lect21: Clustering

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/150aaf12-1b78-4d31-a24c-af678daffafa/21-clustering_.pdf

- **大规模的系统的三个要求(RAS)？**

RAS的全称为 Reliability, Availability, Serviceability。(课件上还加上了scalability)

Reliability（可靠性）：系统必须尽可能的可靠，不会意外的崩溃，重启甚至导致系统物理损坏，这意味着一个具有可靠性的系统必须能够对于某些小的错误能够做到自修复，对于无法自修复的错误也尽可能进行隔离，保障系统其余部分正常运转。Availability（可用性）：系统必须能够确保尽可能长时间工作而不下线，即使系统出现一些小的问题也不会影响整个系统的正常运行，在某些情况下甚至可以进行 Hot Plug 的操作，替换有问题的组件，从而严格的确保系统的宕机时间在一定范围内。Serviceability：系统能够提供便利的诊断功能，如系统日志，动态检测等手段方便管理人员进行系统诊断和维护操作，从而及早的发现错误并且修复错误。

- 集群：优点

The main principle behind clustering is that of redundancy.

- Reliability
  - Remove single points of failure
- Availability
  - Overall availability is $1-(1-f\%)^n$
- Serviceability
  - More complex than a single application server
  - But we could get ability for hot upgrade
- Scalability
  - It is cheaper to build a cluster using standard hardware than to rely on multiprocessor machines.
  - Extending a cluster by adding extra servers can be done during operation and hence is less disruptive than plugging in another CPU board.

负载均衡的策略：

1. 轮流 2. 最小负载 3. 随机

会话粘滞性问题：loadbalence 一开始把一个session开到Aserver上，如果下次调到B上，A上面写的看不见了。

方法1：无论A多忙每次都调用到同一台机器上，这样破坏了负载均衡的意义

方法2：session 放到一个第三方上（redis），redis成为了瓶颈如果redis 崩坏了就die了

这两个都写了example 放在canvas了。

nginx

```
server {
 location / {
   root /data/www;
 }
 location /images/ {
   root /data;
 }
}
```

gateway

src= /image/1.png -> /data/1.png

```
server {
 location / {
  proxy_pass http://localhost:8080/;
 }
 location ~ \.(gif|jpg|png)$ {
  root /data/images;
 }
}
```

以这三种结尾的，导到这个路径

- Load balancing methods
  - The following load balancing mechanisms (or methods) are supported in nginx:
    - round-robin — requests to the application servers are distributed in a round-robin fashion,
      只考虑连接数，但是每个连接化多少时间不管
    - least-connected — next request is assigned to the server with the least number of active connections,
      把用户的ip hash 一下，这样就不均衡了
    - ip-hash — a hash-function is used to determine what server should be selected for the next request (based on the client's IP address).

– The simplest configuration for load balancing with nginx may look like the following:

```
http {
 upstream myapp1 {
  server srv1.example.com;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80; 访问端口80，就分发访问srv1,2,3
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

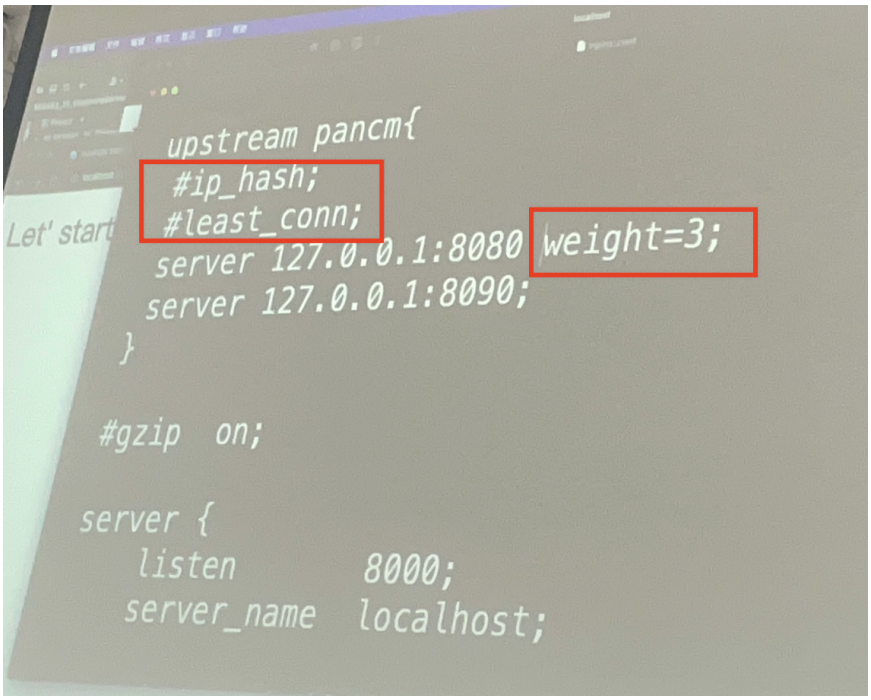
– Reverse proxy implementation in nginx includes load balancing for HTTP, HTTPS, FastCGI, uwsgi, SCGI, and memcached.

- Weighted load balancing

```
http {
 upstream myapp1 {
  server srv1.example.com weight=3;
  server srv2.example.com;  性能比较强，多承担一点，可以设置weight
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

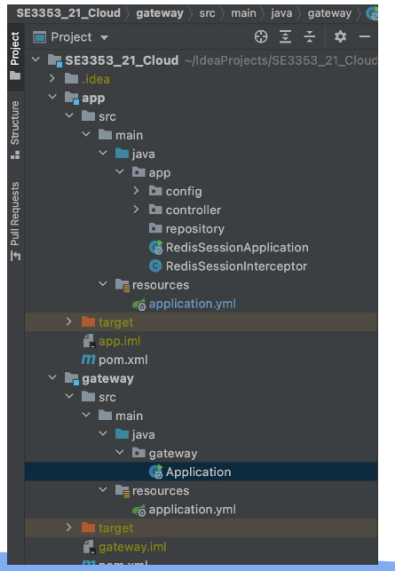
```
http {
  upstream pancm{
   #ip_hash;
   #least_conn;
   server 127.0.0.1:8080; #weight=3;
   server 127.0.0.1:8090;
  }
  server {
    listen      8000;
    server_name  localhost;
    location / {
       root   html;
       proxy_pass http://pancm;
       index  index.html index.htm;
    }
```

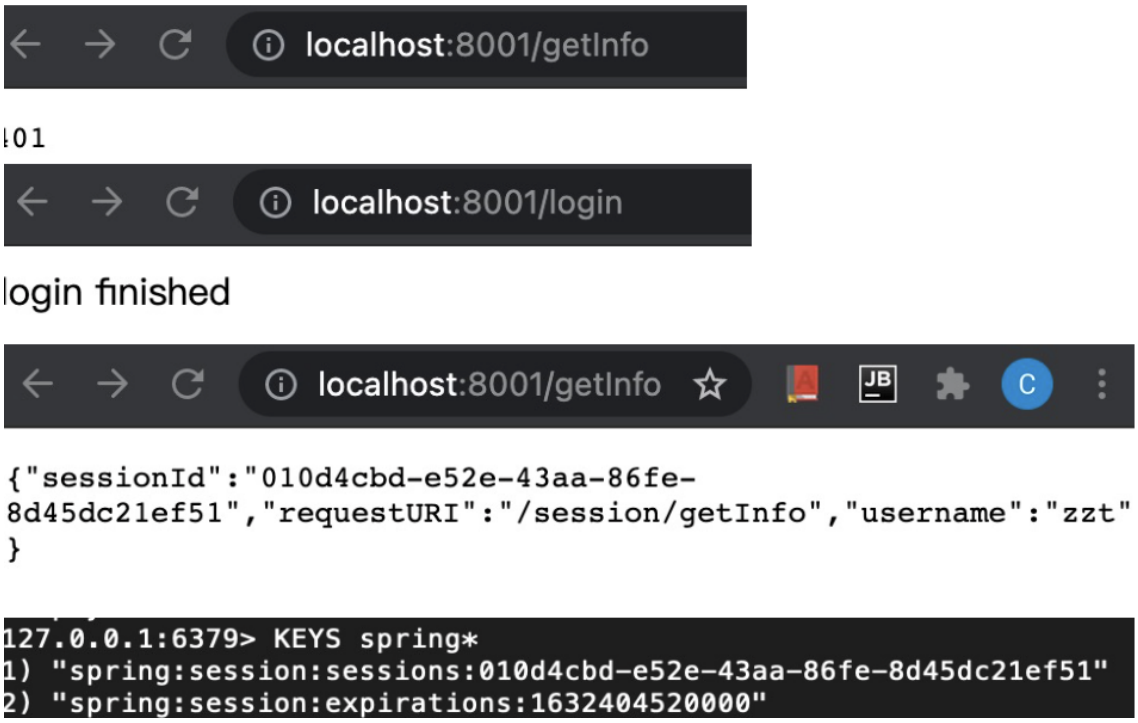
导到这个集群



load balence 是公网IP， sever 上的是内网IP，别人想访问只能访问LB

- Getway Module
  - Run at 8001 port and route request
- App Module
  - Run at 8002 port and process request
- Redis
  - Run at 6379 port and store sessions





要先login 来创建session

```
@Override
public void addInterceptors(InterceptorRegistry registry){
    registry.addInterceptor(redisSessionInterceptor)
            .addPathPatterns("/session/**")
            .excludePathPatterns("/session/login");
    super.addInterceptors(registry);
}
}
```

除了login, 都要被拦截检查一下里面有没有session

之后再访问别的就可以检查到有session了

- 方案比较：

  小集群：ip_hash, 不保证负载均衡，保证session一定在一个机器上

  大集群： 第三方（redis、memcache), 有一台机器来存redis, 但是这样的话redis就成为了单一故障节点，需要多备份一台机器。

- 什么是正向代理和反向代理？
  正向代理就是一个人发送一个请求直接就到达了目标的服务器。

反方代理就是请求统一被nginx接收，nginx反向代理服务器接收到之后，按照一定的规则分发给了后端的业务处理服务器进行处理。

- Nginx应用场景？
  1、http服务器。Nginx是一个http服务可以独立提供http服务。可以做网页静态服务器。
  2、虚拟主机。可以实现在一台服务器虚拟出多个网站，例如个人网站使用的虚拟机。
  3、反向代理，负载均衡。当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群可以使用nginx做反向代理。并且多台服务器可以平均分担负载，不会应为某台服务器负载高宕机而某台服务器闲置的情况。
  4、Nginx 中也可以配置安全管理、比如可以使用Nginx搭建API接口网关,对每个接口服务进行拦截。
- 请列举Nginx和Apache之间的不同点？

最核心的区别在于 Apache 是同步多进程模型，一个连接对应一个进程；Nginx是异步的，多个连接（万级别）可以对应一个进程。

- 动态资源、静态资源分离

动态资源、静态资源分离，是让动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后我们就可以根据静态资源的特点将其做缓存操作，这就是网站静态化处理的核心思路。

在我们的软件开发中，有些请求是需要后台处理的（如：.jsp,.do 等等），有些请求是不需要经过后台处理的（如：css、html、jpg、js 等等文件），这些不需要经过后台处理的文件称为静态文件，否则动态文件。

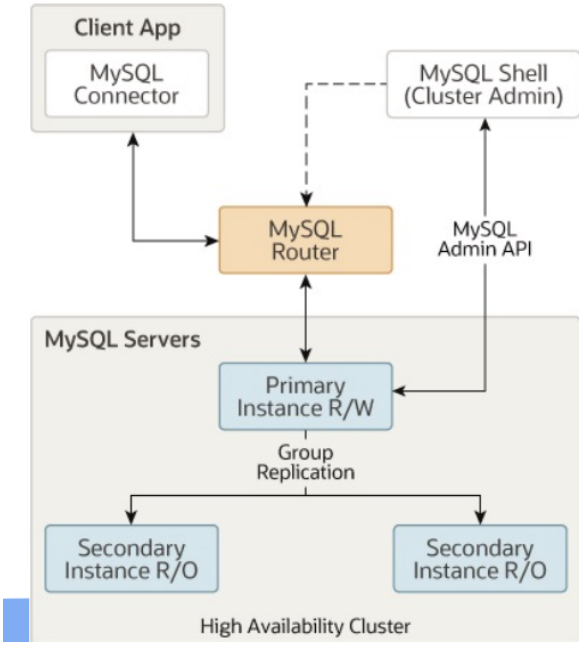
因此我们后台处理忽略静态文件，但是这样后台的请求次数就明显增多了。在我们对资源的响应速度有要求的时候，我们应该使用这种动静分离的策略去解决动、静分离将网站静态资源（HTML，JavaScript，CSS，img等文件）与后台应用分开部署，提高用户访问静态代码的速度，降低对后台应用访问。

这里我们将静态资源放到Nginx中，动态资源转发到Tomcat服务器中去。

MySQL InnoDB cluster

上面讲了tomcat如何建立集群，那么数据库如何集群？

写只能在主节点，读可以负载均衡




```
mysql-js> var cluster = dba.createCluster('testCluster')
Validating instance at icadmin@ic-1:3306...
This instance reports its own address as ic-1
Instance configuration is suitable.
Creating InnoDB cluster 'testCluster' on 'icadmin@ic-1:3306'...
Adding Seed Instance...
Cluster successfully created. Use Cluster.addInstance() to add MySQL instances.
At least 3 instances are needed for the cluster to be able to withstand up to one server failure.
```

种子节点

集群是动态的，可以再添加

```
mysql-js> cluster.addInstance('icadmin@ic-2:3306')
A new instance will be added to the InnoDB cluster. Depending on the amount of data on the cluster this might take from a few seconds to several hours.
Please provide the password for 'icadmin@ic-2:3306': ********
Adding instance to the cluster ...
Validating instance at ic-2:3306...
This instance reports its own address as ic-2 Instance configuration is suitable.
The instance 'icadmin@ic-2:3306' was successfully added to the cluster.
```

primary和secondary的备份

默认刚开始的是primary，后来加入的是secondary

但是如果一个secondary一直是secondary就consistency不太好，最好是每隔一段时间secondary和primary角色互换。

cold 冷备份： secondary一直是secondary， 由primary同步

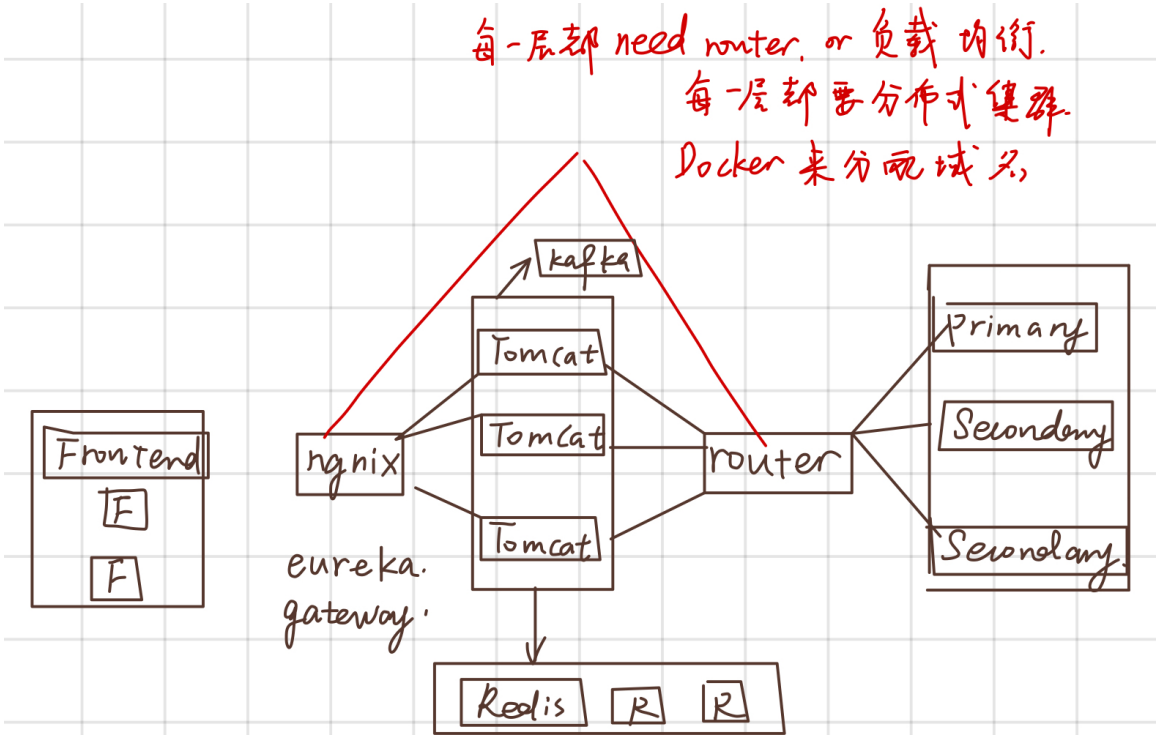
warm 暖备份： 每隔一段时间secondary和primary角色互换。

hot 热备份： 一个事务在primary和secondary都跑，如果返回两个返回值，丢了一个就行。

cold→ warm → hot : 可靠性+++ 耗电+++

改变拓扑结构

可以多primary



总结： tomcat 集群： 维护内存里的状态

database 集群：维护持久化状态