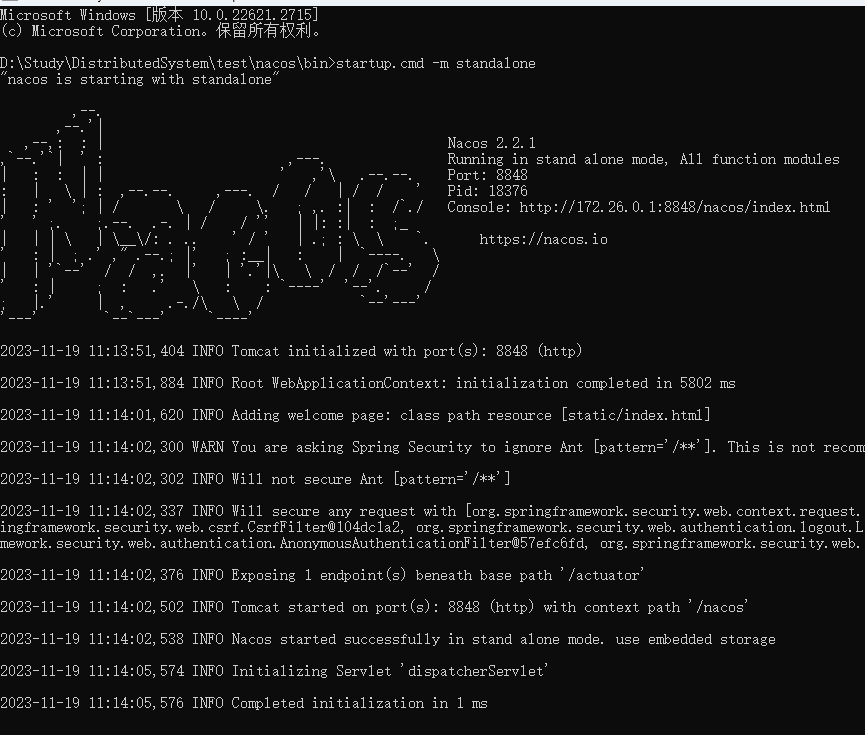
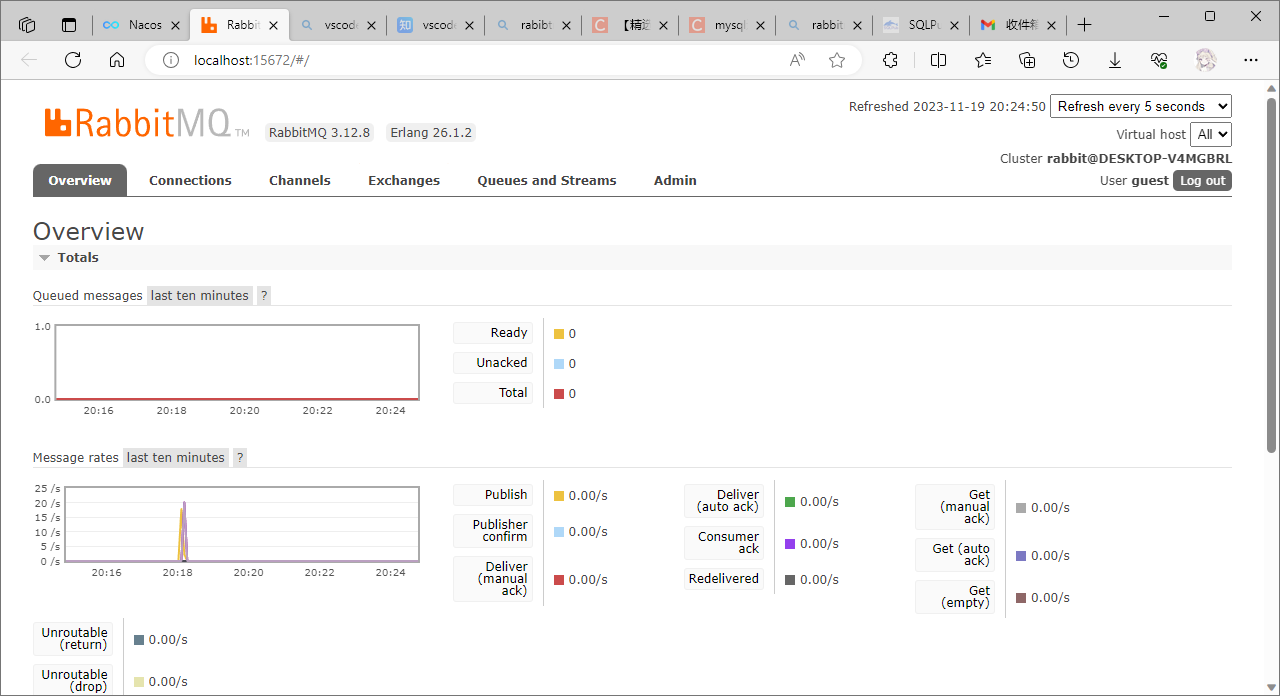
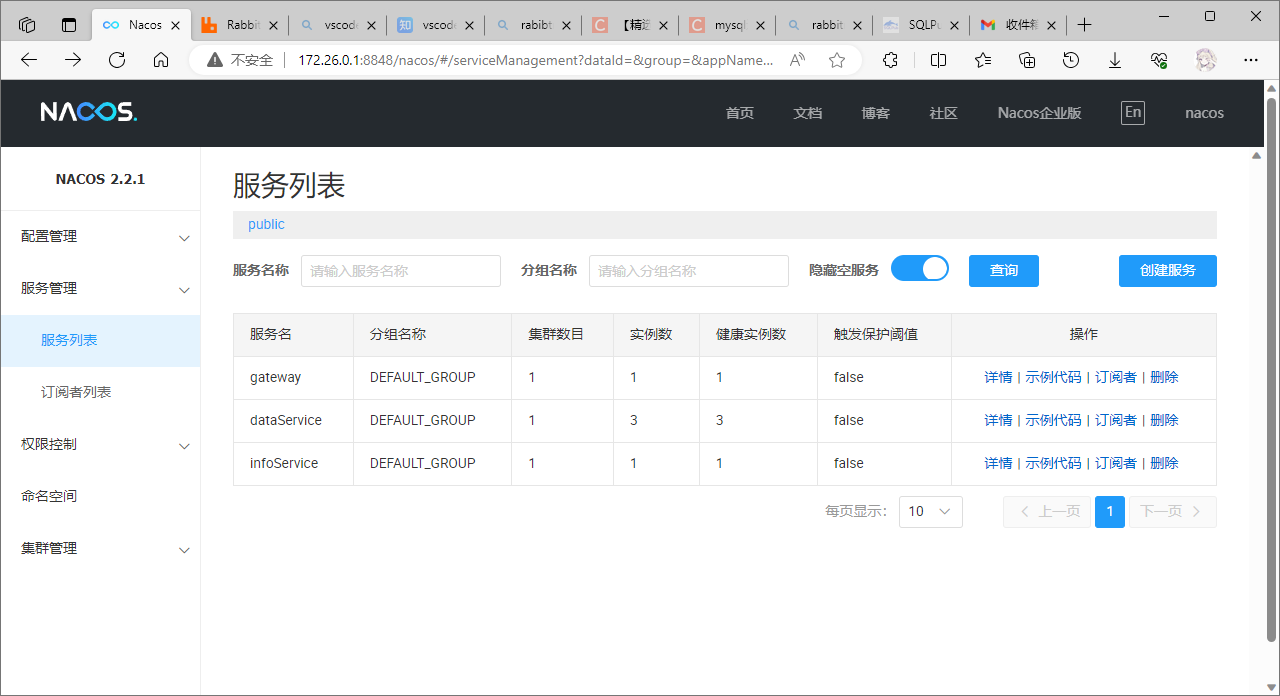
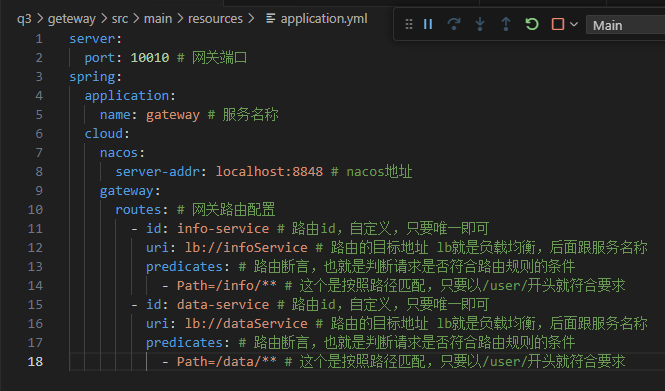
\* 本问题本人采用springboot框架实现。

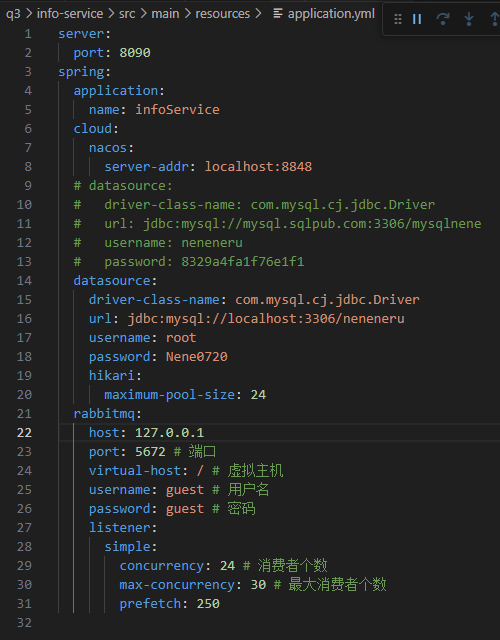
首先我们先开启nacos服务注册中心和在机器上配置RabbitMQ消息队列服务，然后在不同接口启动一个gateway网关服务，一个info微服务，三个data微服务，如下图。





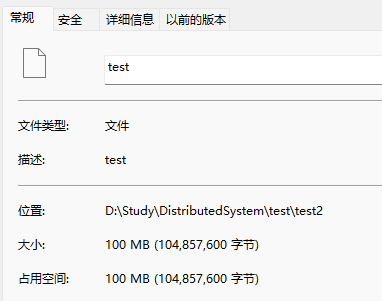
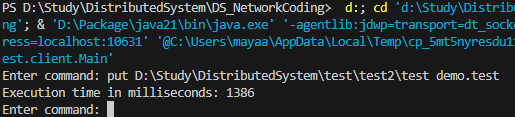
Nacos和RabbitMQ的版本和配置文件和配置说明在源文件夹里有说明，我们观察上面的图可以发现有五个微服务注册到了nacos注册中心上，分别是一个网关，一个infoservice，还有三个dataservice形成的集群，gateway和infoservice和dataservice的配置如下：



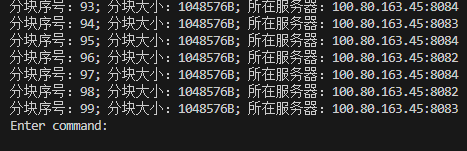
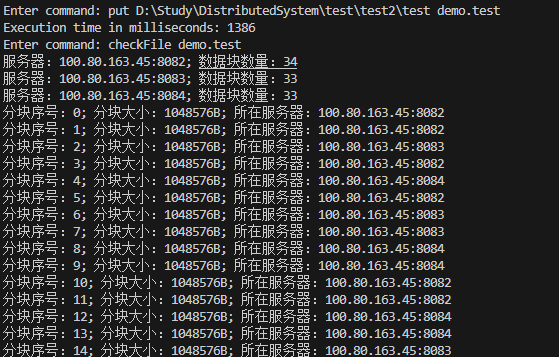


由上面的图可见他们分别有不同的端口（data微服务虽然上面写着一个端口，但是启动的时候是在cmd里配置了三个不同的端口）并且按照各自的需求连接上了mysql数据库/RabbitMQ消息队列/nacos服务中心；网关配置了负载均衡和路由规则；info微服务在与数据库和消息队列通信的时候还用了连接池实现异步多线程/多消费者。

此时我们启动客户端程序，输入指令：put D:\Study\DistributedSystem\test\test2\test demo.test （将D:\Study\DistributedSystem\test\test2\test 这个文件上传到服务器，服务端文件为demo.test（demo目录下的test文件）），用时1386ms（因为是在本地测，我还开了多线程上传，这个时间没什么意义我感觉，下面也会列出时间，但是不会再赘述这个时间没啥意义了），test文件为100MB，分块大小为1MB（后面都用这个test文件测试）



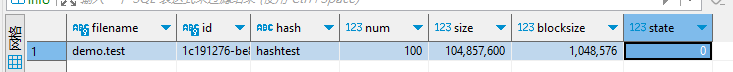
接着我们输入指令：checkFile demo.test （检查在服务器上的demo.test 文件的信息）可以得到如下结果，可以发现，三个data服务器数据块分配均匀，而且分块信息详尽。



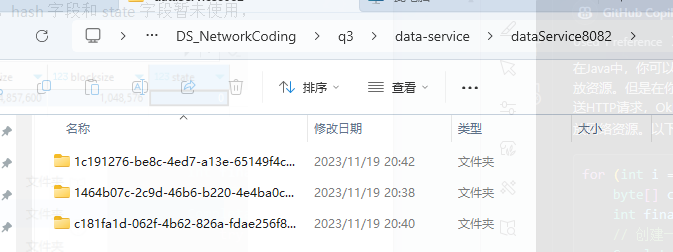
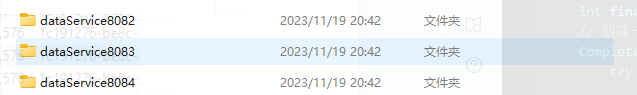
后台检查数据库，也会发现有100个数据分块，符合预期（这部分的逻辑是data微服务在准确无误地接收完一个数据块后异步给info微服务发该数据块的完成消息（通过RabbitMQ），info微服务接收消息写入数据库记录data条目）

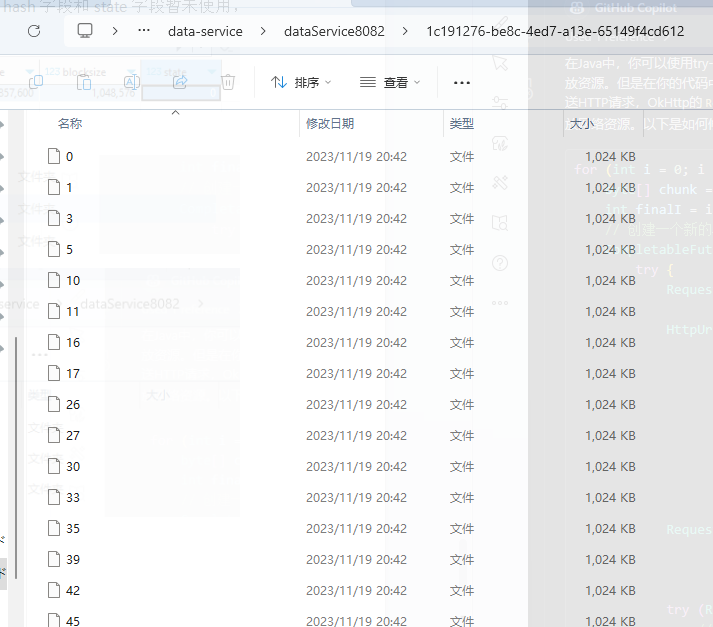


下为在服务器数据库的文件信息，filename就是我们刚才输入的demo.test，然后还有分块数量，文件总大小（单位B），分块大小（单位B）等信息，hash字段和state字段暂未使用，均符合预期

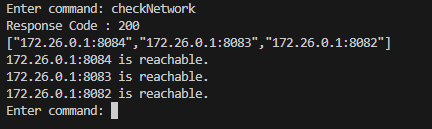


下面是存储在服务器本地的文件系统的格式，其中第二层目录是文件uuid，第三层就是分块。

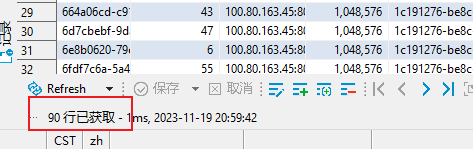




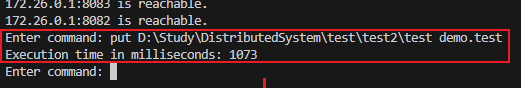
我们还可以使用checkNetwork 指令来查看三个data微服务的可用性：



然后，因为本地测试速度过快，本人没手速瞬间断开一个data微服务的连接模拟出一个服务器掉线，所以直接在数据库中手动删除几项data记录代表有一些data分块因为服务器掉线等原因没有传输上去，见下图，删了10条data分块记录

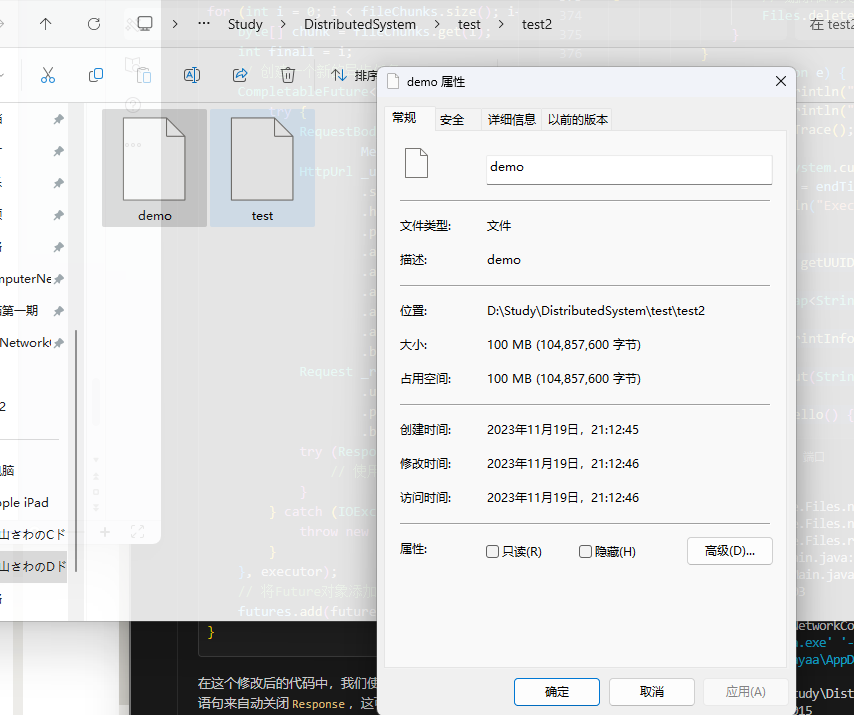


继续采用put指令，发现时间基本没有变化，他还是把100个分块重新上传并覆盖了之前已经上传的文件，证明put没有续传功能。

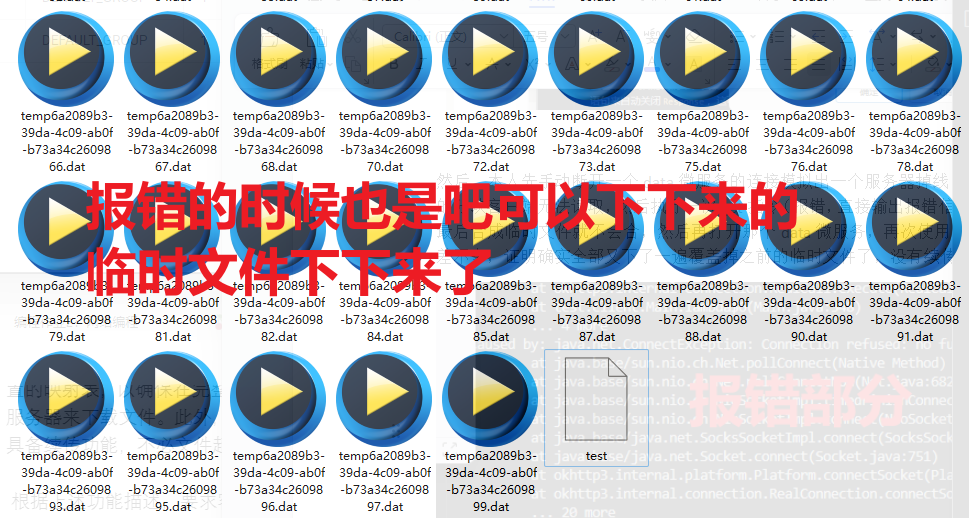
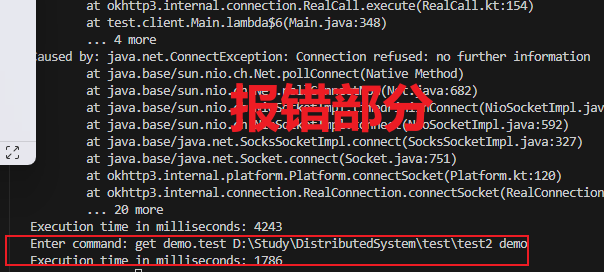


接下来我们使用指令：get demo.test D:\Study\DistributedSystem\test\test2 demo （把服务端上的demo.test文件下载到D:\Study\DistributedSystem\test\test2目录的demo文件里），这部分的处理逻辑是多线程分块下载，然后将临时的分块下载文件合在一起，用了1834ms。下面还有下载下来的demo文件的属性，刚好为100MB。



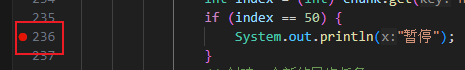


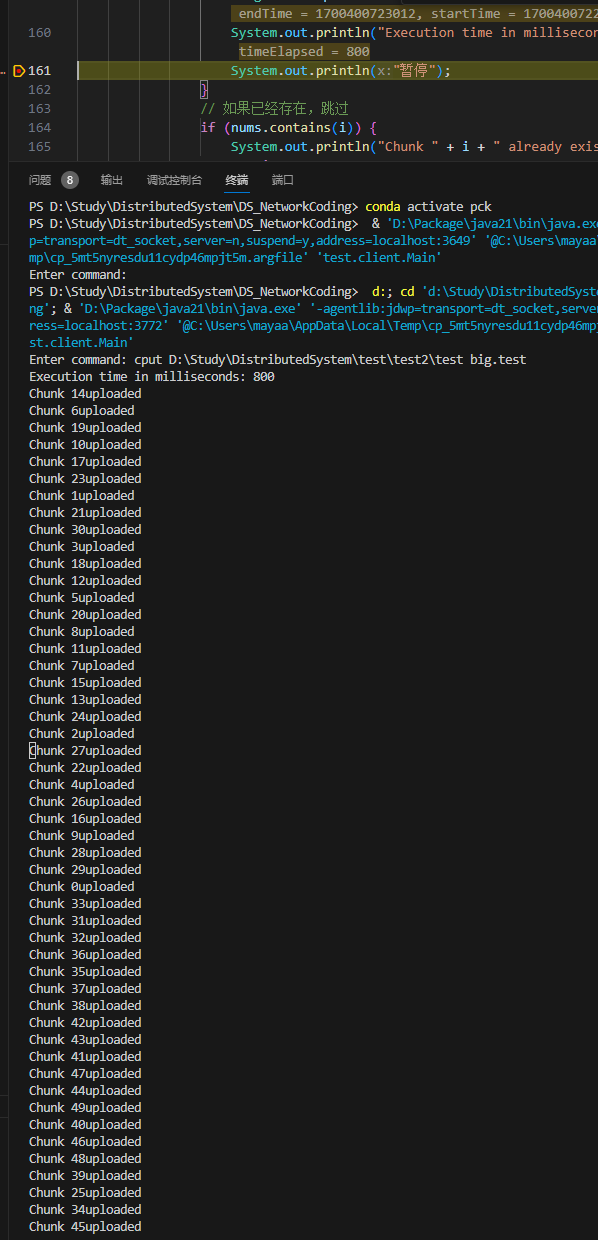
然后，本人先手动断开一个data微服务的连接模拟出一个服务器掉线，这样那个服务器上的分块客户端无法读取，然后执行一次get指令（报错，直接输出报错信息了，有点不好看），最后合成临时文件就不会合，然后再打开那个data微服务，再次使用get命令，结果用时差不多，证明确实全部又下了一遍覆盖掉之前的临时文件了，没有续传功能。（下图）



接下来展示具有续传功能的上传和下载，因为基本逻辑和没有续传功能的相同，下面不赘述基本逻辑，只会说实现续传的逻辑。

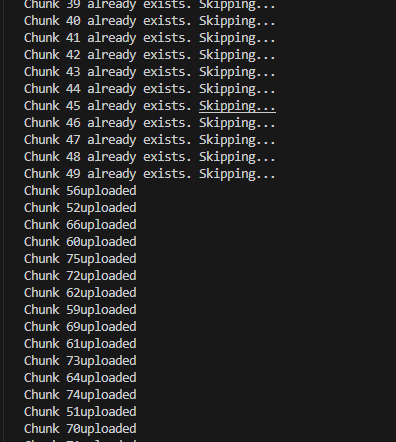
我们在使用cput指令的时候在处理第50个分块的时候打上断点，在断点停下之后关闭客户端，模拟客户端掉线。





可见我们已经完成可前50个分块的上传，用时：800ms

然后我们再重启客户端，继续执行和上面一样的cget操作，从下图可以看出，50之前的都检测到已经存在，直接跳过，直接上传没上传的分块。这里的逻辑就是我不是有一个记录data块信息的表吗，每一次data微服务接受一个块成功都会异步发消息给info微服务，info微服务会往数据库记录的，所以只要检查数据库里关于这个文件已上传成功哪些块号就可以确定还有哪些分块没有上传了。

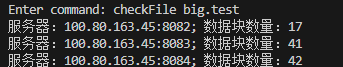




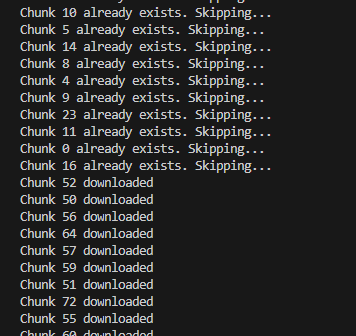
最后这次用时为1085ms，和上50个数据的时间（800ms）差不多，证明这次也基本是上传了50个数据。

总用时为1885ms（因为续传需要处理更多额外的信息，在本地100mb文件的情况下显然对比起来时间没什么优势，而且服务器也掉了一个，确实会慢点）

然后我们在使用cput指令的时候在处理第50个分块的时候打上断点，在断点停下之后关闭一个data微服务，模拟服务端掉线。我们进行一次cput，结果如下图所示，发现第一个服务器的分块数明显比较少，但总数还是100，这是因为springboot框架下的网关gateway服务发现有微服务掉线请求失败会自动把请求转发给其他可用的微服务，而不会丢失数据，具有高可用性和一致性。



接下来我们来测试cget，同样在下载了50个分块的时候打上断点，接着结束客户端模拟客户端掉线。然后重启客户端继续下载。如下图，监测到客户端本地已经有前50个分块的临时文件了，就跳过，直接下后50个，总用时为2067ms。这里的逻辑就是每一次下载都是有把临时文件下载下来然后最后合在一起的，所以只要检查本地有哪几个临时文件就可以确定哪些分块已经下载了，哪些分块还没有下载，然后去请求没有下载的分块来下载就好啦。





接下来是在cget前先掉线一个data微服务，此时cget不能获得那个服务器上的数据，所以获取的数据不完整，然后打开那个data微服务，再进行一次相同的操作，就可以把之前没下的下载下来了。

