拉勾教育
—互联网人实战大学—

《Java性能优化与面试21讲》

李国

— 拉勾教育出品 —

20 | SpringBoot 服务性能优化

前言

比如你的服务用到了缓存，就需要把缓存命中率这些数据进行收集

用到了数据库连接池，就需要把连接池的参数给暴露出来

SpringBoot 如何开启监控?

拉勾教育
互联网人实战大学

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-core</artifactId>
</dependency>
```

SpringBoot 如何开启监控?


```
management.endpoint.metrics.enabled=true
management.endpoints.web.exposure.include=*
management.endpoint.prometheus.enabled=true
management.metrics.export.prometheus.enabled=true
```

SpringBoot 如何开启监控?


```
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual mach
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'profiled nmethods'",} 9961472.0
jvm_memory_committed_bytes{area="heap",id="G1 Survivor Space",} 4194304.0
jvm_memory_committed_bytes{area="heap",id="G1 Old Gen",} 2.8311552E7
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",} 4.233216E7
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'non-nmethods'",} 2555904.0
jvm_memory_committed_bytes{area="heap",id="G1 Eden Space",} 2.7262976E7
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space",} 5898240.0
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'non-profiled nmethods'",} 2555904.0
# HELP http_server_requests_seconds
# TYPE http_server_requests_seconds summary
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actua
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuato
# HELP http_server_requests_seconds_max
# TYPE http_server_requests_seconds_max gauge
http_server_requests_seconds_max{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuato
# HELP jvm_buffer_count_buffers An estimate of the number of buffers in the pool
# TYPE jvm_buffer_count_buffers gauge
jvm_buffer_count_buffers{id="mapped - 'non-volatile memory'",} 0.0
jvm_buffer_count_buffers{id="mapped",} 0.0
jvm_buffer_count_buffers{id="direct",} 5.0
# HELP process_files_max_files The maximum file descriptor count
# TYPE process_files_max_files gauge
```

SpringBoot 如何开启监控?

拉勾教育
互联网人实战大学

```
@Autowired
MeterRegistry registry;

@GetMapping("/test")
@ResponseBody
public String test() {
    registry.counter("test",
            "from", "127.0.0.1",
            "method", "test"
    ).increment();

    return "ok";
}
```

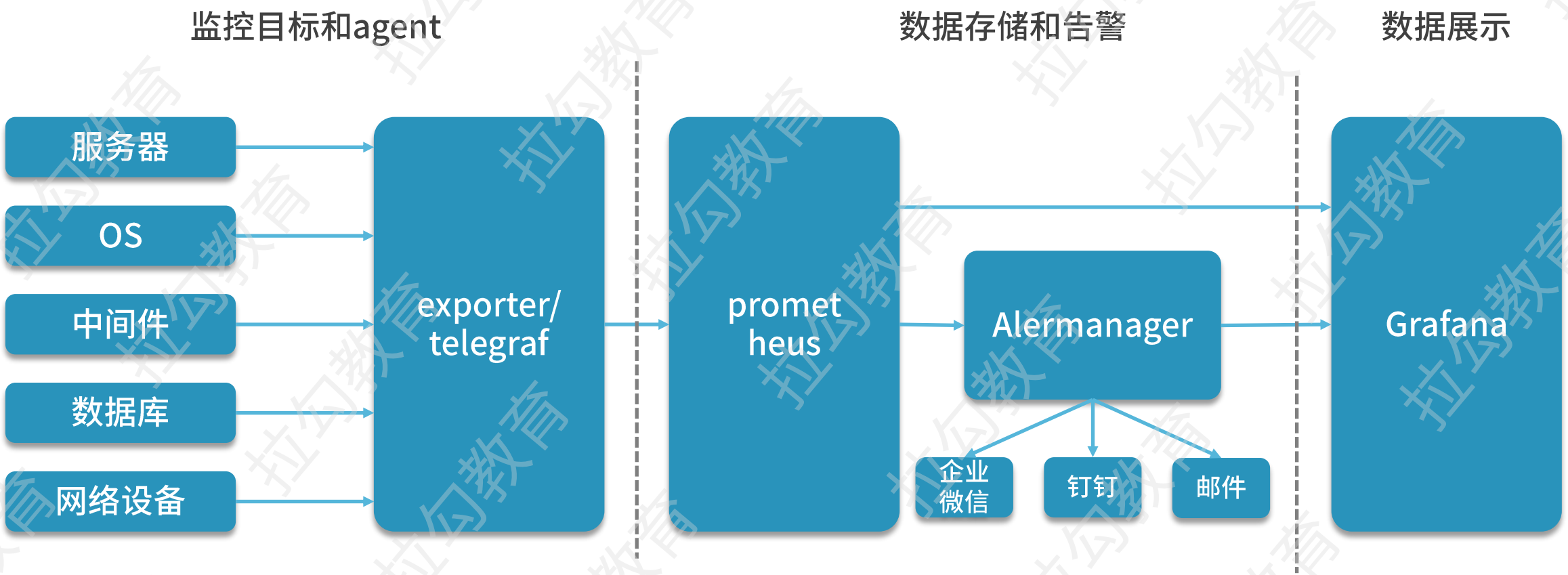
SpringBoot 如何开启监控?

拉勾教育
互联网人实战大学

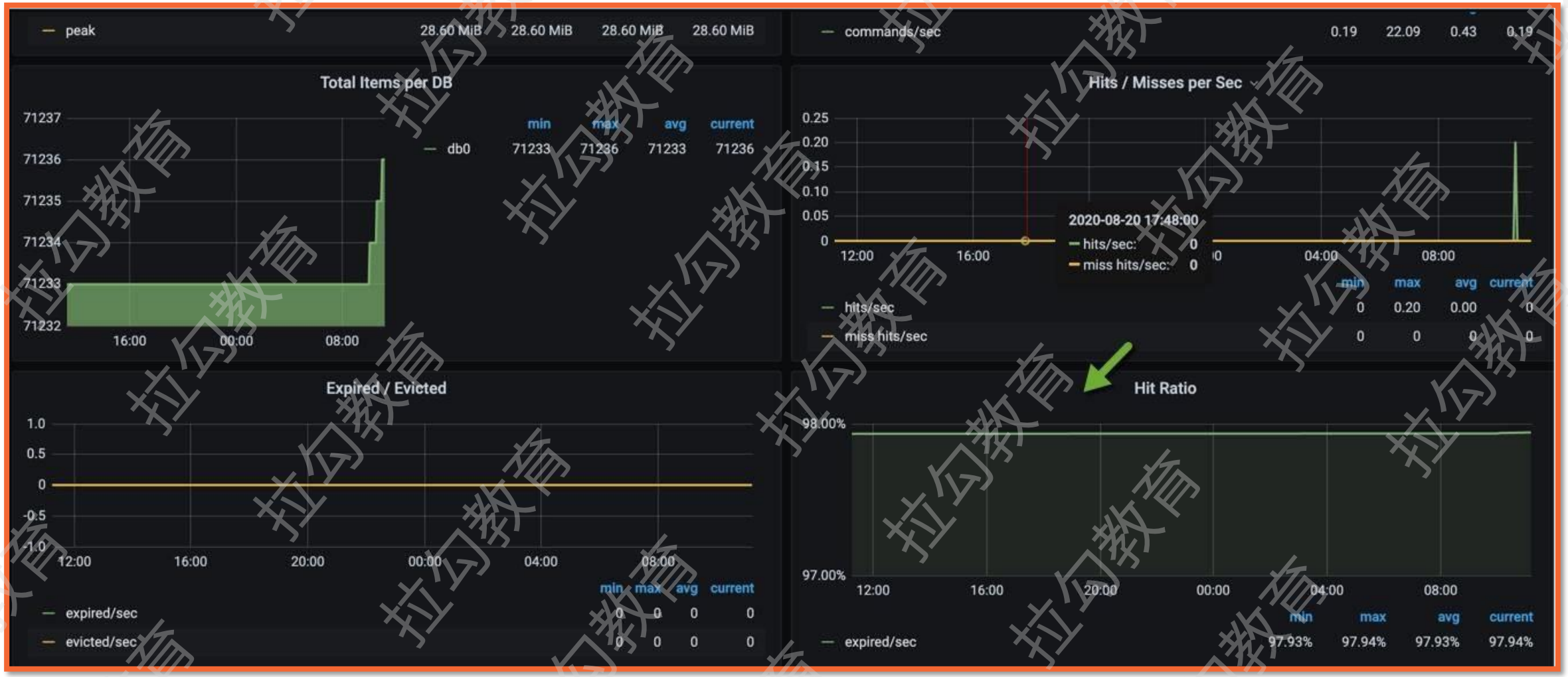
```
test_total{from="127.0.0.1",method="test",} 5.0
```

SpringBoot 如何开启监控?

拉勾教育
互联网人实战大学



SpringBoot 如何开启监控?



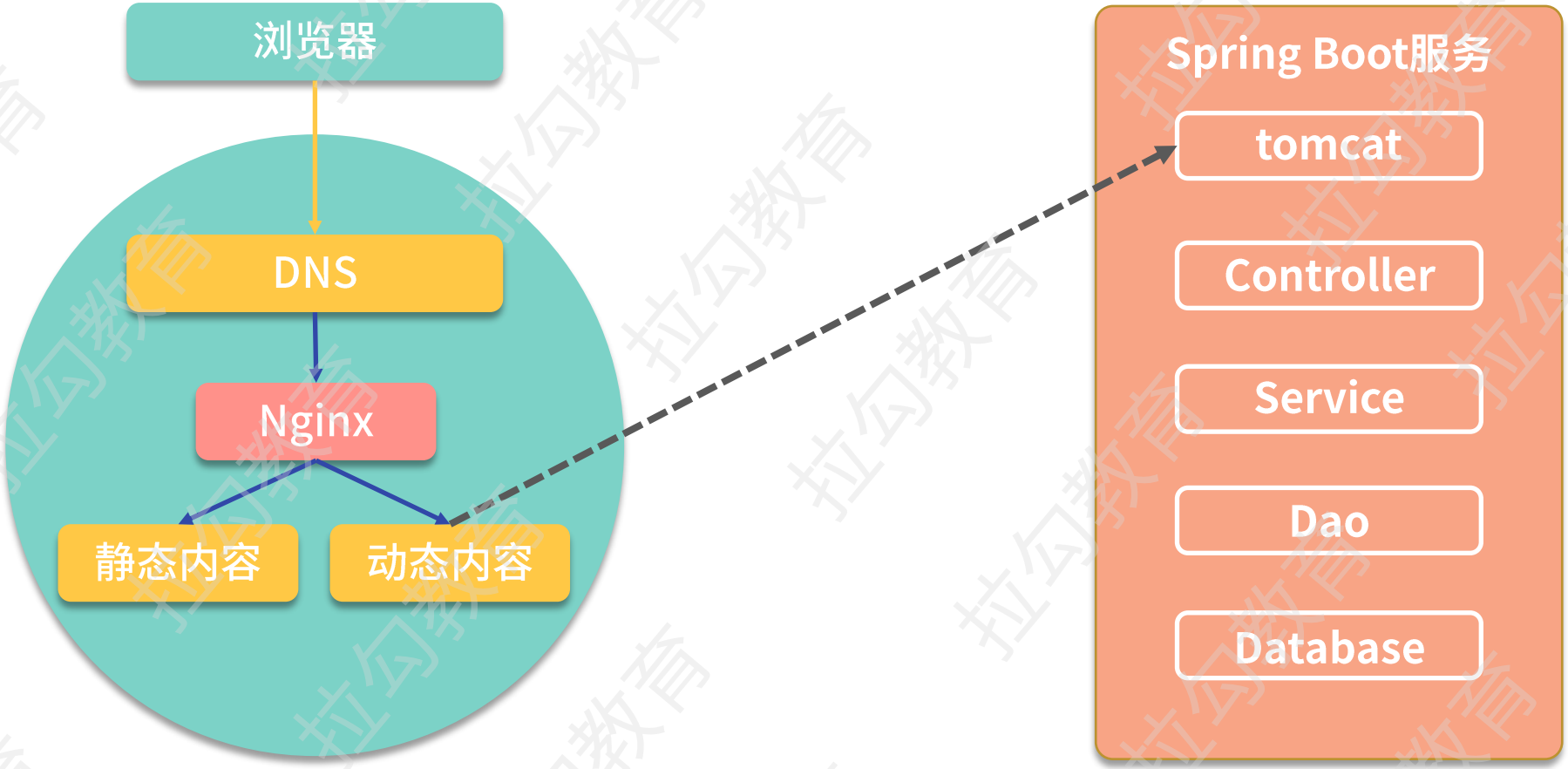
Java 生成火焰图


```
java -agentpath:/root/build/libasyncProfiler.so=start,svg,file=profile.svg -jar
spring-petclinic-2.3.1.BUILD-SNAPSHOT.jar
```

Java 生成火焰图


org/springframework/boot/SpringApplication.run
org/springframework/samples/petclinic/PetClinicApplic
sun/reflect/NativeMethodAccessorImpl.invoke0
sun/reflect/NativeMethodAccessorImpl.invoke
sun/reflect/DelegatingMethodAccessorImpl.invoke
java/lang/reflect/Method.invoke
org/springframework/boot/loader/Launcher.launch
org/springframework/boot/loader/JarLauncher.main
all

优化思路



HTTP 优化

1. 使用 CDN 加速文件获取

比较大的文件，尽量使用 CDN（Content Delivery Network）分发

甚至是一些常用的前端脚本、样式、图片等，都可以放到 CDN 上

CDN 通常能够加快这些文件的获取，网页加载也更加迅速

2. 合理设置 Cache-Control 值

浏览器会判断 HTTP 头 Cache-Control 的内容，用来决定是否使用浏览器缓存

这在管理一些静态文件的时候，非常有用，相同作用的头信息还有 Expires

Cache-Control 表示多久之后过期；Expires 则表示什么时候过期

HTTP 优化


```
location ~* ^.+\.(ico|gif|jpg|jpeg|png)$ {
        # 缓存1年
        add_header Cache-Control: no-cache, max-age=31536000;
}
```

HTTP 优化

3. 减少单页面请求域名的数量

减少每个页面请求的域名数量，尽量保证在 4 个之内

因为浏览器每次访问后端的资源，都需要先查询一次 DNS

然后找到 DNS 对应的 IP 地址，再进行真正的调用

4. 开启 gzip

开启 gzip，可以先把内容压缩后，浏览器再进行解压

由于减少了传输的大小，会减少带宽的使用，提高传输效率

HTTP 优化

4. 开启 gzip

开启 gzip，可以先把内容压缩后，浏览器再进行解压

由于减少了传输的大小，会减少带宽的使用，提高传输效率

```
gzip on;
gzip_min_length 1k;
gzip_buffers 4 16k;
gzip_comp_level 6;
gzip_http_version 1.1;
gzip_types text/plain application/javascript text/css;
```

HTTP 优化

5. 对资源进行压缩

对 JavaScript 和 CSS，甚至是 HTML 进行压缩

道理类似，现在流行的前后端分离模式，一般都是对这些资源进行压缩的

6. 使用 keepalive

由于连接的创建和关闭，都需要耗费资源

用户访问我们的服务后，后续也会有更多的互动，所以保持长连接可以显著减少网络交互，提高性能

HTTP 优化

6. 使用 keepalive

由于连接的创建和关闭，都需要耗费资源

用户访问我们的服务后，后续也会有更多的互动，所以保持长连接可以显著减少网络交互，提高性能

```
http {
    keepalive_timeout  120s 120s;
    keepalive_requests 10000;
}
```

HTTP 优化

6. 使用 keepalive

由于连接的创建和关闭，都需要耗费资源

用户访问我们的服务后，后续也会有更多的互动，所以保持长连接可以显著减少网络交互，提高性能

```
location ~ /{
    proxy_pass http://backend;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
```

自定义 Web 容器


```
@SpringBootApplication(proxyBeanMethods = false)
public class App implements WebServerFactoryCustomizer<ConfigurableServletWebServerFactory> {
    public static void main(String[] args) {
        SpringApplication.run(PetClinicApplication.class, args);
    }
    @Override
    public void customize(ConfigurableServletWebServerFactory factory) {
        TomcatServletWebServerFactory f = (TomcatServletWebServerFactory) factory;
        f.setProtocol("org.apache.coyote.http11.Http11Nio2Protocol");

        f.addConnectorCustomizers(c -> {
            Http11NioProtocol protocol = (Http11NioProtocol) c.getProtocolHandler();
            protocol.setMaxConnections(200);
            protocol.setMaxThreads(200);
            protocol.setSelectorTimeout(3000);
            protocol.setSessionTimeout(3000);
            protocol.setConnectionTimeout(3000);
        });
    }
}
```

自定义 Web 容器


```
[root@localhost wrk2-master]# ./wrk -t2 -c100 -d30s -R2000
http://172.16.1.57:8080/owners?lastName=
Running 30s test @ http://172.16.1.57:8080/owners?lastName=
  2 threads and 100 connections
  Thread calibration: mean lat.: 4588.131ms, rate sampling interval: 16277ms
  Thread calibration: mean lat.: 4647.927ms, rate sampling interval: 16285ms
  Thread Stats   Avg     Stdev     Max   +/- Stdev
    Latency    16.49s    4.98s   27.34s    63.90%
    Req/Sec   106.50     1.50   108.00    100.00%
  6471 requests in 30.03s, 39.31MB read
  Socket errors: connect 0, read 0, write 0, timeout 60
Requests/sec:    215.51
Transfer/sec:      1.31MB
```

自定义 Web 容器


```
[root@localhost wrk2-master]# ./wrk -t2 -c100 -d30s -R2000
http://172.16.1.57:8080/owners?lastName=
Running 30s test @ http://172.16.1.57:8080/owners?lastName=
  2 threads and 100 connections
  Thread calibration: mean lat.: 4358.805ms, rate sampling interval: 15835ms
  Thread calibration: mean lat.: 4622.087ms, rate sampling interval: 16293ms
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    17.47s    4.98s   26.90s    57.69%
    Req/Sec   125.50     2.50   128.00    100.00%
  7469 requests in 30.04s, 45.38MB read
  Socket errors: connect 0, read 0, write 0, timeout 4
Requests/sec:    248.64
Transfer/sec:      1.51MB
```

自定义 Web 容器


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
     <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
     </exclusion>
    </exclusions>
   </dependency>
   <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-undertow</artifactId>
   </dependency>
```

自定义 Web 容器

拉勾教育
互联网人实战大学

```
-XX:+UseG1GC -Xmx2048m -Xms2048m -XX:+AlwaysPreTouch
```

Skywalking

对于一个 web 服务来说，最缓慢的地方就在于**数据库操作**

Skywalking

将 agent 的压缩包，解压到相应的目录

```
tar xvf skywalking-agent.tar.gz  -C /opt/
```

Skywalking

将 agent 的压缩包，解压到相应的目录

```
tar xvf skywalking-agent.tar.gz  -C /opt/
```

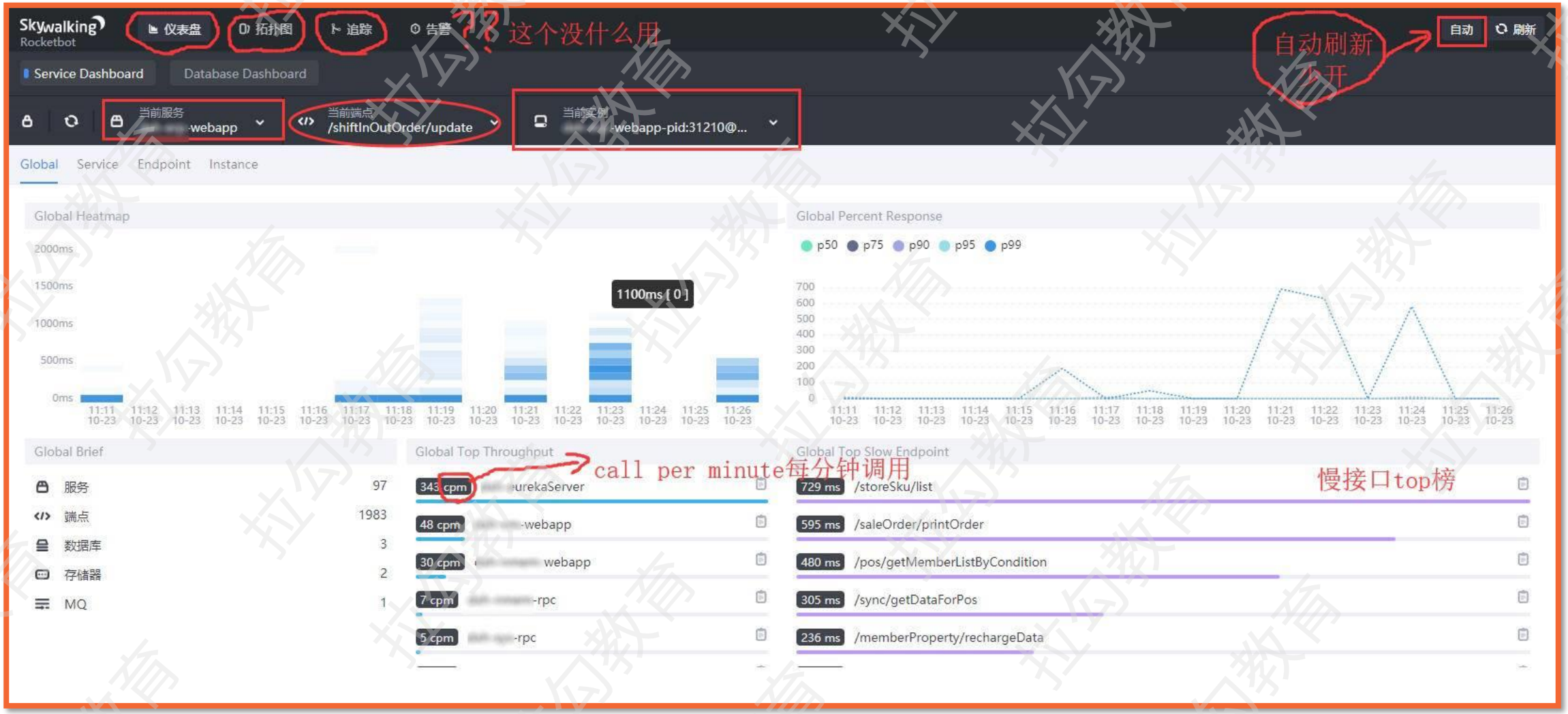
在业务启动参数中加入 agent 的包

```
java -jar /opt/test-service/spring-boot-demo.jar --spring.profiles.active=dev
```

Skywalking


```
java -javaagent:/opt/skywalking-agent/skywalking-agent.jar
-Dskywalking.agent.service_name=the-demo-name  -jar /opt/test-
service/spring-boot-demo.ja  --spring.profiles.active=dev
```

Skywalking


Skywalking
Rocketbot
仪表盘
拓扑图
追踪
告警
这个没什么用
自动刷新
开
自动
刷新
Service Dashboard
Database Dashboard
当前服务
webapp
当前端点
/shiftInOutOrder/update
当前实例
webapp-pid:31210@...
Global
Service
Endpoint
Instance
Global Heatmap
2000ms
1500ms
1000ms
500ms
0ms
1100ms [0]
Global Percent Response
p50
p75
p90
p95
p99
700
600
500
400
300
200
100
0
Global Brief
服务 97
端点 1983
数据库 3
存储器 2
MQ 1
Global Top Throughput
call per minute每分钟调用
343 cpm
48 cpm
webapp
30 cpm
webapp
7 cpm
-rpc
5 cpm
-rpc
Global Top Slow Endpoint
慢接口top榜
729 ms /storeSku/list
595 ms /saleOrder/printOrder
480 ms /pos/getMemberListByCondition
305 ms /sync/getDataForPos
236 ms /memberProperty/rechargeData

各个层次的优化方向

1. Controller 层

controller 层用于接收前端的查询参数，然后构造查询结果

现在很多项目都采用前后端分离的架构，所以 controller 层的方法

一般会使用 @ResponseBody 注解，把查询的结果，解析成 JSON 数据返回（兼顾效率和可读性）

对于一般的服务，保持结果集的精简，是非常有必要的

各个层次的优化方向

2. Service 层

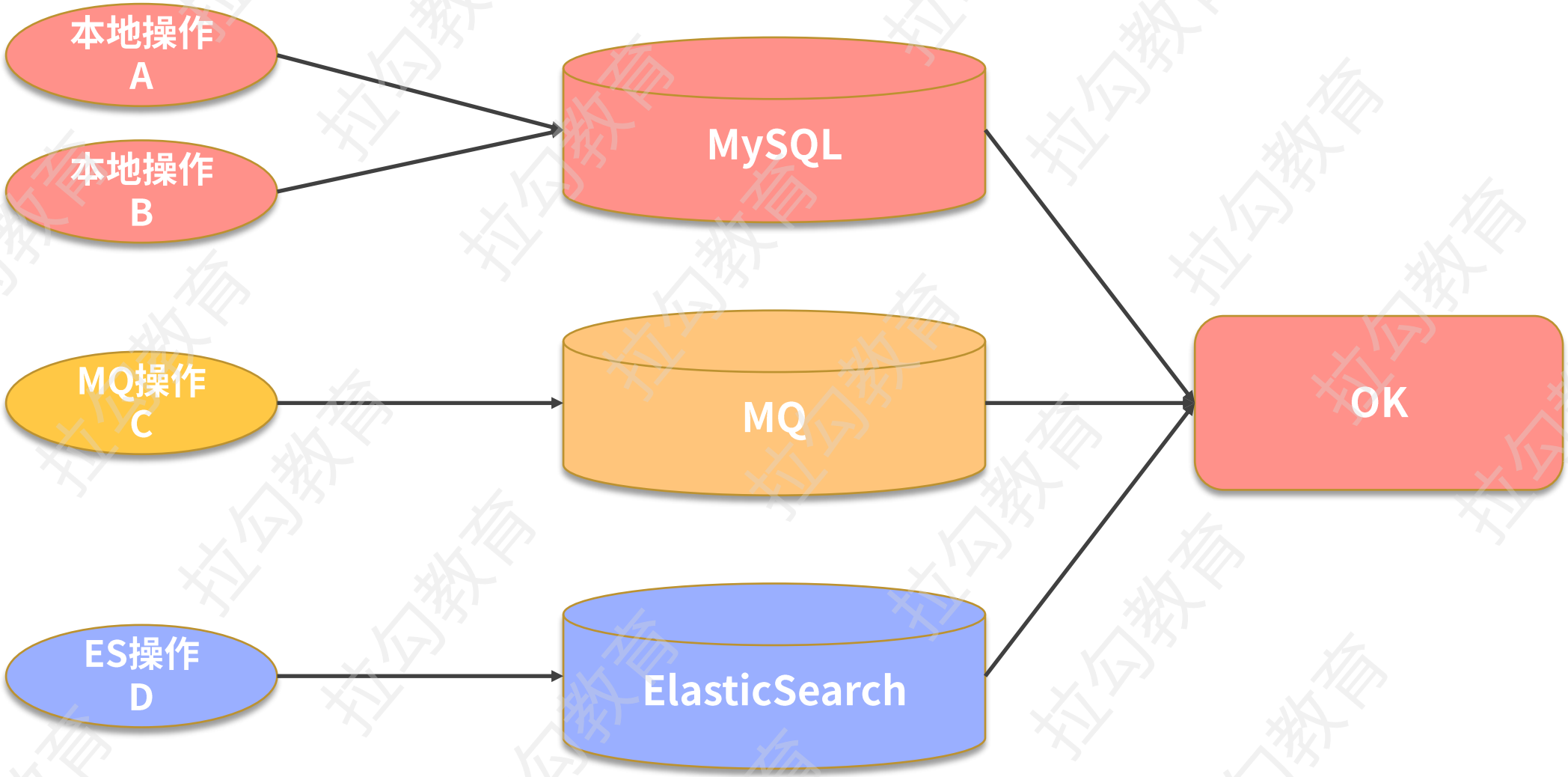
service 层用于处理具体的业务，大部分功能需求都是在这里完成的

service 层一般是使用单例模式（prototype），很少会保存状态，而且可以被 controller 复用

service 层的代码组织，对代码的可读性、性能影响都比较大

service 层会频繁使用更底层的资源，通过组合的方式获取我们所需要的数据

各个层次的优化方向



各个层次的优化方向

各个层次的优化方向

原子性（Atomicity）

一致性（Consistency）

隔离性（Isolation）

持久性（Durability）

各个层次的优化方向

BASE 为 Basically Available、Soft-state、Eventually consistent 三者的缩写

各个层次的优化方向

BASE 为 Basically Available、Soft-state、Eventually consistent 三者的缩写

各个层次的优化方向

3. Dao层

经过合理的数据缓存，我们都会尽量避免请求穿透到 Dao 层

除非你对 ORM 本身提供的缓存特性特别的熟悉

否则，都推荐你使用更加通用的方式去缓存数据

小结

- 监控系统 Prometheus，可以看到一些具体的指标大小
- 火焰图，可以看到具体的代码热点
- Skywalking，可以分析分布式环境中的调用链

Next：第21讲《性能优化的过程方法与求职面经总结》

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容

L / A / G / O / U