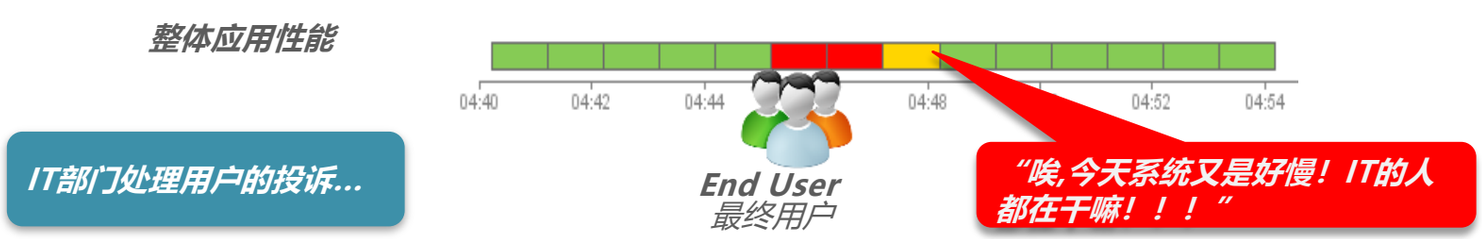
ELASTIC APM
功能深入剖析

Get great thing from data

何为APM


整体应用性能
04:40
04:42
04:44
04:48
04:52
04:54
End User
最终用户
IT部门处理用户的投诉...
“唉,今天系统又是好慢！IT的人都在干嘛！！！”


“是网络问题”
应用的Log都很正常
没有任何Exceptions
应用部门
“网络没问题”
网络流量不大
Ping 时延都正常
网络部门
“与我无关,可能是网络
CPU 是正常的
主机系统部门
“没有什么异常现象”
交易数比平常多一点,不过看起来也是正常的
DBA
问题从哪下手？叫每个部门硬找出问题？
问题时有时无，飘忽不定...怎么办？

何为APM

APM = Application Performance Management，应用性能管理，对企业系统即时监控以实现对应用程序性能管理和故障管理的系统化的解决方案。

APM的覆盖范围包括五个层次的实现：

ELASTIC APM 和 DYNATRACE的比较

dynatrace

VS

elastic

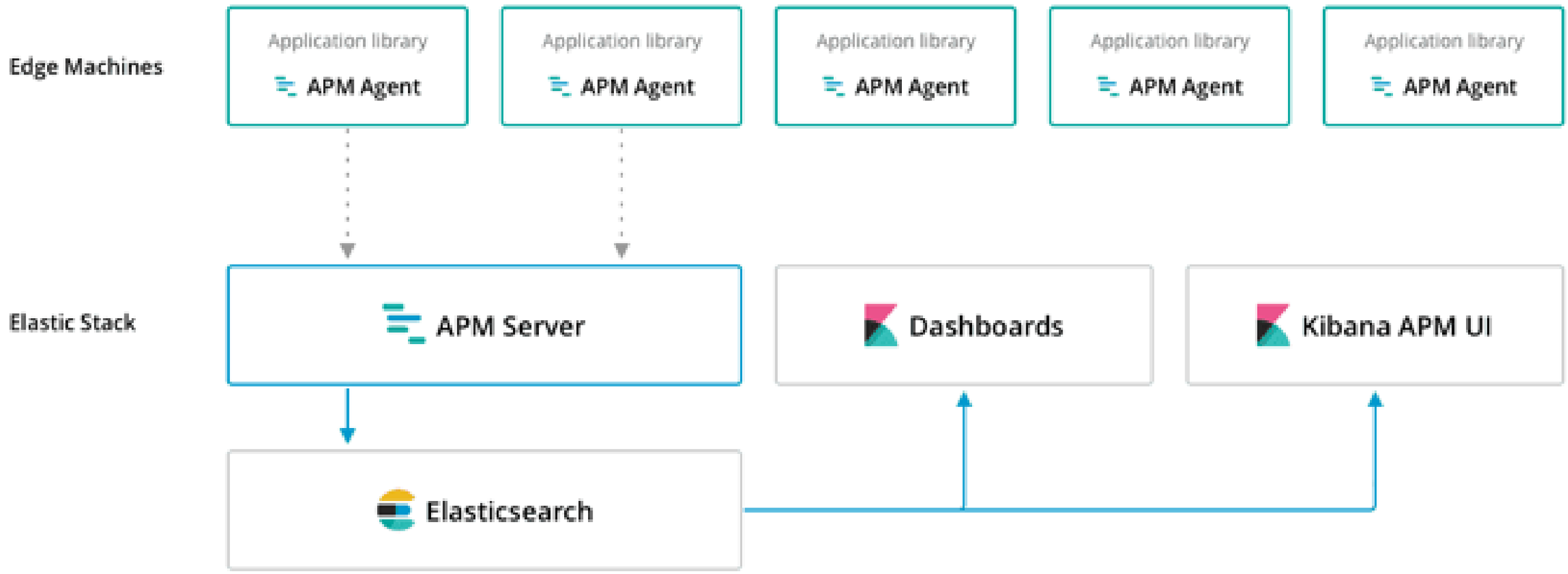
VS

ELASTIC APM 和 DYNATRACE的比较

- 免费 vs 80万
- 开源 vs 闭源
- 手动配置 vs 自动配置
- 多agent vs One Agent
- 组合的解决方案 vs 一体化的解决方案

主要是elastic APM背靠的是elasticsearch，很多时候我们做根因分析，需要集成多方面的数据：海量的日志，实时的在线检索，多维度的条件搜索和模糊查询，以及各种指标等，elastic stack提供的是一个更灵活的解决方案

ELASTIC APM的架构



ELASTIC APM 基本架构

APM

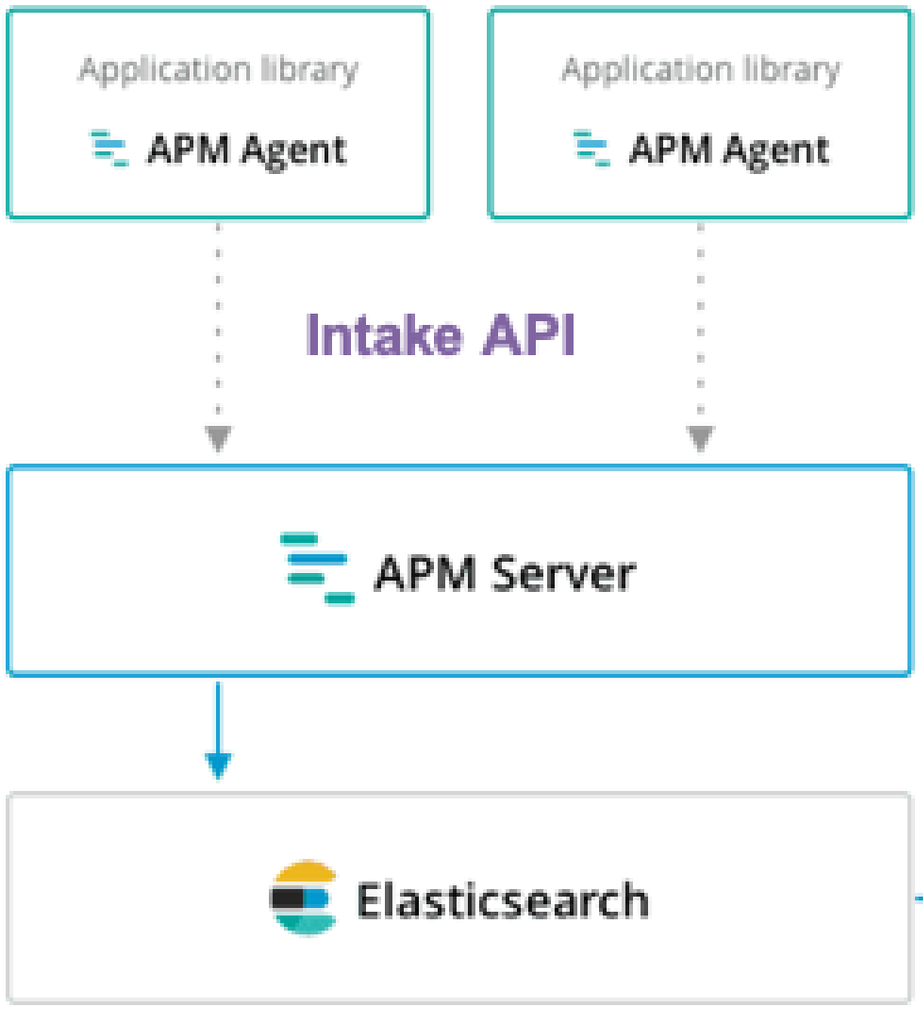
APM Server是一个用Go编写的开源应用程序，通常在专用服务器上运行。它默认侦听端口8200，并通过JSON HTTP API从代理接收数据。然后，它根据该数据创建文档并将其存储在Elasticsearch中。

Elasticsearch 用于存储APM性能指标并利用其聚合。

Kibana

Kibana 提供专用APM UI 和 dashboard来可视化APM数据。

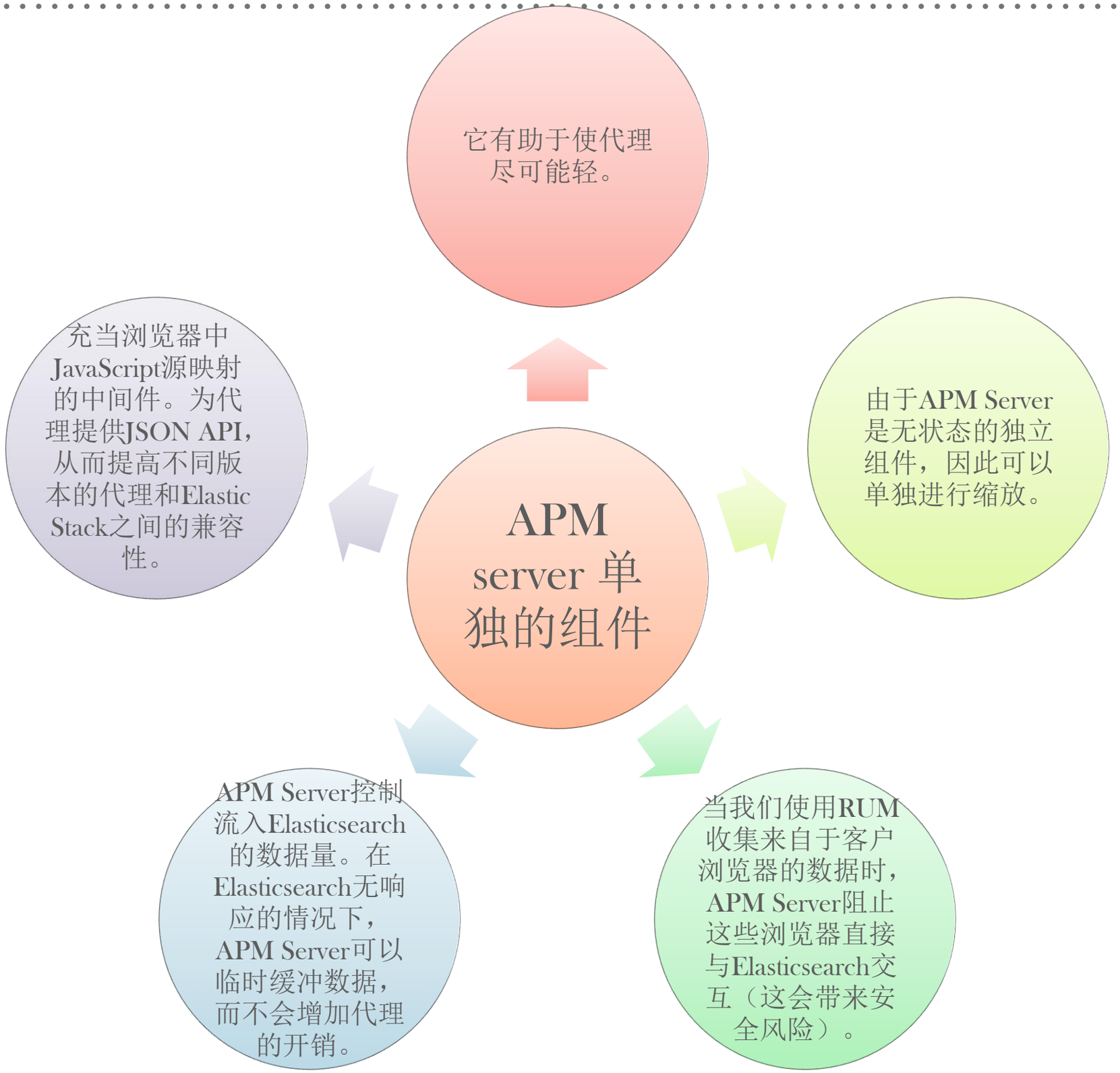
Elastic APM – 组件间通信



- APM服务器是一个单独的组件 - 它有助于保持代理轻量级，防止某些安全风险，并提高整个elastic Stack和APM的兼容性。

- Intake API是的APM agent和APM server进行通信的内部协议。在APM server验证并处理来自APM agent的事件（通过Intake API）后，server将数据转换为Elasticsearch文档并将其存储在相应的Elasticsearch索引中。只需几秒钟，您就可以开始在Kibana中查看应用程序性能数据。

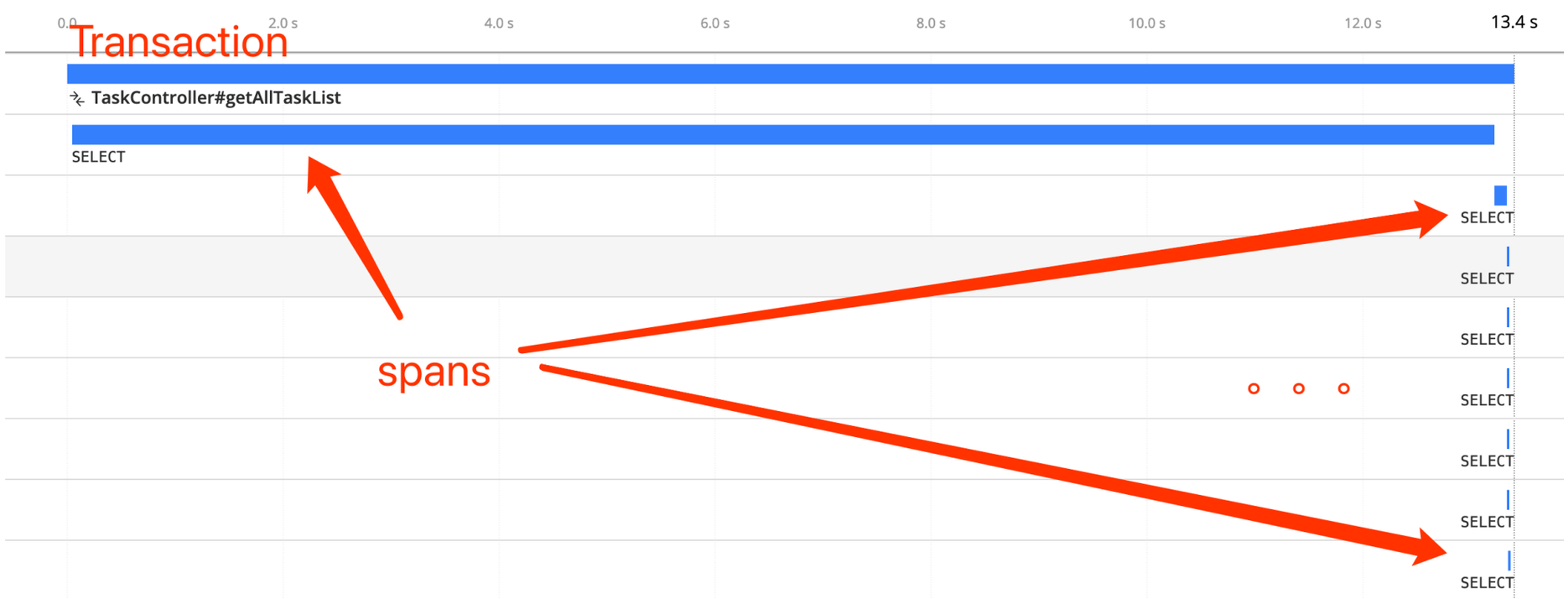
Elastic APM – 组件设计原则



ELASTIC APM的基本要素

APM agent从其已监测的应用程序中捕获不同类型的信息，称为事件。事件可以是Errors，Spans或Transactions。然后将这些事件流式传输到APM server，由server验证并处理事件。

- *Errors 包含捕获的错误或异常的相关信息。*
- *Spans包含已执行的特定代码路径的相关信息。它们从活动的开始到结束进行测量，并且可以与其他跨度建立父/子关系。*
- *Transactions是一种特殊的跨度，具有与之关联的额外元数据。您可以将Transactions视为您在服务中衡量的最高级别的工作。例如，提供HTTP请求或运行特定的后台作业。*



ELASTIC APM 要素- 默认监控

ELASTIC JAVA APM AGENT

https://www.elastic.co/guide/en/apm/agent/java/current/supported-technologies-details.html

栗子干货

SQL的监控

apm java agent默认会监控HTTP和SQL的请求，agent启动后，会在jvm中主动拦截常见数据库驱动层的类以进行监控。但并非所有的SQL访问都会被APM所监控，只有在判定为事务(transaction)或者跨度(span)的事件(event)中才会被监测(instrumented)。而目前，APM自动创建一个transaction的依据是，这是一个HTTP API（即Restful API或Web service，SOAP等）。因此，如果SQL语句位于一个HTTP API的调用链中，则该SQL肯定会被监测；如果你的SQL语句位于非HTTP API调用的函数中，则需要手动创建事务以进行监测。

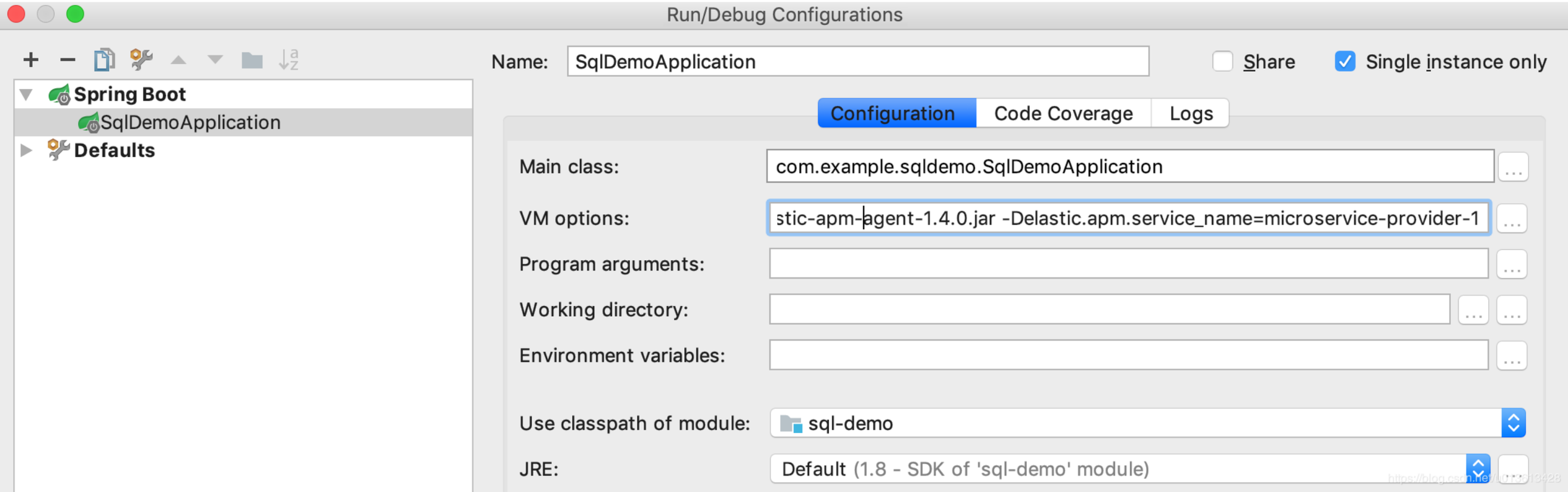
以下是今天各种栗子的基本信息：

- *为了简便，我们先将SQL语句放在HTTP请求的处理函数中。*
- *因为已经很少有人直接使用JDBC，栗子中使用比较流行的Mybatis作为数据层框架*
- 为了PPT能容得下，今天只展示最基本的代码

启动APM

默认的，今天所有的栗子，在java app启动的时候，没有特殊说明都是通过以下方式来启动apm

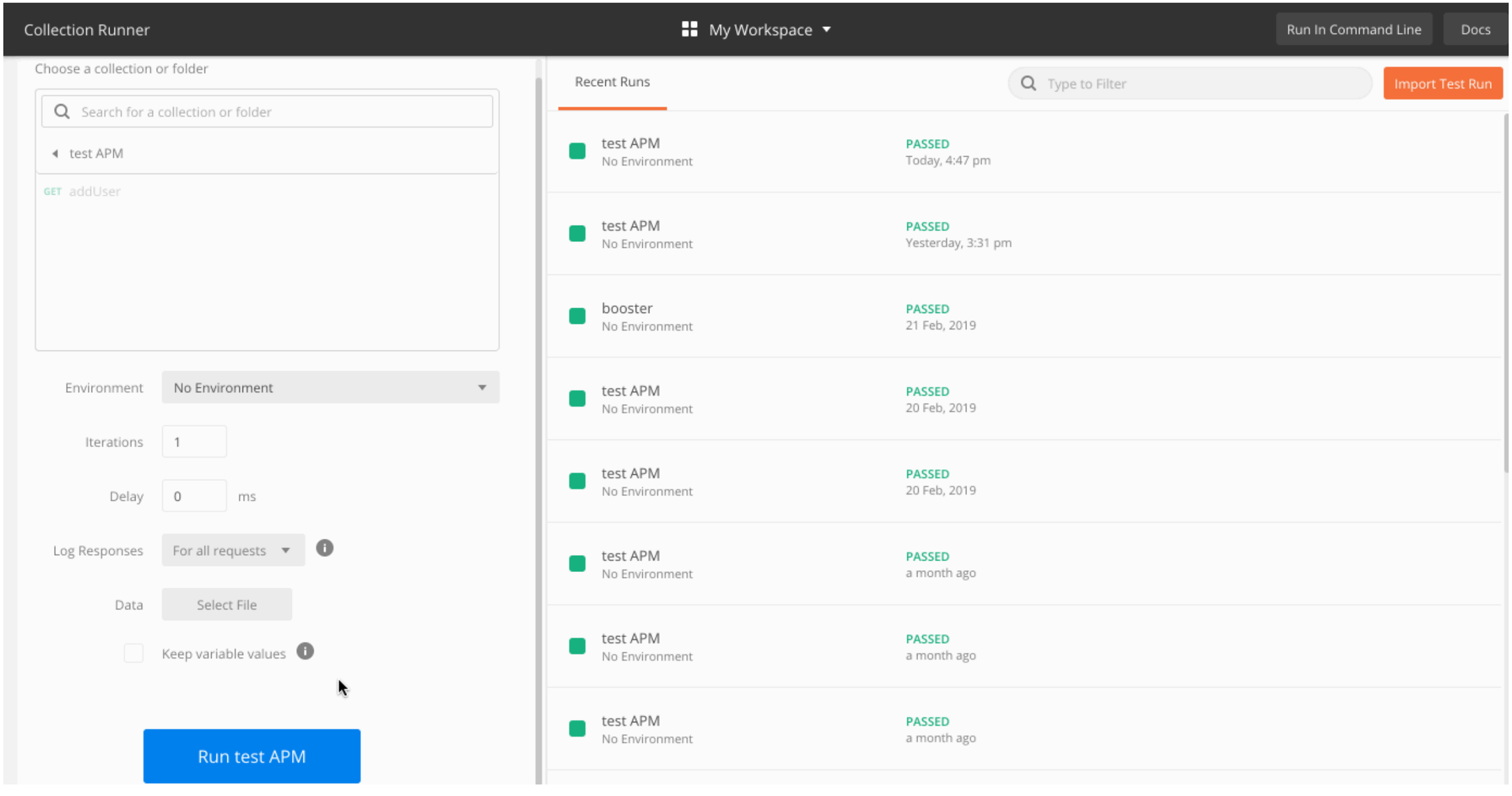
- 通过-javaagent参数把elastic apm java agent作为java探针加载
- 通过service_name来指定服务名称
- 其余信息，比如apm server地址等，配置elasticapm.properties文件中



SQL监控

```
    @RequestMapping( "/addUser" )
    @ResponseBody
    public String addUser() throws ArithmeticException
    {
        boolean AllSuccess = true;
        List<User> users = createRandomUsers();

        try
        {
            for( User user : users )
            {
                AllSuccess &= this.userService.addUser( user );
            }
        }
        catch( Exception e )

        {
            AllSuccess &= false;
        }

        return AllSuccess ? "All success" : "Some failure";
}
```

SQL监控


Collection Runner
My Workspace
Run In Command Line
Docs
Choose a collection or folder
Search for a collection or folder
test APM
GET addUser
Environment
No Environment
Iterations
1
Delay
0
ms
Log Responses
For all requests
Data
Select File
Keep variable values
Run test APM
Recent Runs
Type to Filter
Import Test Run
test APM
No Environment
PASSED
Today, 4:47 pm
test APM
No Environment
PASSED
Yesterday, 3:31 pm
booster
No Environment
PASSED
21 Feb, 2019
test APM
No Environment
PASSED
20 Feb, 2019
test APM
No Environment
PASSED
20 Feb, 2019
test APM
No Environment
PASSED
a month ago
test APM
No Environment
PASSED
a month ago
test APM
No Environment
PASSED
a month ago

SQL监控

现在，我们优化一下代码，在mybatis中加入batch操作：

```
    <insert id ="insertUserBatch" parameterType="java.util.List" useGeneratedKeys="true"
        <selectKey resultType ="java.lang.Integer" keyProperty= "id"
                   order= "AFTER">
            SELECT LAST_INSERT_ID()
        </selectKey>
        insert into user_t
        (id, userName, password, age)
        values
        <foreach collection="list" item="User" index= "index" separator =",">

            ( #{User.id,jdbcType=INTEGER}, #{User.userName,jdbcType=VARCHAR}, #{User.passworc

        </foreach >
    </insert >
```

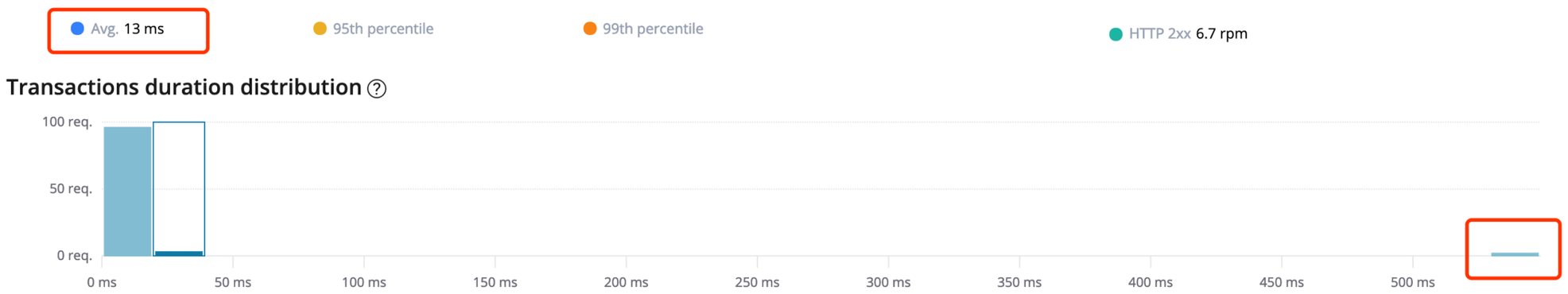
复制

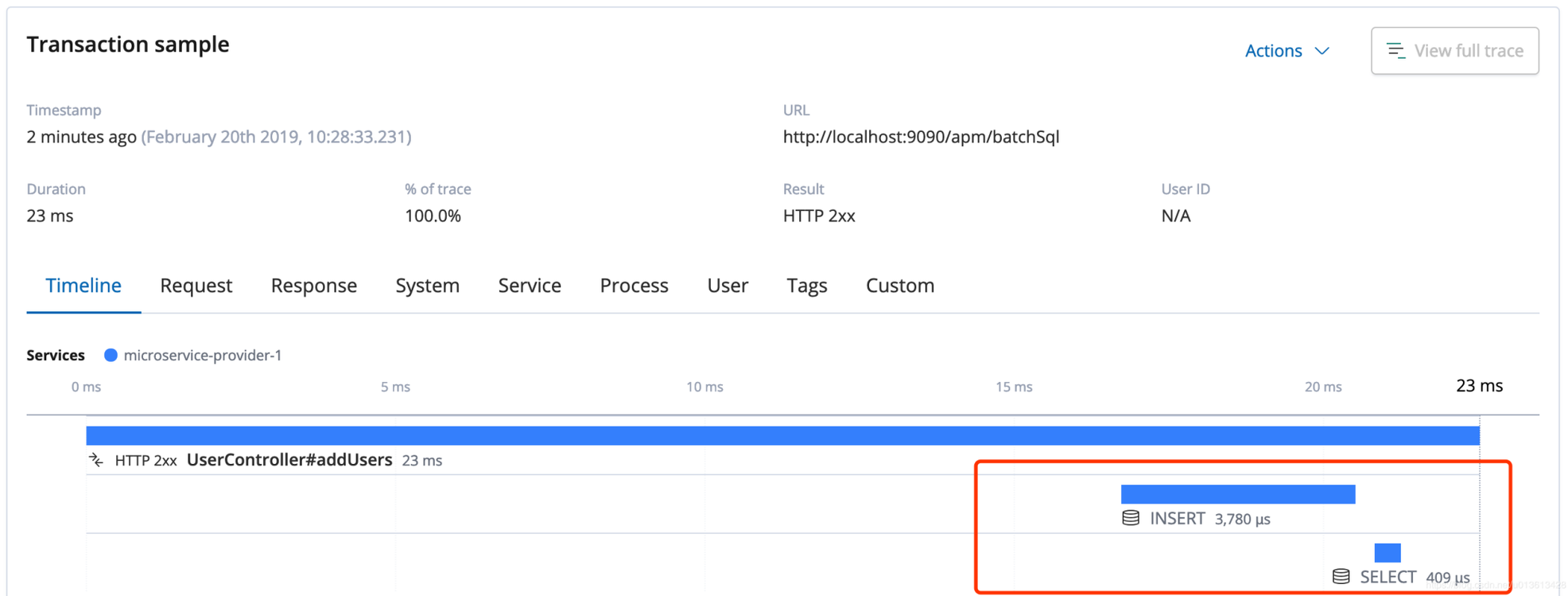
然后接口中新增一个*boolean addUsers(List<User> records);*函数，增加一个新的HTTP接口

```
    @ResponseBody
    public void addUsers()
    {
        this.userService.addUsers( createRandomUsers() );
    }
```

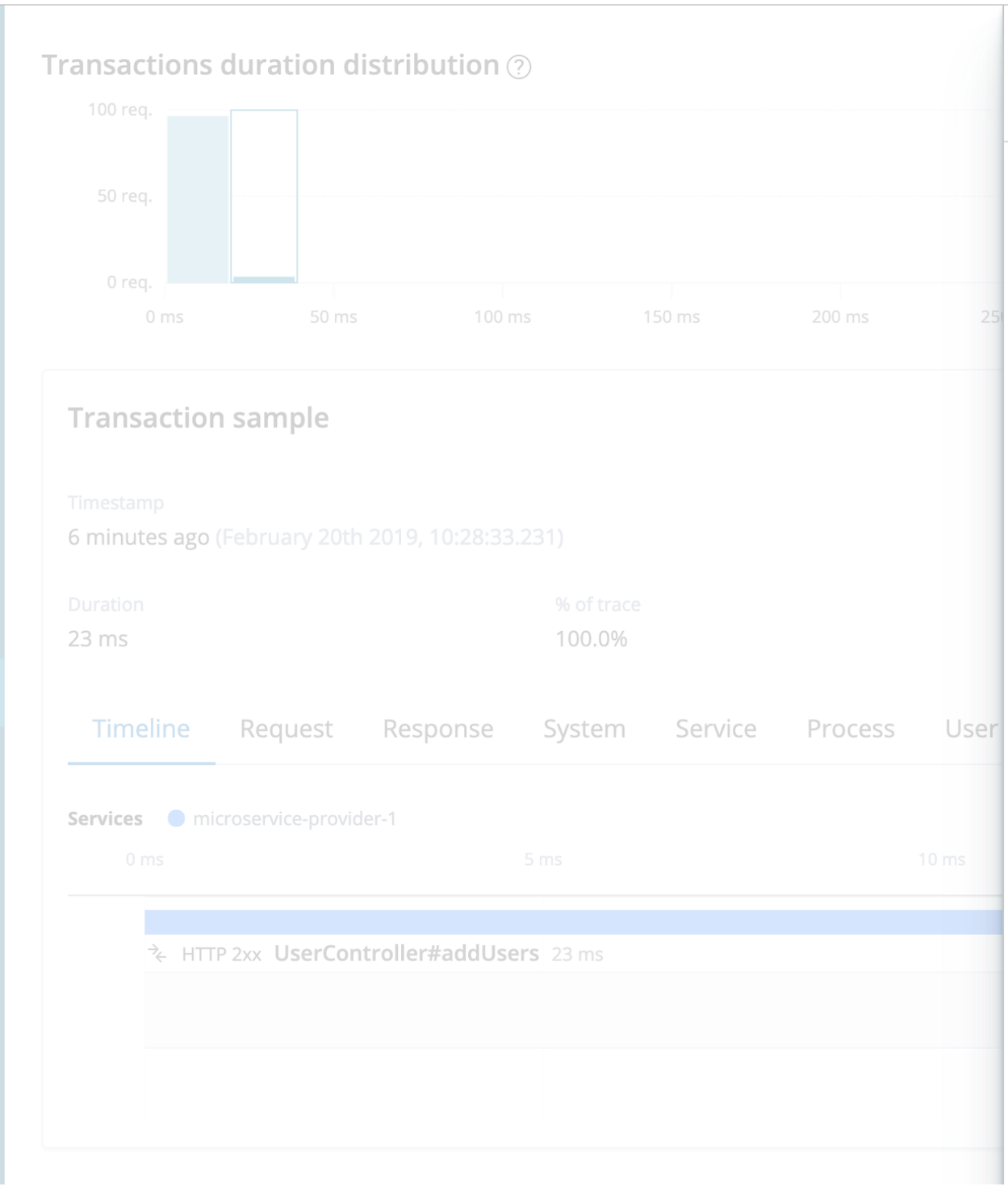
SQL监控

重启服务，这次通过postman运行100次，访问 *addUsers*。再次打开APM的界面，可以看到有几个明显的变化，平均的响应时间只有13ms，另外，SQL语句只剩下一条insert和一条select，点击后，还可以看到变成了batch insert





SQL监控



Span details

View span in Discover

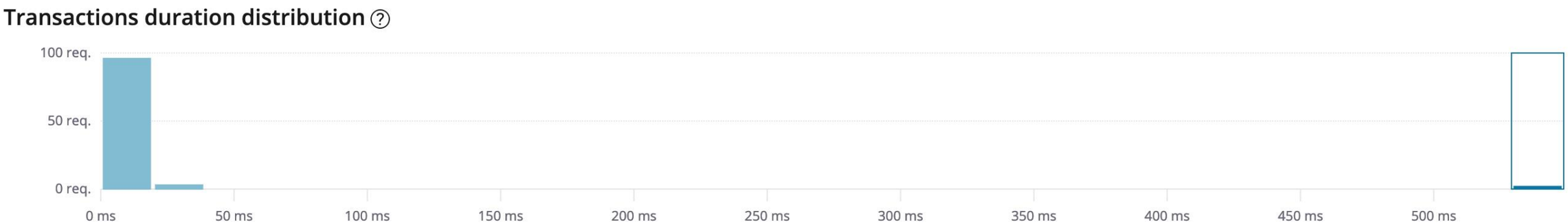
Stack Trace | Tags

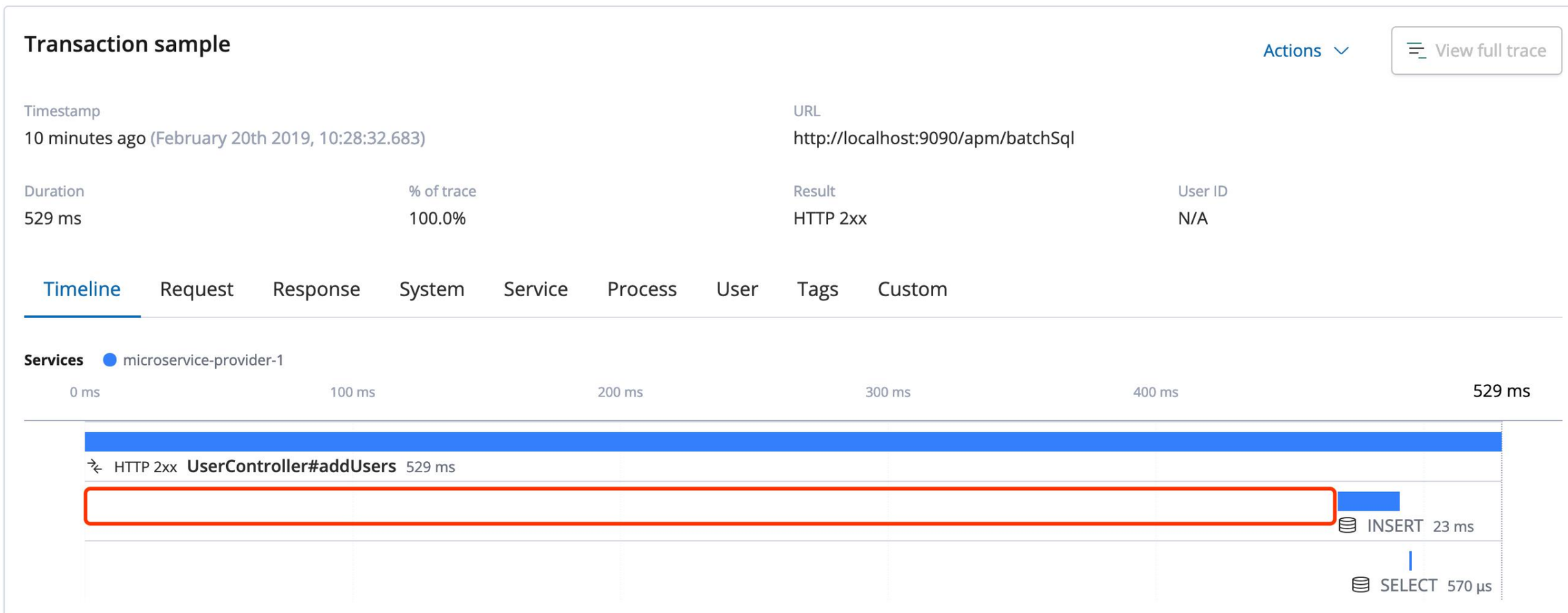
Database statement

```
insert into user_t
        (id, userName, password, age)
        values

        ( ?, ?, ?,?)
      ,
        ( ?, ?, ?,?)
      ,
        ( ?, ?, ?,?)
      ,
        ( ?, ?, ?,?)
      ,
        ( ?, ?, ?,?)
      ,
```

SQL监控

但，我们仍然有一次调用的时间特别的长。而且529毫秒，对于没有100% UT 覆盖，或者没有使用测试工具，纯手工测试的同学，500毫秒仍然是无法感知的性能问题。





手动添加跨度

我们就需要手动将一些函数加入到我们到监测范围当中首先，是我们的函数createRandomUsers()，其次，是我们的数据访问层的mybatis包;

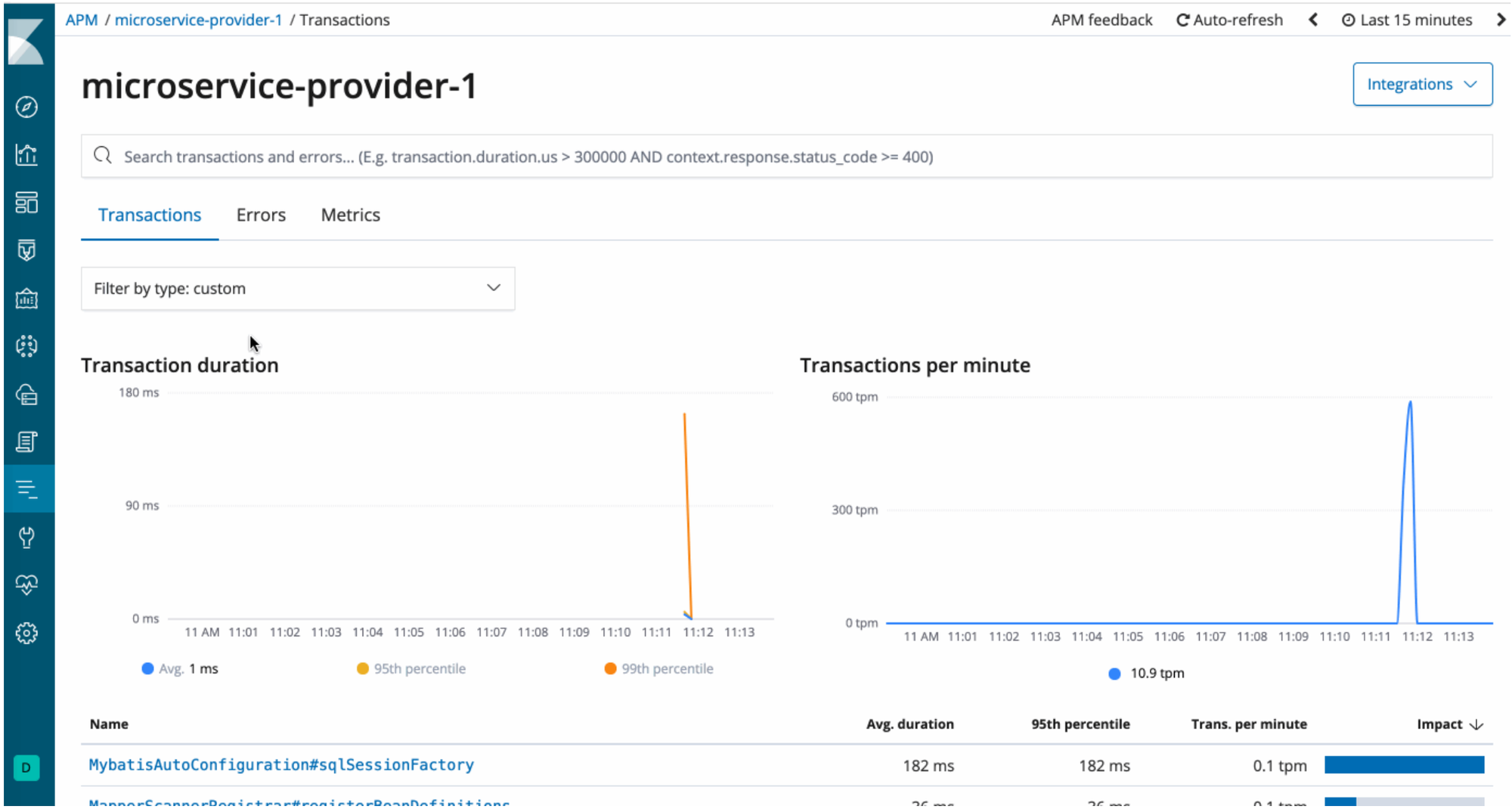
方法是修改apm的配置文件：elasticapm.properties。注意，该文件必须和apm java agent放在同一目录。增加以下内容:

```
trace_methods=com.example.sqldemo.controller.UserController#createRandomUsers,org.mybatis.*
```

这里需要注意的是:

- trace_methods接受一个列表，列表中的对象或者方法，会作为transation或者span的触发者
- 列表中的元素以,进行分隔
- 元素可以是package,class或method，可使用通配符
- class和method之间需要用#分隔

手动添加跨度


APM / microservice-provider-1 / Transactions
APM feedback
Auto-refresh
Last 15 minutes
microservice-provider-1
Integrations
Search transactions and errors... (E.g. transaction.duration.us > 300000 AND context.response.status_code >= 400)
Transactions
Errors
Metrics
Filter by type: custom
Transaction duration
180 ms
90 ms
0 ms
11 AM 11:01 11:02 11:03 11:04 11:05 11:06 11:07 11:08 11:09 11:10 11:11 11:12 11:13
Avg. 1 ms
95th percentile
99th percentile
Transactions per minute
600 tpm
300 tpm
0 tpm
11 AM 11:01 11:02 11:03 11:04 11:05 11:06 11:07 11:08 11:09 11:10 11:11 11:12 11:13
10.9 tpm
Name
Avg. duration
95th percentile
Trans. per minute
Impact
MybatisAutoConfiguration#sqlSessionFactory
182 ms
182 ms
0.1 tpm

定时任务中的远程通信监控

那除了HTTP和SQL之外，我们在做开发的时候，仍然有很多其他的组件需要我们进行远程调用和访问，并有可能变为性能瓶颈。这里我们以用得比较多的redis来举例，看看elastic APM如何完成任务。

另外，除了向外暴露的接口调用需要我们注意性能问题之外，我们在应用程序中也会创建很多定时任务来完成特殊的工作，它们无法被外部直接感知，但仍然有可能是性能瓶颈或者是关键任务，需要被监测。

下面的栗子，我们将对redis的访问，放在定时任务中。

定时任务中的远程通信监控

这里假设我们有一个任务:

- 需要每20秒运行一次
- 需要在redis上读取某些状态
- 然后对DB进行操作

```java
@Component
public class MyTask
{
    @Resource
    private UserService userService;

    @Scheduled( cron = "0/20 * * * * *" )
    public void work()
    {
        System.out.println( Thread.currentThread().getName() );
        Jedis jedis = new Jedis( "x.x.x.x", 6611 );
        jedis.auth( "123456" );

        Set<String> keys = jedis.keys( "*" );
        Iterator<String> it = keys.iterator();
        while( it.hasNext() )
        {
            String key = it.next();
//
        }

        // 对DB的访问
        System.out.println( "getUserById" );
        userService.getUserById( 1 );
    }
}
```

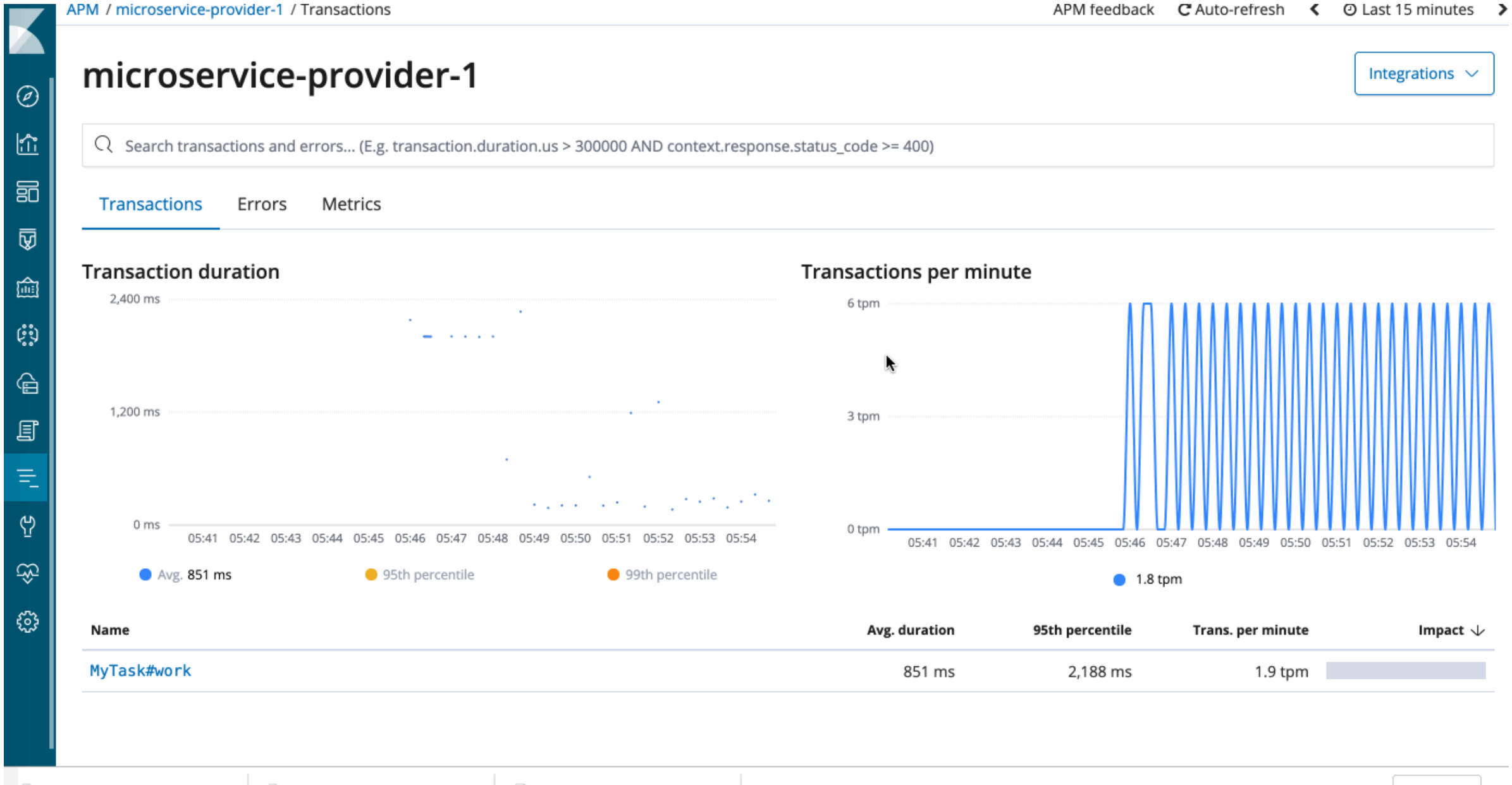
定时任务中的远程通信监控

因为这是一个内部运行的定时任务，**elastic APM**不会对其进行主动监测。因此，我们需要在启动项里面增加对该类的监控。另外，因为引用了Jedis包，为了探测Jedis的性能，我们也需要增加额外的对Jedis的监测。

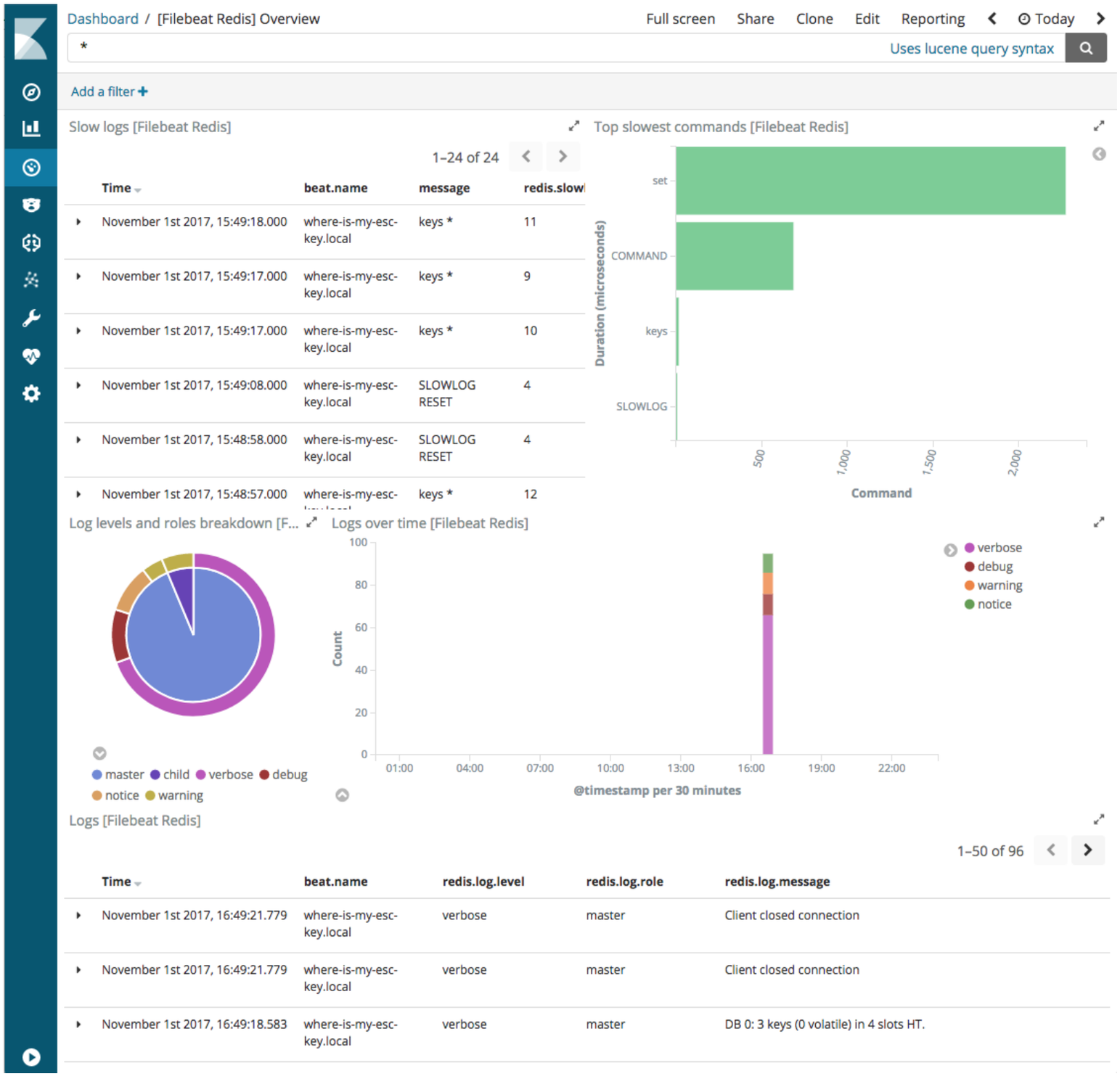
```
trace_methods=com.example.sqldemo.service.*,redis.clients.jedis.Jedis#keys(),redis.clients.jedis.BinaryJedis#*
```

- com.example.sqldemo.service.MyTask#*是对MyTask的监控
- redis.clients.jedis.Jedis#keys是看看我们获取所有的keys需要多少时间
- redis.clients.jedis.BinaryJedis#*是看看连接到Redis的性能

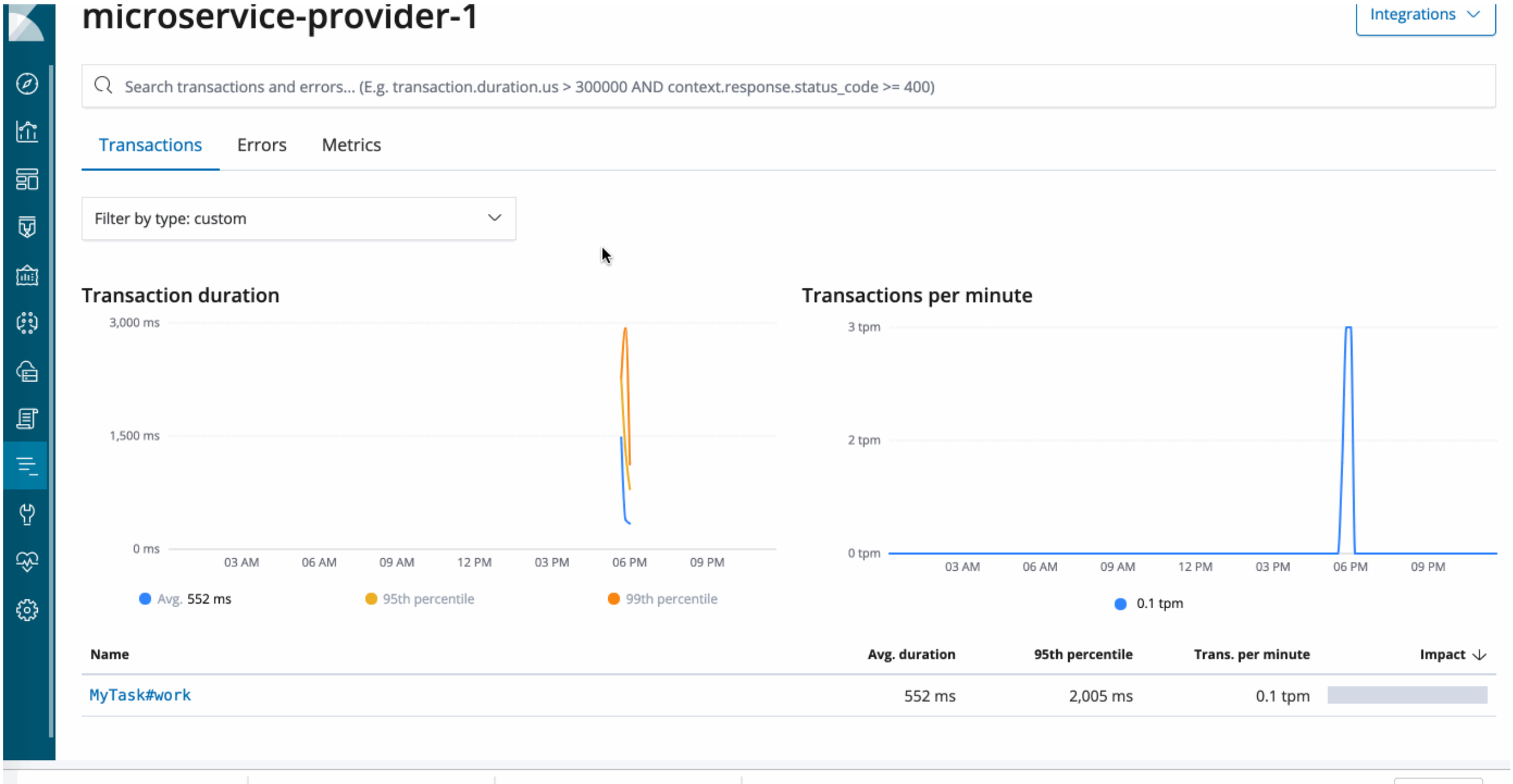
定时任务中的远程通信监控



结合日志分析问题


Dashboard / [Filebeat Redis] Overview
Full screen
Share
Clone
Edit
Reporting
Today
*
Uses lucene query syntax
Add a filter +
Slow logs [Filebeat Redis]
1–24 of 24
Time
beat.name
message
redis.slowl
November 1st 2017, 15:49:18.000 where-is-my-esc-key.local keys * 11
November 1st 2017, 15:49:17.000 where-is-my-esc-key.local keys * 9
November 1st 2017, 15:49:17.000 where-is-my-esc-key.local keys * 10
November 1st 2017, 15:49:08.000 where-is-my-esc-key.local SLOWLOG RESET 4
November 1st 2017, 15:48:58.000 where-is-my-esc-key.local SLOWLOG RESET 4
November 1st 2017, 15:48:57.000 where-is-my-esc- keys * 12
Top slowest commands [Filebeat Redis]
set
COMMAND
keys
SLOWLOG
Duration (microseconds)
500
1,000
1,500
2,000
Command
Log levels and roles breakdown [F...
master child verbose debug
notice warning
Logs over time [Filebeat Redis]
100
80
60
40
20
0
Count
01:00
04:00
07:00
10:00
13:00
16:00
19:00
22:00
@timestamp per 30 minutes
verbose
debug
warning
notice
Logs [Filebeat Redis]
1–50 of 96
Time
beat.name
redis.log.level
redis.log.role
redis.log.message
November 1st 2017, 16:49:21.779 where-is-my-esc-key.local verbose master Client closed connection
November 1st 2017, 16:49:21.779 where-is-my-esc-key.local verbose master Client closed connection
November 1st 2017, 16:49:18.583 where-is-my-esc-key.local verbose master DB 0: 3 keys (0 volatile) in 4 slots HT.

异常的捕获


microservice-provider-1
Integrations
Search transactions and errors... (E.g. transaction.duration.us > 300000 AND context.response.status_code >= 400)
Transactions
Errors
Metrics
Filter by type: custom
Transaction duration
3,000 ms
1,500 ms
0 ms
03 AM
06 AM
09 AM
12 PM
03 PM
06 PM
09 PM
Avg. 552 ms
95th percentile
99th percentile
Transactions per minute
3 tpm
2 tpm
0 tpm
0.1 tpm
Name
Avg. duration
95th percentile
Trans. per minute
Impact
MyTask#work
552 ms
2,005 ms
0.1 tpm

异常的捕获

Error occurrence

View 14 occurrences in Discover

Timestamp
22 minutes ago (March 6th 2019, 17:48:02.007)

URL
N/A

Request method
N/A

Handled
N/A

Transaction sample ID
88f07945d3eab8b4

User ID
N/A

Exception stacktrace | System | Service | Process | User | Tags | Custom

6 library frames

MyTask.java in work at line 25

ScheduledMethodRunnable.java in run at line 84

DelegatingErrorHandlingRunnable.java in run at line 54

ReschedulingRunnable.java in run at line 93

7 library frames

- 我们还可以通过链接跳转到对应的事务中。但经过测试，如果抛出异常的代码，不在transaction或者spans当中的话，则该异常无法被捕获到

异步执行的函数监控

在上一个栗子上，我们在函数里面定时的去远程查询redis，然后再执行下一步操作。

但在日常的开发中，定时任务里面可能需要执行一系列的操作，他们只需要被定时触发，却没有上下文的联系。特别是那些阻塞式的，依赖网络环境的函数调用，往往是被异步执行的。

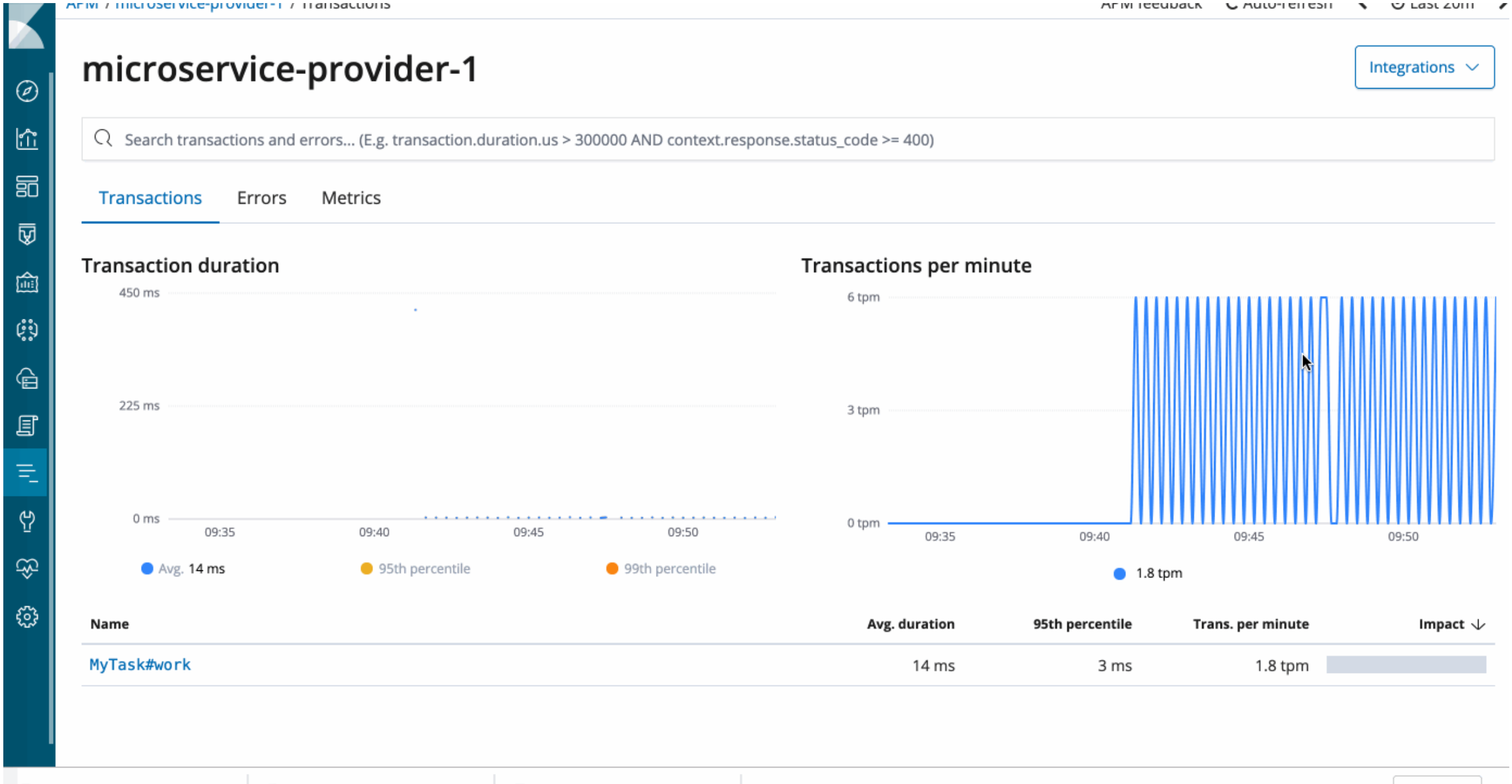
1

```java
@Component
public class RedisServiceImpl
                implements RedisService
{
    @Override
    @Async
    public void reachRedis()
    {
        System.out.println( Thread.currentThread().getName() );
        Jedis jedis = new Jedis( "*.*.*.*", 6611 );
        jedis.auth( "123456" );

        Set<String> keys = jedis.keys( "*" );
        Iterator<String> it = keys.iterator();
        while( it.hasNext() )
        {
            String key = it.next();
        }
    }
}
```

2

```java
@Component
public class MyTask
{
    @Resource
    private UserService userService;
    @Autowired
    private RedisService redisService;

    @Scheduled( cron = "0/20 * * * * *" )
    public void work()
    {
        System.out.println( Thread.currentThread().getName() );
        redisService.reachRedis();
        // 对DB的访问
        System.out.println( "getUserById" );
        userService.getUserById( 1 );
    }

}
```

异步执行的函数监控


microservice-provider-1
Integrations
Search transactions and errors... (E.g. transaction.duration.us > 300000 AND context.response.status_code >= 400)
Transactions
Errors
Metrics
Transaction duration
450 ms
225 ms
0 ms
09:35
09:40
09:45
09:50
Avg. 14 ms
95th percentile
99th percentile
Transactions per minute
6 tpm
3 tpm
0 tpm
09:35
09:40
09:45
09:50
1.8 tpm
Name
Avg. duration
95th percentile
Trans. per minute
Impact
MyTask#work
14 ms
3 ms
1.8 tpm

异步执行的函数监控

总结：通过APM，我们同时观察到了异步函数的运行情况。这点，对于实际的监控来说是相当重要的。**因为很多时候，我们为了提高用户的体验，一些耗时操作往往是被异步化了的，但这些异步操作才是性能的瓶颈，所以，我们即要监控与用户体验相关接口的性能也要同时监控后台执行的程序的性能。**

REAL USER MONITORING

- Real User Monitoring用于捕获用户与Web浏览器等客户端的交互
- javascript agent是Elastic的RUM agent。要使用它，您需要在APM server中启用RUM
- 与监视请求和响应的Elastic APM后端agent不同，RUM JavaScript agent监视客户端应用程序中的真实用户体验和交互
- RUM JavaScript agent也与框架无关，这意味着它可以与任何前端JavaScript应用程序一起使用。您将能够测量诸如time to first byte之类的指标。而domInteractive，domComplete这类指标可以帮助你发现客户端应用程序中的性能问题以及与服务器端应用程序通信延迟的相关问题

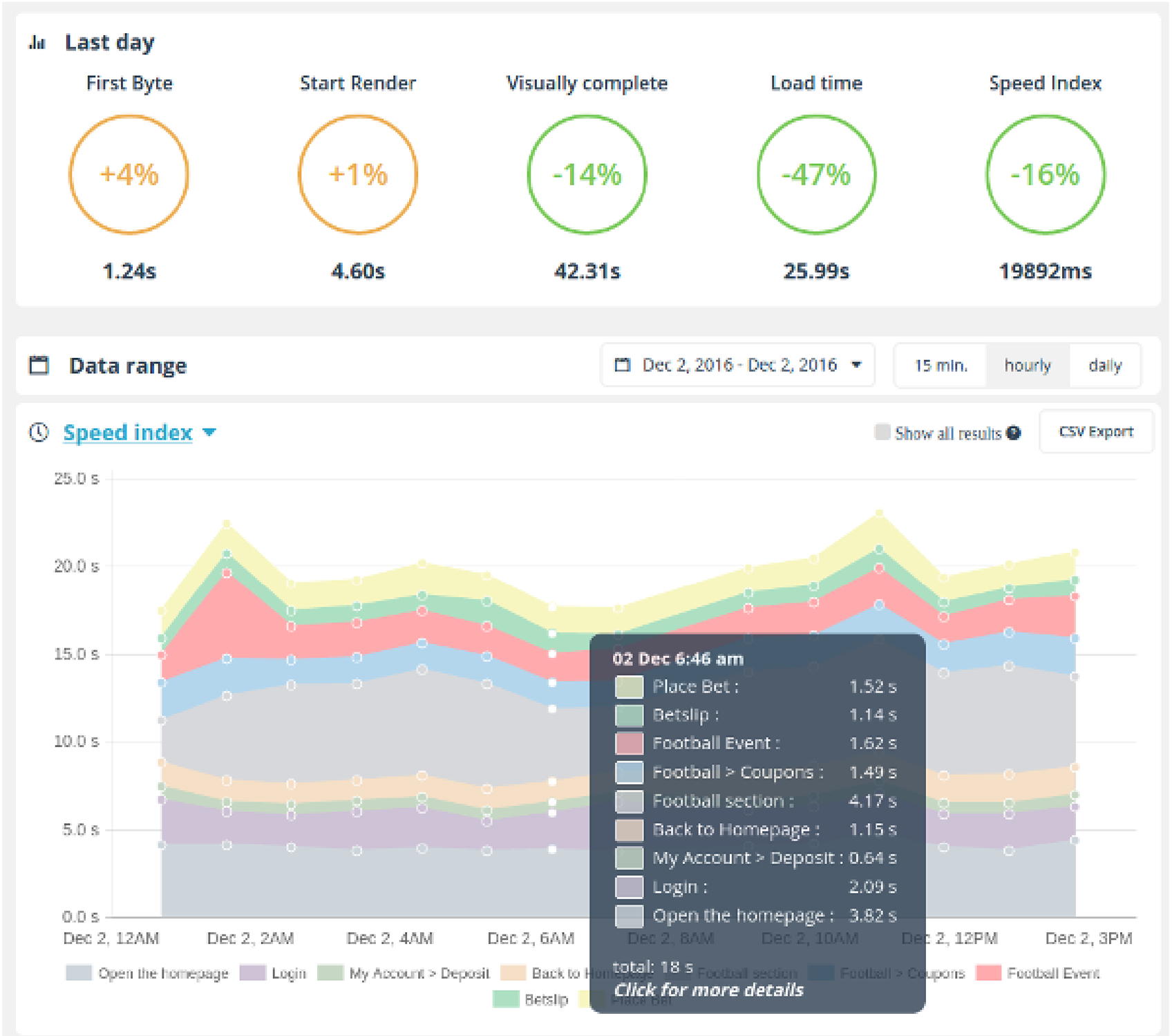
REAL USER MONITORING

- 我们在kibana自己的启动代码里面，启动APM

```
/**
 * The CoreSystem is the root of the new platform, and starts all parts
 * of Kibana in the UI, including the LegacyPlatform which is managed
 * by the LegacyPlatformService. As we migrate more things to the new
 * platform the CoreSystem will get many more Services.
 */

import { init as initApm } from 'elastic-apm-js-base'
const apm = initApm({

  // Set required service name (allowed characters: a-z, A-Z, 0-9, -, _, and space)
  serviceName: 'kibana-rum',

  // Set custom APM Server URL (default: http://localhost:8200)
  serverUrl: 'http://localhost:8200',

  // Set service version (required for sourcemap feature)
  serviceVersion: '',

  pageLoadTransactionName: "kibana-home"

})
```

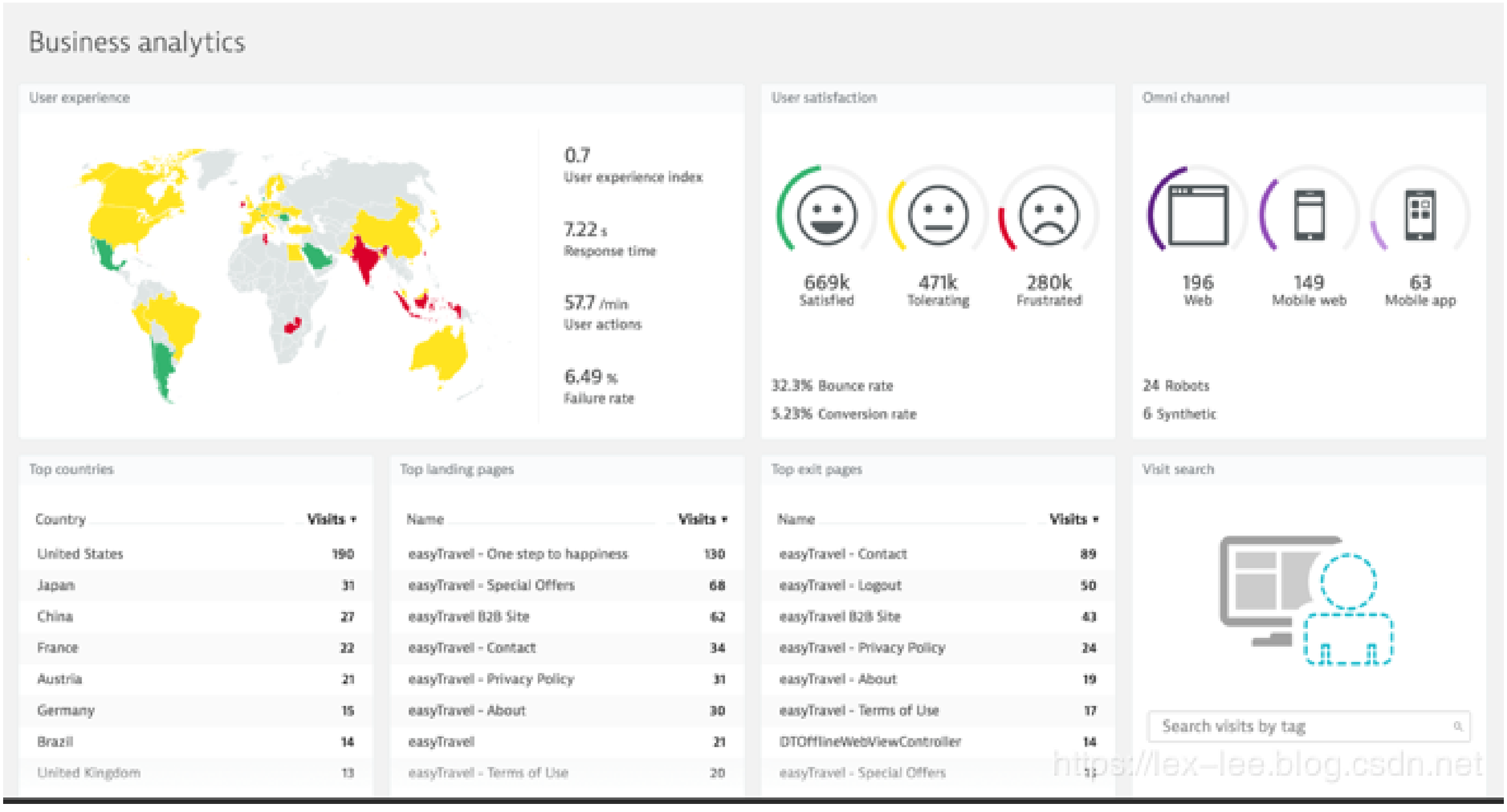
REAL USER MONITORING

REAL USER MONITORING



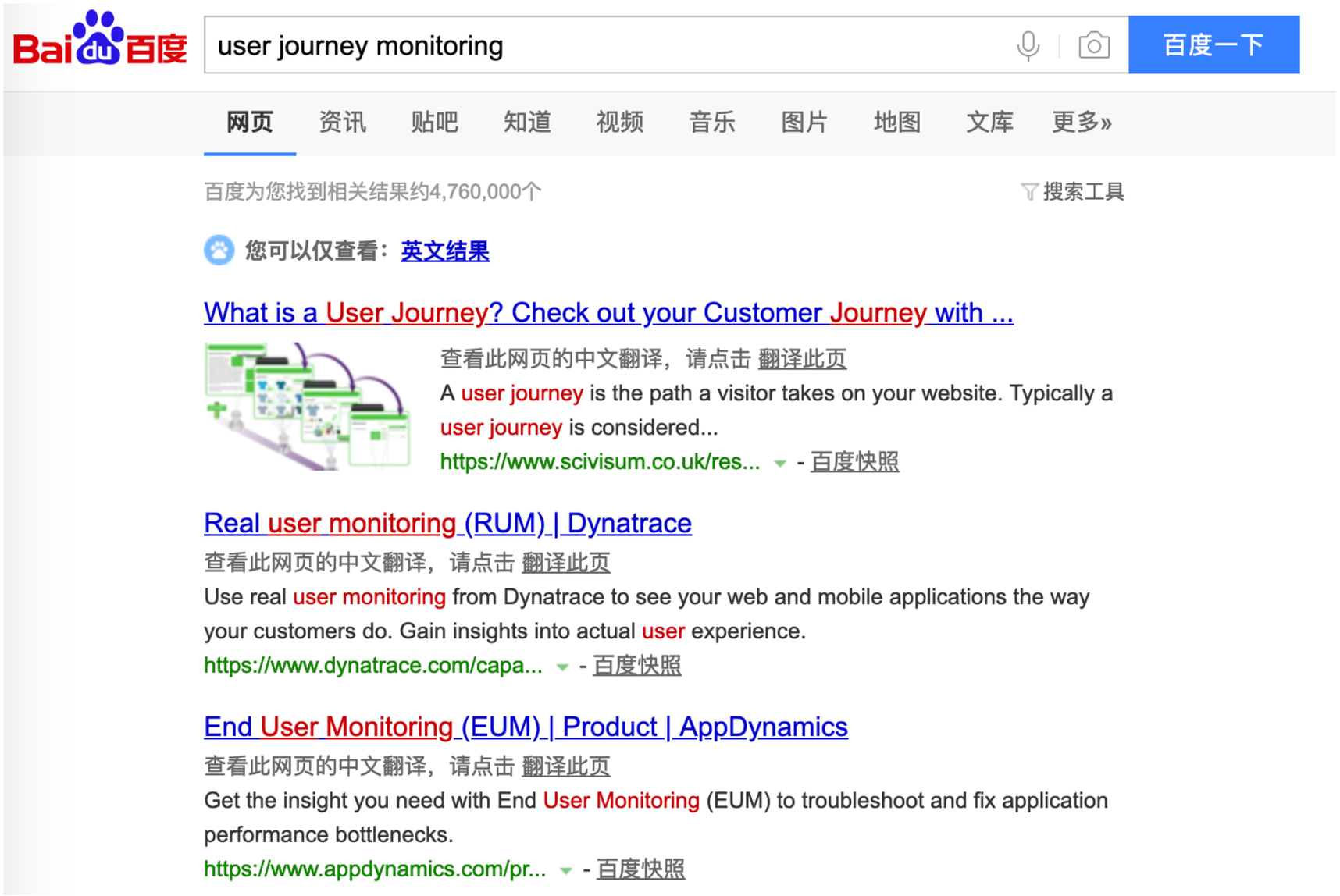
彩蛋：USER JOURNEY MONITORING

实际上，我们想要的功能是这种



USER JOURNEY MONITORING

- 这个概念好像还不是很流行。。。我们姑且把它叫做用户轨迹监控



USER JOURNEY MONITORING

要实现用户的轨迹追踪，必须实现几个基本的要素：

- 页面需能与后端服务器通信，以记录各种用户行为轨迹相关的数据
- 知晓用户发生轨迹切换的事件 - 登录，跳转，离开等
- 当事件发生后，需要记录相关的指标 - 用户名，停留时间，跳转前的页面地址等

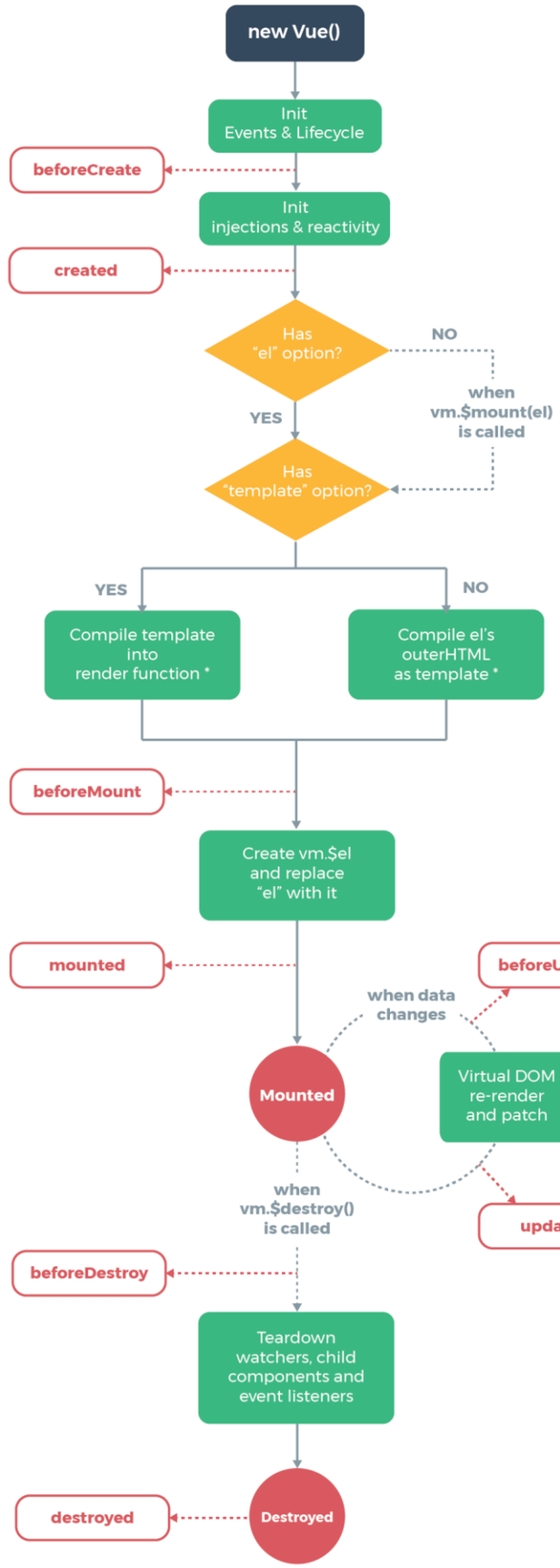
单页应用VS多页应用

为什么我们要讨论单页应用和多页应用的问题，最主要的区别在于：

- 当你的应用是一个多页应用的时候，每个页面的跳转需要访问后端的浏览器获取html相关资源，即你的轨迹记录是可以从类似apache,nginx等日志中提取的，但日志数据是否能够完整包含轨迹数据，存疑；而且，对应的，因为页面之间缺乏直接的联系，页面跳转后，记录需要在页面间传递的轨迹相关数据会比较麻烦。
- 当你的应用是一个单页应用时，页面的“跳转”实际上是浏览器重新渲染页面的一个过程，即用户的访问轨迹并不会出现在日志当中，也不会记录在浏览器的访问历史当中，但相对的，因为是一个整体内部的切换，通过页面上的数据治理，很容易在页面的“跳转”之间的数据传递。

前端框架的组件生命周期

现在流行的web框架，比如，vue, angularJS, react等，都提供了组件生命周期管理的勾子函数，以Vue这个在中国最流行的框架来举例，每个页面都是一个vue的组件（当然，组件里面还有很多子组件）



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

解决方案

从上图可以看到，created，activated，deactivated，destroyed等函数所代表的事件，是和我们需要记录的用户轨迹的事件，如：进入，离开等事件是吻合的。

并且，作为一个勾子函数，我们可以在函数中访问框架提供的组件间通信数据以及公共数据，

如此，我们可以在函数中得到如下信息：

- 登录的用户名，以及其他用户信息
- 进入组件的时间，离开组件的时间，组件停留时间
- 发生组件切换的前后关系

POC代码

```
import {init as initApm} from 'elastic-apm-js-base'

const apm = initApm({

  // Set required service name (allowed characters: a-z, A-Z, 0
  serviceName: 'qsa_smart_advisor',

  // Set custom APM Server URL (default: http://localhost:8200)
  serverUrl: 'http://localhost:8200',

  // Set service version (required for sourcemap feature)
  serviceVersion: '',

  pageLoadTransactionName: 'home'
})
```

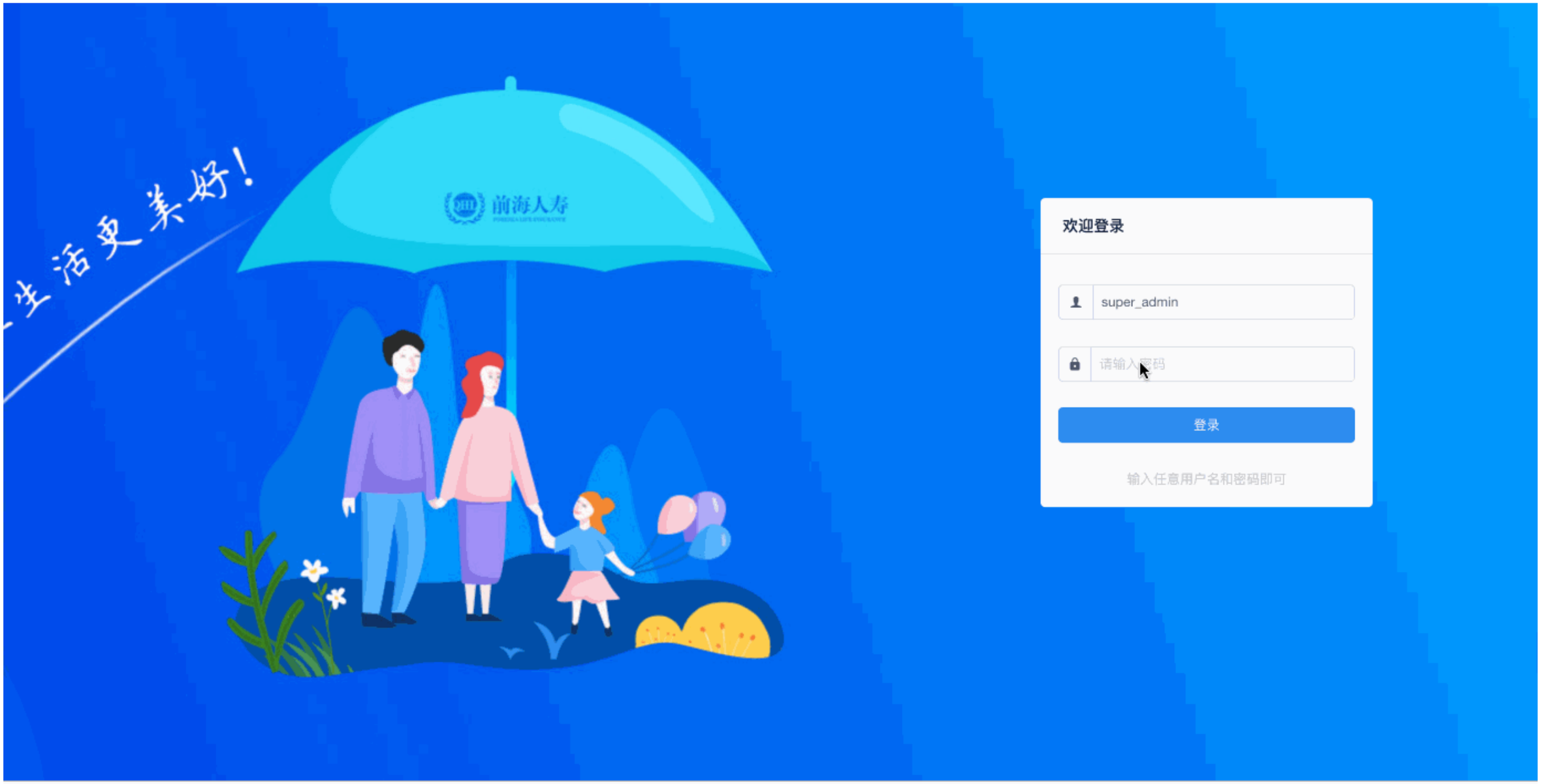
POC代码

```
function startUJMTransaction(theVue) {
  var enableAPM = theVue.$options.enableAPM;
  if (enableAPM) {
    store.dispatch('getUserInfo').then(user => {
      apm.setUserContext({
        id: user.id,
        username: user.name
      })
    });
    if (theVue.$options.transaction) {
      endUJMTransaction(theVue);
    }
    theVue.$options.transaction = apm.startTransaction(theVue.$options.name, 'custom');
    let startTime = new Date().getTime();
    theVue.$options.transaction_start = startTime;
    apm.addTags({int_start: new Date().getTime()});
    console.log("create " + theVue.$options.transaction.name + " with tag: " + {start: st
  }
}

function endUJMTransaction(theVue) {
  var enableAPM = theVue.$options.enableAPM;
  if (enableAPM && theVue.$options.transaction) {
    let endTime = new Date().getTime();

    apm.addTags({int_end: endTime});
    apm.addTags({int_duration: endTime - theVue.$options.transaction_start});

    theVue.$options.transaction.end();

    theVue.$options.transaction = null;
    theVue.$options.transaction_start = null;
```

POC代码

```
Vue.mixin({
  created: function () {
    startUJMTransaction(this)
  },

  activated: function () {
    let currentTrans = apm.getCurrentTransaction();
    if (currentTrans) {
      endUJMTransaction(this)
    }
    startUJMTransaction(this)
  },

  deactivated: function () {
    endUJMTransaction(this)
  },

  destroyed: function () {
    endUJMTransaction(this)
  }
});
```

展示



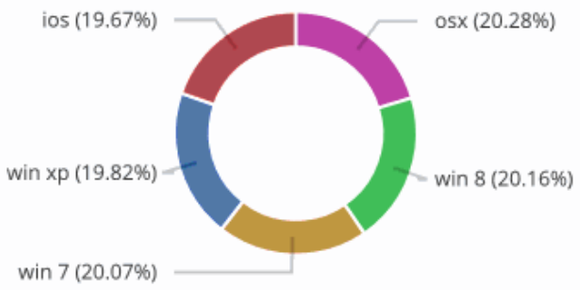
展示

Add a filter +

User Journey Montoring

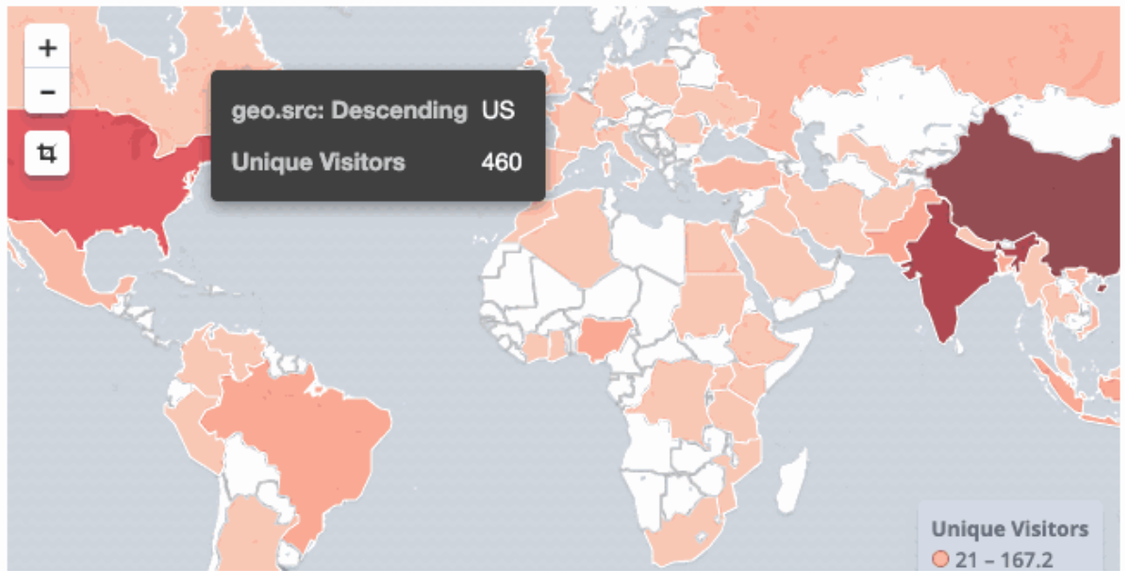
用户	用户使用的组件	访问次数	最大停留时间(ms)	平均停留时间(ms)
super_admin	ranking	2	31,120	17,558.5
super_admin	futures	1	3,289	3,289
super_admin	dashboard	4	31,685	10,210
lex	dashboard	4	6,167	2,423
admin	ranking	1	6,579	6,579
admin	futures	1	7,176	7,176
admin	dashboard	5	40,088	9,556.2

[Logs] Article Tags

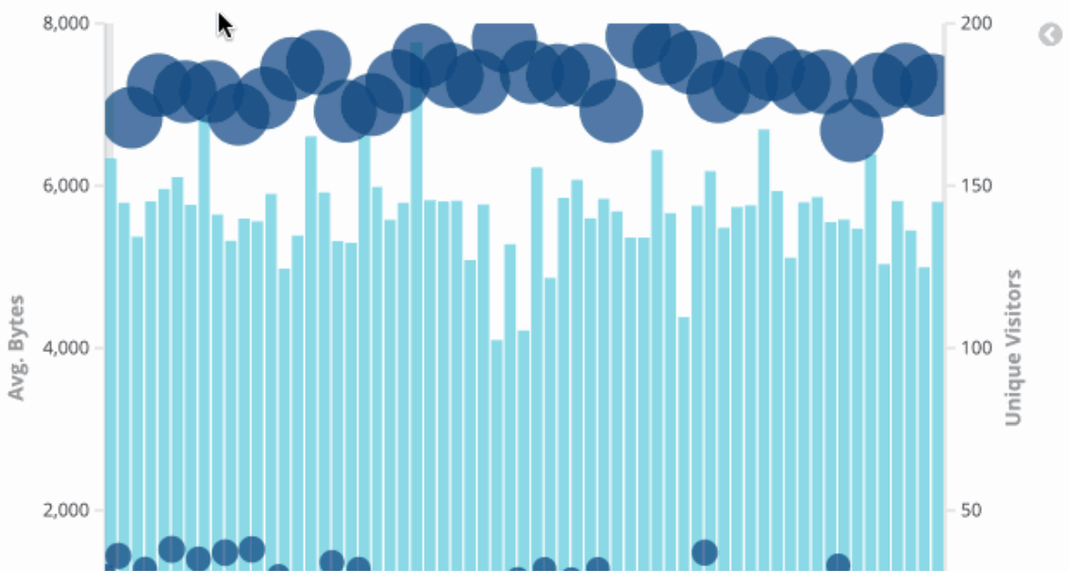


Unique Visitors

[Logs] Unique Visitors by Country



[Logs] Unique Visitors vs. Average Bytes



ELASTIC APM在分布式追踪上的应用

- **elastic APM 深入测试一 （无嵌套调用的分布式微服务监控）**

- **elastic APM 深入测试二 基于spring cloud微服务框架的分布式追踪**

elastic

中文社区

专业、垂直、纯粹的 Elastic 开源技术交流社区

https://elasticsearch.cn/