老树新花-Java异步服务开发

主讲人：饿了么-框架工具组-朱杰

WHO AM I

朱杰

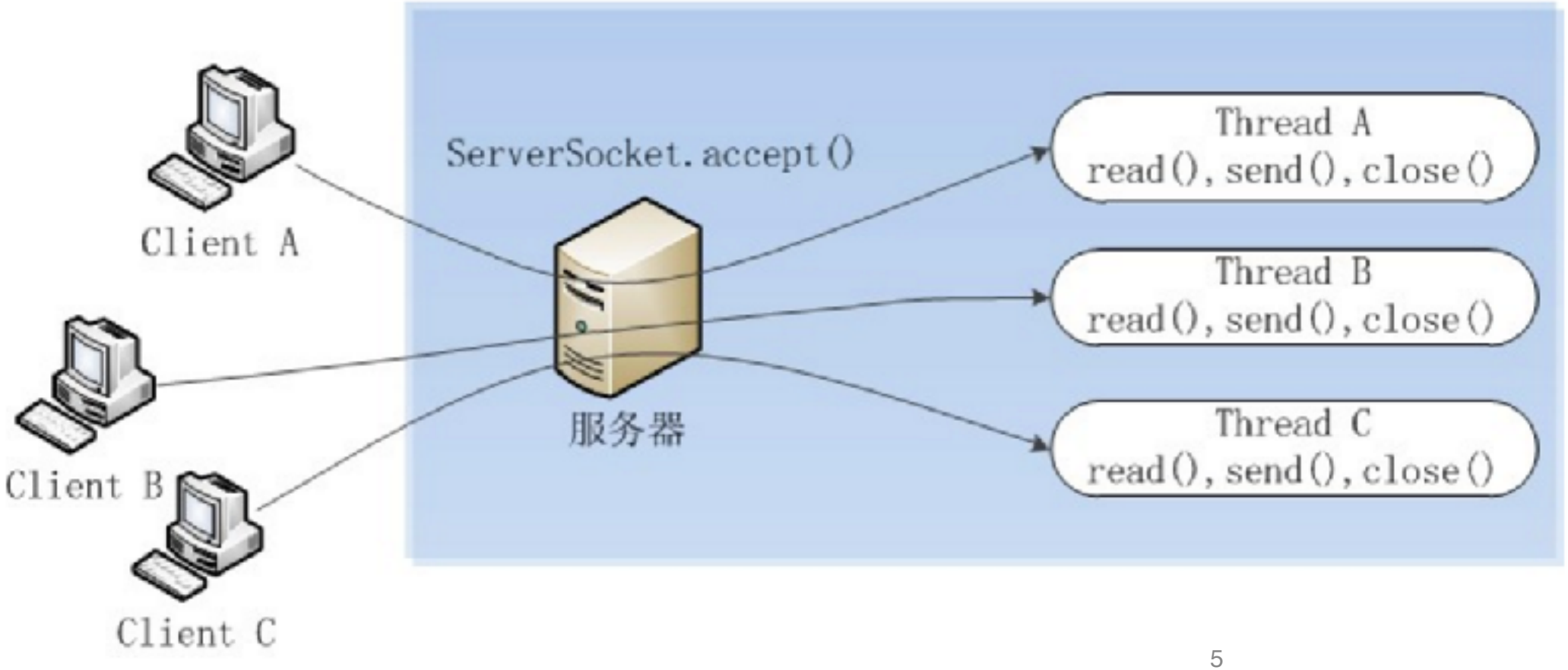
- 曾就职于爱立信,携程
- 2015.4 加入饿了么
- 有幸开发了饿了么MYSQL(POSTGRES)数据库中间件
- 两年来踩坑无数, 填坑无数

目录

同步模型

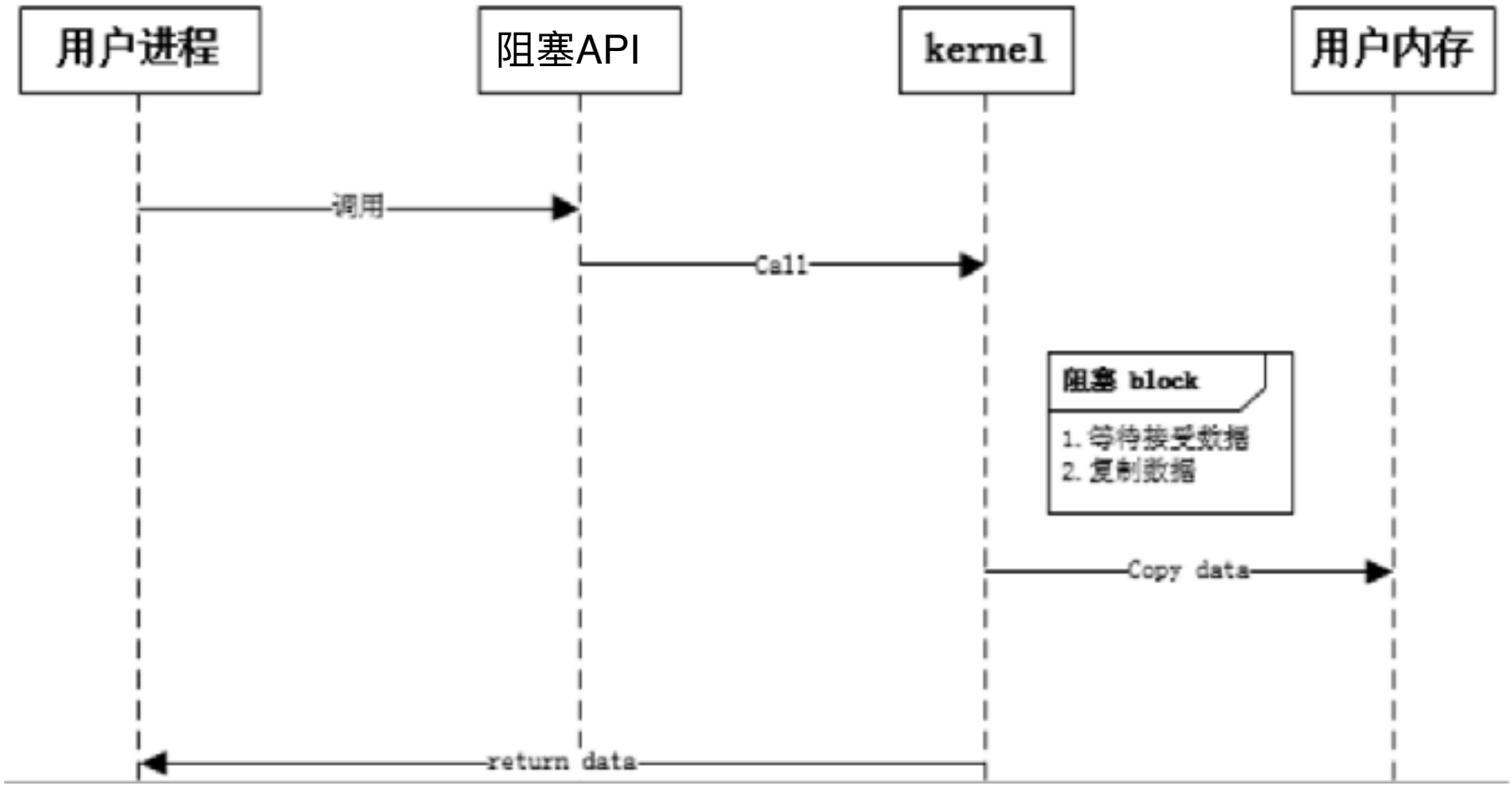
同步模型

阻塞I/O


Client A
Client B
Client C
ServerSocket.accept()
服务器
Thread A
read(), send(), close()
Thread B
read(), send(), close()
Thread C
read(), send(), close()

同步模型

阻塞I/O


用户进程
阻塞API
kernel
用户内存
调用
Call
阻塞 block
1. 等待接受数据
2. 复制数据
Copy data
return data

同步模型

阻塞I/O

- Connection conn = **DriverManager**.*getConnection*(url);

```
"main" #1 prio=5 os_prio=31 tid=0x00007f8c1100c800 nid=0x1703 runnable [0x00007000002170000]
   java.lang.Thread.State: RUNNABLE
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:345)
        - locked <0x000000c07960311e0> (a java.net.SocksSocketImpl)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
```

```
        at java.sql.DriverManager.getConnection(DriverManager.java:664)
        at java.sql.DriverManager.getConnection(DriverManager.java:270)
        at ele.me.javatraining.JDBCTest.main(JDBCTest.java:41)
```

同步模型

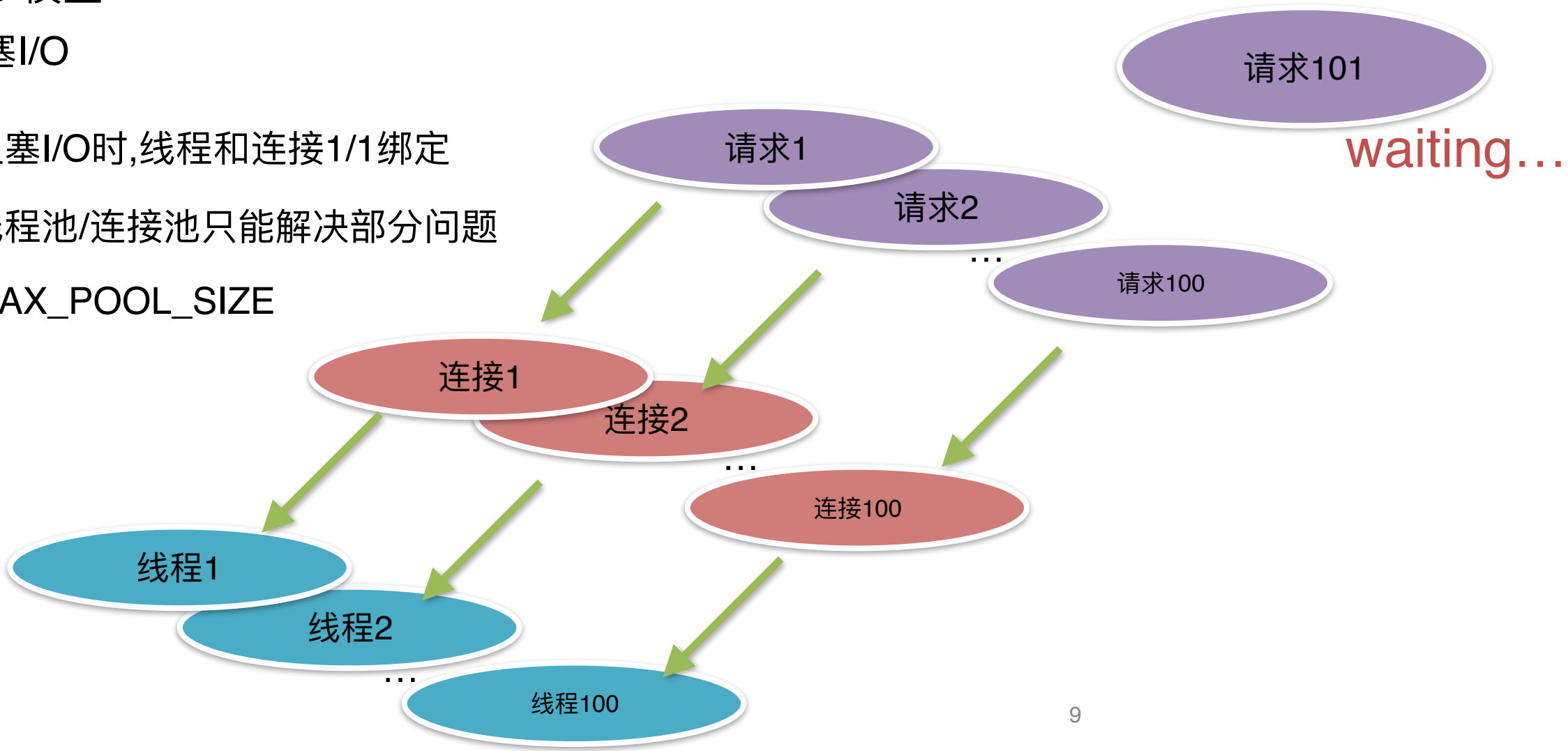
阻塞I/O

- 符合人的思考逻辑
- 线程阻塞
- 让出CPU
- 不适用于高并发

同步模型

阻塞I/O

- 阻塞I/O时,线程和连接1/1绑定
- 线程池/连接池只能解决部分问题
- MAX_POOL_SIZE



同步模型

阻塞I/O

- 线程池/连接池的作用
  - 减少创建或销毁线程/连接的开销
  - 池化并非直接影响到QPS
- 单机线程数是有限的
- 高并发的情况下产生瓶颈
- 去除阻塞!!!

异步模型

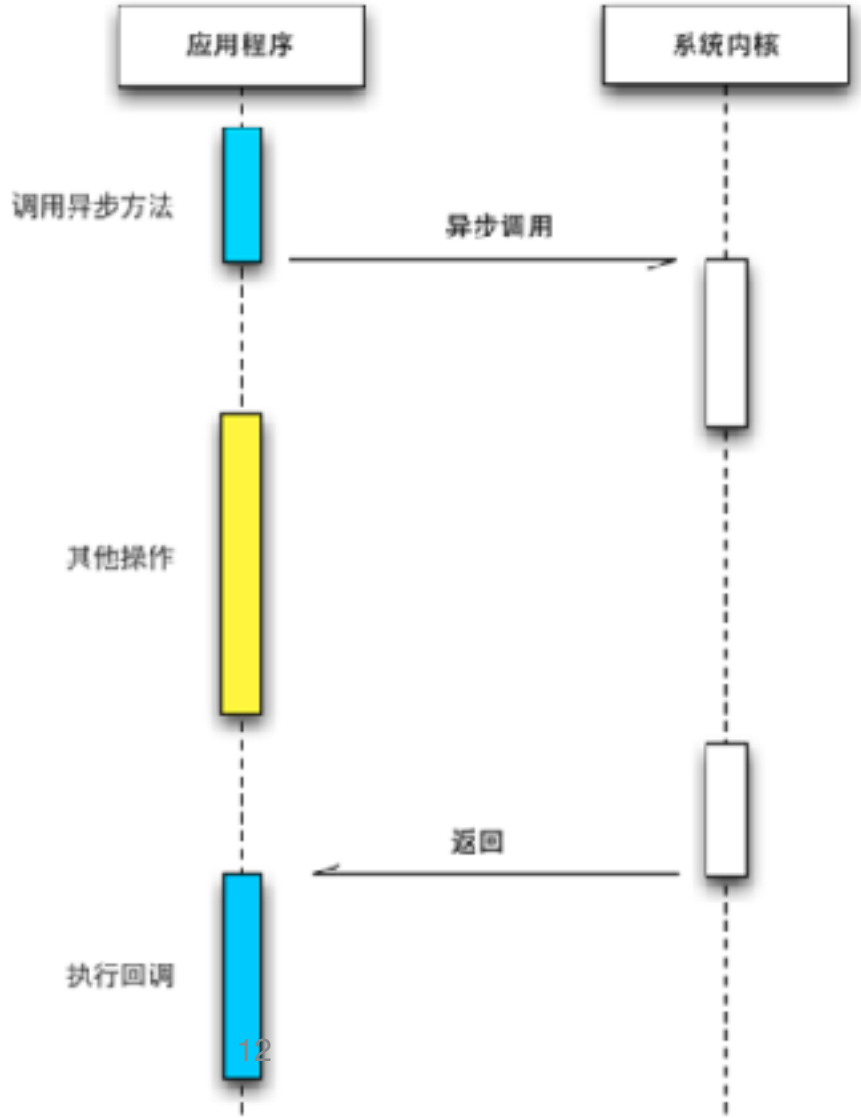
异步模型

异步I/O模型介绍

- 非阻塞

- 不等待结果即返回

- 处理结果不保证在调用API的线程



异步模型

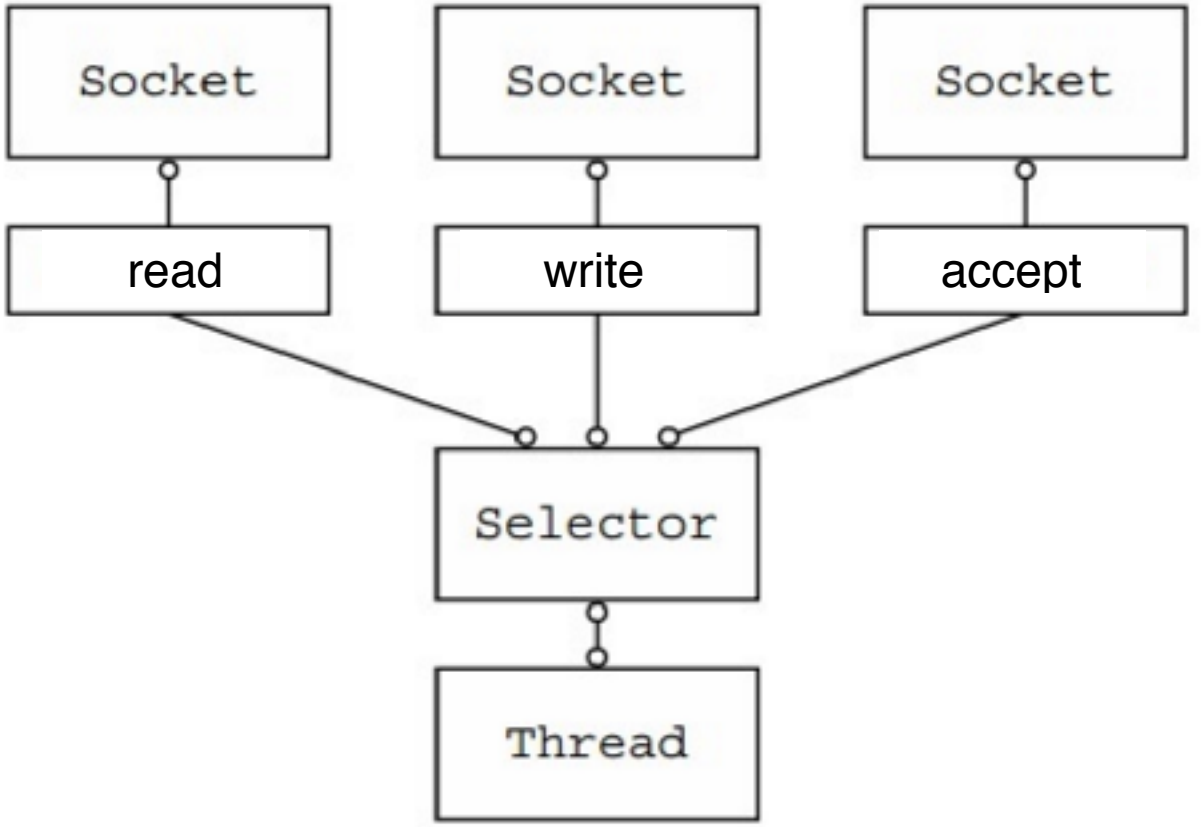
Java中的异步模型

- 基于操作系统的模型
  - EPOLL/KQUEUE/IOCP
- JAVA/C异步模型的基石

- 少数线程等待事件

- 根据类型处理事件

图1.2 使用Selector的非阻塞I/O



Java异步模型

Java中的异步模型

```
ServerSocketChannel serverChannel = ServerSocketChannel.open();
Selector selector = Selector.open();
serverChannel.register(selector, SelectionKey.OP_ACCEPT);//连接事件
while (true) {
    selector.select();//阻塞轮询有无感兴趣的事件发生
    for (SelectionKey key : selector.selectedKeys()) {
        if (key.isAcceptable()) { // 连接事件
            ServerSocketChannel server = (ServerSocketChannel) key.channel();
            SocketChannel client = server.accept();
            client.register(selector, SelectionKey.OP_READ);//数据读取事件
        } else if (key.isReadable()) {
            ((SocketChannel) key.channel()).read(...);//处理数据
        }
    }
}
```

异步模型

题外话: 协程

- 行为:
  - 轻量级线程,可当做线程使用
  - 阻塞I/O API 阻塞的是协程,不阻塞线程
- 消耗:
  - 用户态资源, 用户态调度, 消耗极低,可启动数万个
- 实现:
  - 和线程非1对1的关系
  - 难点在于编程语言内部实现, 而非使用(JAVA未实现,PYTHON部分实现,GO完美实现)

异步模型

其他语言协程

- PYTHON:
  - 支持协程
  - 全局解释锁GIL - 同一时刻只有单个CPU在运行
  - 多进程 + 协程
- GO
  - 语言层完美支持不阻塞线程的I/O操作
  - 多协程

异步模型

我们到底要什么

- 如同步I/O一样编写代码
- 不会创建过多数量的线程
- 尽量让CPU处理忙碌状态而非等待
- 寻找尽量满足以上条件的JAVA库

异步模型

其他语言异步模型

- JAVA协程三方库 (QUASAR)
  - 没有解决根本问题
  - 不支持NATIVE I/O的异步,(如输入流的READ仍然是同步的)
  - 只支持库自定制的同步改异步的I/O(修改JDK输入流为库定制的版本)
  - 支持非阻塞I/O的协程切换

```
IntStream.range(0, 100000).forEach(new Fiber(() -> {
    System.out.println("This is a fiber not thread");
    Fiber.sleep(...);
}).start());
```

异步模型

其他语言异步模型

- JAVA异步工具库(VERT.X)
  - 异步JDBC
  - 异步HTTP CLIENT
  - 基于NETTY
- 处理不一定在当前线程

```
vertx.createHttpClient().getNow(8080, "localhost", "/",
 response -> {
  response.handler(body -> {
    context.assertTrue(body.toString().contains("Hello"));
    async.complete();
  });
});
```

```
connection.query("SELECT ID, FNAME, LNAME, SHOE_SIZE from PEOPLE", res -> {
  if (res.succeeded()) {
    // Get the result set
    ResultSet resultSet = res.result();
  } else {
    // Failed!
  }
});
```

异步模型

服务异步化需要做的事

- 查看接口是否可支持异步 (服务异步 OR API异步)
- 使用JAVA异步工具库
  - 异步数据库访问方式
  - 异步HTTP CLIENT
- 如果是框架,那么修改调用方式
  - 异步回调 (SPRING @ASYNC)
  - 事件监听
- 注意线程安全问题 (重要的事情说三遍)

异步模型

异步化

- 优势:
  - 极大提高I/O密集型业务的性能 (10-100倍)
  - 解决线程数创建过多的问题
- 劣势
  - 增加编程难度,包括状态保存,回调处理,线程安全等
  - 👻 可能会压垮下游服务 (双核机器启动数万个连接)

老树新花 – 基于Netty的Java异步模型

Netty异步模型

Netty特性

1. **Netty/All In One** **634** usages

io.netty » netty-all Apache

Netty/All In One

- 基于JAVA原生的异步模型,封装并优化
- 修复(绕过)JDK中一些BUG
- 提供多种辅助类方便开发
- 编程模型简单且高效, 开发者只需关心具体实现逻辑即可
  - 基本不用花精力做JAVA网络层面的优化
- 成熟,稳定, 业界使用多

Netty异步模型

事件驱动

- 连接事件

```
public void channelActive(ChannelHandlerContext ctx) throws Exception;
```

- 读取事件 (接收数据事件)

```
protected void decode(ChannelHandlerContext ctx, I msg, List<Object> out)
```

- 异常事件

```
public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause)
```

- 自定义事件

```
public void userEventTriggered(ChannelHandlerContext ctx, Object evt)
```

Netty异步模型

Netty特性

- 异步I/O的API, 全完消除阻塞
  - BIND/CONNECT/FLUSH/WRITE/CLOSE
- 事件回调
  - FUTURE模式 + 事件监听LISTENER模式
- 线程模型
  - 单个连接上的所有操作(I/O事件,消息处理)由同一个线程执行
  - 避免线程安全问题(之后会提到)

Netty异步模型

Netty特性

- I/O异步回调

io.netty.channel.Channel

ChannelFuture bind(SocketAddress localAddress)

Request to bind to the given SocketAddress and notify the ChannelFuture once the operation completes, either because the operation was successful or because of an error.

```
ChannelFuture close();

ChannelFuture write(Object msg);

ChannelFuture connect(SocketAddress remoteAddress);
```

Netty异步模型

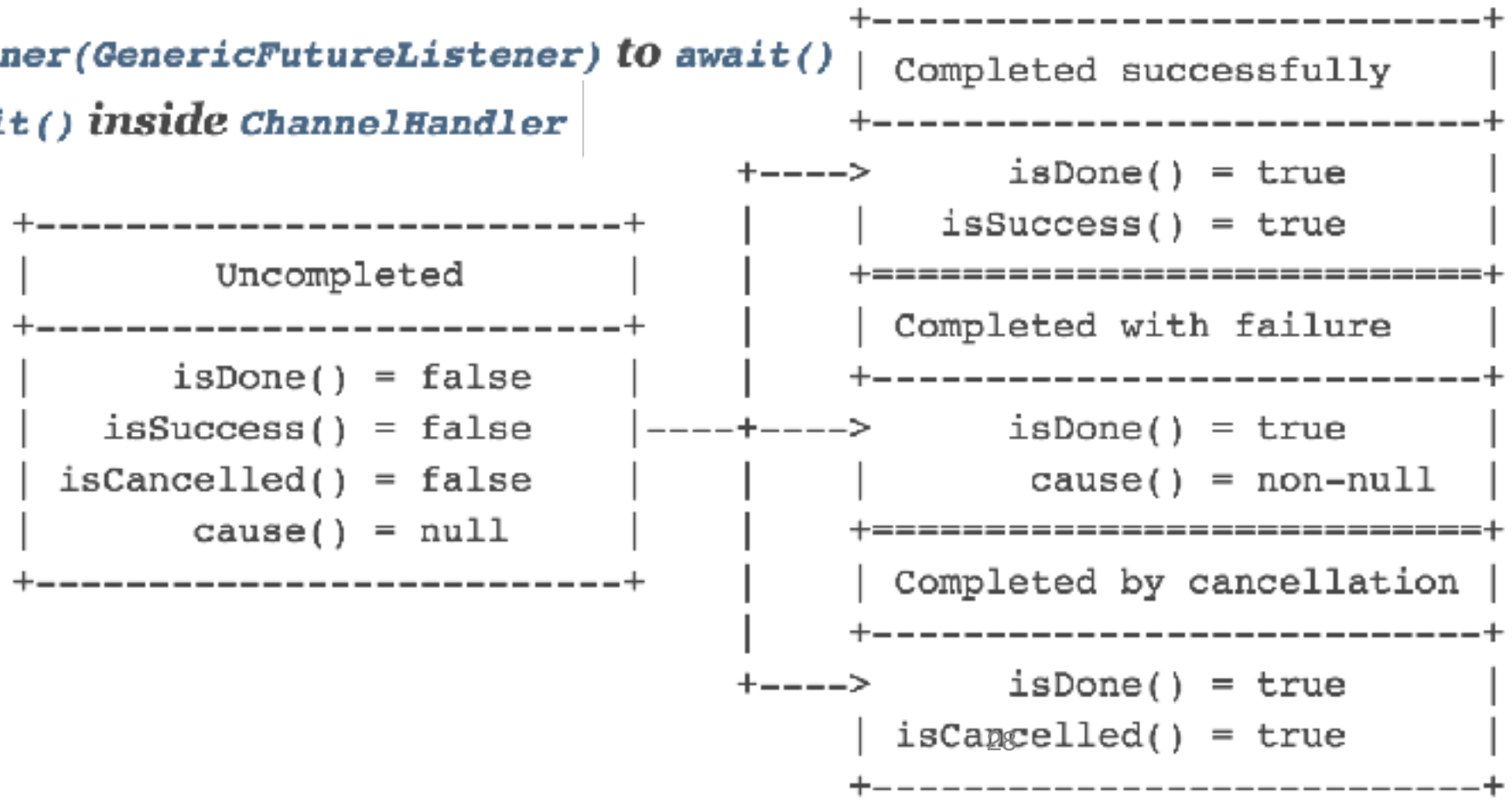
Netty特性

```
 @Override
public void channelRead(ChannelHandlerContext ctx, Object msg) {
    ChannelFuture future = ctx.channel().close(); //关闭连接
    future.addListener(new ChannelFutureListener() { //添加事件监听
        public void operationComplete(ChannelFuture future) {
            // Perform post-closure operation
            // ...
        }
    });
```

Netty异步模型

Netty特性

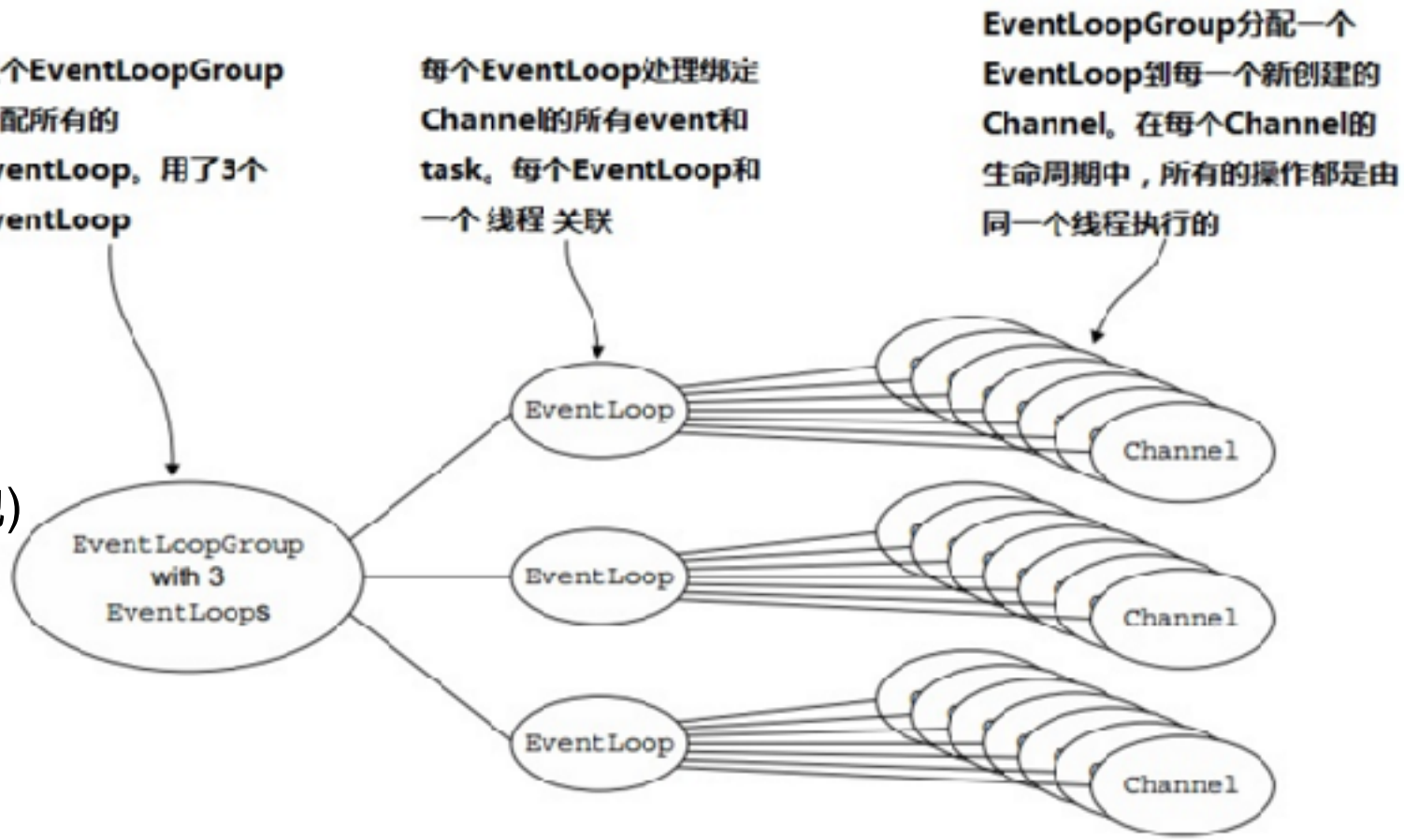
- ***Prefer*** `addListener(GenericFutureListener)` ***to*** `await()`
- ***Do not call*** `await()` ***inside*** `ChannelHandler`



Netty异步模型

Netty特性

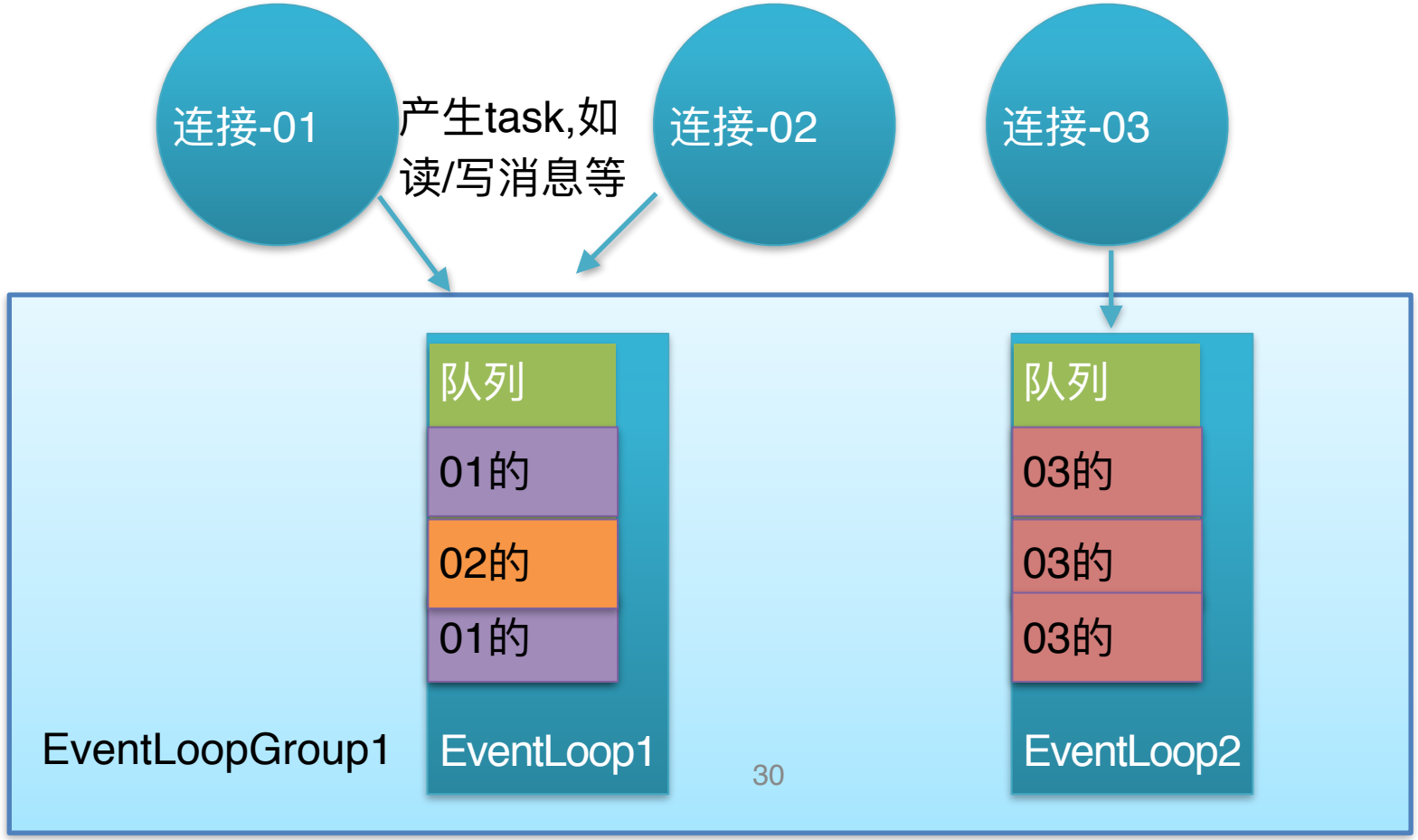
- 单个连接绑定一个线程
- EVENTLOOP即一个线程
- EVENTLOOPGROUP即线程组(池)



Netty异步模型

Netty特性

- 任何I/O API都是产生一个任务
- 放入该连接对应的线程里执行
- 局部串行化



Netty异步模型

Netty特性总结

- 事件驱动

- 所有I/O API使用异步+回调监听处理消息

- 单个连接的处理都在同一线程内执行

案例 – 饿了么数据库中间件

饿了么数据库中间件

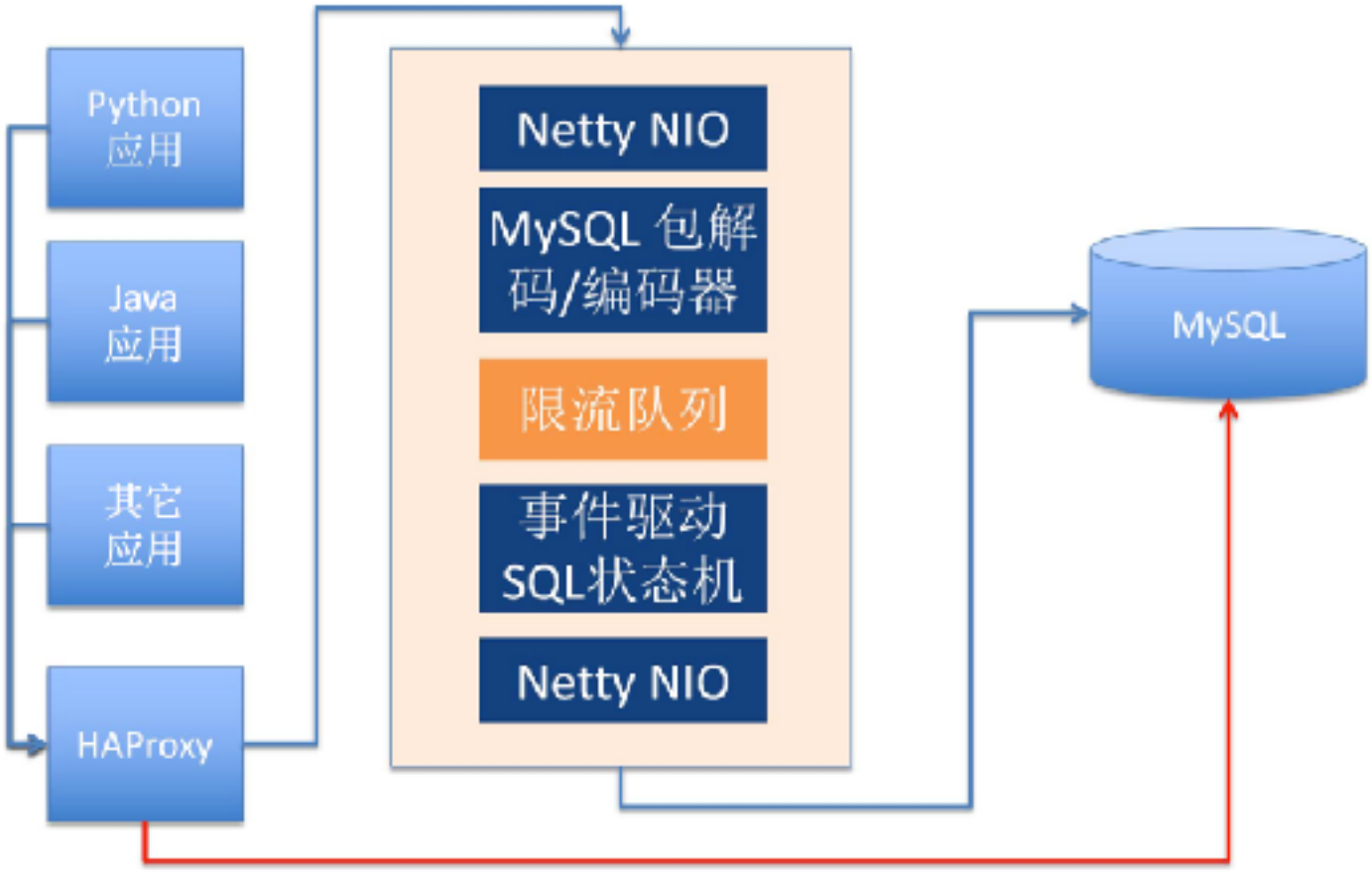
技术选型

- 实现MYSQL协议的中间代理服务
  - 上游支持上万客户端连接
  - 下游支持上千数据库连接
- 基于GITHUB阻塞I/O开源库(主要是MYSQL协议解析)
- 首要任务即同步改为异步

饿了么数据库中间件

前后端异步

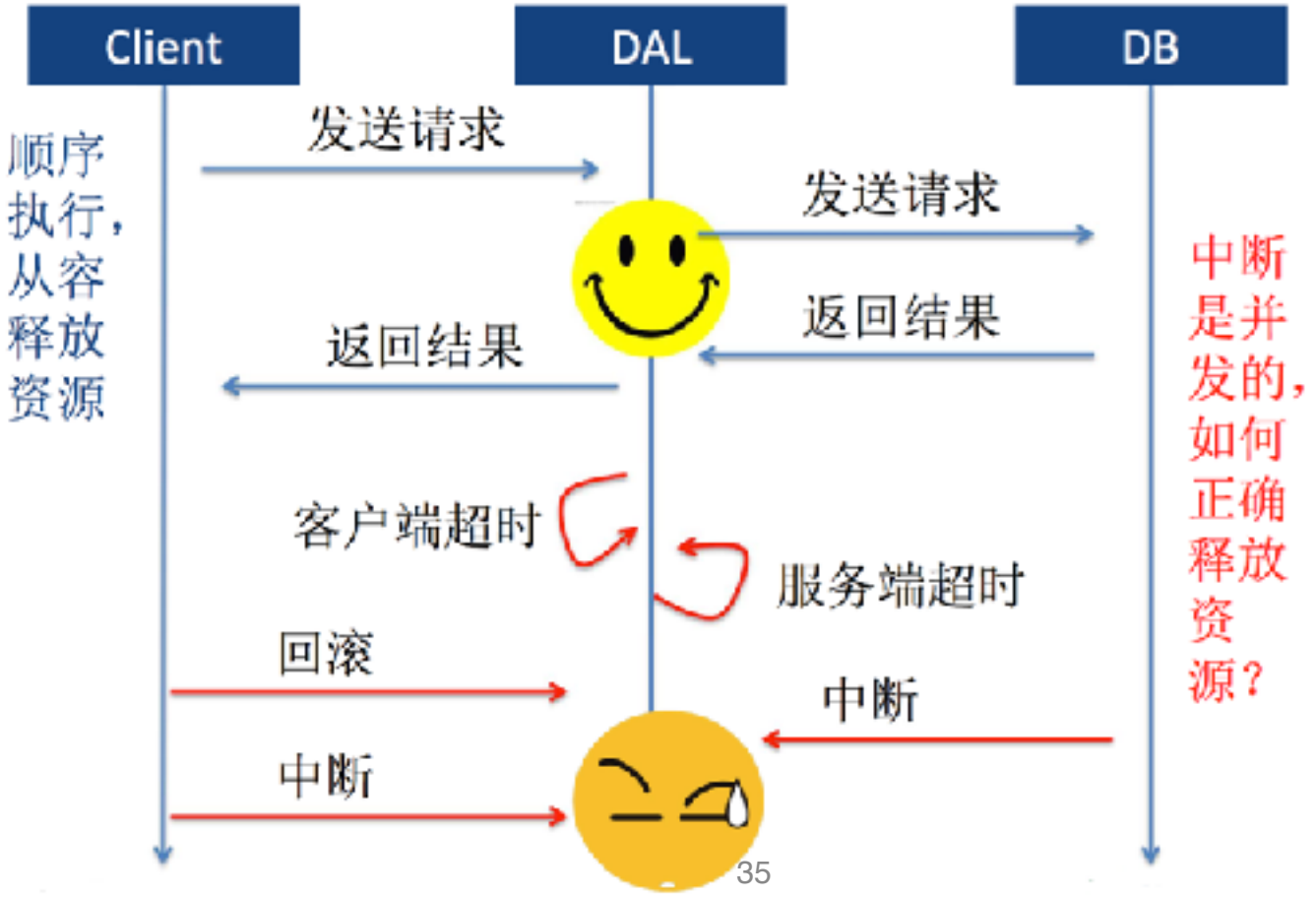
- 线程模型
  - 上下游各NETTY一个线程组
  - 中间件内部有一个处理业务的线程组
- 问题
  - 三个线程组之间的线程安全问题
  - 稳定性测试
    - 1秒KILL所有下游数据库连接



饿了么数据库中间件

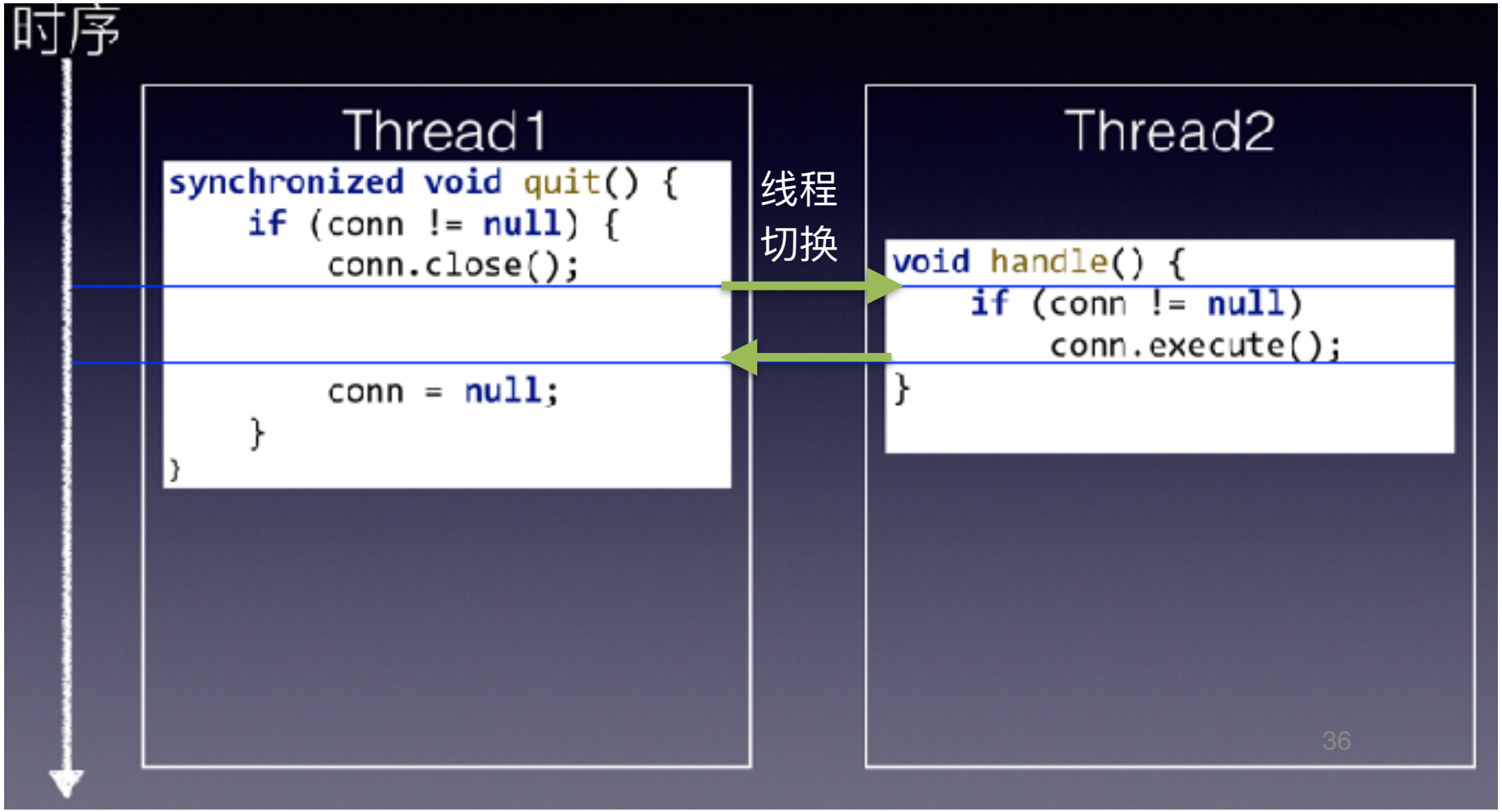
前后端异步

- 正常流程由协议保证顺序执行
- 异常中断是并发执行的



饿了么数据库中间件

前后端异步



饿了么数据库中间件

前后端异步

- 每个连接绑定一个单线程池

Thread1

```
//SingleThreadEventExecutor es;
void quit() {
    Task t = new QuitTask();
    es.submit(t);
}
```

Thread2

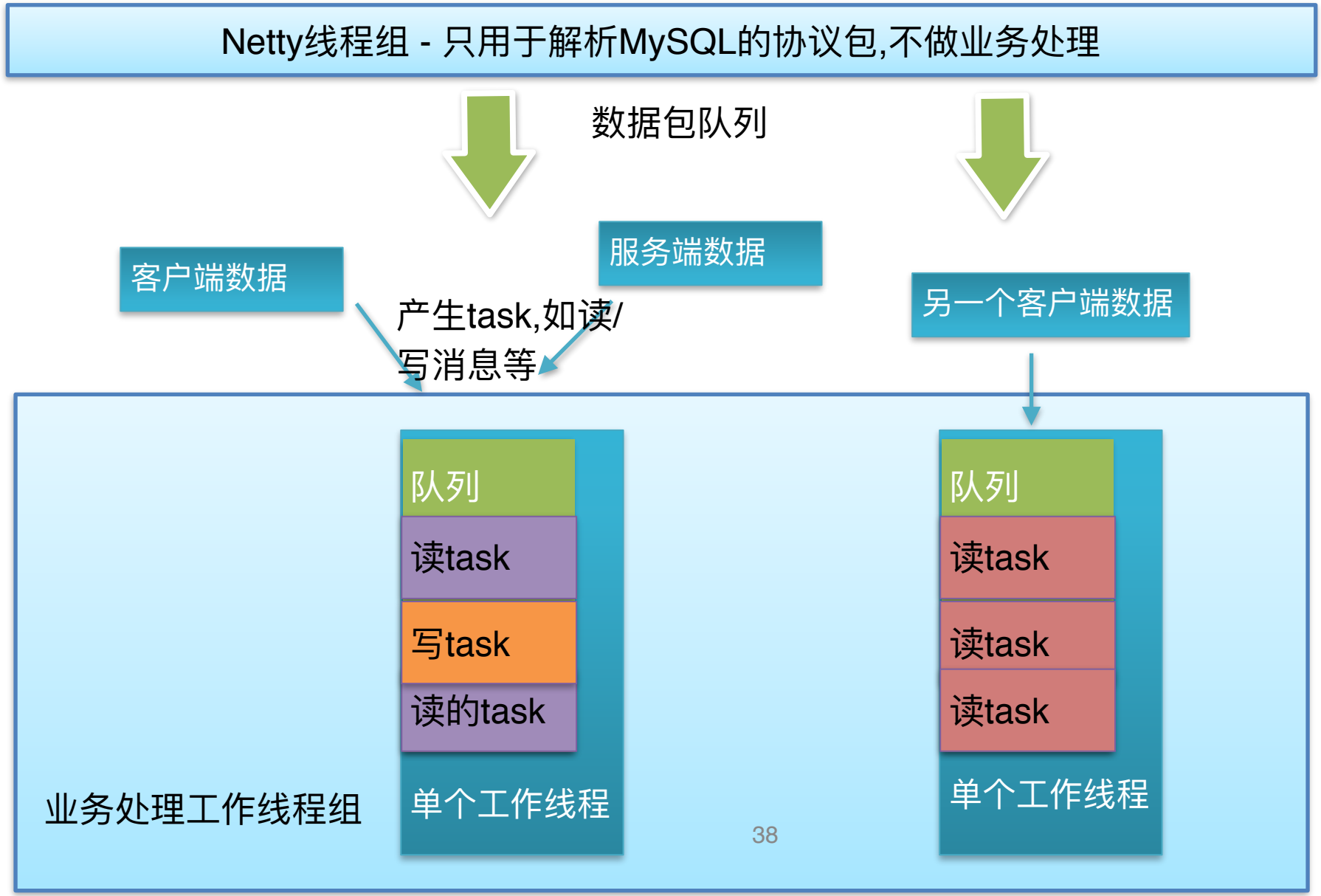
```
void handle() {
    Task t=new HandleTask();
    es.submit(t);
}
```

饿了么数据库中间件

前后端异步

- 仿照NETTY
- 上下游只生产协议包
- 后传入业务线程组处理

- 去锁
- 局部串行化



饿了么数据库中间件

前后端异步

- 优点
  - 模型简单
  - 便于后续修改

- 观点:
  - 过多的WAIT/NOTIFY/LOCK不一定代表良好的多线程编程能力
  - 却可能代表这是不够优雅的设计

饿了么数据库中间件

信号量控制异步

- 场景
  - 假设控制数据库最大连接数
- 方案:
  - 定制化信号量

java.util.concurrent.Semaphore

```
public void acquire()
            throws InterruptedException
```

Acquires a permit from this semaphore, blocking until one is available, or the thread is interrupted.

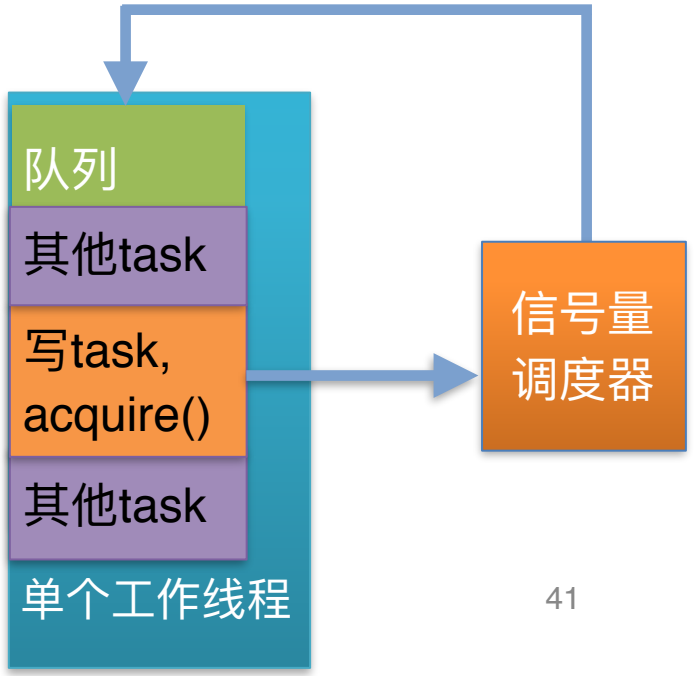
饿了么数据库中间件

信号量控制异步

- 传入回调,即获取到信号量时执行的代码

```
AsyncSemaphore sem = new AsyncSemaphore();
SemaphoreAcquiredCallback callback = () -> { continueBusiness() };
sem.acquire(callbck);
// 线程在调用acquire()之后马上返回
```

- 单个连接级别的串行化



饿了么数据库中间件

其他异步

- 日志异步
- 心跳异步
- JOB异步
- 配置变更异步

饿了么数据库中间件

总结

- 高效：异步化所有API以此去除阻塞
- 稳定: 局部串行化解决线程安全问题
- 简单：模型简单,易于修改和理解

老树新花-Java异步服务开发

分享总结

- 同步异步概念介绍
- 使用JAVA来开发异步化服务
- 回调监听模式所遇到的问题和解决

Q & A

THANKS !