

实验4： ZooKeeper协调服务

2学时 · 起点镜像 L04

今天的成果：**观察一次有证据的主节点交接**

本次要分清的角色

ZooKeeper实例：提供协调服务

候选worker：借助协调服务竞争业务主节点

客户端会话：连接、心跳与超时

worker选举决定业务主节点；ZooKeeper leader协调服务端写入。

实验任务与90分钟安排

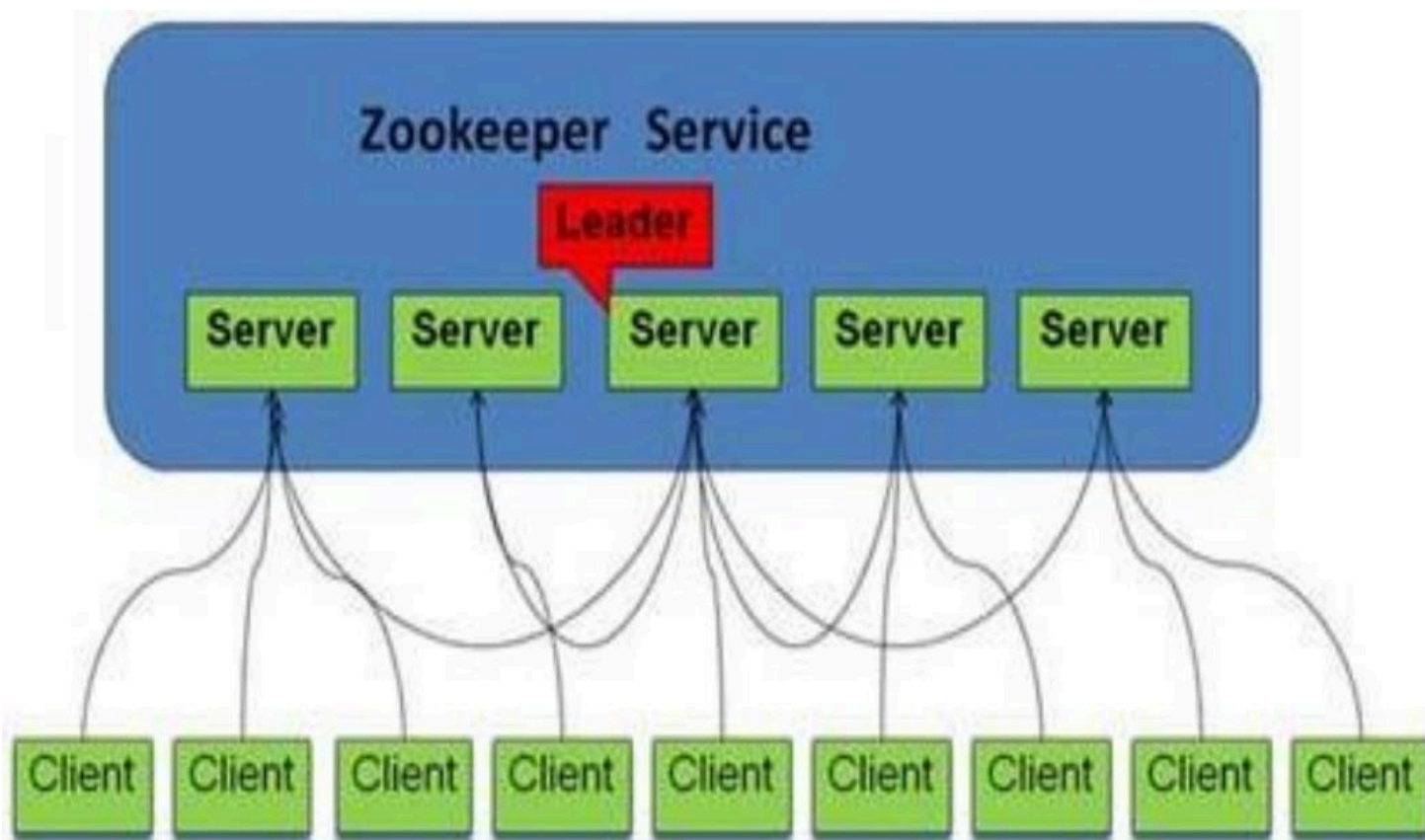
阶段	时间	观察结果
概念与启动	15分钟	三实例与客户端连接
节点与监听	20分钟	数据变化、临时节点删除
候选选举	25分钟	唯一主节点及继任者
多数派	15分钟	2实例可写、1实例等待
核对	15分钟	日志、记录与解释

ZooKeeper的客户端与服务集群

客户端通过会话访问节点树

集群内服务器协调更新

应用用节点表达配置、成员和候选状态



三个实例的教学环境

```
labctl stop  
labctl start lab04  
labctl status  
cd ~/work/lab04
```

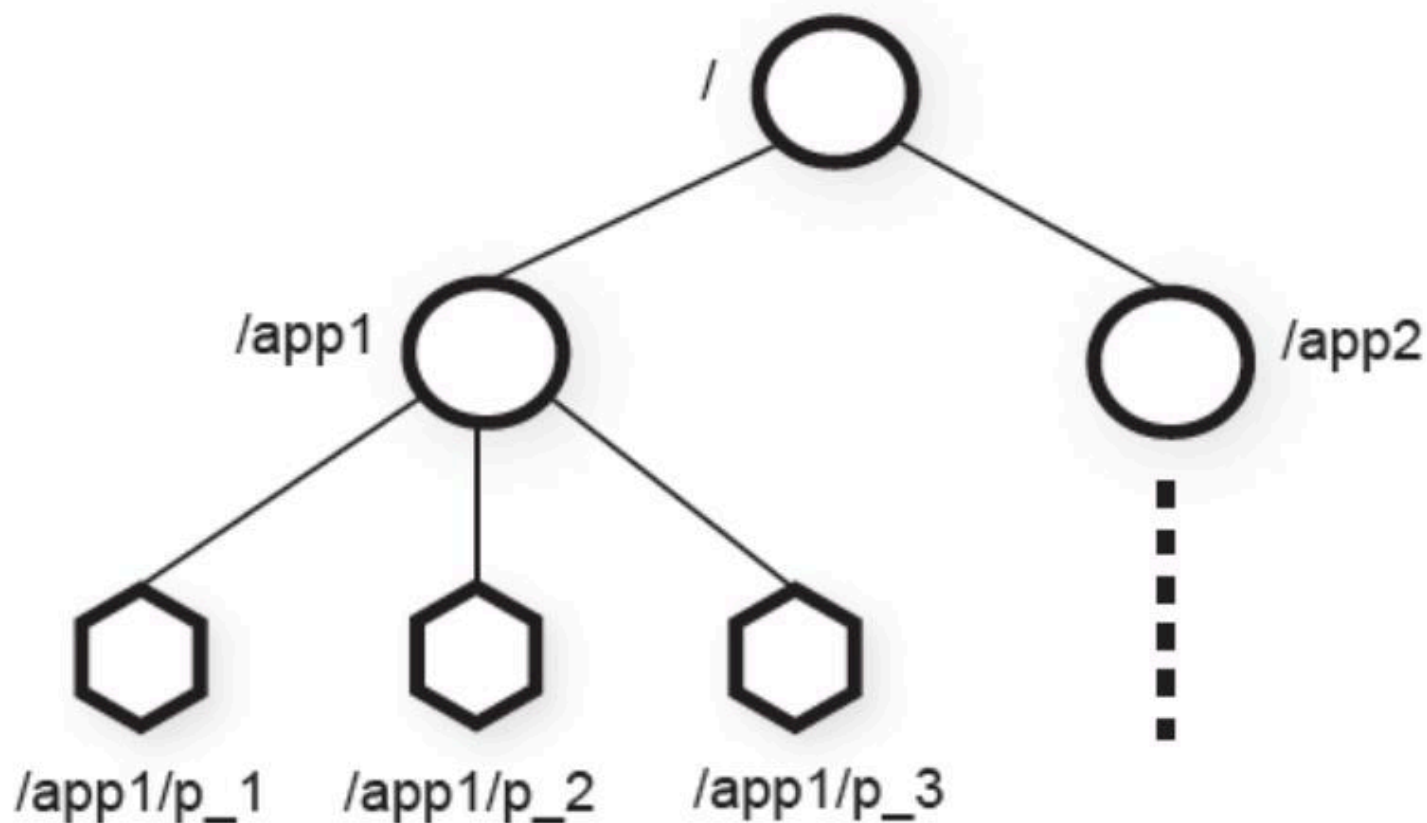
端口2181、2182、2183，各用独立数据目录。

znode树与路径

/ 为根节点， /lesson 为本次练习节点

节点可同时保存少量数据并拥有子节点

路径标识逻辑协调状态



持久节点与临时节点

类型	创建方式	生命周期
持久节点	<code>create /lesson hello</code>	保留到显式删除
临时节点	<code>create -e /temporary hello</code>	绑定创建者会话

关闭创建会话后，观察两类节点各自的状态。

两个客户端共享一份节点状态

终端A	终端B
create /lesson hello	get /lesson
set /lesson changed	get /lesson
create -e /temporary hello	ls /
quit	ls /

每次操作后记录另一个客户端看到的變化。

用命令行读写节点

```
/opt/zookeeper/bin/zkCli.sh -server 127.0.0.1:2181
```

```
create /lesson hello
```

```
get /lesson
```

```
set /lesson changed
```

监听一次变化

```
get -w /lesson  
set /lesson another-value  
get /lesson
```

记录watch事件；本次普通watch触发后需重新注册。

watch的注册、触发与再次注册

```
get -w /lesson  
set /lesson v2  
set /lesson v3  
get -w /lesson  
set /lesson v4
```

普通watch触发一次后消耗；再次注册后接收后续变化。

会话连接状态与超时

客户端与ZooKeeper建立会话

短暂断线时，会话可等待重连

正常关闭会话会结束其临时节点

异常断线后，服务在会话超时后清理节点

实验记录操作方式和观察到的时间顺序。

临时节点依赖会话

```
create -e /temporary hello
```

用第二个zkCli查看，再退出第一个会话。

正常关闭或会话超时后，临时节点消失。

应用选举与服务集群选举

层次	角色	本次查看方式
ZooKeeper服务	协调协议的leader与follower	三实例日志与状态
worker应用	执行业务任务的LEADER	election.py输出

worker通过ZooKeeper节点竞争应用主节点。

顺序候选与继任关系

候选者创建带顺序编号的临时节点

队列顺序用于决定谁获得选举资格

当前持有者退出后，后续候选继续竞争

Kazoo Election提供这套协调流程

具体候选节点与编号可在程序输出中观察。

election.py中的关键位置

```
client = KazooClient(hosts="127.0.0.1:2181,...")
election = client.Election("/course/election",
args.name)
thread = threading.Thread(target=lambda:
election.run(lead))
```

`/course/election`：本组选举路径

`lead()`：当选后执行的函数

`stop.wait()`：维持当前进程，等待结束信号

启动两个候选

```
# SSH终端A  
python election.py worker-A  
# SSH终端B  
python election.py worker-B
```

同一时刻只应有一个进程输出LEADER。

主节点退出后的交接

记录当前LEADER名称

在该进程终端按Ctrl+C

观察另一个候选成为LEADER

保存前后日志和实际名称

以实际LEADER输出确认首次当选者。

主节点交接的日志证据

终端A: LEADER worker-A

操作: 在A按Ctrl+C

终端B: LEADER worker-B

记录实际当选顺序、退出操作和继任输出。

`evidence.json` 填写日志中观察到的before和after。

记录观察结果

在evidence.json中填写实际名称：

```
{"before": "worker-A", "after": "worker-B"}
```

```
labctl check lab04
```

三个实例需要多数派

可用实例	能否获得多数派
3	能
2	能
1	不能提交新的写入

多数派为2，避免两个少数派各自决定更新。

多数派的计算

$$q = \left\lfloor \frac{n}{2} \right\rfloor + 1$$

当 $n = 3$ 时, $q = 2$:

3个实例存活: 可获得多数派

2个实例存活: 仍可获得多数派

1个实例存活: 写入等待多数派恢复

多数派集合相交, 为更新决策提供共同成员。

多数派实验的分步验证

步骤	操作	客户端验证
1	停3号实例	连接2181, set /lesson two
2	停2号实例	新写入等待或连接重试
3	启2号实例	重新连接并验证写入
4	启3号实例	恢复完整三实例

客户端等待约15秒后可Ctrl+C，进入恢复步骤。

停止与恢复实例

```
labctl zk-stop 3  
labctl zk-stop 2  
labctl zk-start 2  
labctl zk-start 3
```

每一步用客户端观察；等待超时可Ctrl+C退出。

三实例端口与数据目录

实例	客户端端口	本地数据
1	2181	zookeeper/1
2	2182	zookeeper/2
3	2183	zookeeper/3

`~/lab-state/zookeeper/1/zoo.cfg` 等文件保存各实例配置。

同一虚拟机提供进程级故障观察环境。

单机三进程的故障范围

三个进程有独立目录和端口

虚拟机整体关闭时，三个进程共同停止

实例共享主机、内存与供电

本拓扑用于观察进程级故障、会话和多数派行为。

报告截图核对

第1步：图1 watch；图2、图3 临时节点退出前后

第2步：图4、图5 主节点交接前后

第3步：图6 单实例等待；图7 恢复写入

截图内容与图号按Word模板填写，直接插入对应位置。

常见现象与诊断方法

现象	诊断
NodeExists	get查看节点，按目标set更新
ConnectionLoss	检查端口、实例数和服务日志
候选等待	查另一终端是否已当选
临时节点仍在	确认所属会话及超时过程
交接记录不符	对照两终端实际日志重填

完成观察后恢复三个实例，再停止课程服务。

提交Word报告

“源码与运行说明”： election.py全文

完成各步骤实际结果、规定截图和已有书面栏目

AI使用情况： 简短说明或一张截图； 未使用填“未使用”

保存后重新打开，核对文本和截图清晰可读

正式提交一份填写完成的Word报告。