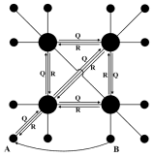
DATAGURU 炼数成金

Leader-us系列



大型分布式系统案例实战 第2课

DATAGURU专业数据分析社区

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

分布式系统之网络篇

- 80%的程序员所欠缺的网络知识
- 网络的最前沿技术之网络虚拟化
- 关于NIO于AIO的那些小秘密
- 网络编程中的一朵金花之Netty
- 网络编程中的瑞士军刀之Zookeeper

80%的程序员所欠缺的网络知识

服务器网卡的一些秘密

硬件不同

传统的PC机器的PCI接口66MHz 133Mb/s
而服务器的主板支持PCI-E X16 8bit 2.5GHz 8Gb/s
TCP/IP卸载引擎(TOE)技术

价格昂贵

Intel EXPX9501AFXSR
适用网络类型：万兆以太网
网线接口类型：LC
主芯片：Intel 82598
适用领域：服务器
评分：★★★★☆ 4.0 点评(1)

¥6700
2015-04-25
11家商家报价
查询底价

对比

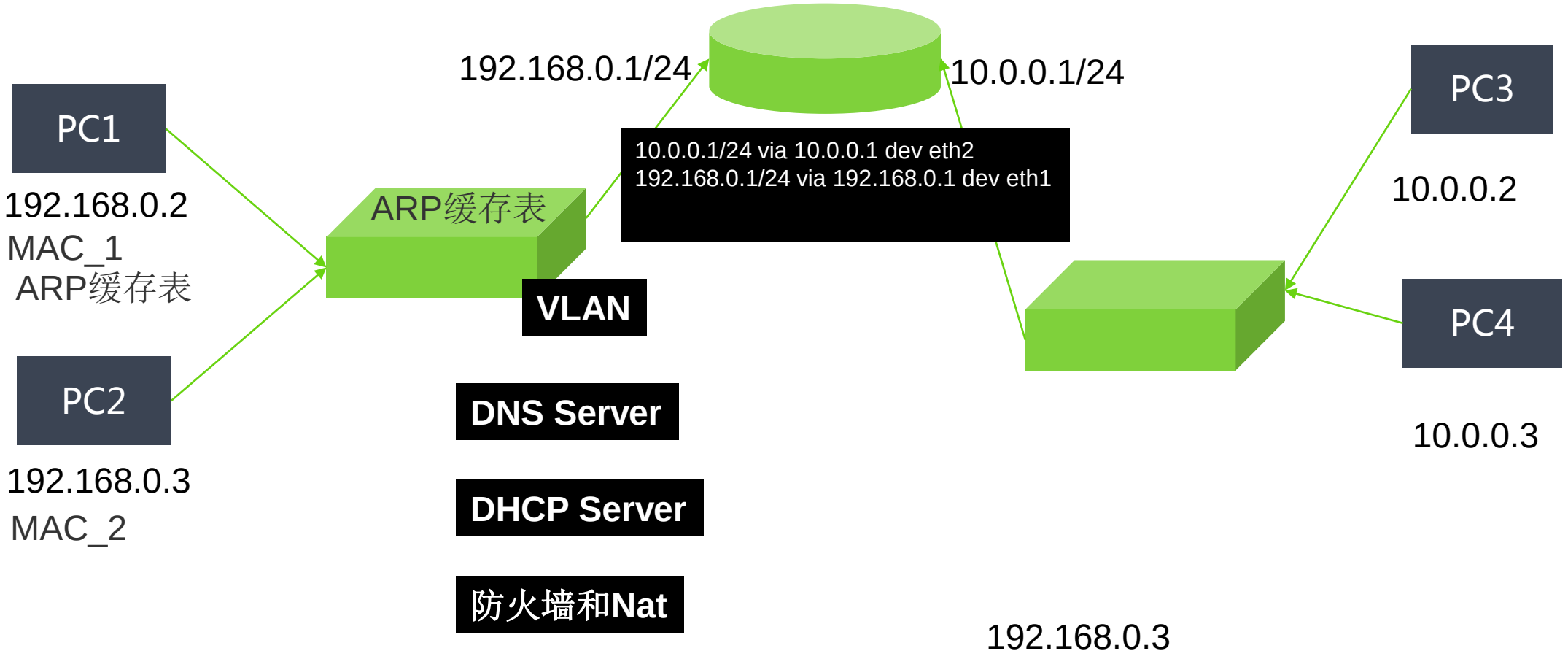
性能出众

Intel x520-t2 10Gbase-t万兆网卡（4800元报价）测试，
单端口单向实际测试结果是1248MB/s(9984Mbps)
，达到理论峰值的99.84%

技术门槛高

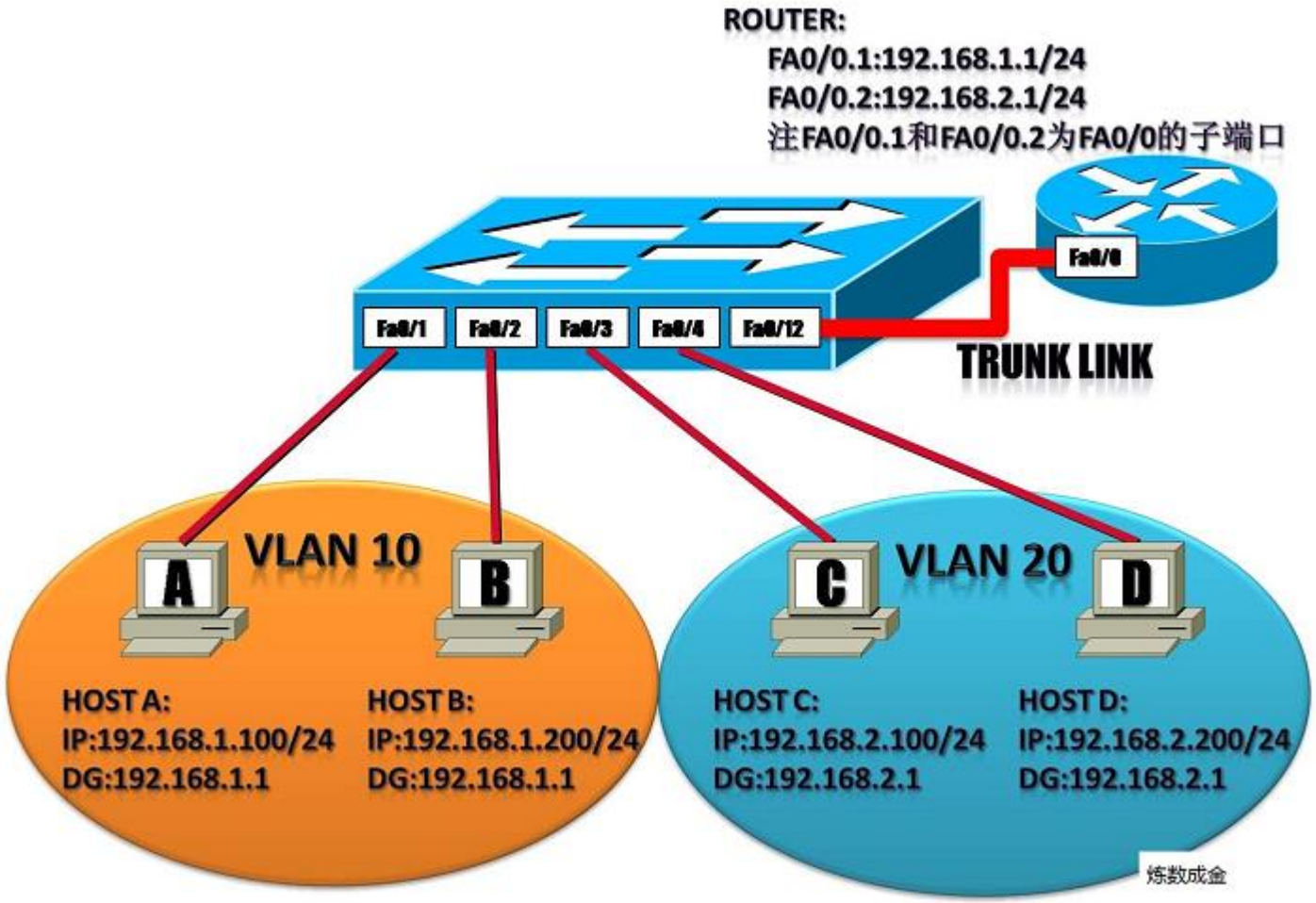
80%的程序员所欠缺的网络知识

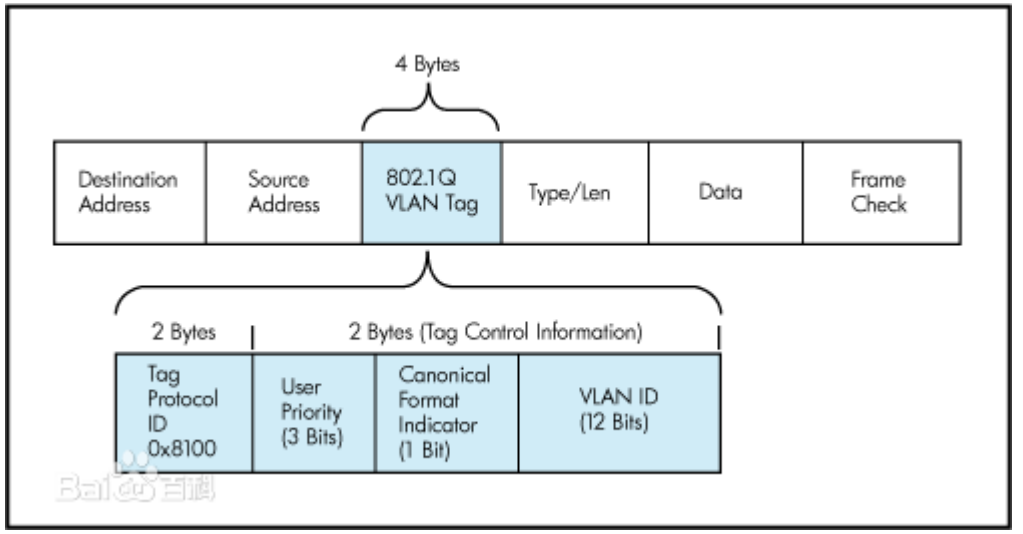
MAC地址、IP、子网掩码、网关及路由



80%的程序员所欠缺的网络知识

VLAN的原理

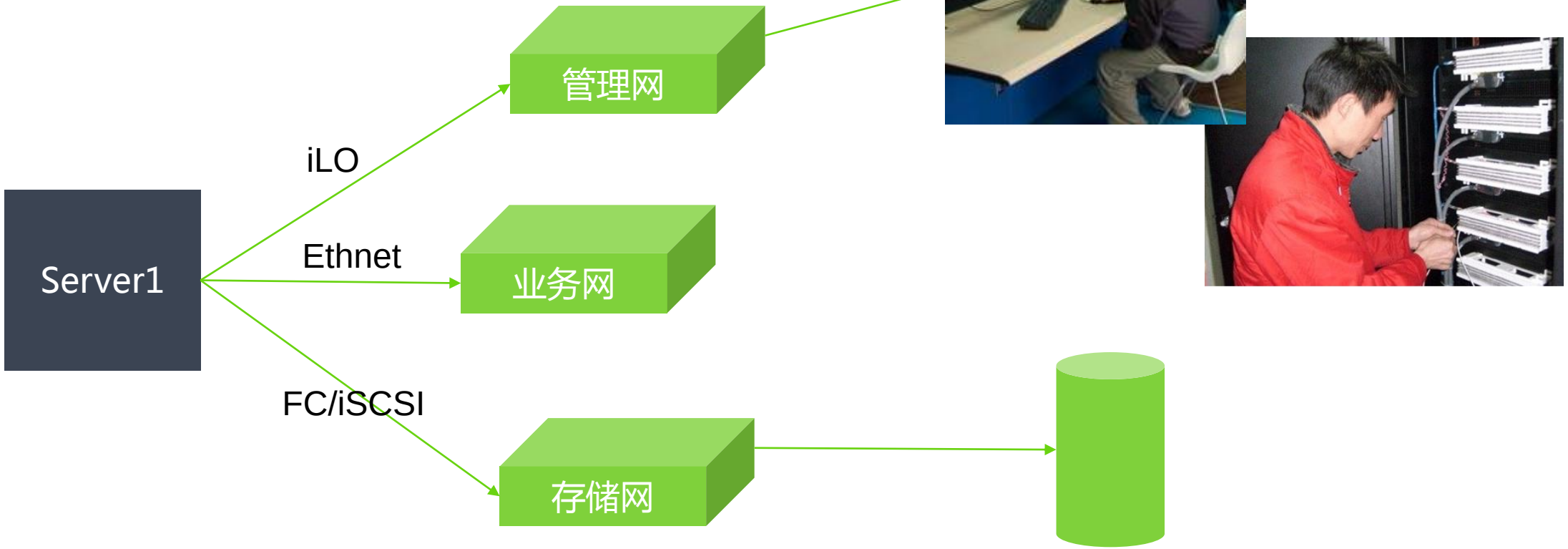




802.1Q报文格式

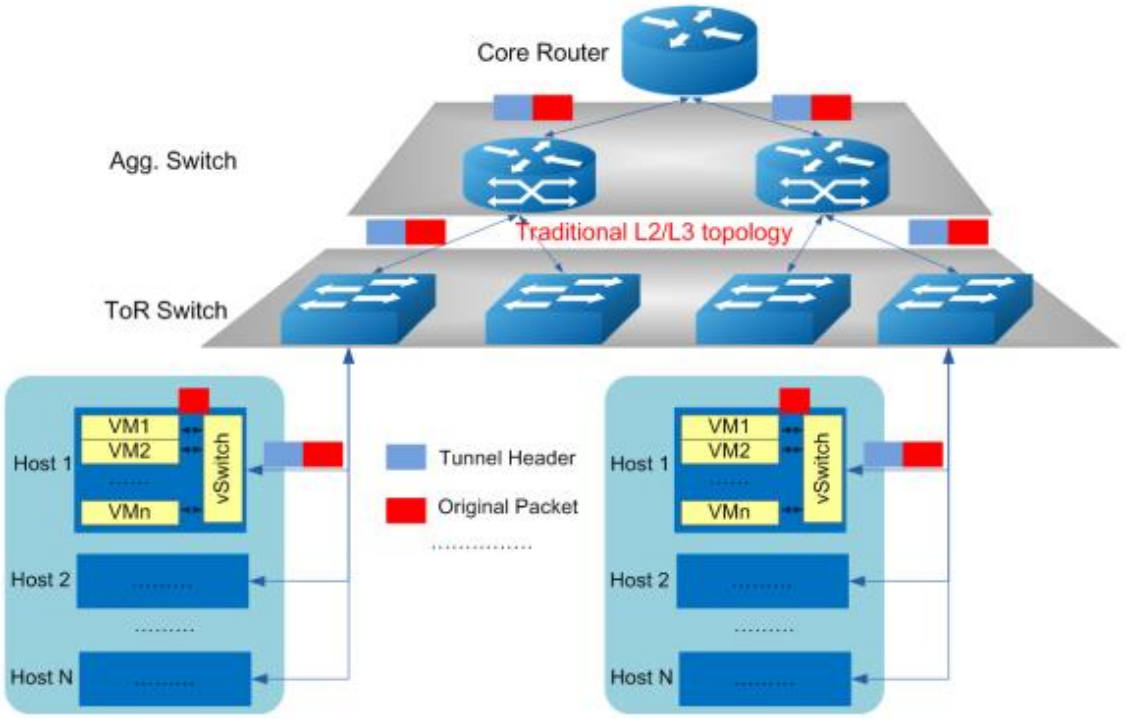
80%的程序员所欠缺的网络知识

管理网络、业务网络、存储网络



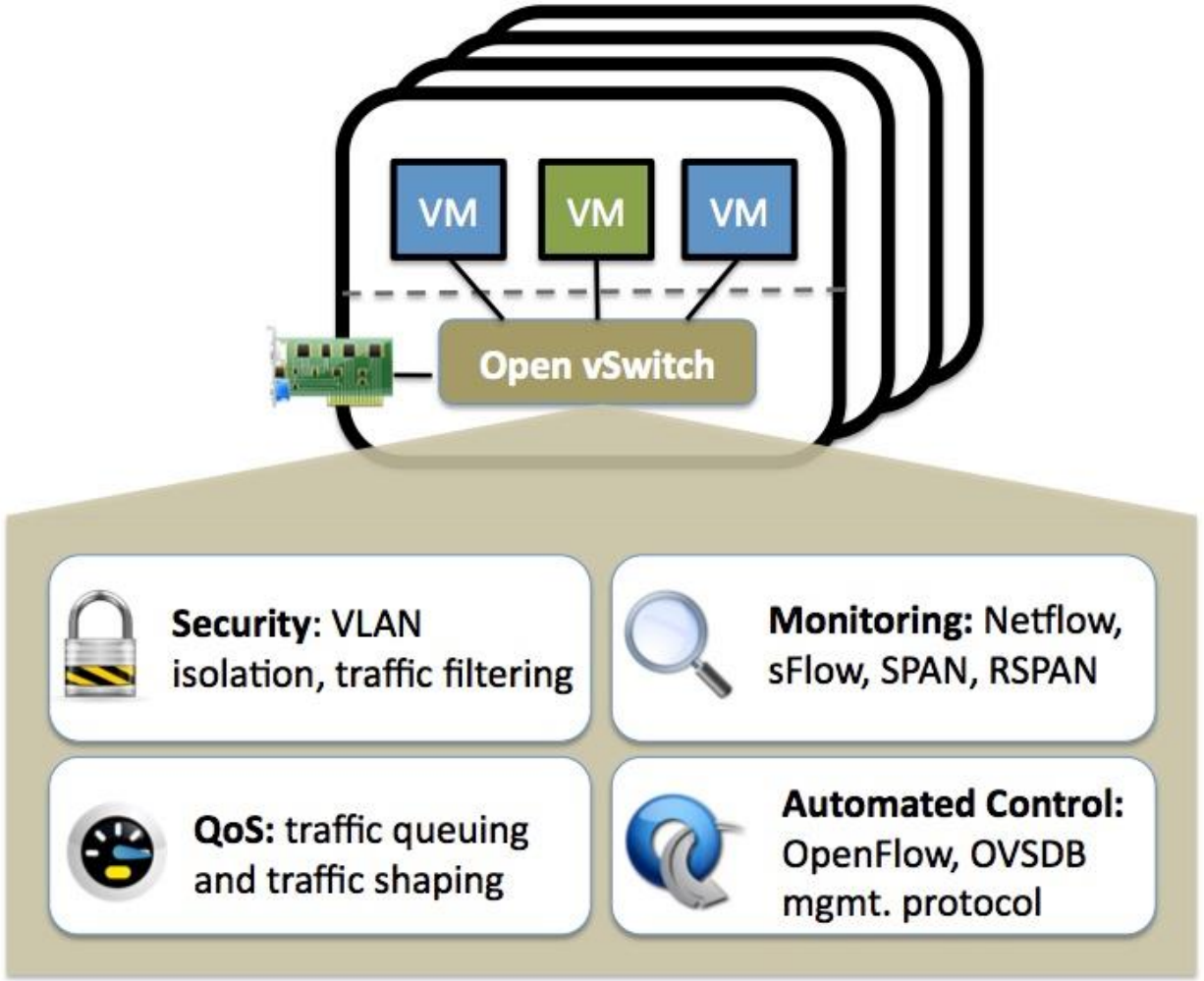
网络的最前沿技术之网络虚拟化

服务器虚拟化技术与网络虚拟化技术相辅相成



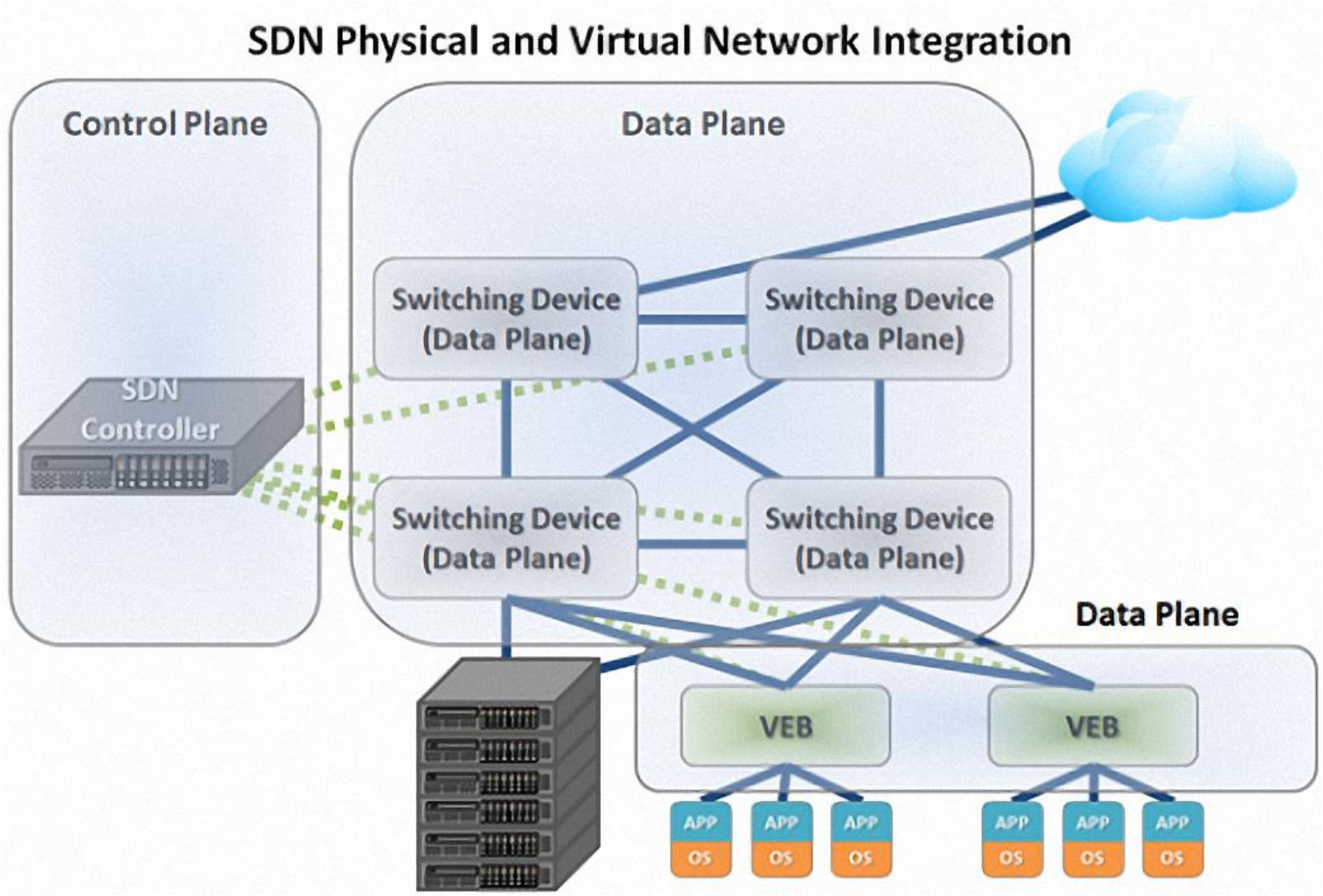
网络的最前沿技术之网络虚拟化

Open vswitch



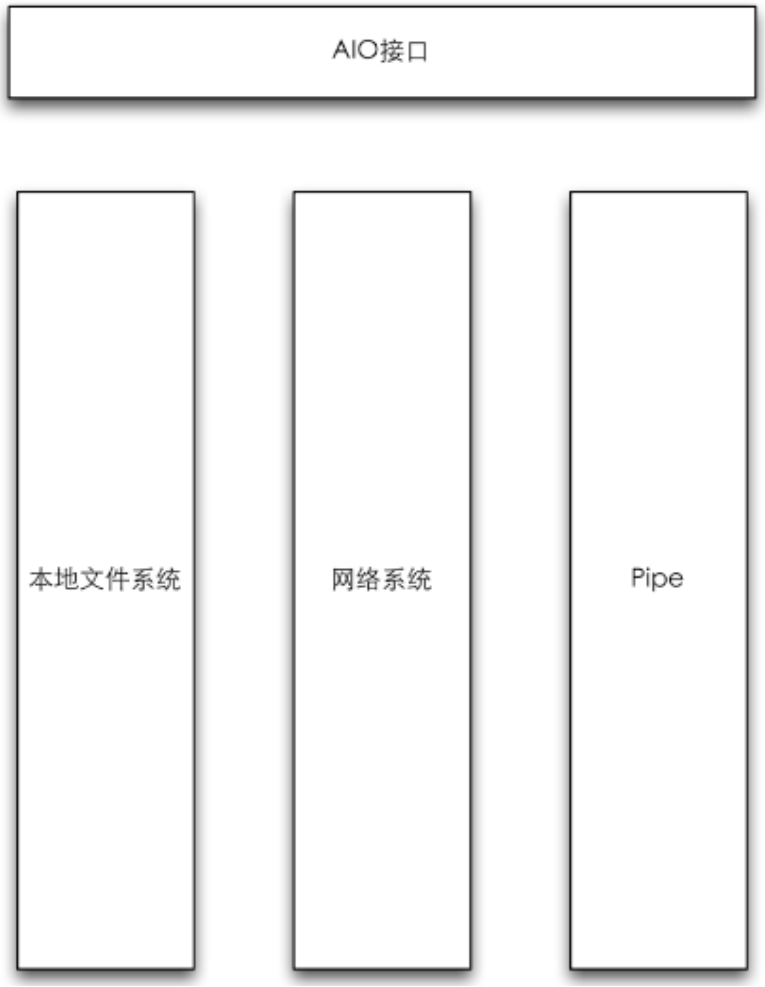
网络的最前沿技术之网络虚拟化

SDN&网络虚拟化



关于NIO与AIO的那些小秘密

Linux AIO的奇葩十年路



目标VS现实

- Glibc AIO 被逐渐淘汰
- 2003年新的Linux Kemal的AIO方案应用到Linux Kernel 2.4
- IBM来接手，新的 retry在2.6内核实现，仍然没有解决遗留问题
- 2007年，Oracle尝试用syslet方案实现新的AIO，也没有了下文
- 直到今天，Linux AIO仍然陷入泥潭，没有清晰的路线图和答案

关于NIO与AIO的那些小秘密

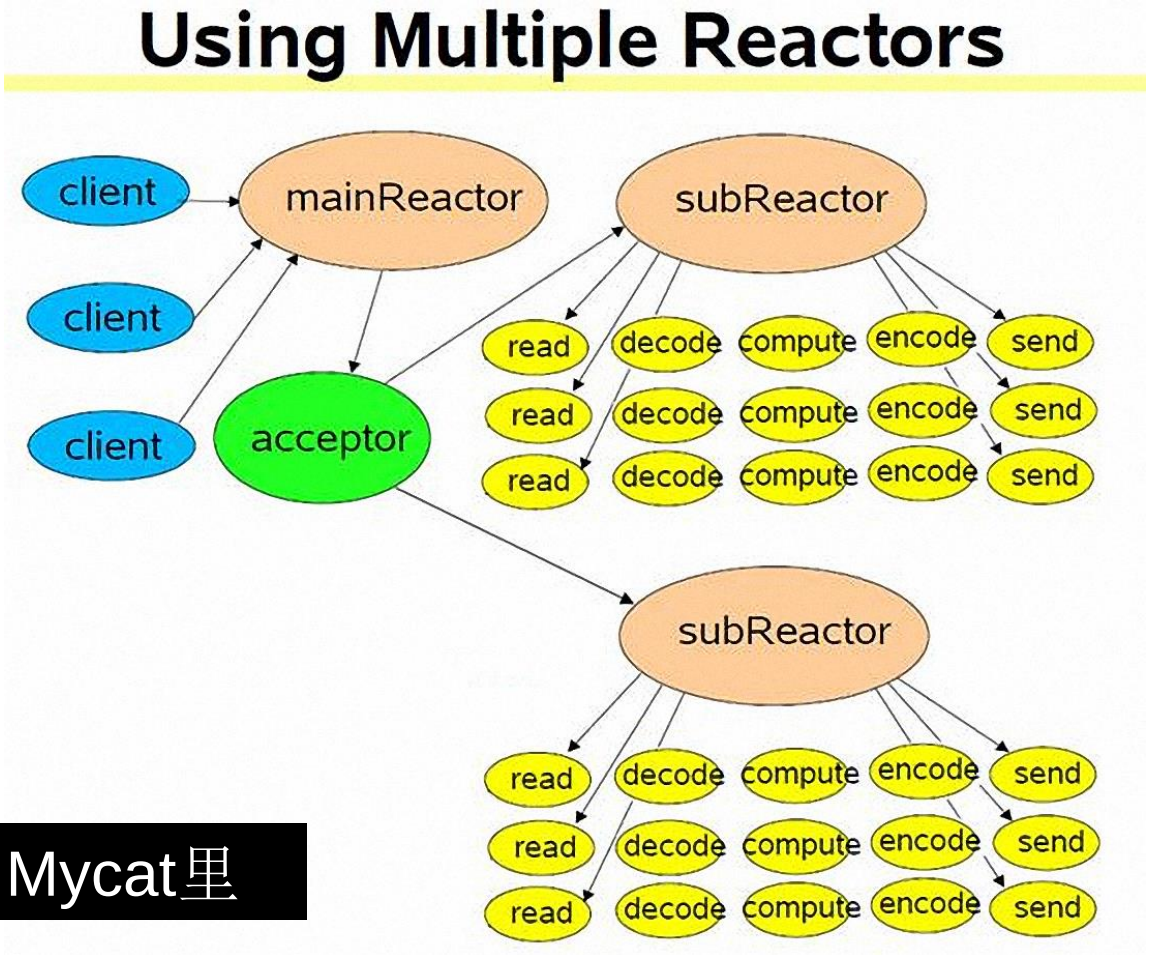
Java NIO VS AIO

AIO带来了编程的大幅简化和优化
AIO性能目前不如NIO
AIO当前比较适合大文件的读写
现阶段网络编程主要还是NIO

关于NIO与AIO的那些小秘密

NIO最佳实践

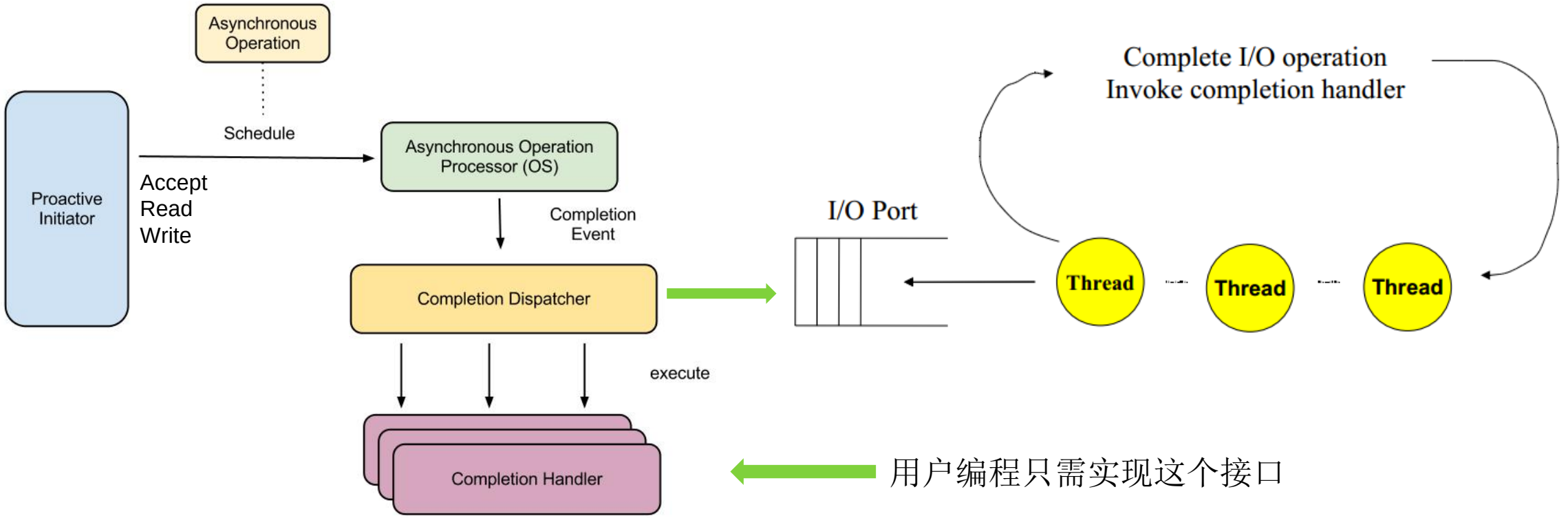
- Reactor线程的数量
- Direct Buffer Pool的使用
- Reactor线程与异步线程池的合理使用



国内最好的NIO实践，就在开源项目Mycat里

关于NIO与AIO的那些小秘密

Java AIO编程框架

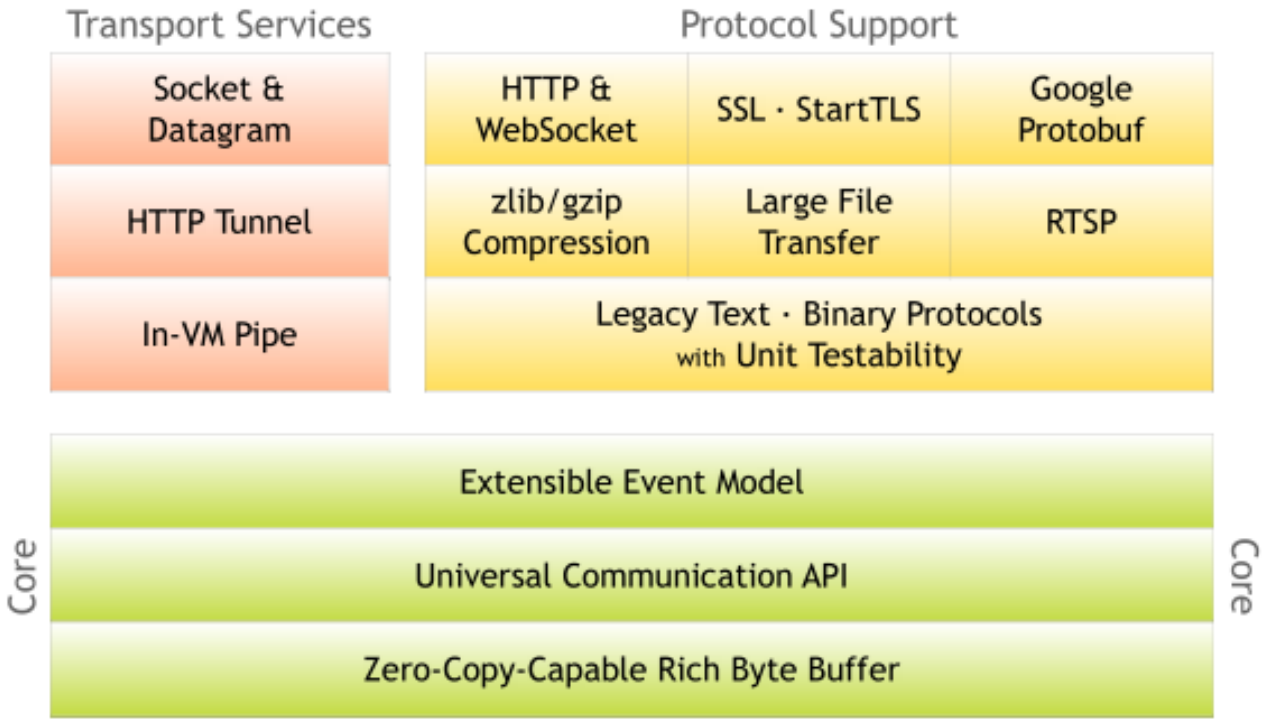


网络编程中的一朵金花之Netty

为什么选择Netty

- NIO入门门槛高，精通很困难
- 除了NIO本身，企业级的Socket Server还需要大量外围代码开发
- Netty是业界最流行的NIO框架之一，几乎没有对手
- Netty的作者也是开发了大名鼎鼎的Apache Mina的作者

网络编程中的一朵金花之Netty

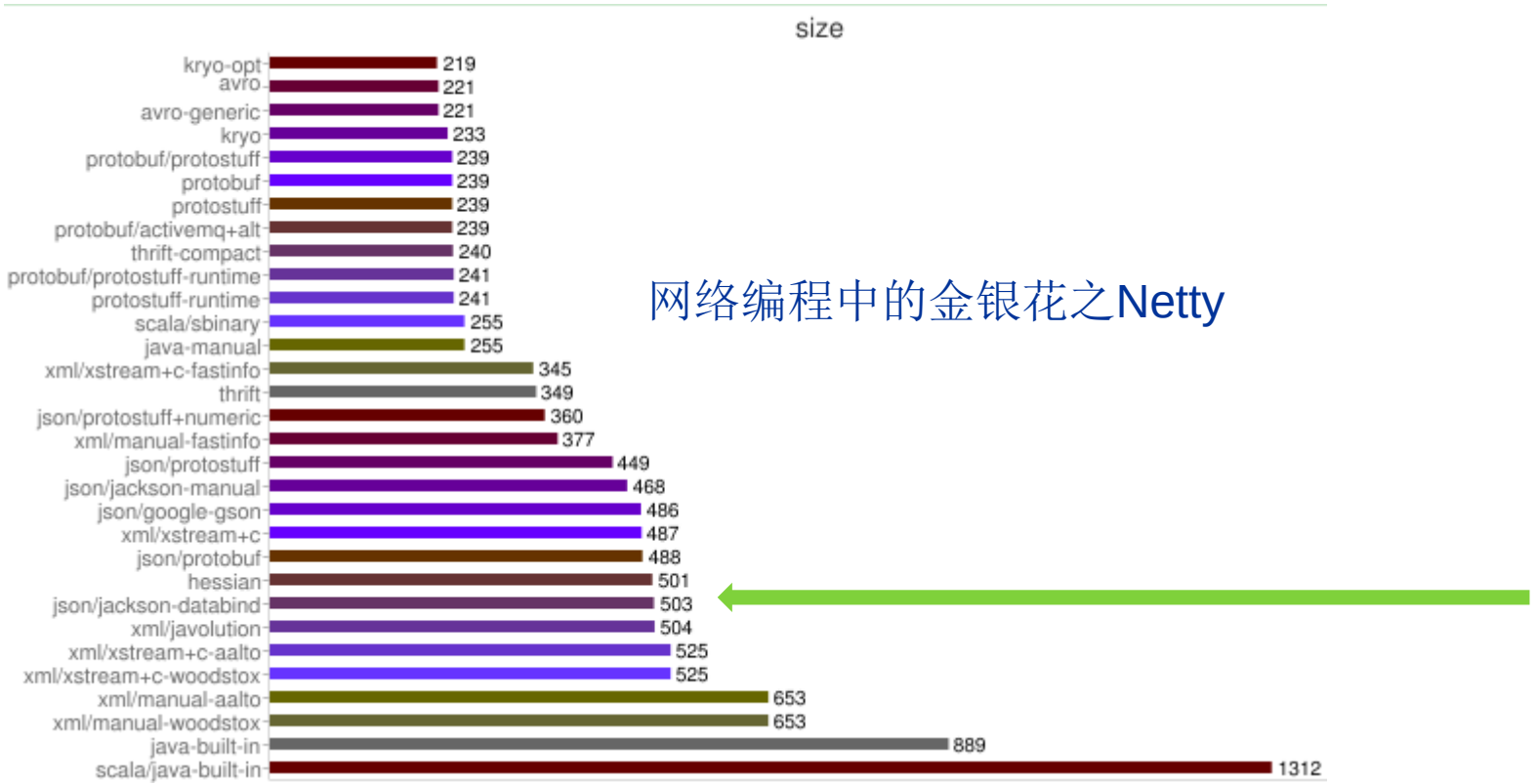


Netty的优势

- 多种Reactor线程池模式
- 网络数据串行处理，减少线程切换
- 强大的Buffer池
- 娴熟的高并发编程技巧（Volatile、CAS、ThreadLocal等的使用）
- 作者很多年网络编程经验的积累与提升

网络编程中的一朵金花之Netty

Netty默认提供了对Google Protobuf的支持，通过扩展Netty的编解码接口，用户可以实现其它的高性能序列化框架，例如Thrift的压缩二进制编解码框架。

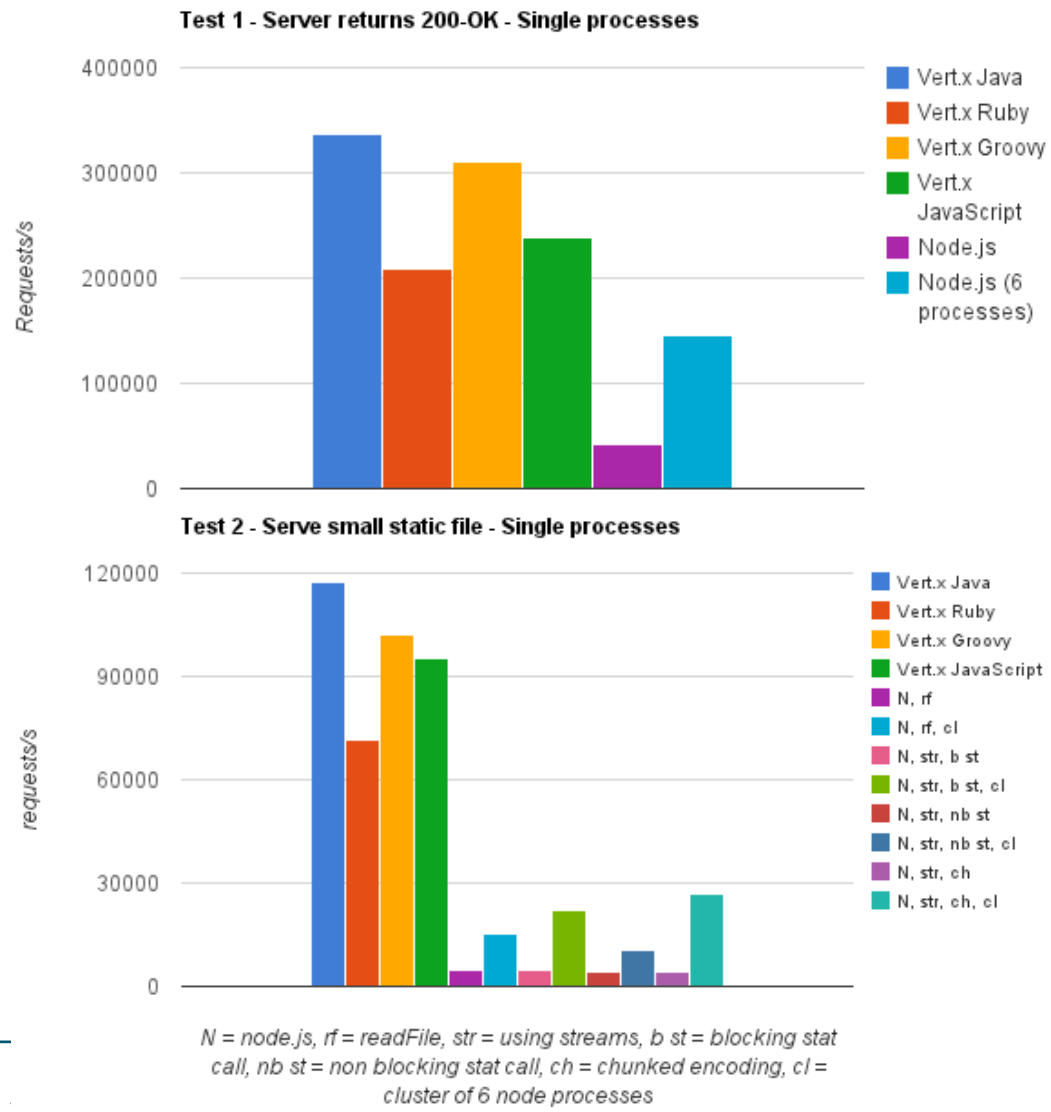


网络编程中的金银花之Netty

网络编程中的一朵金花之Netty

ATAGURU 炼数成金

Vert.x—— JVM上的Node.js



Peak JSON responses per second, EC2 large

Framework	Peak performance
netty	36,717 100.0%
servlet	28,745 78.3%
vertx	28,012 76.3%
gemini	25,264 68.8%
go	13,472 36.7%
compojure	13,135 35.8%
nodejs	10,541 28.7%
wicket	8,431 23.0%
express	7,258 19.8%
webgo	6,881 18.7%
play	5,181 14.1%
php	5,054 13.8%
rack-jruby	4,336 11.8%
tapestry	3,901 10.6%
spring	3,874 10.6%
wsgi	3,138 8.5%
rack-ruby	2,164 5.9%
grails	2,045 5.6%
sinatra-ruby	1,469 4.0%
django	1,156 3.1%
rails-jruby	871 2.4%
sinatra-jruby	692 1.9%
rails-ruby	687 1.9%
cake	59 0.2%

Higher is better. These data represent processes per second in response to HTTP requests with keep-alives enabled.

In this test, the server's response is a single trivial newly-instantiated in-memory object serialized to JSON. This is an EC2 test run on a pair of Large instances. WeighHTTP was configured to use two threads (to utilize the virtual machine's two cores) and results were captured at various WeighHTTP concurrency levels ranging from 2^3 to 2^8.

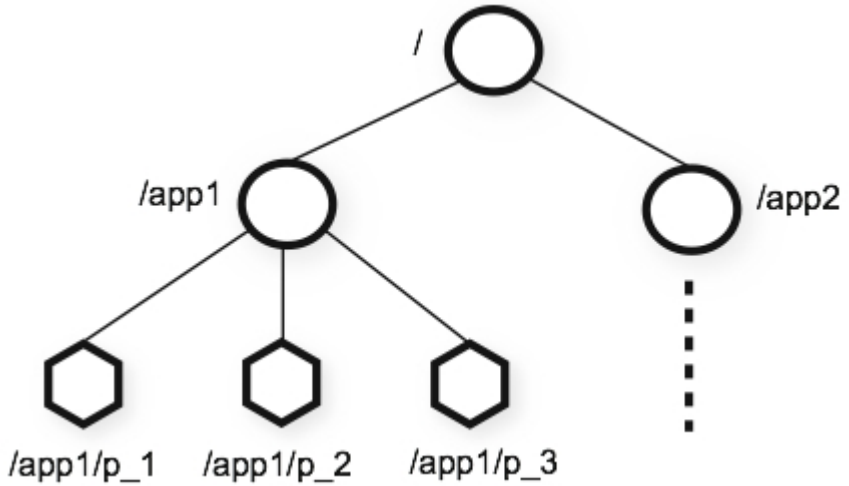
网络编程中的瑞士军刀之Zookeeper

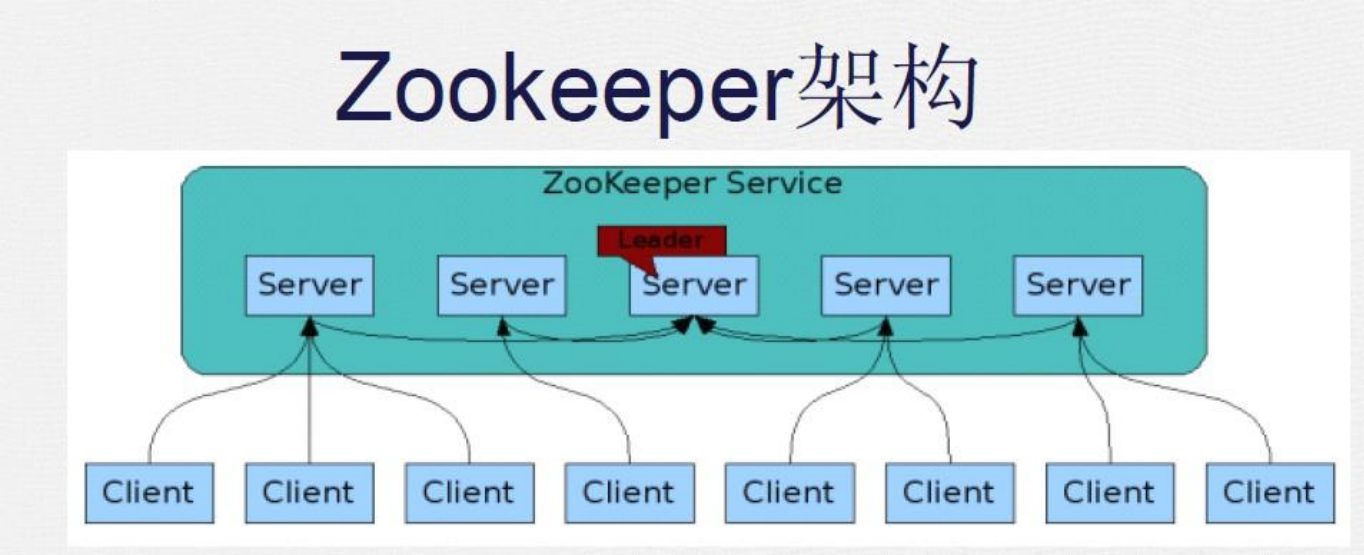
Zookpeer能做什么

- 类似JNDI的**命名服务**
- 实现分布式系统中的**配置服务**
- 提供简单好用的**分布式同步服务**
- 提供简单好用的**分布式协调框架**
- 可以作为一个简单的可靠的**消息队列**

网络编程中的瑞士军刀之Zookeeper

Zookpeer的原理





- ZK提供类似文件目录树的数据结构，每个节点可以设置byte[]数据
- 节点类型可以是持久化保存的，也可以是临时的(EPHEMERAL)

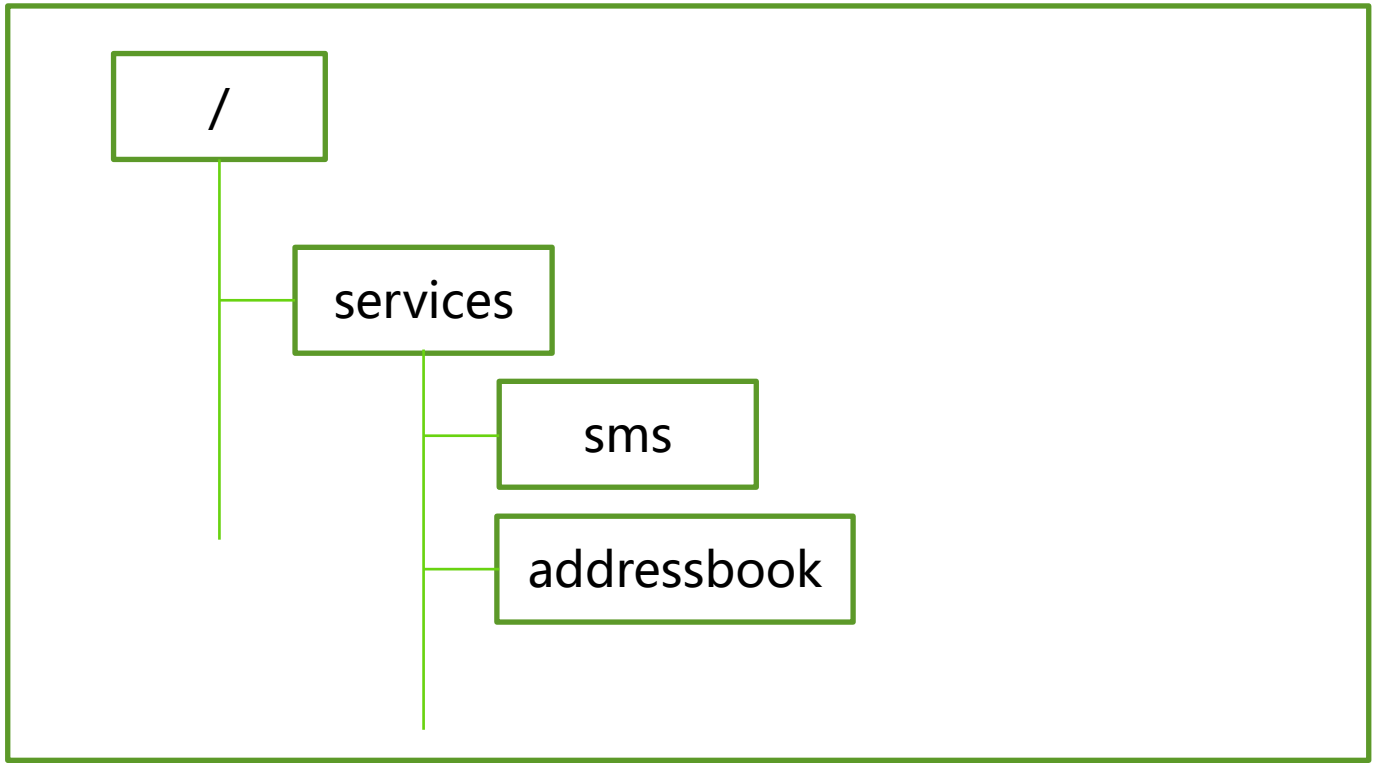
网络编程中的瑞士军刀之Zookeeper

- **Zookeeper 特点**
- 原子性：更新要么成功，要么失败，不会出现部分更新。
- 可靠性：一旦数据更新成功，将一直保持，直到新的更新。
- 单一性 ：无论客户端连接哪个server，都会看到同一个视图。
- 及时性：客户端会在一个确定的时间内得到最新的数据。
- 等待无关：慢的或者失效的client不得干预快速的client的请求，使得每个client都能有效的等待。
- 顺序一致性：按照客户端发送请求的顺序更新数据，包括全局有序和偏序两种：全局有序是指如果在一台服务器上消息a在消息b前发布，则在所有Server上消息a都将在消息b前被发布；偏序是指如果一个消息b在消息a后被同一个发送者发布，a必将排在b前面

网络编程中的瑞士军刀之Zookeeper

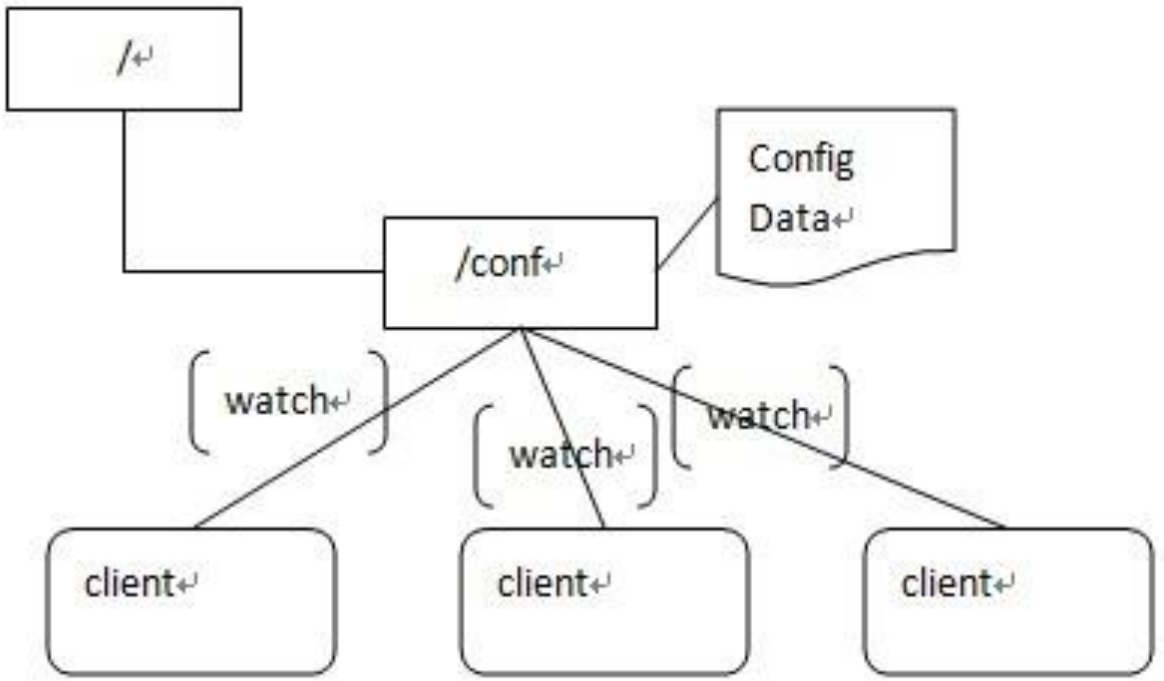
ZK应用场景之命名服务

ZK创建一个节点后，节点的路径就是全局唯一的，可以作为全局名称使用。



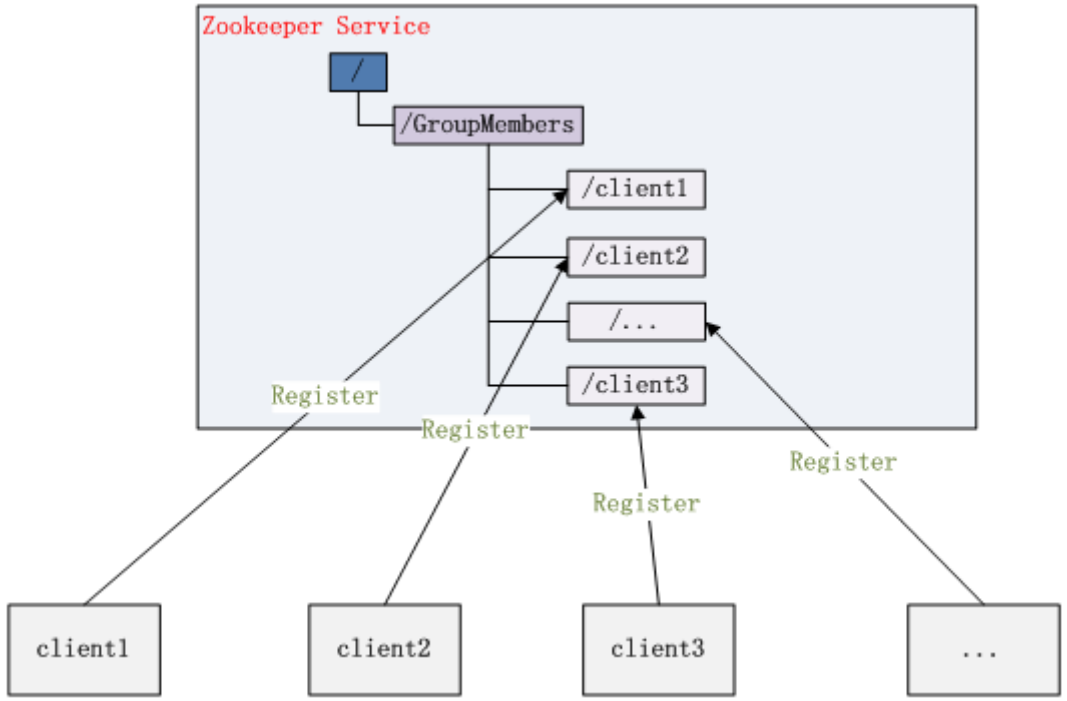
网络编程中的瑞士军刀之Zookeeper

ZK应用场景之分布式配置管理



网络编程中的瑞士军刀之Zookeeper

ZK应用场景之分布式集群管理

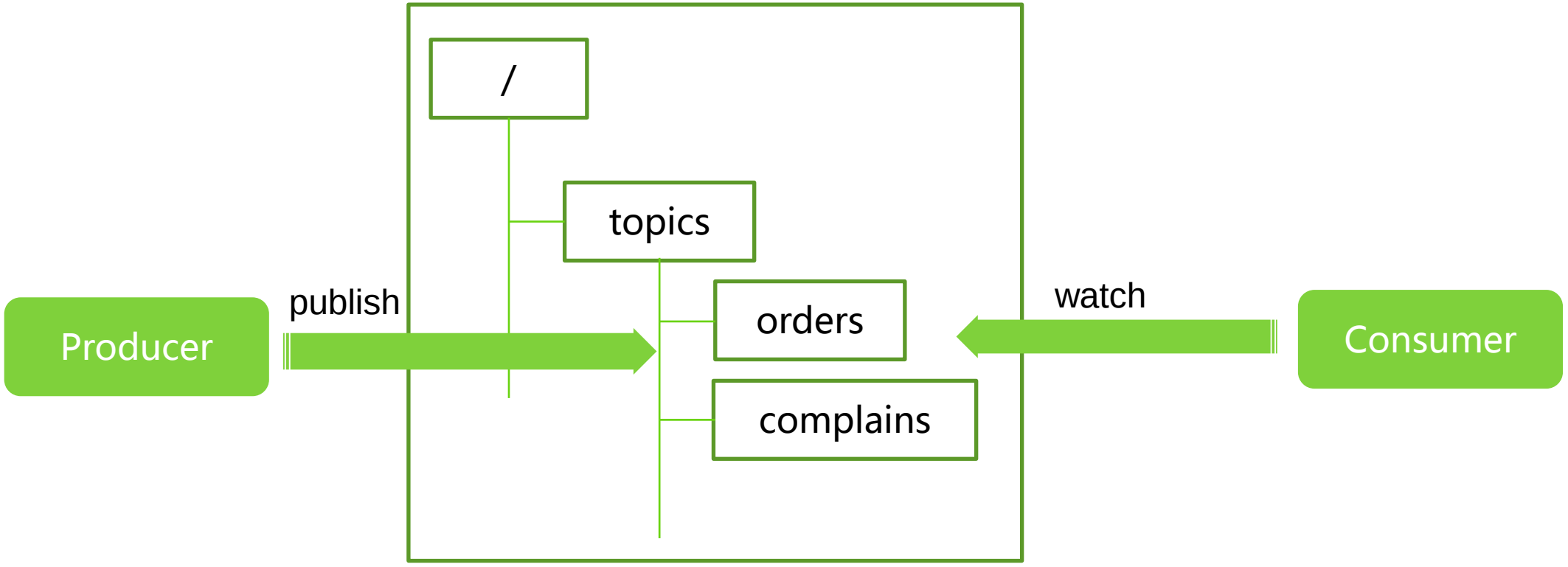


- 每个节点（进程）启动的时候在ZK路径GroupMembers下建立子路径（节点ID为路径名），通信地址Endpoint则写到路径的Data中。
- 每个节点监听ZK路径GroupMembers，当集群中增加新节点或故障节点下线，每个节点都会获得通知

网络编程中的瑞士军刀之Zookeeper

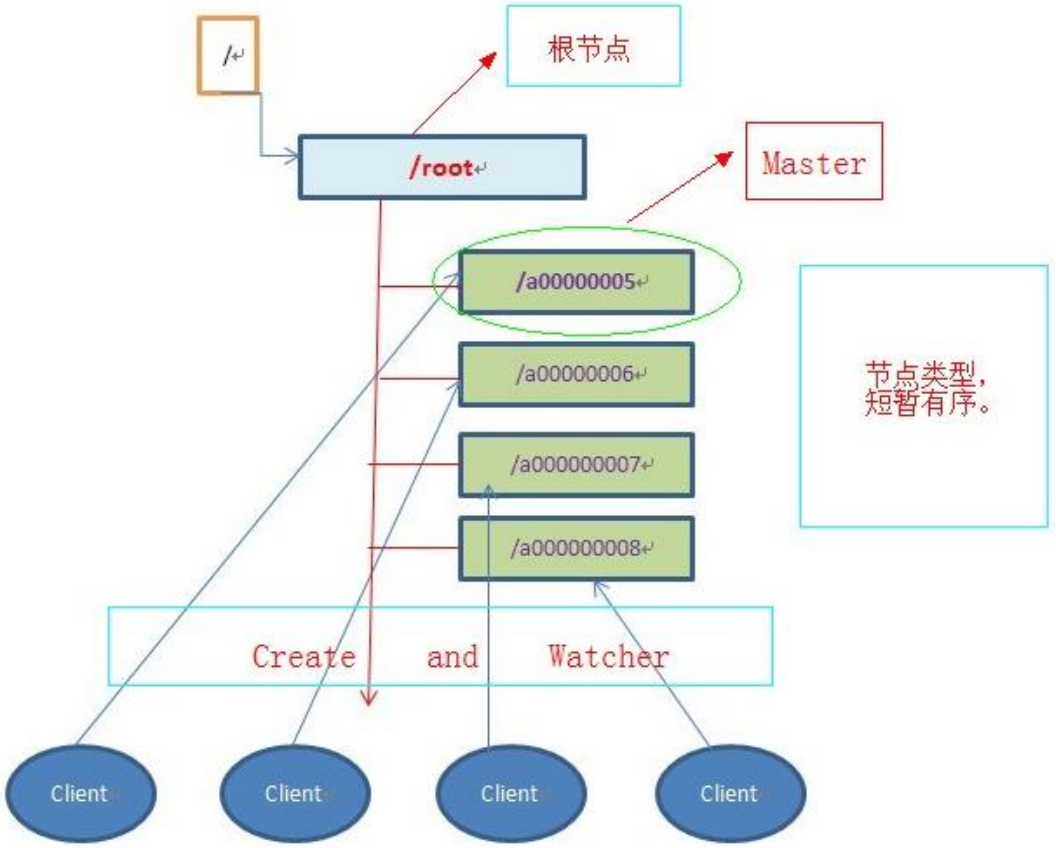
ZK应用场景之消息队列

消息发送方在ZK上创建一个Path，发送消息时，将消息信息设置为该Path的内容，而消息接收方则监听此Path，实现了简单可靠的发布订阅模式的消息队列



网络编程中的瑞士军刀之Zookeeper

ZK应用场景之分布式锁

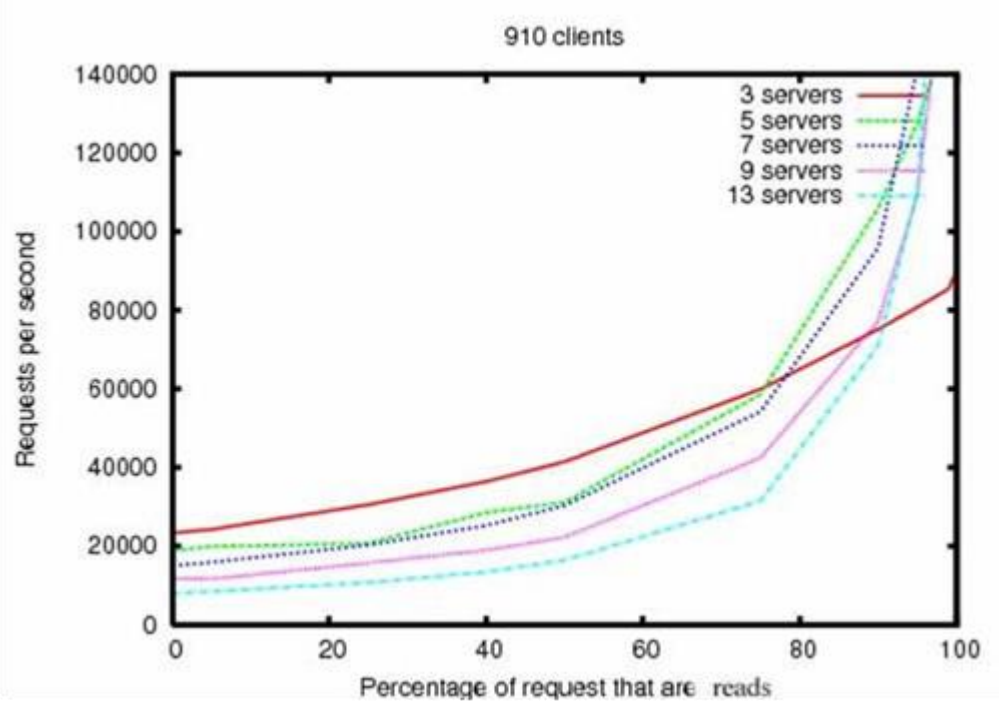


Zookeeper能保证数据的强一致性，用户任何时候都可以相信集群中每个节点的数据都是相同的。一个用户创建一个节点作为锁，另一个用户检测该节点，如果存在，代表别的用户已经锁住，如果不存在，则可以创建一个节点，代表拥有一个锁。

网络编程中的瑞士军刀之Zookeeper

Zookeeper的性能调优和规模

- ZK的日志文件和snapshort文件分别放在两块硬盘上
- Leader节点不允许Client连接



网上所能见到的信息，最大为1万个连接（客户端），5节点的ZK集群

网络编程中的瑞士军刀之Zookeeper

Zookeeper客户端Curator

> "Guava is to Java what Curator is to ZooKeeper"
> Patrick Hunt, ZooKeeper committer

Fluent Style风格

```
zk = CuratorFrameworkFactory
        .builder()
        .namespace("myapp")
        .connectString(zk_address)
        .connectionTimeoutMs(
            session_timeout < 5000 ? session_timeout : 5000)
        .sessionTimeoutMs(this.session_timeout)
        .retryPolicy(new RetryOneTime(10000)).build();
```

NetFlix出品

极大程度上减少了程序猿的负担

接管了Client与Server的连接重连问题
提供了各种常用ZK应用场景的抽象封装（如共享锁服务、Leader选举）

网络编程中的瑞士军刀之Zookeeper

Curator的关键用法

列出子目录：CuratorFramework.getChildren().forPath(path)

```
public boolean exists(String parentPath, String path) throws Exception {
    Stat stat = zk.checkExists().forPath(parentPath + "/" + path);
    return (stat != null);
```

创建目录

```
result = zk.create().withMode(createMode).
                        forPath(path,nodeData);
```

监听ZK节点的变化并做出相应的处理

```
private void watchZKPath(String path) {
  PathChildrenCacheListener plis = new PathChildrenCacheListener() {
    @Override
    public void childEvent(CuratorFramework client,
        PathChildrenCacheEvent event) throws Exception {
      String linkPath = ZKPaths.getNodeFromPath(event.getData()
          .getPath());
      switch (event.getType()) {
      case CHILD_ADDED: {

      break;
      }

      case CHILD_REMOVED: {
        break;
      }
      default: {
        LOGGER.warn("not handled event " + event);
      }
      }

    }
  };
  PathChildrenCache cache = new PathChildrenCache(zk, path, false);
  cache.getListenable().addListener(plis);
  cache.start();
```

网络编程中的瑞士军刀之Zookeeper

Curator的几个问题

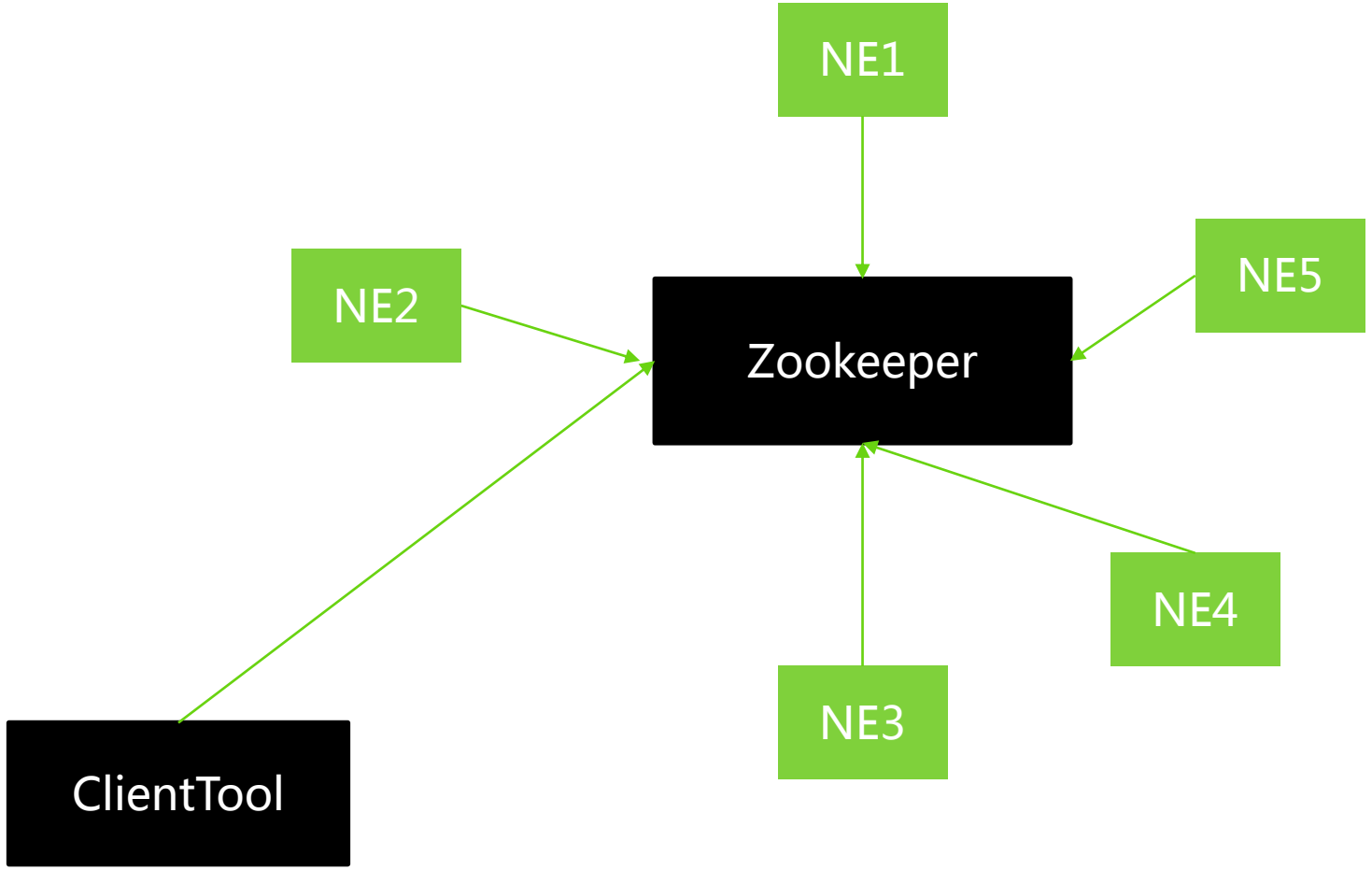
若ZK Server还未启动，用户程序先启动了，则虽然Curator能够后来自动重连上，
但之前的创建节点和监听事件将不会起效.

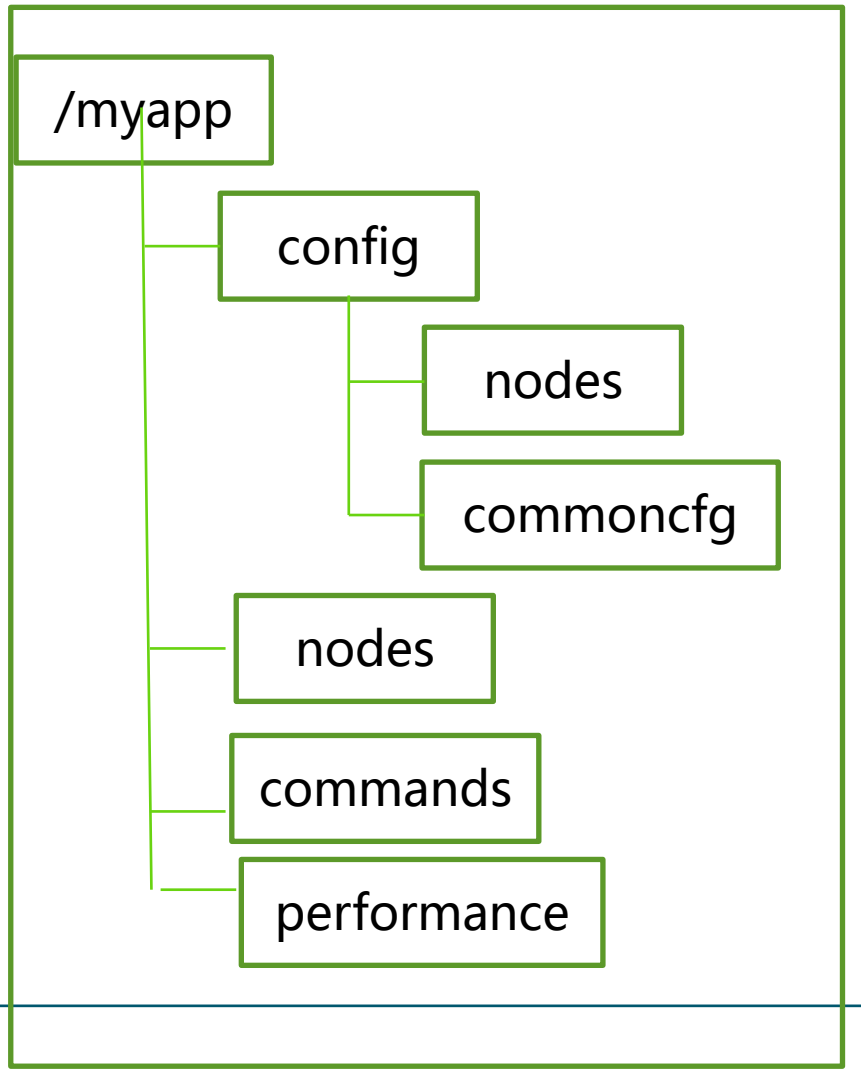
```
List<PathChildrenCache> allZKWathers = new LinkedList<PathChildrenCache>();

PathChildrenCache cache = new PathChildrenCache(zk, path, false);
cache.getListenable().addListener(plis);
cache.start();
allZKWathers.add(cache);
```

方法一：org.apache.zookeeper.Watcher来监听Connection的状态，
CuratorZookeeperClient(connectString,**TimeoutMs, int connectionTimeoutMs,watcher, retryPolicy)**
当连接建立后触发PathChildrenCache的rebuild方法，重新监听路径
方法二：定时线程，每隔几分钟调用PathChildrenCache的rebuild方法

Netty+Zookeeper实践项目分析





Thanks

FAQ时间