

假不理 Lv4

2018年11月30日 阅读 649

[关注](#)

Hystrix都停更了，我为什么还要学？

最近小主看到很多公众号都在发布Hystrix停更的文章，spring cloud体系的使用者和拥护者一片哀嚎，实际上，spring作为Java最大的家族，根本不需要担心其中一两个零件的废弃，Hystrix的停更，只会催生更多或者更好的零件来替代它，因此，我们需要做的是：**知道Hystrix是干嘛，怎么用的，这样要找替代者就易于反掌了。**

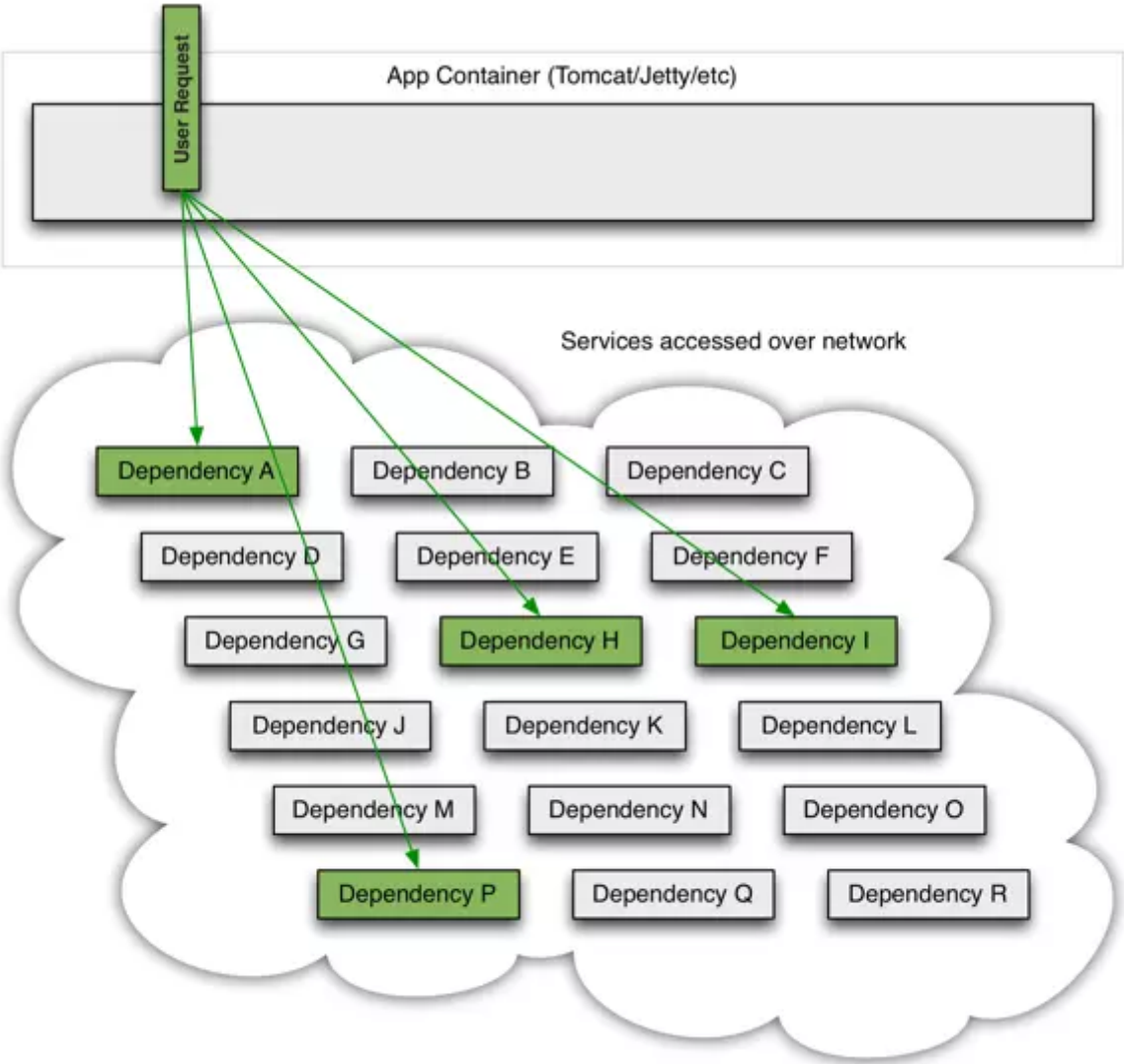
文章提纲：

1. 为什么需要Hystrix？
2. Hystrix如何解决依赖隔离
3. 如何使用Hystrix

1. 为什么需要Hystrix？

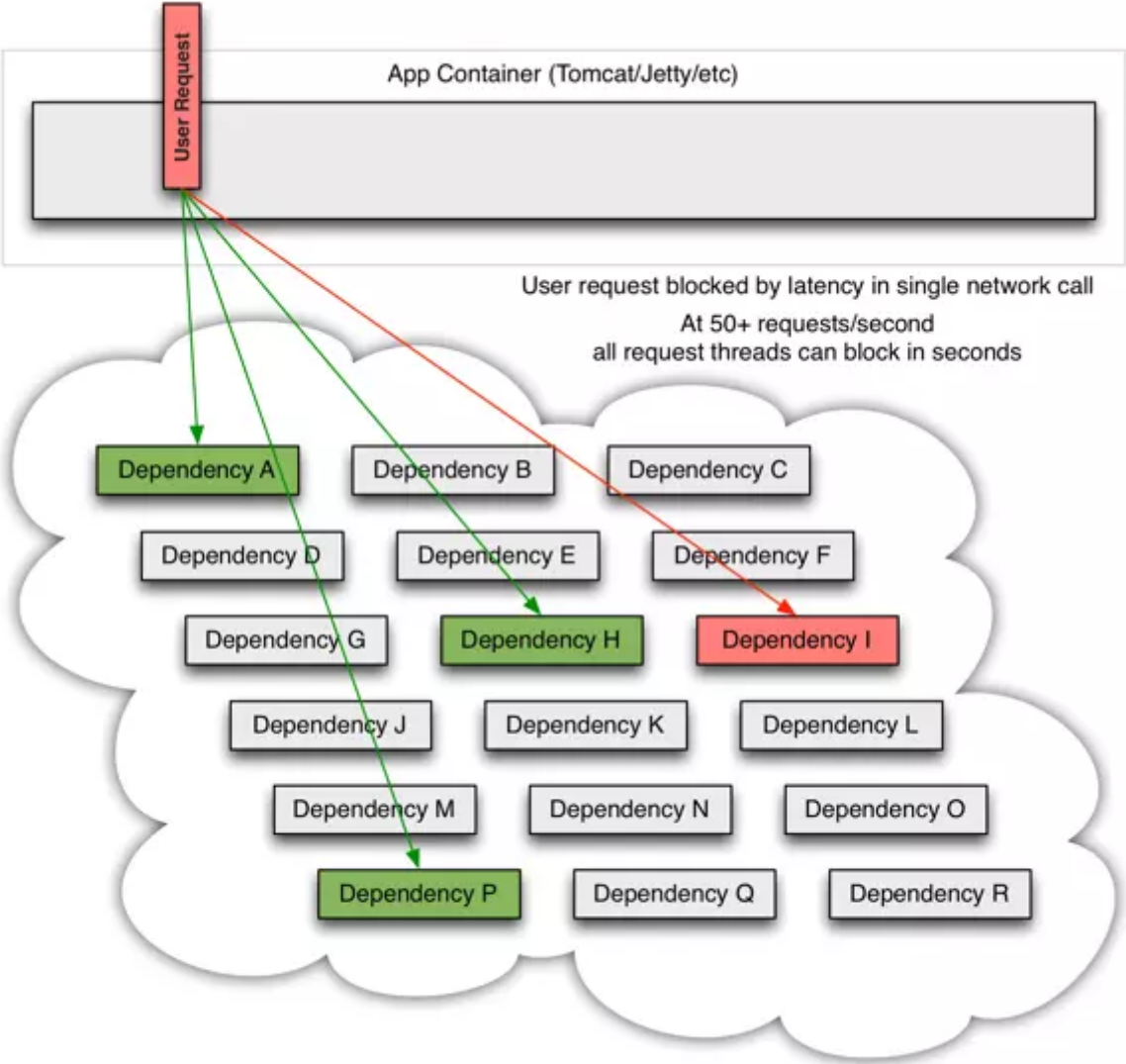
在大中型分布式系统中，通常系统很多依赖(HTTP,hession,Netty,Dubbo等)，如下图：

[首页](#) ▼

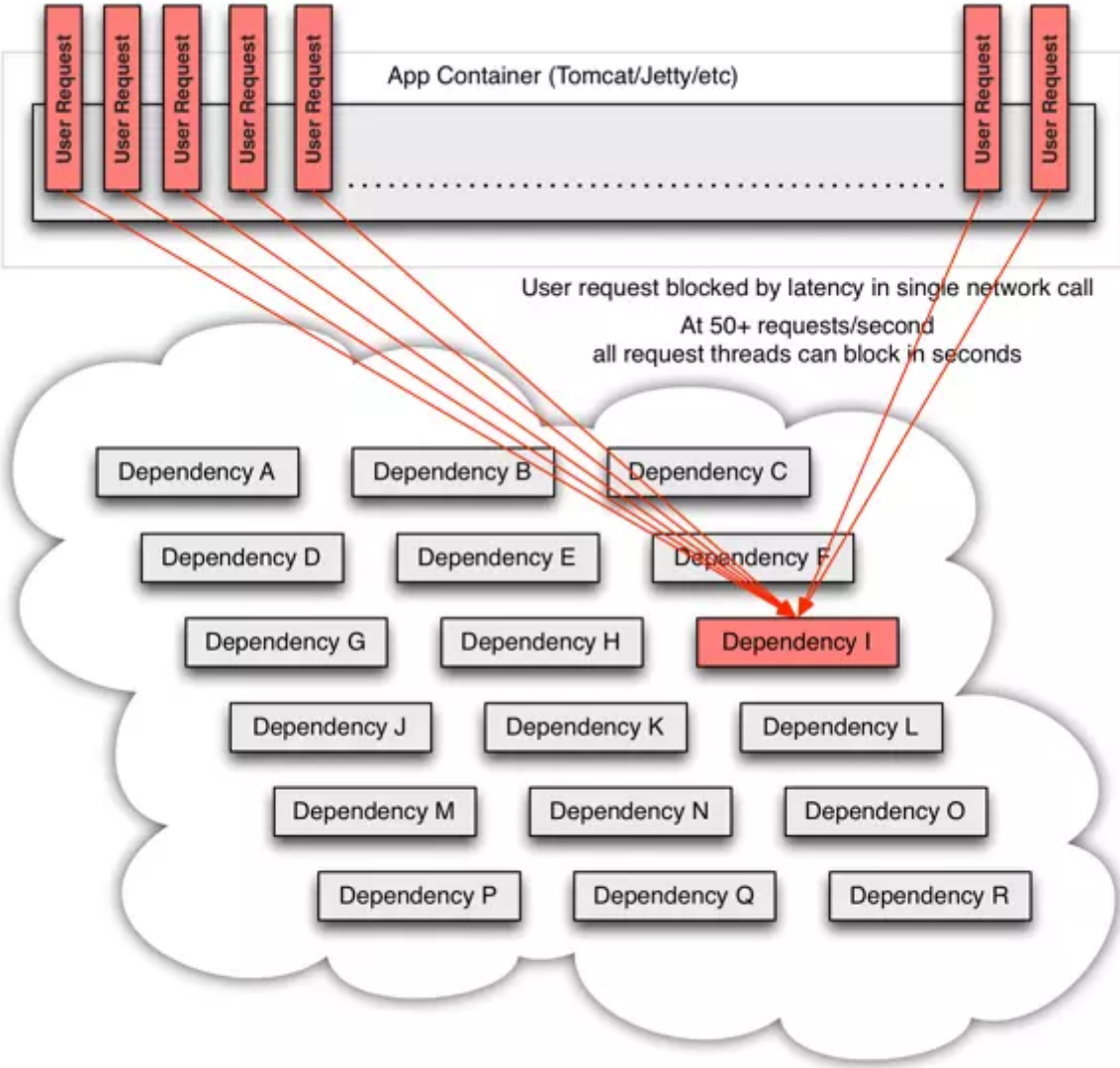


在高并发访问下,这些依赖的稳定性与否对系统的影响非常大,但是依赖有很多不可控问题:如网络连接缓慢,资源繁忙,暂时不可用,服务脱机等.

如下图: QPS为50的依赖 I 出现不可用,但是其他依赖仍然可用.



当依赖I 阻塞时,大多数服务器的线程池就出现阻塞(BLOCK),影响整个线上服务的稳定性. 如下图:



在复杂的分布式架构的应用程序有很多的依赖，都会不可避免地在某些时候失败。高并发的依赖失败时如果没有隔离措施，当前应用服务就有被拖垮的风险。

- 1 例如:一个依赖30个SOA服务的系统,每个服务99.99%可用。
- 2 99.99%的30次方 ≈ 99.7%
- 3 0.3% 意味着一亿次请求 会有 3,000,00次失败
- 4 换算成时间大约每月有2个小时服务不稳定。
- 5 随着服务依赖数量的变多，服务不稳定的概率会成指数性提高。

解决问题方案:对依赖做隔离,Hystrix就是处理依赖隔离的框架,同时也是可以帮我们做依赖服务的治理和监控.

Netflix 公司开发并成功使用Hystrix,使用规模如下:

- 1 he Netflix API processes 10+ billion HystrixCommand executions per day using thread isolation.
- 2 Each API instance has 40+ thread-pools with 5-20 threads in each (most are set to 10).
- 3



1. Hystrix使用命令模式HystrixCommand(Command)包装依赖调用逻辑，每个命令在单独线程中/信号授权下执行。
2. 可配置依赖调用超时时间,超时时间一般设为比99.5%平均时间略高即可.当调用超时时，直接返回或执行fallback逻辑。
3. 为每个依赖提供一个小的线程池（或信号），如果线程池已满调用将被立即拒绝，默认不采用排队.加速失败判定时间。
4. 依赖调用结果分:成功，失败（抛出异常），超时，线程拒绝，短路。请求失败(异常，拒绝，超时，短路)时执行fallback(降级)逻辑。
5. 提供熔断器组件,可以自动运行或手动调用,停止当前依赖一段时间(10秒)，熔断器默认错误率阈值为50%，超过将自动运行。
6. 提供近实时依赖的统计和监控

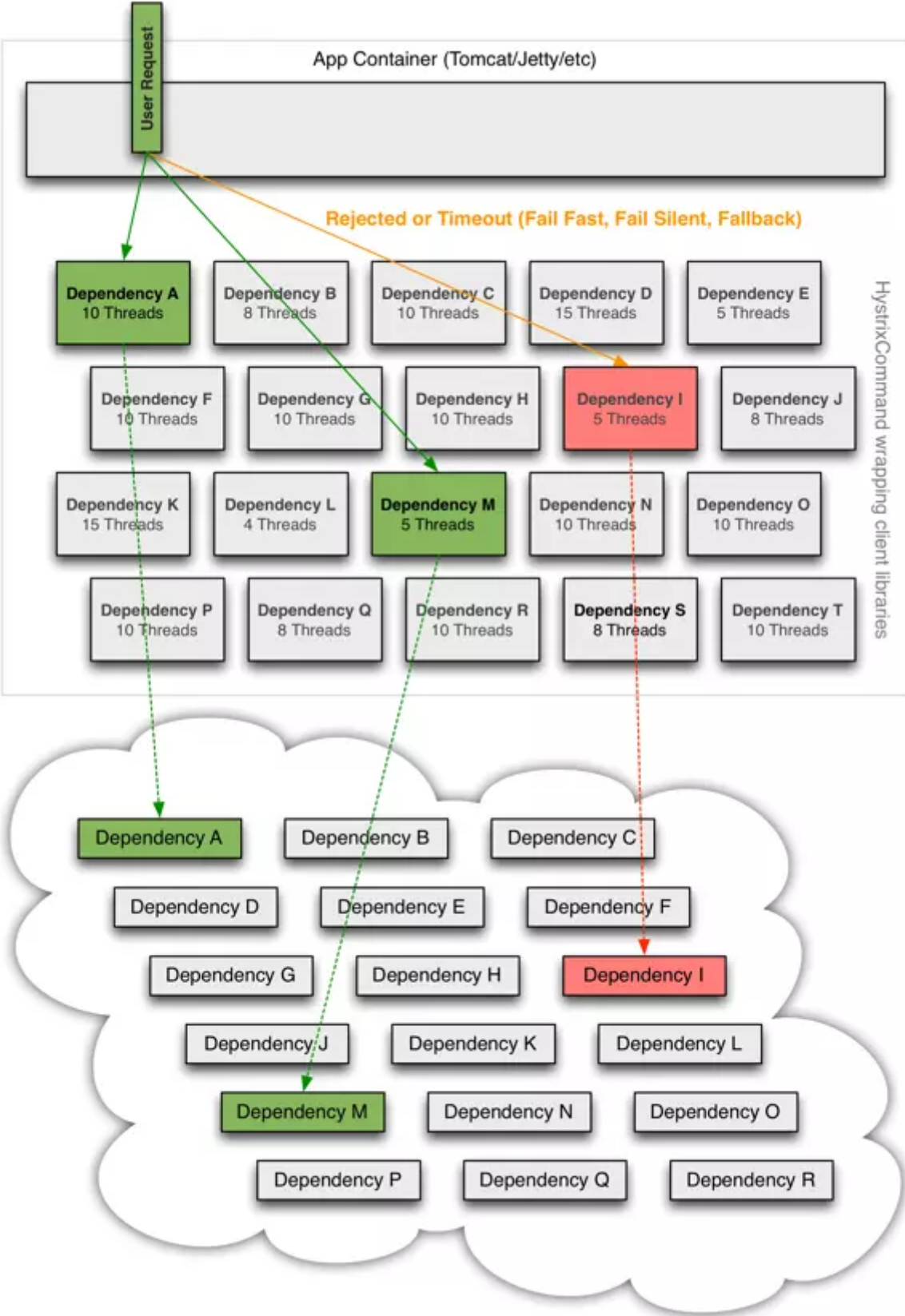
Hystrix依赖的隔离架构,如下图:



首页 ▾

搜索更新啦





3. 如何使用Hystrix



首页 ▾

搜索更新啦



```

1 <!-- 依赖版本 -->
2 <hystrix.version>1.3.16</hystrix.version>
3 <hystrix-metrics-event-stream.version>1.1.2</hystrix-metrics-event-stream.version>
4
5 <dependency>
6   <groupId>com.netflix.hystrix</groupId>
7   <artifactId>hystrix-core</artifactId>
8   <version>${hystrix.version}</version>
9 </dependency>
10 <dependency>
11   <groupId>com.netflix.hystrix</groupId>
12   <artifactId>hystrix-metrics-event-stream</artifactId>
13   <version>${hystrix-metrics-event-stream.version}</version>
14 </dependency>
15 <!-- 仓库地址 -->
16 <repository>
17   <id>nexus</id>
18   <name>local private nexus</name>
19   <url>http://maven.oschina.net/content/groups/public/</url>
20   <releases>
21     <enabled>true</enabled>
22   </releases>
23   <snapshots>
24     <enabled>false</enabled>
25   </snapshots>
26 </repository>

```

2. 使用命令模式封装依赖逻辑

```

1 public class HelloWorldCommand extends HystrixCommand<String> {
2   private final String name;
3   public HelloWorldCommand(String name) {
4     //最少配置:指定命令组名(CommandGroup)
5     super(HystrixCommandGroupKey.Factory.asKey("ExampleGroup"));
6     this.name = name;
7   }
8   @Override
9   protected String run() {
10    // 依赖逻辑封装在run()方法中
11    return "Hello " + name + " thread:" + Thread.currentThread().getName();
12  }
13  //调用实例
14  public static void main(String[] args) throws Exception{
15    //每个Command对象只能调用一次,不可以重复调用,
16    //重复调用对应异常信息:This instance can only be executed once. Please instantiate a new instance
17    HelloWorldCommand helloWorldCommand = new HelloWorldCommand("Synchronous-hystrix");
18    //使用execute()同步调用代码,效果等同于:helloWorldCommand.queue().get();
19    String result = helloWorldCommand.execute();
20    System.out.println("result=" + result);
21
22    helloWorldCommand = new HelloWorldCommand("Asynchronous-hystrix");
23    //异步调用,可自由控制获取结果时机,
24    Future<String> future = helloWorldCommand.queue();
25    //get操作不能超过command定义的超时时间,默认:1秒
26    result = future.get(100, TimeUnit.MILLISECONDS);
27    System.out.println("result=" + result);
28    System.out.println("mainThread=" + Thread.currentThread().getName());
29  }
30
31 }
32 //运行结果: run()方法在不同的线程下执行
33 // result=Hello Synchronous-hystrix thread:hystrix-HelloWorldGroup-1

```


[首页](#)

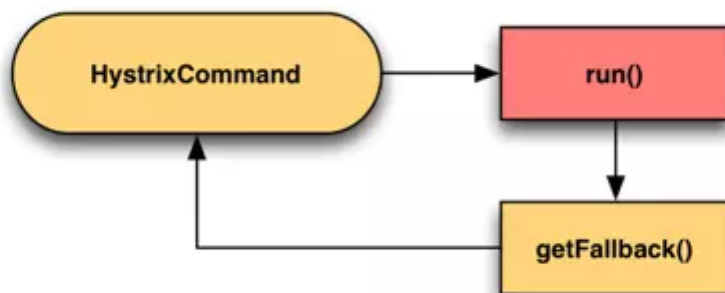


note:异步调用使用 `command.queue().get(timeout, TimeUnit.MILLISECONDS)`;同步调用使用`command.execute()` 等同于 `command.queue().get()`;

3. 注册异步事件回调执行

```
1 //注册观察者事件拦截
2 Observable<String> fs = new HelloWorldCommand("World").observe();
3 //注册结果回调事件
4 fs.subscribe(new Action1<String>() {
5     @Override
6     public void call(String result) {
7         //执行结果处理,result 为HelloWorldCommand返回的结果
8         //用户对结果做二次处理.
9     }
10 });
11 //注册完整执行生命周期事件
12 fs.subscribe(new Observer<String>() {
13     @Override
14     public void onComplete() {
15         // onNext/onError完成之后最后回调
16         System.out.println("execute onComplete");
17     }
18     @Override
19     public void onError(Throwable e) {
20         // 当产生异常时回调
21         System.out.println("onError " + e.getMessage());
22         e.printStackTrace();
23     }
24     @Override
25     public void onNext(String v) {
26         // 获取结果后回调
27         System.out.println("onNext: " + v);
28     }
29 });
30 /* 运行结果
31 call execute result=Hello observe-hystrix thread:hystrix-HelloWorldGroup-3
32 onNext: Hello observe-hystrix thread:hystrix-HelloWorldGroup-3
33 execute onComplete
34 */
```

4. 使用Fallback() 提供降级策略




```

5      super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("HelloWorldGroup"))
6          /* 配置依赖超时时间,500毫秒*/
7          .andCommandPropertiesDefaults(HystrixCommandProperties.Setter().withExecutionIsolation
8      this.name = name;
9  }
10 @Override
11 protected String getFallback() {
12     return "exeucute Falled";
13 }
14 @Override
15 protected String run() throws Exception {
16     //sleep 1 秒,调用会超时
17     TimeUnit.MILLISECONDS.sleep(1000);
18     return "Hello " + name + " thread:" + Thread.currentThread().getName();
19 }
20 public static void main(String[] args) throws Exception{
21     HelloWorldCommand command = new HelloWorldCommand("test-Fallback");
22     String result = command.execute();
23 }
24 }
25 /* 运行结果:getFallback() 调用运行
26 getFallback executed
27 */

```

NOTE: 除了HystrixBadRequestException异常之外,所有从run()方法抛出的异常都算作失败,并触发降级getFallback()和断路器逻辑。

1 HystrixBadRequestException用在非法参数或非系统故障异常等不应触发回退逻辑的场景。

5. 依赖命名:CommandKey

```

1 public HelloWorldCommand(String name) {
2     super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("ExampleGroup"))
3         /* HystrixCommandKey工厂定义依赖名称 */
4         .andCommandKey(HystrixCommandKey.Factory.asKey("HelloWorld")));
5     this.name = name;
6 }

```

NOTE: 每个CommandKey代表一个依赖抽象,相同的依赖要使用相同的CommandKey名称。依赖隔离的根本就是对相同CommandKey的依赖做隔离。

6. 依赖分组:CommandGroup

命令分组用于对依赖操作分组,便于统计,汇总等。

```

1 //使用HystrixCommandGroupKey工厂定义
2 public HelloWorldCommand(String name) {
3     Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("HelloWorldGroup"))
4 }

```

7. 线程池/信号:ThreadPoolKey

```
1 public HelloWorldCommand(String name) {
2     super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("ExampleGroup"))
3         .andCommandKey(HystrixCommandKey.Factory.asKey("HelloWorld")))
4     /* 使用HystrixThreadPoolKey工厂定义线程池名称*/
5     .andThreadPoolKey(HystrixThreadPoolKey.Factory.asKey("HelloWorldPool")));
6     this.name = name;
7 }
```

NOTE: 当对同一业务依赖做隔离时使用CommandGroup做区分,但是对同一依赖的不同远程调用如(一个是redis 一个是http),可以使用HystrixThreadPoolKey做隔离区分.

1 最然在业务上都是相同的组，但是需要在资源上做隔离时，可以使用HystrixThreadPoolKey区分。

8. 请求缓存 Request-Cache

```
1 public class RequestCacheCommand extends HystrixCommand<String> {
2     private final int id;
3     public RequestCacheCommand( int id) {
4         super(HystrixCommandGroupKey.Factory.asKey("RequestCacheCommand"));
5         this.id = id;
6     }
7     @Override
8     protected String run() throws Exception {
9         System.out.println(Thread.currentThread().getName() + " execute id=" + id);
10        return "executed=" + id;
11    }
12    //重写getCacheKey方法,实现区分不同请求的逻辑
13    @Override
14    protected String getCacheKey() {
15        return String.valueOf(id);
16    }
17
18    public static void main(String[] args){
19        HystrixRequestContext context = HystrixRequestContext.initializeContext();
20        try {
21            RequestCacheCommand command2a = new RequestCacheCommand(2);
22            RequestCacheCommand command2b = new RequestCacheCommand(2);
23            Assert.assertTrue(command2a.execute());
24            //isResponseFromCache判定是否是在缓存中获取结果
25            Assert.assertFalse(command2a.isResponseFromCache());
26            Assert.assertTrue(command2b.execute());
27            Assert.assertTrue(command2b.isResponseFromCache());
28        } finally {
29            context.shutdown();
30        }
31        context = HystrixRequestContext.initializeContext();
32        try {
33            RequestCacheCommand command3b = new RequestCacheCommand(2);
34            Assert.assertTrue(command3b.execute());
35            Assert.assertFalse(command3b.isResponseFromCache());
36        } finally {
37            context.shutdown();
38        }
39    }
40 }
```



NOTE:请求缓存可以让(CommandKey/CommandGroup)相同的情况下,直接共享结果,降低依赖调用次数,在高并发和CacheKey碰撞率高场景下可以提升性能.

Servlet容器中,可以直接实用Filter机制Hystrix请求上下文

```

1  public class HystrixRequestContextServletFilter implements Filter {
2      public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
3          throws IOException, ServletException {
4          HystrixRequestContext context = HystrixRequestContext.initializeContext();
5          try {
6              chain.doFilter(request, response);
7          } finally {
8              context.shutdown();
9          }
10     }
11 }
12 <filter>
13     <display-name>HystrixRequestContextServletFilter</display-name>
14     <filter-name>HystrixRequestContextServletFilter</filter-name>
15     <filter-class>com.netflix.hystrix.contrib.request.servlet.HystrixRequestContextServletFilter<
16 </filter>
17 <filter-mapping>
18     <filter-name>HystrixRequestContextServletFilter</filter-name>
19     <url-pattern>/*</url-pattern>
20 </filter-mapping>
21

```

9. 信号量隔离:SEMAPHORE

隔离本地代码或可快速返回远程调用(如memcached,redis)可以直接使用信号量隔离,降低线程隔离开销.

```

1  public class HelloWorldCommand extends HystrixCommand<String> {
2      private final String name;
3      public HelloWorldCommand(String name) {
4          super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("HelloWorldGroup")))
5              /* 配置信号量隔离方式,默认采用线程池隔离 */
6              .andCommandPropertiesDefaults(HystrixCommandProperties.Setter().withExecutionIsolationType(ExecutionIsolationType.SEMAPHORE));
7          this.name = name;
8      }
9      @Override
10     protected String run() throws Exception {
11         return "HystrixThread:" + Thread.currentThread().getName();
12     }
13     public static void main(String[] args) throws Exception{
14         HelloWorldCommand command = new HelloWorldCommand("semaphore");
15         String result = command.execute();
16         System.out.println(result);
17         System.out.println("MainThread:" + Thread.currentThread().getName());
18     }
19 }
20 /** 运行结果
21 HystrixThread:main
22 MainThread:main
23 */

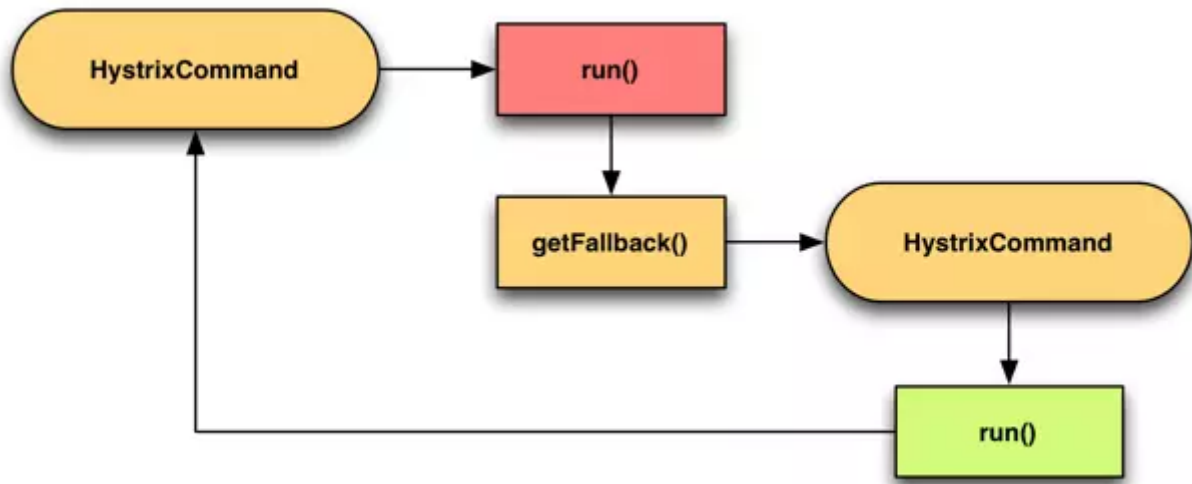
```



首页 ▾

搜索更新啦





用场景:用于fallback逻辑涉及网络访问的情况,如缓存访问。

```

1  public class CommandWithFallbackViaNetwork extends HystrixCommand<String> {
2      private final int id;
3
4      protected CommandWithFallbackViaNetwork(int id) {
5          super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("RemoteServiceX"))
6                .andCommandKey(HystrixCommandKey.Factory.asKey("GetValueCommand")));
7          this.id = id;
8      }
9
10     @Override
11     protected String run() {
12         // RemoteService.getValue(id);
13         throw new RuntimeException("force failure for example");
14     }
15
16     @Override
17     protected String getFallback() {
18         return new FallbackViaNetwork(id).execute();
19     }
20
21     private static class FallbackViaNetwork extends HystrixCommand<String> {
22         private final int id;
23         public FallbackViaNetwork(int id) {
24             super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("RemoteServiceX"))
25                   .andCommandKey(HystrixCommandKey.Factory.asKey("GetValueFallbackCommand"))
26                   // 使用不同的线程池做隔离，防止上层线程池跑满，影响降级逻辑。
27                   .andThreadPoolKey(HystrixThreadPoolKey.Factory.asKey("RemoteServiceXFallback")));
28             this.id = id;
29         }
30         @Override
31         protected String run() {
32             MemCacheClient.getValue(id);
33         }
34
35         @Override
36         protected String getFallback() {
37             return null;
38         }
39     }
40 }
  
```



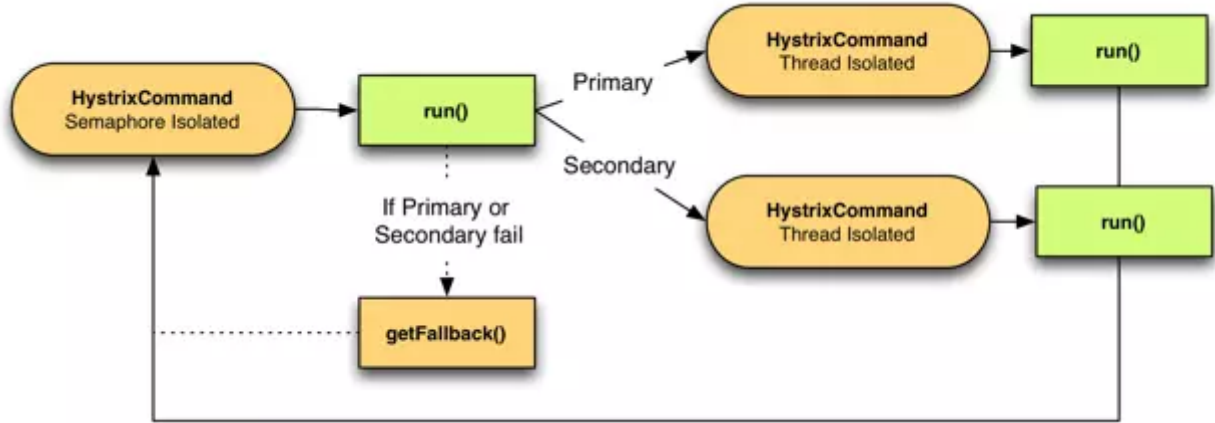
首页 ▾

搜索更新啦



NOTE: 依赖调用和降级调用使用不同的线程池做隔离，防止上层线程池跑满，影响二级降级逻辑调用。

11. 显示调用fallback逻辑,用于特殊业务处理



```

1 public class CommandFacadeWithPrimarySecondary extends HystrixCommand<String> {
2     private final static DynamicBooleanProperty usePrimary = DynamicPropertyFactory.getInstance().get
3     private final int id;
4     public CommandFacadeWithPrimarySecondary(int id) {
5         super(Setter
6             .withGroupKey(HystrixCommandGroupKey.Factory.asKey("SystemX"))
7             .andCommandKey(HystrixCommandKey.Factory.asKey("PrimarySecondaryCommand"))
8             .andCommandPropertiesDefaults(
9                 HystrixCommandProperties.Setter()
10                .withExecutionIsolationStrategy(ExecutionIsolationStrategy.SEMAPHORE)));
11         this.id = id;
12     }
13     @Override
14     protected String run() {
15         if (usePrimary.get()) {
16             return new PrimaryCommand(id).execute();
17         } else {
18             return new SecondaryCommand(id).execute();
19         }
20     }
21     @Override
22     protected String getFallback() {
23         return "static-fallback-" + id;
24     }
25     @Override
26     protected String getCacheKey() {
27         return String.valueOf(id);
28     }
29     private static class PrimaryCommand extends HystrixCommand<String> {
30         private final int id;
31         private PrimaryCommand(int id) {
32             super(Setter
33                 .withGroupKey(HystrixCommandGroupKey.Factory.asKey("SystemX"))
34                 .andCommandKey(HystrixCommandKey.Factory.asKey("PrimaryCommand"))
35                 .andThreadPoolKey(HystrixThreadPoolKey.Factory.asKey("PrimaryCommand"))
36                 .andCommandPropertiesDefaults(
37                     // we default to a 600ms timeout for primary
38                     HystrixCommandProperties.Setter().withExecutionTimeoutInMilliseconds(600)));
39             this.id = id;

```



首页 ▾

搜索更新啦



```

43     // perform expensive 'primary' service call
44     return "responseFromPrimary-" + id;
45 }
46 }
47 private static class SecondaryCommand extends HystrixCommand<String> {
48     private final int id;
49     private SecondaryCommand(int id) {
50         super(Setter
51             .withGroupKey(HystrixCommandGroupKey.Factory.asKey("SystemX"))
52             .andCommandKey(HystrixCommandKey.Factory.asKey("SecondaryCommand"))
53             .andThreadPoolKey(HystrixThreadPoolKey.Factory.asKey("SecondaryCommand"))
54             .andCommandPropertiesDefaults(
55                 // we default to a 100ms timeout for secondary
56                 HystrixCommandProperties.Setter().withExecutionTimeoutInMilliseconds(100)));
57         this.id = id;
58     }
59     @Override
60     protected String run() {
61         // perform fast 'secondary' service call
62         return "responseFromSecondary-" + id;
63     }
64 }
65 public static class UnitTest {
66     @Test
67     public void testPrimary() {
68         HystrixRequestContext context = HystrixRequestContext.initializeContext();
69         try {
70             ConfigurationManager.getConfigInstance().setProperty("primarySecondary.usePrimary", true);
71             assertEquals("responseFromPrimary-20", new CommandFacadeWithPrimarySecondary(20).execute());
72         } finally {
73             context.shutdown();
74             ConfigurationManager.getConfigInstance().clear();
75         }
76     }
77     @Test
78     public void testSecondary() {
79         HystrixRequestContext context = HystrixRequestContext.initializeContext();
80         try {
81             ConfigurationManager.getConfigInstance().setProperty("primarySecondary.usePrimary", false);
82             assertEquals("responseFromSecondary-20", new CommandFacadeWithPrimarySecondary(20).execute());
83         } finally {
84             context.shutdown();
85             ConfigurationManager.getConfigInstance().clear();
86         }
87     }
88 }
89 }

```

NOTE:显示调用降级适用于特殊需求的场景,fallback用于业务处理,fallback不再承担降级职责,建议慎重使用,会造成监控统计混乱等问题.

12. 命令调用合并:HystrixCollapser

命令调用合并允许多个请求合并到一个线程/信号下批量执行。

执行逻辑图如下。

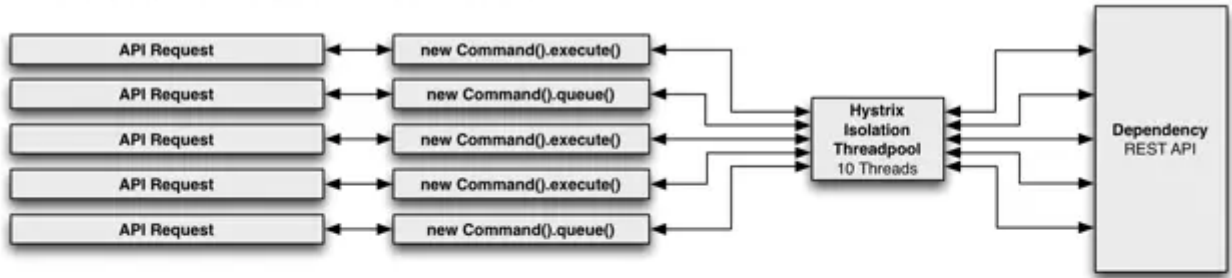


首页 ▾

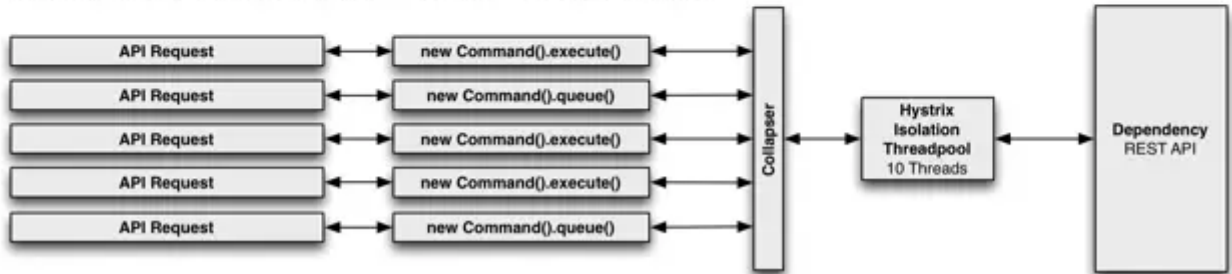
搜索更新啦



Without Collapsing: Request == Thread == Network Connection



With Collapsing: Requests within 'window' == 1 Thread == 1 Network Connection



```

1 public class CommandCollapserGetValueForKey extends HystrixCollapser<List<String>, String, Integer>
2     private final Integer key;
3     public CommandCollapserGetValueForKey(Integer key) {
4         this.key = key;
5     }
6     @Override
7     public Integer getRequestArgument() {
8         return key;
9     }
10    @Override
11    protected HystrixCommand<List<String>> createCommand(final Collection<CollapsedRequest<String,
12        //创建返回command对象
13        return new BatchCommand(requests);
14    }
15    @Override
16    protected void mapResponseToRequests(List<String> batchResponse, Collection<CollapsedRequest<String, Integer>> requests) {
17        int count = 0;
18        for (CollapsedRequest<String, Integer> request : requests) {
19            //手动匹配请求和响应
20            request.setResponse(batchResponse.get(count++));
21        }
22    }
23    private static final class BatchCommand extends HystrixCommand<List<String>> {
24        private final Collection<CollapsedRequest<String, Integer>> requests;
25        private BatchCommand(Collection<CollapsedRequest<String, Integer>> requests) {
26            super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("ExampleGroup"))
27                .andCommandKey(HystrixCommandKey.Factory.asKey("GetValueForKey")));
28            this.requests = requests;
29        }
30        @Override
31        protected List<String> run() {
32            ArrayList<String> response = new ArrayList<String>();
33            for (CollapsedRequest<String, Integer> request : requests) {
34                response.add("ValueForKey: " + request.getArgument());
35            }
36            return response;
37        }
38    }
39    public static class UnitTest {
40        HystrixRequestContext context = HystrixRequestContext.initializeContext();
41        try {
42            Future<String> f1 = new CommandCollapserGetValueForKey(1).queue();

```



首页 ▾

搜索更新啦




```
46 assertEquals("ValueForKey: 1", f1.get());
47 assertEquals("ValueForKey: 2", f2.get());
48 assertEquals("ValueForKey: 3", f3.get());
49 assertEquals("ValueForKey: 4", f4.get());
50 assertEquals(1, HystrixRequestLog.getCurrentRequest().getExecutedCommands().size());
51 HystrixCommand<?> command = HystrixRequestLog.getCurrentRequest().getExecutedCommands().to
52 assertEquals("GetValueForKey", command.getCommandKey().name());
53 assertTrue(command.getExecutionEvents().contains(HystrixEventType.COLLAPSED));
54 assertTrue(command.getExecutionEvents().contains(HystrixEventType.SUCCESS));
55 } finally {
56     context.shutdown();
57 }
58 }
59 }
```

NOTE:使用场景:HystrixCollapser用于对多个相同业务的请求合并到一个线程甚至可以合并到一个连接中执行，降低线程交互次和IO数,但必须保证他们属于同一依赖.

原文出自：

<http://hot66hot.iteye.com/blog/2155036>

觉得本文对你有帮助？请分享给更多人 关注「编程无界」，提升装逼技能



关注下面的标签，发现更多相似文章

Redis

后端

架构

Spring

假不理 Lv4

Java @ 编程无界 (公众号)

获得点赞 4,262 · 获得阅读 97,953

关注



首页 ▾

搜索更新啦



评论

输入评论...

安拉

666

7月前



回复

闪电皮卡丘 工程师 @ WC

这个切入手法，666

7月前



回复

假不理 Lv4 (作者) Java @ 编程无界...

回复 闪电皮卡丘: 😊😊

7月前

相关推荐

专栏 · 衣舞晨风 · 16小时前 · Spring

关于Spring AOP与IOC的个人思考[精品长文]

3

专栏 · 小姐姐味道 · 4天前 · Java / 架构

必看！java后端，亮剑诛仙（最全知识点）

585

22

专栏 · 漫话编程 · 2天前 · 后端 / Java

漫话：如何给女朋友解释为什么必应搜索（Bing）无法访问了？

57

12

专栏 · 江南一点雨 · 10天前 · Spring Boot / 后端

公司倒闭 1 年了，而我当年的项目上了 GitHub 热榜

405

47

专栏 · 林冠宏 · 2天前 · Android / 后端

base16，base32，base64 编码方式的通俗讲解

37

12



[专栏](#) · [小姐姐味道](#) · 1天前 · [Linux / 后端](#)

Echo , Linux上最忧伤的命令


 21

 9

[专栏](#) · [DevYK](#) · 1天前 · [架构](#)

移动架构 (二) Android 中 Handler 架构分析 , 并实现自己简易版本 Handler 框架

 13



[专栏](#) · [DevYK](#) · 3天前 · [架构](#)

移动架构 (一) 架构第一步 , 学会画各种 UML 图。

 42

 6

[专栏](#) · [shanyue](#) · 3天前 · [Node.js / Redis](#)

谈谈 redis 在项目中的常见使用场景

 35

 3

