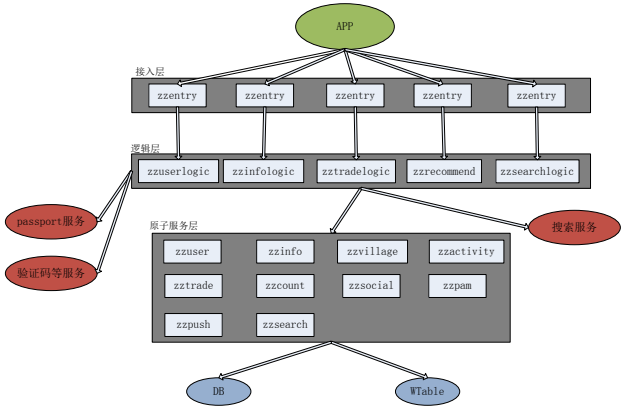
- 逻辑层如何设计?
  - ALL IN ONE（业务垂直划分）方式
    - 缺点
      - 物理上一个模块
      - 编译成本高
      - 一个业务修改，重新上线
      - 重启影响所有业务
    - 适用场景
      - 互联网公司使用较多
      - 58
      - 百度

逻辑层整体架构

逻辑层整体架构

- 逻辑层如何设计?
  - 业务间物理垂直划分方式
    - 每个业务一个独立的业务模块（进程）





逻辑层整体架构

逻辑层整体架构

- 逻辑层如何设计?
  - 业务间物理垂直划分方式
    - 优点
      - 业务间完全解耦
      - 业务间互不影响
      - 业务模块独立
      - 单独开发、上线、运维
      - 效率高
    - 适用场景
      - 互联公司使用多
      - 58
      - 赶集

18

无状态业务逻辑层如何设计

什么是无状态

- 系统不存储业务的上下文信息
- 仅根据每次请求携带数据进行相应的业务逻辑处理
- 多个模块（子系统）之间完全对称
- 请求提交到任何服务器，处理结果都是完全一样

无状态业务逻辑层如何设计

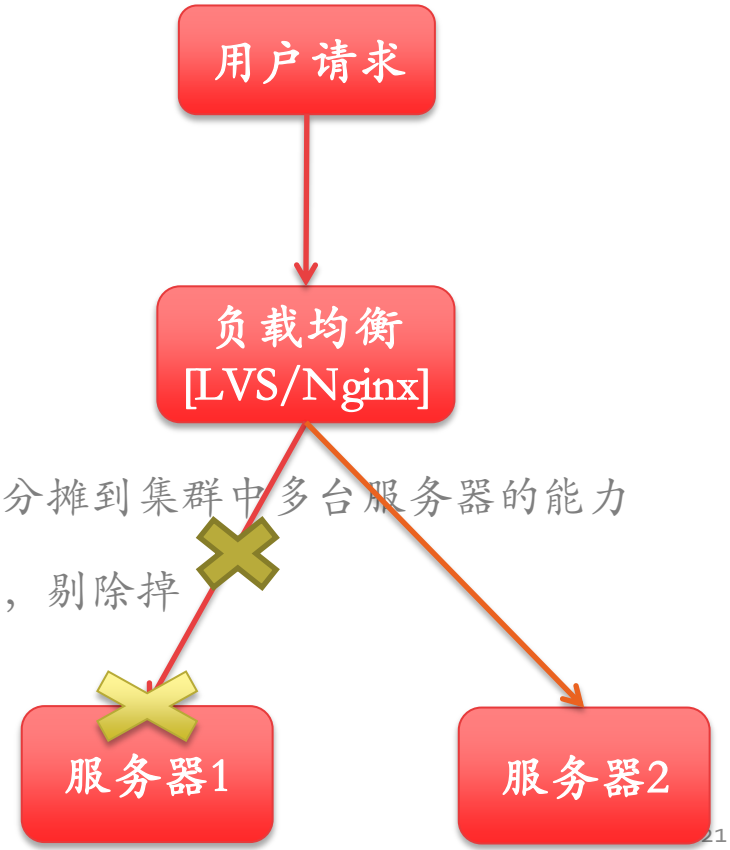
无状态业务逻辑层设计

- 关键因素
  - 业务逻辑层不保存请求状态
  - 业务逻辑层不保存数据
  - 所有业务逻辑层服务器完全对称
  - 当一台或者多台宕机
  - 请求提交到集群中的任意可用服务器
  - 业务逻辑层高可用
  - 实现高可用的关键因素是什么？
    - 负载均衡

无状态业务逻辑层如何设计

无状态业务逻辑层设计

- 负载均衡
  - 服务器可用状态实时监测的机制
  - 自动转移失败任务(机器)的机制
  - 请求量和数据量较高，将流量和数据分摊到集群中多台服务器的能力
  - 通过心跳机制发现下游服务器不可用，剔除掉
  - 一旦服务器可用，可以自动重连恢复



业务逻辑层如何纯异步调用？

纯异步调用

- 什么是同步
  - 发出一个请求调用时，在没有得到结果之前，该调用就不返回
  - 调用者(线程)阻塞模式
- 什么是异步
  - 异步调用发出后，调用者立即返回。结果完成后，通过状态、通知和回调来通知调用者。
  - 调用者(线程)非阻塞模式

业务逻辑层如何纯异步调用？

纯异步调用

- 异步调用优点
  - 线程(调用者)非阻塞，一直Running，CPU利用率高，系统性能高
  - 系统吞吐量高
- 异步调用缺点
  - 实现成本稍高

业务逻辑层如何纯异步调用？

如何异步调用

- 消息队列方案一
  - 通过消息队列（**缓冲、持久化、解决异步**）实现异步调用
  - 例子
    - 新用户注册请求，假设需要2步
      - 第一步：用户名和密码写入数据库
      - 第二步：发送注册成功邮件
    - 同步方式
      - 第一步调用等待->成功
      - 第二步调用等待->成功
      - 如果第二步执行失败，整个请求就执行失败
      - 阻塞串行执行，效率较低

业务逻辑层如何纯异步调用？

如何异步调用

- 消息队列方案一
  - 例子
    - 新用户注册请求，假设需要2步
      - 第一步：用户名和密码写入数据库
      - 第二步：发送注册成功邮件
    - 异步方式
      - 新用户注册请求写入消息队列，直接返回成功给请求调用者
      - 业务逻辑层通过成消息队列中读取请求，异步执行第一步和第二步
      - 调用者不阻塞，效率高
      - 系统性能高

业务逻辑层如何纯异步调用?

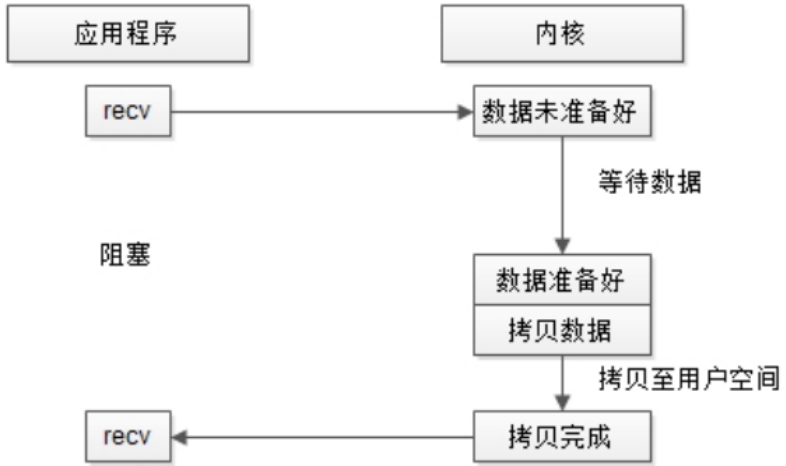
如何异步调用

- 异步调用的场景
  - I/O
  - 特别是网络I/O
  - I/O模型
    - 阻塞I/O模型
    - 轮询非阻塞I/O模型
    - I/O复用模型

业务逻辑层如何纯异步调用?

如何异步调用

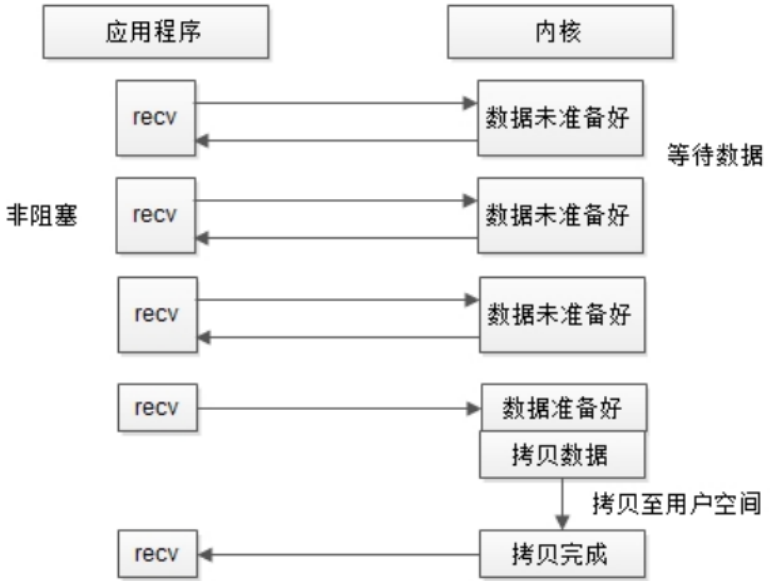
- 异步调用的场景
  - 阻塞I/O模型



业务逻辑层如何纯异步调用?

如何异步调用

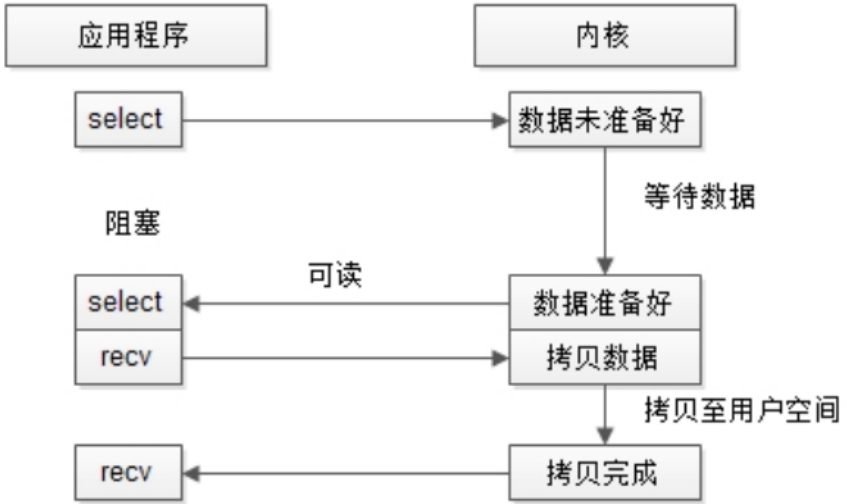
- 异步调用的场景
  - 轮询非阻塞I/O模型



业务逻辑层如何纯异步调用?

如何异步调用

- 异步调用的场景
  - I/O复用模型
    - select
      - » 1024
    - poll
    - epoll



业务逻辑层如何纯异步调用？

如何异步调用

- 高性能纯异步网络调用设计
  - server端连接池+server端收发队列；
  - client端连接池+client端收发队列；
  - 超时队列与超时管理器；
  - 上下文管理器+状态机；
- 案例里详细介绍

业务逻辑层如何分级管理?

分级管理

- 硬件分级层面
  - 核心系统使用好的机器、
    - CPU、内存、磁盘、网卡
  - 边缘系统使用差的机器
- 部署层面
  - 服务部署隔离
  - 避免故障带来的连锁反应
  - 核心系统部署在物理机上
  - 核心系统部署不同的机房
  - 边缘系统部署虚拟机
  - 边缘系统公用机器

业务逻辑层如何分级管理?

- 分级管理
  - 监控分级层面
    - 核心服务更多类型的监控
      - 进程
      - 语义
      - 错误日志
      - ……
    - 监控粒度更细致
    - 邮件和短信发送通知
  - 响应分级层面
    - 核心服务开发响应迅速
    - 核心服务上线响应迅速
    - 核心服务运维响应迅速
    - 核心服务上线问题处理迅速

业务逻辑层如何设置合理的超时?

设置合理超时

- 业务逻辑层和下游模块交互次数多
- 设置合理超时非常重要
- 下游服务宕机、线程死锁等，请求得到不到响应
- 请求占用资源
- 调用方得不到响应，用户体验糟糕

业务逻辑层如何设置合理的超时?

设置合理超时

- 请求的超时设置根据请求的平均响应延迟
  - 经验值
    - 超时时间是平均响应延迟的2倍，避免过长时间等待
  - 响应延迟高，超时时间设置长些
    - 3S
  - 响应延迟低，超时时间设置短些
    - 100MS
- 下游请求超时后，业务层根据预设的调度策略
- 继续重试
  - 一般3次
  - 多次无好处
- 请求转移到下游其他同样服务上

业务逻辑层服务降级如何设计?

业务逻辑层服务降级设计

- 网站高峰期，并发量大
  - 服务能力有限
  - 性能下降
  - 服务宕机
  - 系统雪崩情况
- 怎么办?
  - 服务降级

业务逻辑层服务降级如何设计?

业务逻辑层服务降级设计

- 保证核心服务可用
- 非核心服务弱可用，甚至不可用
- 降级设计方案
  - 拒绝部分请求
  - 关闭请求

业务逻辑层服务降级如何设计?

业务逻辑层服务降级设计

- 拒绝部分请求
  - 拒绝低优先级服务的调用
  - 减少服务调用并发数
  - 确保核心服务正常使用
  - 队列方式
    - 入队、出队时间，超过一定时间，直接丢弃
    - 58帮帮例子
  - 优先级请求方式
    - 非核心请求直接丢弃
    - 58帮帮例子
  - 随机拒绝方式
    - 随机丢弃一定比例请求
    - 网站一会可用，一会不可用，大都是这样的处理

业务逻辑层服务降级如何设计?

业务逻辑层服务降级设计

- 关闭部分服务
  - 非核心服务直接关闭
  - 业务逻辑层屏蔽掉
  - 不再调用
  - 节约系统开销
  - 保证核心服务的正常响应
  - 秒杀活动
  - 双11.11活动
    - 关闭社交

业务逻辑层如何做到幂等设计?

服务器幂等设计

- 请求失败后，会继续重试
  - 各种失败原因
- 保证服务重复调用和一次调用结果相同（幂等性）
- 不能保证幂等性
  - 结果将是灾难性的
    - 转账
    - 交易
    - 支付

业务逻辑层如何做到幂等设计?

服务器幂等设计

- 天然幂等
  - 离线消息设置为已读
    - 多次设置都是一样
- 非幂等->幂等设计
  - 支付
    - 支付ID，支付状态
    - 每次支付前，判断支付状态，未支付状态继续进行，已支付了就中止
  - 转账
    - 转账状态判断

业务逻辑层高可用设计最佳实践是什么？

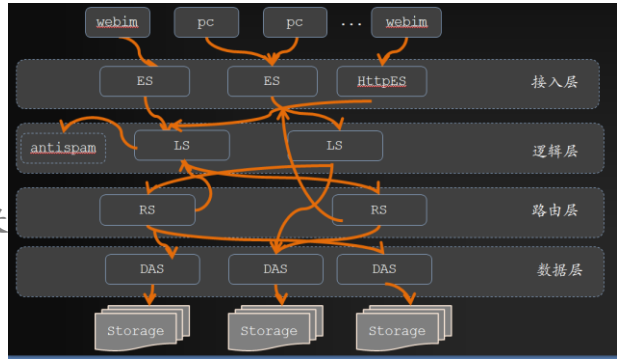
最佳实践

- 无状态
- 冗余部署
- 异步
- 超时机制
- 请求分级
- 幂等设计
- 高性能

我们的实践案例

58帮帮业务场景（加好友）

- IM的加好友服务，用户A将好友添加到一个分组中
  - 拉取好友列表，看看B是否是好友
  - 拉取分组，看分组team是否存在
  - 检查B是否是IM用户
  - 检查是否符合antispam策略，加好友频率不能过快
  - 拉B的阻止列表，看B是否阻止A
  - 拉B的加好友策略，看是否需要验证
  - ……
- 列举的步骤需要帮帮业务逻辑层访问下游服务



我们的实践案例

58帮帮业务场景（加好友）

- 业务复杂
- 涉及多个下游模块
- 需要和多个下游模块网络交换
- 需要高性能、高可用
  - 网络异步模型
  - 业务逻辑层无状态
  - 冗余部署
  - 动态的扩展
- 给出上下游关系图
  - 如何设计?

我们的实践案例

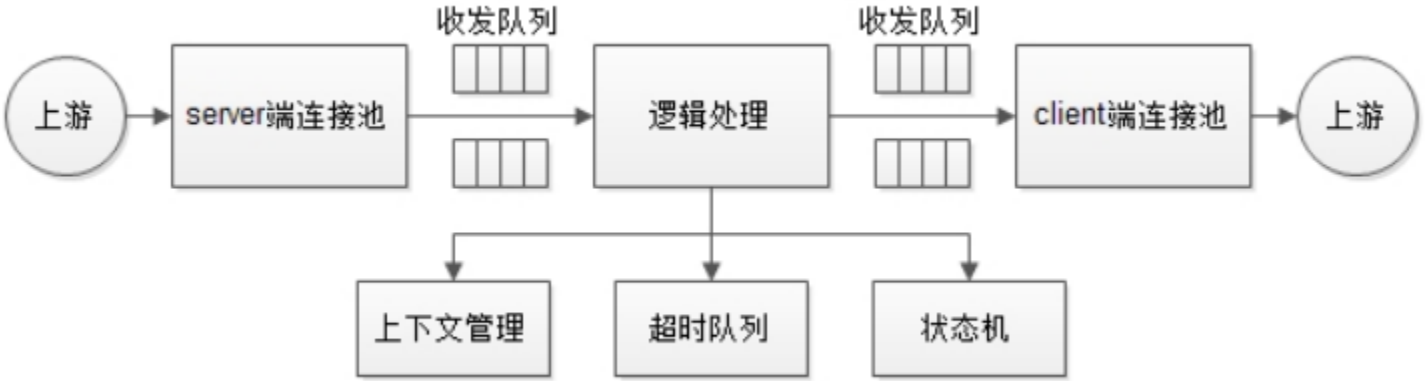
58帮帮业务场景（加好友）

- 名词解释
  - 客户端
    - 网络通信中主动发起通信的一端
  - 服务端
    - 网络通信中被动通信的一端，为客户端提供服务器，一般情况下一个服务端可对多个客户端提供服务。
  - 异步通信
    - 客户端在于服务器通信的过程中可以并发的发送多个请求，而不用没发送一个请求就停下来等待服务器的响应，收到服务端的响应后底层通过某种机制通知上层应用（比如函数回调）

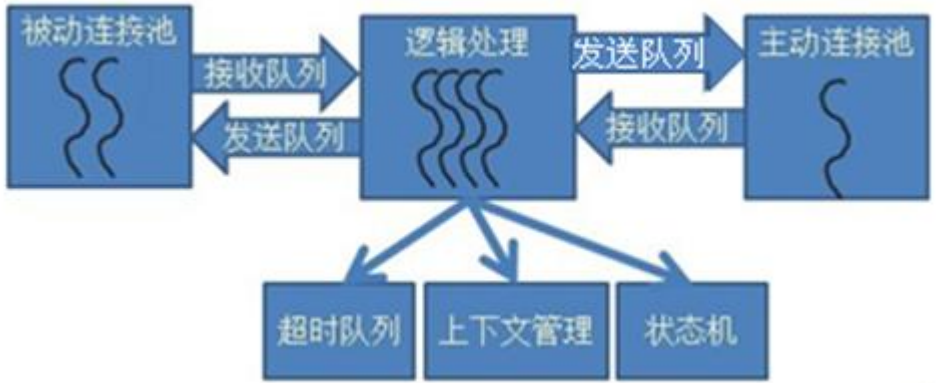
我们的实践案例

58帮帮业务场景（加好友）

- 框架结构



我们的实践案例



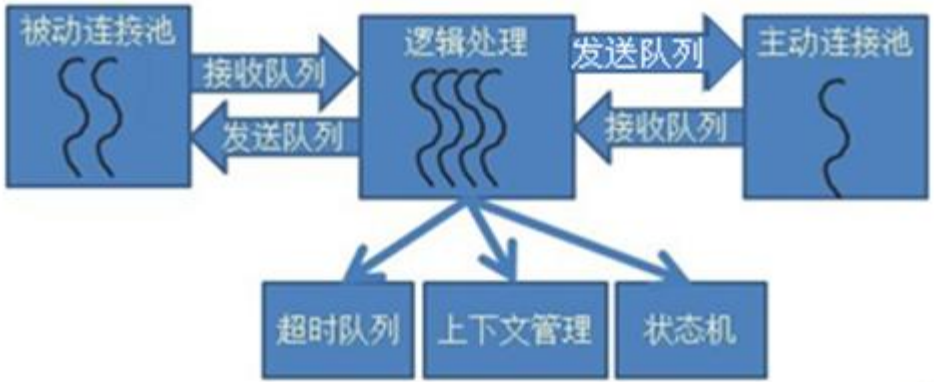
58帮帮业务场景（加好友）

- 具体实施
  - 1.进程启动下游连接加入主动连接池；
  - 2.建立上游客户端连接，并加入被动连接池；
  - 3.上游客户端数据放入接收队列；
  - 4.工作线程开始处理；
  - 5.发送到下游服务器，请求到客户端的发送队列，并加入超时队列；
  - 6.收到下游服务端响应，请求接入接收队列，删除超时队列，通知上层回调；
  - 7.下游超时，删除，并回调；
  - 8.响应加入到服务端发送队列，回复给客户端；

我们的实践案例

58帮帮业务场景（加好友）

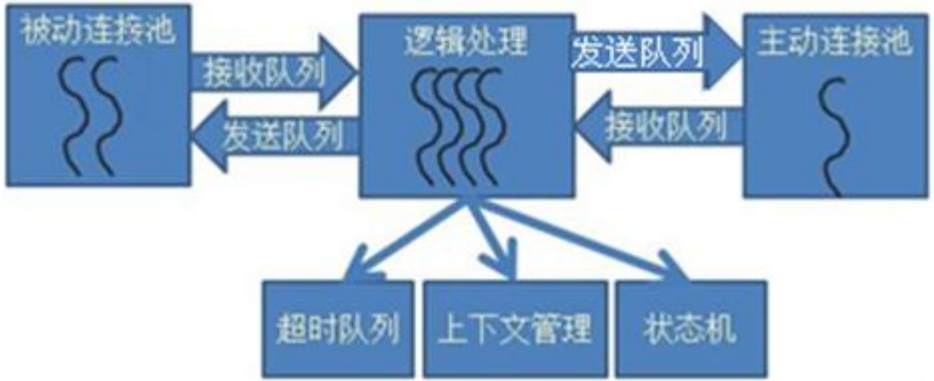
- 超时管理器
  - 发送下游包的超时管理
  - 避免无限等待
  - 单独线程
  - 定时扫描
  - 超时处理



我们的实践案例

58帮帮业务场景（加好友）

- 上下文管理器
  - 请求上下文
  - 请求的唯一标示
  - package_key
  - 超时等删除上下文



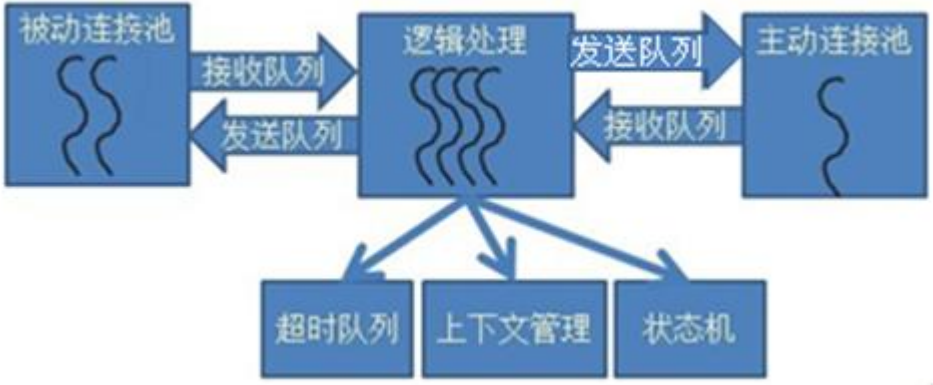
我们的实践案例

58帮帮业务场景（加好友）

- 状态机管理器
  - 异步调用的状态机
  - 标志请求的状态
  - 串行执行的状态机

```
enum EState
{
    STATE_WAIT_D        RIEND = 1,
    STATE_WAIT_D        EAM_LIST = 2,
    STATE_WAIT_D        BLOCK_B = 3,
    STATE_WAIT_DS_Q     B_BLOCK_A = 4,
    STATE_WAIT_DS_      FRIEND = 5,
    STATE_WAIT_DS_      RIEND_LEVEL = 6,

    STATE_WAIT_DS_B_R   ADD_A = 7,
    STATE_WAIT_DS_A_F   B = 8,
    STATE_WAIT_DS_A_    MODIFY_B = 9,
    STATE_WAIT_DS_A     ADD_B_UNVERIFY = 10,
    STATE_WAIT_DS_A     E_ADD_B = 11,
    STATE_WAIT_DS_B_    DIFY_A = 12,
    STATE_WAIT_RS_KI    ER = 13,
    STATE_WAIT_SEND_    OUND_MSGNOTIFY = 14,
    STATE_WAIT_DS_SA    NE_MSG = 15,
    STATE_WAIT_RS_REL   ND_ADD_NOTIFY = 16,
};
```



思考

思考点

- epoll的fd设置成阻塞有什么问题?
  - 读取不到会阻塞
  - 需要40-，只读取了30个
- 同步、异步
  - 消息通信
- 阻塞、非阻塞区别
  - I/O
- 同步阻塞可行吗?
- 异步非阻塞可行吗?
- 异步阻塞可行吗?
  - 不行
- 同步非阻塞可行吗?
- 同步阻塞实现异步?
  - SCF异步调用
  - 请求者数据-》队列-》线程池同步阻塞处理-》回调请求者

本课总结

总结

- 逻辑层都做什么?
- 逻辑层整体架构设计
- 无状态业务逻辑层如何设计?
- 业务逻辑层如何纯异步调用?
- 业务逻辑层如何分级管理?
- 业务逻辑层如何设置合理的超时?
- 业务逻辑层服务降级如何设计?
- 业务逻辑层如何做到幂等设计?
- 业务逻辑层高可用设计最佳实践是什么?
- 我们的实践案例;

THANK YOU