

应用性能监控 接入指南



腾讯云

【 版权声明 】

©2013–2024 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

接入指南

接入 GO 应用

通过 Skywalking 协议上报

通过 Jaeger 协议上报

通过 Jaeger 原始 SDK 上报

通过 gin Jaeger 中间件上报

通过 goredis 中间件上报

通过 gRPC-Jaeger 拦截器上报

通过 OpenTelemetry 上报应用数据

通过 opentelemetry - grpc-go 拦截器上报

接入 JAVA 应用

通过 OpenTelemetry 增强探针上报

通过 Skywalking 协议上报

通过 TAPM 上报

接入 Python 应用

通过 Jaeger 协议上报

接入 PHP 应用

通过 OpenTelemetry 协议上报

接入 Node.js 应用

通过 Jaeger 原始 SDK 上报

接入指南

接入 GO 应用

通过 Skywalking 协议上报

最近更新时间：2023-07-04 16:17:41

Go2sky 是 Golang 提供给开发者实现 SkyWalking agent 探针的包，可以通过它来实现向 SkyWalking Collector 上报数据。本文将为您介绍如何使用 Skywalking 协议上报 Go 应用数据。

操作流程

步骤1：获取接入点和 Token

进入 [应用性能监控控制台](#) 应用监控 > 应用列表页面，单击接入应用，在接入应用时选择 Go 语言与 Skywalking 的数据采集方式。

在选择接入方式步骤获取您的接入点和 Token。



步骤2：上报应用数据

通过 Skywalking 协议上报 Go 应用数据：

1. 接入埋点。

参见 [Go2Sky 文档](#)，自行对 Go 的跨服务调用埋点。Go 语言应用在使用 Skywalking 上报数据时有一定改造成本，您需要改造少量业务代码以完成接入埋点。

2. 修改上报配置。

将 reporter 的 serverAddr 修改为 APM 的接入点，将 reporter 的 auth 修改为 Token。

3. 重启服务，开始上报数据。

4. 接入验证。

向应用发送请求，在收到响应后，在应用性能监控控制台查看调用数据。您可以在1分钟内通过[链路追踪](#) > [调用查询](#) 查找调用详情。监控曲线与统计数据将在1分钟后开始正常显示。

Go2Sky 改造示例

以下是基于 Go2Sky 的 Demo 改造示例，您可根据实际情况进行修改。

1. 在 NewGRPCReporter 下上报地址和 Authentication（上报地址与 Token 的获取方式参见 [步骤1](#)。

```
report, err = reporter.NewGRPCReporter(
    "ap-guangzhou.tencentservicewatcher.com:11800",
    reporter.WithAuthentication("tsw_site@xxxxxxxxxx"))
```

⚠ 注意

请根据控制台给出的私网接入点和 Token 进行改造。

2. 进行 Server 端配置，Demo 如下：

```
import (
    "flag"
    "github.com/SkyAPM/go2sky"
    v3 "github.com/SkyAPM/go2sky-plugins/gin/v3"
    "github.com/SkyAPM/go2sky/reporter"
    "github.com/gin-gonic/gin"
    "log"
    "net/http"
)
var (
    grpc      bool
    oapServer string
    listenAddr string
    serviceName string
    client *http.Client
)
func init() {
    flag.BoolVar(&grpc, "grpc", false, "use grpc reporter")
    //9.223.77.222:11800 需替换为 APM 的私网接入点
    flag.StringVar(&oapServer, "oap-server", "9.223.77.222:11800", "oap server address")
    flag.StringVar(&listenAddr, "listen-addr", "0.0.0.0:8809", "listen address")
    flag.StringVar(&serviceName, "service-name", "go2sky-server", "service name")
}
```

```

func main() {
    flag.Parse()
    log.Println("reporter.NewGRPCReporter start")
    var report go2sky.Reporter
    var err error
    /*
        参数说明:
        @oapServer:SkyWalking 后端收集器地址
    */
    report, err = reporter.NewGRPCReporter(
        oapServer,
        reporter.WithAuthentication("c944279f910baee6d2e102817270696f"))
    //c944279f910baee6d2e102817270696f 需替换成您的 Token
    //report, err = reporter.NewLogReporter()
    if err != nil {
        log.Fatalf("crate grpc reporter error: %v \n", err)
    }
    /*
        参数说明:
        @service 服务名字, 以 @结尾代表该服务所在 DMP 租户。
        @opts 固定格式, 一个Reporter的实例
    */
    log.Println("go2sky.NewTracer")
    tracer, err := go2sky.NewTracer(serviceName, go2sky.WithReporter(report))
    if err != nil {
        log.Fatalf("crate tracer error: %v \n", err)
    }
    gin.SetMode(gin.ReleaseMode)
    r := gin.New()
    /*
        go2sky的中间件实现路径追踪
        v3 是 github.com/SkyAPM/go2sky-plugins/gin/v3 的缩写
    */
    r.Use(v3.Middleware(r, tracer))
    r.GET("/ping", func(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "hi gin",
        })
    })
    log.Println("0.0.0.0:8809")
    r.Run(listenAddr)
}
    
```


通过 Jaeger 协议上报

通过 Jaeger 原始 SDK 上报

最近更新时间：2023-04-26 15:52:07

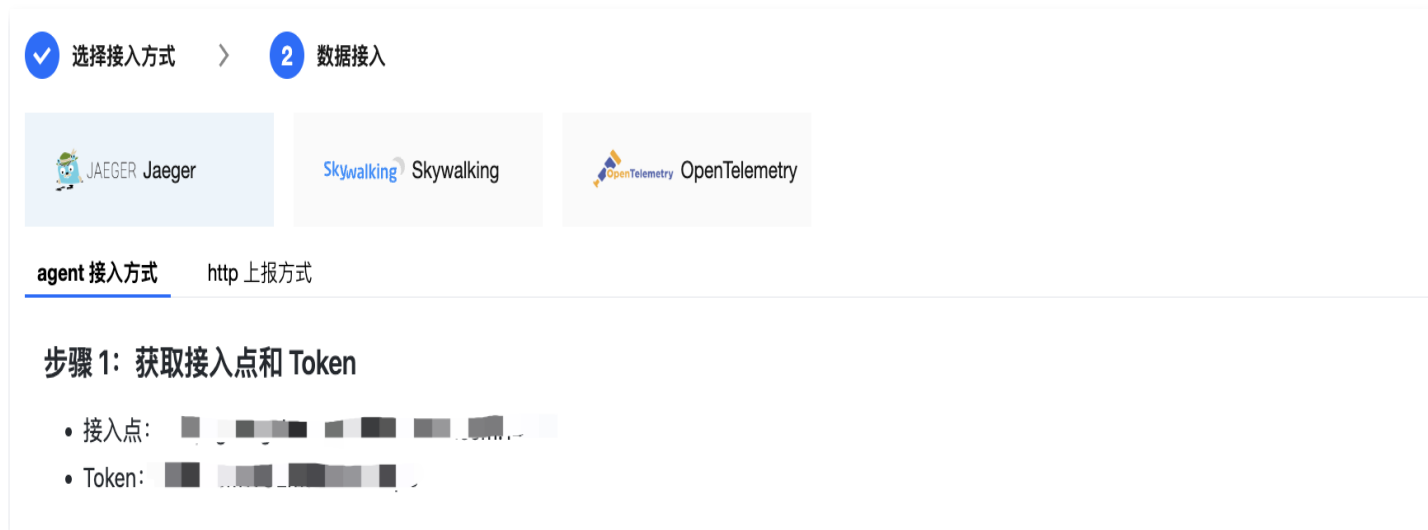
本文将为您介绍如何使用 Jaeger 原始 SDK 上报 Go 应用数据。

操作步骤

步骤1: 获取接入点和 Token

进入 [应用性能监控控制台](#) [应用监控](#) > [应用列表](#) 页面，单击 [接入应用](#)，在接入应用时选择 GO 语言与 Jaeger 原始 SDK 的数据采集方式。

在选择接入方式步骤获取您的接入点和 Token，如下图所示：



步骤2: 安装 Jaeger Agent

1. 下载 [Jaeger Agent](#)。
2. 执行下列命令启动 Agent 。

```
nohup ./jaeger-agent --reporter.grpc.host-port={{接入点}} --agent.tags=token={{token}}
```

❗ 说明

对于 Jaeger Agent v1.15.0及以下版本，请将启动命令中 `--agent.tags` 替换为 `--jaeger.tags`。

步骤3: 上报数据

通过 Jaeger 原始 SDK 上报数据:

1. 客户端侧由于需要模拟 HTTP 请求, 引入 `opentracing-contrib/go-stdlib/nethttp` 依赖

- 依赖路径: `github.com/opentracing-contrib/go-stdlib/nethttp`
- 版本要求: `≥ dv1.0.0`

2. 配置 Jaeger, 创建 Trace 对象。示例如下:

```
cfg := &jaegerConfig.Configuration{
    ServiceName: ginClientName, //对其发起请求的的调用链, 叫什么服务
    Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置, 详情见4.1.1
        Type: "const",
        Param: 1,
    },
    Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息, 所有字段都是可选的
        LogSpans: true,
        LocalAgentHostPort: endPoint,
    },
    //Token配置
    Tags: []opentracing.Tag{ //设置tag, token等信息可存于此
        opentracing.Tag{Key: "token", Value: token}, //设置token
    },
}

tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据配置得到tracer
```

3. 构建 span 并把 span 放入 conext 中, 示例如下:

```
span := tracer.StartSpan("CallDemoServer") //构建span
ctx := opentracing.ContextWithSpan(context.Background(), span) //将span的引用放入conext中
```

4. 构建带 tracer 的 Request 请求, 示例如下:

```
//构建http的请求
req, err := http.NewRequest(
    http.MethodGet,
    fmt.Sprintf("http://localhost%s/ping", ginPort),
```

```
    nil,  
    )  
    req = req.WithContext(ctx)  
    //构建带tracer的请求  
    req, ht := nethttp.TraceRequest(tracer, req)
```

5. 发起 HTTP 请求，并获得返回结果。

```
httpClient := &http.Client{Transport: &nethttp.Transport{}} //初始化http客户端  
res, err := httpClient.Do(req)  
//.省略err判断  
body, err := ioutil.ReadAll(res.Body)  
//.省略err判断  
log.Printf(" %s receive: %s\n", clientServerName, string(body))
```

完整代码如下：

```
// Copyright © 2019-2020 Tencent Co., Ltd.  
  
// This file is part of tencent project.  
// Do not copy, cite, or distribute without the express  
// permission from Cloud Monitor group.  
  
package gindemo  
  
import (  
    "context"  
    "fmt"  
    "github.com/opentracing-contrib/go-stdlib/nethttp"  
    "github.com/opentracing/opentracing-go"  
    "github.com/opentracing/opentracing-go/ext"  
    opentracingLog "github.com/opentracing/opentracing-go/log"  
    "github.com/uber/jaeger-client-go"  
    jaegerConfig "github.com/uber/jaeger-client-go/config"  
    "io/ioutil"  
    "log"  
    "net/http"  
)  
  
const (  
    // 服务名 服务唯一标示，服务指标聚合过滤依据。  
    ginClientName = "demo-gin-client"
```

```
ginPort    = ":8080"
endPoint   = "xxxxx:6831" // 本地agent地址
token     = "abc"
)

// StartClient gin client 也是标准的 http client.
func StartClient() {
    cfg := &jaegerConfig.Configuration{
        ServiceName: ginClientName, //对其发起请求的的调用链，叫什么服务
        Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置，详情见4.1.1
            Type: "const",
            Param: 1,
        },
        Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息，所有字
段都是可选的
            LogSpans:      true,
            LocalAgentHostPort: endPoint,
        },
        //Token配置
        Tags: []opentracing.Tag{ //设置tag， token等信息可存于此
            opentracing.Tag{Key: "token", Value: token}, //设置token
        },
    }

    tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据
配置得到tracer
    defer closer.Close()
    if err != nil {
        panic(fmt.Sprintf("ERROR: fail init Jaeger: %v\n", err))
    }
    //构建span，并将span放入context中
    span := tracer.StartSpan("CallDemoServer")
    ctx := opentracing.ContextWithSpan(context.Background(), span)
    defer span.Finish()

    // 构建http请求
    req, err := http.NewRequest(
        http.MethodGet,
        fmt.Sprintf("http://localhost%s/ping", ginPort),
        nil,
    )
    if err != nil {
        HandlerError(span, err)
        return
    }
}
```

```
// 构建带tracer的请求
req = req.WithContext(ctx)
req, ht := nethttp.TraceRequest(tracer, req)
defer ht.Finish()

// 初始化http客户端
httpClient := &http.Client{Transport: &nethttp.Transport{}}
// 发起请求
res, err := httpClient.Do(req)
if err != nil {
    HandlerError(span, err)
    return
}
defer res.Body.Close()
body, err := ioutil.ReadAll(res.Body)
if err != nil {
    HandlerError(span, err)
    return
}
log.Printf(" %s receive: %s\n", ginClientName, string(body))
}

// HandlerError handle error to span.
func HandlerError(span opentracing.Span, err error) {
    span.SetTag(string(ext.Error), true)
    span.LogKV(opentracingLog.Error(err))
}
```

通过 gin Jaeger 中间件上报

最近更新时间：2023-07-04 16:17:41

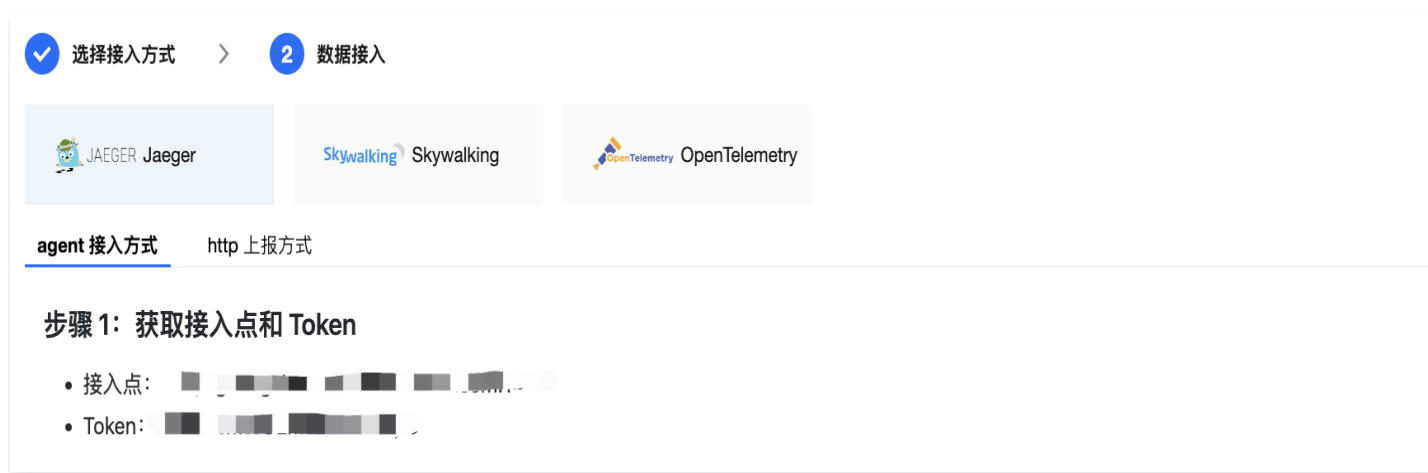
本文将为您介绍如何使用 gin Jaeger 中间件上报 Go 应用数据。

操作步骤

步骤1: 获取接入点和 Token

进入 [应用性能监控控制台](#) [应用监控](#) > [应用列表](#)，单击接入应用页面，在接入应用时选择 GO 语言与 gin Jaeger 中间组件的数据采集方式。

在选择接入方式步骤获取您的接入点和 Token，如下图所示：



步骤2: 安装 Jaeger Agent

1. 下载 [Jaeger Agent](#)。
2. 执行下列命令启动 Agent。

```
nohup ./jaeger-agent --reporter.grpc.host-port={{collectorRPCHostPort}} --  
agent.tags=token={{token}}
```

步骤3: 选择上报端类型上报应用数据

选择上报端类型，通过 gin Jaeger 中间件上报 Go 应用数据：

服务端

1. 在服务端侧引入 `opentracing-contrib/go-gin` 依赖
 - 依赖路径：`github.com/opentracing-contrib/go-gin`

- 版本要求: \geq v0.0.0-20201220185307-1dd2273433a4

2. 配置 Jaeger, 创建 Trace 对象。示例如下:

```
cfg := &jaegerConfig.Configuration{
    ServiceName: ginServerName, //对服务发起请求的调用链, 填写服务名称
    Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置, 详情见4.1.1
        Type: "const",
        Param: 1,
    },
    Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息, 所有字段都是可选的
        LogSpans: true,
        LocalAgentHostPort: endPoint,
    },
    //Token配置
    Tags: []opentracing.Tag{ //设置tag, token等信息可存于此
        opentracing.Tag{Key: "token", Value: token}, //设置token
    },
}

tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据配置得到tracer
```

3. 配置中间件

```
r := gin.Default()
//传入tracer
r.Use(ginhttp.Middleware(tracer))
```

❗ 说明

官方默认 `OperationName` 是 `HTTP + HttpMethod`, 建议使用 `HTTP + HttpMethod + URL` 可以分析到具体接口, 接口主要 URL 应是参数名, 不是具体参数值。具体用法如下:

正确: `/user/{id}`, 错误: `/user/1`

```
r.Use(ginhttp.Middleware(tracer, ginhttp.OperationNameFunc(func(r *http.Request)
string {
    return fmt.Sprintf("testtestheling HTTP %s %s", r.Method, r.URL.String())
})))
```

完整代码如下：

```
// Copyright © 2019-2020 Tencent Co., Ltd.

// This file is part of tencent project.
// Do not copy, cite, or distribute without the express
// permission from Cloud Monitor group.

package gindemo

import (
    "fmt"
    "github.com/gin-gonic/gin"
    "github.com/opentracing-contrib/go-gin/ginhttp"
    "github.com/opentracing/opentracing-go"
    "github.com/uber/jaeger-client-go"
    jaegerConfig "github.com/uber/jaeger-client-go/config"
    "net/http"
)

// 服务名 服务唯一标示，服务指标聚合过滤依据。
const ginServerName = "demo-gin-server"

// StartServer
func StartServer() {
    //初始化jaeger，得到tracer
    cfg := &jaegerConfig.Configuration{
        ServiceName: ginServerName, //对服务发起请求的调用链，填写服务名称
        Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置，详情见4.1.1
            Type: "const",
            Param: 1,
        },
        Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息，所有字
段都是可选的
            LogSpans: true,
            LocalAgentHostPort: endPoint,
        },
        //Token配置
        Tags: []opentracing.Tag{ //设置tag，token等信息可存于此
            opentracing.Tag{Key: "token", Value: token}, //设置token
        },
    }
}
```

```
tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据
配置得到tracer
if err != nil {
    panic(fmt.Sprintf("ERROR: fail init Jaeger: %v\n", err))
}
defer closer.Close()
r := gin.Default()
//这里说明一下, 官方默认 OperationName 是 HTTP + HttpMethod,
//建议使用 HTTP + HttpMethod + URL 可以分析到具体接口, 具体用法如下
//PS: Restful 接口主要URL应该是参数名, 不是具体参数值。如: 正确: /user/{id}, 错
误: /user/1
r.Use(ginhttp.Middleware(tracer, ginhttp.OperationNameFunc(func(r
*http.Request) string {
    return fmt.Sprintf("HTTP %s %s", r.Method, r.URL.String())
})))
r.GET("/ping", func(c *gin.Context) {
    c.JSON(200, gin.H{
        "message": "pong",
    })
})
r.Run() // 监听 0.0.0.0:8080
}
```

客户端

1. 客户端侧由于需要模拟 HTTP 请求, 引入 `opentracing-contrib/go-stdlib/nethttp` 依赖

- 依赖路径: `github.com/opentracing-contrib/go-stdlib/nethttp`
- 版本要求: `≥ v1.0.0`

2. 配置 Jaeger, 创建 Trace 对象。示例如下:

```
cfg := &jaegerConfig.Configuration{
    ServiceName: ginClientName, //对服务发起请求的的调用链, 填写服务名称
    Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置, 详情见4.1.1
        Type: "const",
        Param: 1,
    },
    Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息, 所有字段都
是可选的
        LogSpans: true,
        LocalAgentHostPort: endPoint,
    },
    //Token配置
    Tags: []opentracing.Tag{ //设置tag, token等信息可存于此
```



```
    opentracing.Tag{Key: "token", Value: token}, //设置token
  },
}

tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据配置得到tracer
```

3. 构建 span 并把 span 放入 conext 中，示例如下：

```
span := tracer.StartSpan("CallDemoServer") //构建span
ctx := opentracing.ContextWithSpan(context.Background(), span) //将span的引用放入conext中
```

4. 构建带 tracer 的 Request 请求，示例如下：

```
//构建http的请求
req, err := http.NewRequest(
    http.MethodGet,
    fmt.Sprintf("http://localhost%s/ping", ginPort),
    nil,
)
req = req.WithContext(ctx)
//构建带tracer的请求
req, ht := nethttp.TraceRequest(tracer, req)
```

5. 发起 HTTP 请求，并获得返回结果。

```
httpClient := &http.Client{Transport: &nethttp.Transport{}} //初始化http客户端
res, err := httpClient.Do(req)
//.省略err判断
body, err := ioutil.ReadAll(res.Body)
//.省略err判断
log.Printf(" %s receive: %s\n", clientServerName, string(body))
```

完整代码如下：

```
// Copyright © 2019-2020 Tencent Co., Ltd.
```

```
// This file is part of tencent project.
// Do not copy, cite, or distribute without the express
// permission from Cloud Monitor group.

package gindemo

import (
    "context"
    "fmt"
    "github.com/opentracing-contrib/go-stdlib/nethttp"
    "github.com/opentracing/opentracing-go"
    "github.com/opentracing/opentracing-go/ext"
    opentracingLog "github.com/opentracing/opentracing-go/log"
    "github.com/uber/jaeger-client-go"
    jaegerConfig "github.com/uber/jaeger-client-go/config"
    "io/ioutil"
    "log"
    "net/http"
)

const (
    // 服务名 服务唯一标示, 服务指标聚合过滤依据。
    ginClientName = "demo-gin-client"
    ginPort      = ":8080"
    endPoint     = "xxxxx:6831" // 本地agent地址
    token       = "abc"
)

// StartClient 下的 gin client 也是标准的 http client.
func StartClient() {
    cfg := &jaegerConfig.Configuration{
        ServiceName: ginClientName, //对服务发起请求的调用链, 填写服务名称
        Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置, 详情见4.1.1
            Type: "const",
            Param: 1,
        },
        Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息, 所有字段
            都是可选的
            LogSpans:      true,
            LocalAgentHostPort: endPoint,
        },
        //Token配置
        Tags: []opentracing.Tag{ //设置tag, token等信息可存于此
            opentracing.Tag{Key: "token", Value: token}, //设置token
        },
    }
```

```
}

tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据配置得到tracer
defer closer.Close()
if err != nil {
    panic(fmt.Sprintf("ERROR: fail init Jaeger: %v\n", err))
}
//构建span, 并将span放入context中
span := tracer.StartSpan("CallDemoServer")
ctx := opentracing.ContextWithSpan(context.Background(), span)
defer span.Finish()

// 构建http请求
req, err := http.NewRequest(
    http.MethodGet,
    fmt.Sprintf("http://localhost%s/ping", ginPort),
    nil,
)
if err != nil {
    HandlerError(span, err)
    return
}
// 构建带tracer的请求
req = req.WithContext(ctx)
req, ht := nethttp.TraceRequest(tracer, req)
defer ht.Finish()

// 初始化http客户端
httpClient := &http.Client{Transport: &nethttp.Transport{}}
// 发起请求
res, err := httpClient.Do(req)
if err != nil {
    HandlerError(span, err)
    return
}
defer res.Body.Close()
body, err := ioutil.ReadAll(res.Body)
if err != nil {
    HandlerError(span, err)
    return
}
log.Printf(" %s receive: %s\n", ginClientName, string(body))
}
```

```
// HandlerError handle error to span.  
func HandlerError(span opentracing.Span, err error) {  
    span.SetTag(string(ext.Error), true)  
    span.LogKV(opentracingLog.Error(err))  
}
```

通过 goredis 中间件上报

最近更新时间：2023-04-26 15:52:07

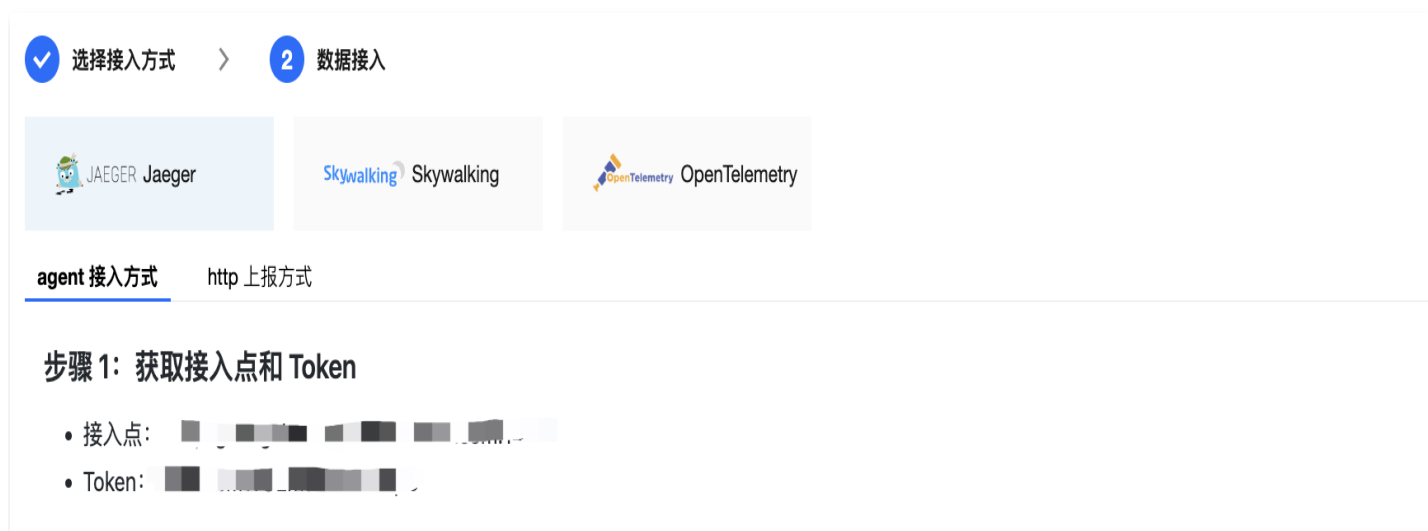
本文将为您介绍如何使用 go redis 中间件上报Go应用数据

操作步骤

步骤1: 获取接入点和 Token

进入 [应用性能监控控制台](#) [应用监控](#) > [应用列表](#) 页面，单击[接入应用](#)，在接入应用时选择 GO 语言与 goredis 中间件的数据采集方式。

在选择接入方式步骤获取您的接入点和 Token，如下图所示：



步骤2: 安装 Jaeger Agent

1. 下载 [Jaeger Agent](#)。
2. 执行下列命令启动 Agent 。

```
nohup ./jaeger-agent --reporter.grpc.host-port={{collectorRPCHostPort}} --  
agent.tags=token={{token}}
```

步骤3: 选择上报端类型上报应用数据

选择上报端类型，通过 go redis 中间件上报 Go 应用数据：

客户端

1. 引入 `opentracing-contrib/goredis` 埋点依赖。

- 依赖路径: `github.com/opentracing-contrib/goredis`
- 版本要求: `≥ v0.0.0-20190807091203-90a2649c5f87`

2. 配置 Jaeger，创建Trace对象并设置 GlobalTracer。示例如下：

```

cfg := &jaegerConfig.Configuration{
    ServiceName: clientServerName, //对其发起请求的的调用链，叫什么服务
    Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置，详情见4.1.1
        Type: "const",
        Param: 1,
    },
    Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息，所有字段都是可选的
        LogSpans: true,
        LocalAgentHostPort: endPoint,
    },
    //Token配置
    Tags: []opentracing.Tag{ //设置tag，token等信息可存于此
        opentracing.Tag{ Key: "token", Value: token }, //设置token
    },
}
tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据配置得到tracer
    
```

3. 初始化 Redis 连接，示例如下：

```

func InitRedisConnector() error {
    redisClient = redis.NewUniversalClient(&redis.UniversalOptions{
        Addrs: []string{redisAddress},
        Password: redisPassword,
        DB: 0,
    })
    if err := redisClient.Ping().Err(); err != nil {
        log.Println("redisClient.Ping() error:", err.Error())
        return err
    }
    return nil
}
    
```

4. 获取 Redis 连接，示例如下：

```

func GetRedisDBConnector(ctx context.Context) redis.UniversalClient {
    
```

```
client := apmgoredis.Wrap(redisClient).WithContext(ctx)
return client
}
```

完整代码如下:

```
package main

import (
    "context"
    "fmt"
    "github.com/go-redis/redis"
    apmgoredis "github.com/opentracing-contrib/goredis"
    "github.com/opentracing/opentracing-go"
    "github.com/uber/jaeger-client-go"
    jaegerConfig "github.com/uber/jaeger-client-go/config"
    "log"
    "time"
)

const (
    redisAddress    = "127.0.0.1:6379"
    redisPassword  = ""
    clientServerName = "redis-client-demo"
    testKey         = "redis-demo-key"
    endPoint        = "xxxxx:6831" // HTTP 直接上报地址
    token           = "abc"
)

func main() {
    cfg := &jaegerConfig.Configuration{
        ServiceName: clientServerName, //对其发起请求的的调用链, 叫什么服务
        Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置, 详情见4.1.1
            Type: "const",
            Param: 1,
        },
        Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息, 所有字
            段都是可选的
            LogSpans:      true,
            LocalAgentHostPort: endPoint,
        },
        //Token配置
        Tags: []opentracing.Tag{ //设置tag, token等信息可存于此
            opentracing.Tag{Key: "token", Value: token}, //设置token
        },
    }
}
```

```
    },
  }
  tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据
配置得到tracer
  opentracing.SetGlobalTracer(tracer)
  defer closer.Close()
  if err != nil {
    panic(fmt.Sprintf("ERROR: fail init Jaeger: %v\n", err))
  }
  InitRedisConnector()
  redisClient := GetRedisDBConnector(context.Background())
  redisClient.Set(testKey, "redis-client-demo", time.Duration(1000)*time.Second)
  redisClient.Get(testKey)
}

var (
  redisClient redis.UniversalClient
)

func GetRedisDBConnector(ctx context.Context) redis.UniversalClient {
  client := apmgoredis.Wrap(redisClient).WithContext(ctx)
  return client
}

func InitRedisConnector() error {
  redisClient = redis.NewUniversalClient(&redis.UniversalOptions{
    Addrs: []string{redisAddress},
    Password: redisPassword,
    DB: 0,
  })
  if err := redisClient.Ping().Err(); err != nil {
    log.Println("redisClient.Ping() error:", err.Error())
    return err
  }
  return nil
}
```


通过 gRPC-Jaeger 拦截器上报

最近更新时间：2023-04-26 15:52:08

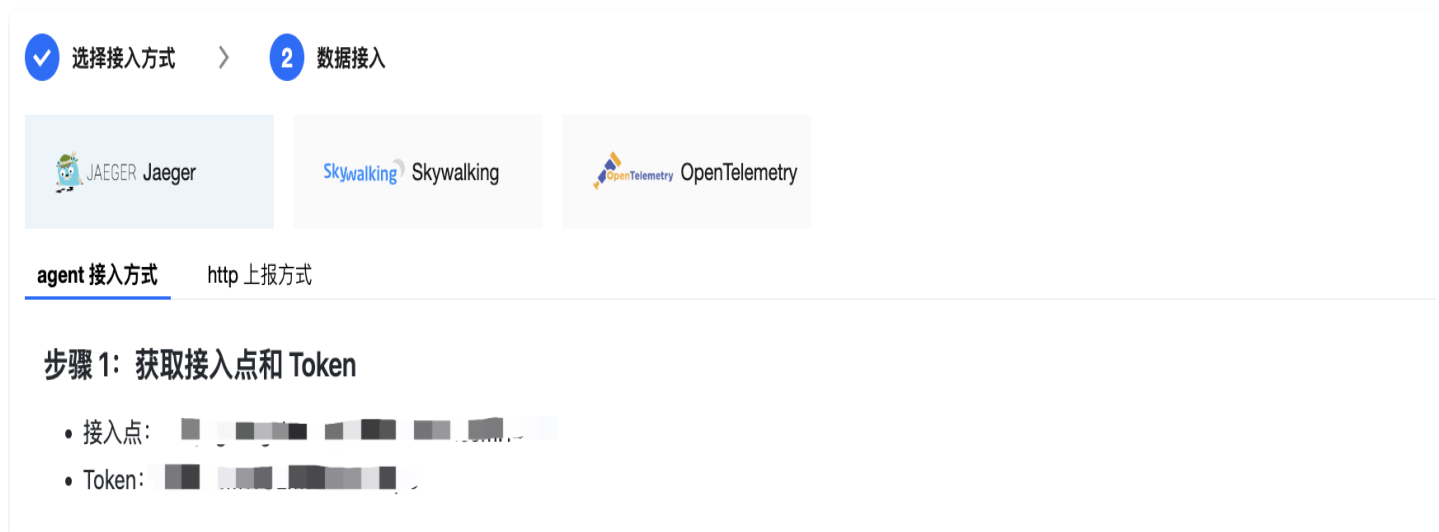
本文将为您介绍如何使用 gRPC-Jaeger 拦截器上报 Go 应用数据。

操作步骤

步骤1: 获取接入点和 Token

进入 [应用性能监控控制台](#) [应用监控](#) > [应用列表](#) 页面，单击 [接入应用](#)，在接入应用时选择 GO 语言与 gRPC-Jaeger 的数据采集方式。

在选择接入方式步骤获取您的接入点和 Token，如下图所示：



步骤2: 安装 Jaeger Agent

1. 下载 [Jaeger Agent](#)。
2. 执行下列命令启动 Agent。

```
nohup ./jaeger-agent --reporter.grpc.host-port={{collectorRPCHostPort}} --  
agent.tags=token={{token}}
```

步骤3: 选择上报端类型上报应用数据

选择上报端类型，通过 gRPC-Jaeger 拦截器上报 Go 应用数据：

服务端

1. 在服务端侧引入 `opentracing-contrib/go-grpc` 埋点依赖。

- 依赖路径: `github.com/opentracing-contrib/go-grpc`
- 版本要求: `≥ v0.0.0-20210225150812-73cb765af46e`

2. 配置 Jaeger, 创建 Trace 对象。示例如下:

```
cfg := &jaegerConfig.Configuration{
    ServiceName: grpcServerName, //对其发起请求的的调用链, 叫什么服务
    Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置, 详情见4.1.1
        Type: "const",
        Param: 1,
    },
    Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息, 所有字段都是可选的
        LogSpans: true,
        LocalAgentHostPort: endPoint,
    },
    //Token配置
    Tags: []opentracing.Tag{ //设置tag, token等信息可存于此
        opentracing.Tag{ Key: "token", Value: token }, //设置token
    },
}
tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据配置得到tracer
```

3. 配置拦截器。

```
s :=
grpc.NewServer(grpc.UnaryInterceptor(otgrpc.OpenTracingServerInterceptor(tracer
)))
```

4. 启动 Server 服务。

```
// 在gRPC服务器处注册我们的服务
pb.RegisterHelloTraceServer(s, &server{})
if err := s.Serve(lis); err != nil {
    log.Fatalf("failed to serve: %v", err)
}
```

完整代码如下:

```
// Copyright © 2019-2020 Tencent Co., Ltd.

// This file is part of tencent project.
// Do not copy, cite, or distribute without the express
// permission from Cloud Monitor group.

package grpcdemo

import (
    "context"
    "fmt"
    "github.com/opentracing/opentracing-go"
    "github.com/uber/jaeger-client-go"
    jaegerConfig "github.com/uber/jaeger-client-go/config"
    "log"
    "net"

    "github.com/opentracing-contrib/go-grpc"
    "google.golang.org/grpc"
)

const (
    // 服务名 服务唯一标示，服务指标聚合过滤依据。
    grpcServerName = "demo-grpc-server"
    serverPort     = ":9090"
)

// server is used to implement proto.HelloTraceServer.
type server struct {
    UnimplementedHelloTraceServer
}

// SayHello implements proto.HelloTraceServer
func (s *server) SayHello(ctx context.Context, in *TraceRequest) (*TraceResponse,
error) {
    log.Printf("Received: %v", in.GetName())
    return &TraceResponse{Message: "Hello " + in.GetName()}, nil
}

// StartServer
func StartServer() {
    cfg := &jaegerConfig.Configuration{
        ServiceName: grpcServerName, //对其发起请求的的调用链，叫什么服务
        Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置，详情见4.1.1
            Type: "const",
```

```
    Param: 1,
  },
  Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息，所有字
段都是可选的
    LogSpans:      true,
    LocalAgentHostPort: endPoint,
  },
  //Token配置
  Tags: []opentracing.Tag{ //设置tag， token等信息可存于此
    opentracing.Tag{Key: "token", Value: token}, //设置token
  },
}
tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据
配置得到tracer
defer closer.Close()
if err != nil {
    panic(fmt.Sprintf("ERROR: fail init Jaeger: %v\n", err))
}
lis, err := net.Listen("tcp", serverPort)
if err != nil {
    log.Fatalf("failed to listen: %v", err)
}
s :=
grpc.NewServer(grpc.UnaryInterceptor(otgrpc.OpenTracingServerInterceptor(tracer
)))

// 在gRPC服务器处注册我们的服务
RegisterHelloTraceServer(s, &server{ })
if err := s.Serve(lis); err != nil {
    log.Fatalf("failed to serve: %v", err)
}
}
```

客户端

1. 客户端侧由于需要模拟 HTTP 请求，引入 `opentracing-contrib/go-stdlib/nethttp` 依赖
 - 依赖路径: `github.com/opentracing-contrib/go-stdlib/nethttp`
 - 版本要求: `≥ v1.0.0`
2. 配置 Jaeger，创建 Trace 对象。

```
cfg := &jaegerConfig.Configuration{
  ServiceName: grpcClientName, //对其发起请求的的调用链，叫什么服务
```

```
Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置，详情见4.1.1
    Type: "const",
    Param: 1,
},
Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息，所有字段都是可选的
    LogSpans: true,
    LocalAgentHostPort: endPoint,
},
//Token配置
Tags: []opentracing.Tag{ //设置tag，token等信息可存于此
    opentracing.Tag{Key: "token", Value: token}, //设置token
},
}
tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据配置得到tracer
```

3. 建立连接，配置拦截器。

```
// 向服务端建立连接，配置拦截器
conn, err := grpc.Dial(serverAddress, grpc.WithInsecure(), grpc.WithBlock(),
    grpc.WithUnaryInterceptor(otgrpc.OpenTracingClientInterceptor(tracer)))
```

4. 进行 gRPC 调用，验证是否接入成功。

完整代码如下：

```
// Copyright © 2019-2020 Tencent Co., Ltd.

// This file is part of tencent project.
// Do not copy, cite, or distribute without the express
// permission from Cloud Monitor group.

package grpcdemo

import (
    "context"
    "fmt"
    "github.com/opentracing-contrib/go-grpc"
    "github.com/opentracing/opentracing-go"
    "github.com/uber/jaeger-client-go"
    jaegerConfig "github.com/uber/jaeger-client-go/config"
    "google.golang.org/grpc"
)
```

```
"log"
"time"
)

const (
    // 服务名 服务唯一标示, 服务指标聚合过滤依据。
    grpcClientName = "demo-grpc-client"
    defaultName    = "TAW Tracing"
    serverAddress  = "localhost:9090"
    endPoint      = "xxxxx:6831" // 本地agent地址
    token         = "abc"
)

// StartClient
func StartClient() {
    cfg := &jaegerConfig.Configuration{
        ServiceName: grpcClientName, //对其发起请求的的调用链, 叫什么服务
        Sampler: &jaegerConfig.SamplerConfig{ //采样策略的配置, 详情见4.1.1
            Type: "const",
            Param: 1,
        },
        Reporter: &jaegerConfig.ReporterConfig{ //配置客户端如何上报trace信息, 所有字
            段都是可选的
            LogSpans:      true,
            LocalAgentHostPort: endPoint,
        },
        //Token配置
        Tags: []opentracing.Tag{ //设置tag, token等信息可存于此
            opentracing.Tag{Key: "token", Value: token}, //设置token
        },
    }
    tracer, closer, err := cfg.NewTracer(jaegerConfig.Logger(jaeger.StdLogger)) //根据
    配置得到tracer
    defer closer.Close()
    if err != nil {
        panic(fmt.Sprintf("ERROR: fail init Jaeger: %v\n", err))
    }
    // 向服务端建立链接, 配置拦截器
    conn, err := grpc.Dial(serverAddress, grpc.WithInsecure(), grpc.WithBlock(),
        grpc.WithUnaryInterceptor(otgrpc.OpenTracingClientInterceptor(tracer)))
    if err != nil {
        log.Fatalf("did not connect: %v", err)
    }
    defer conn.Close()
}
```

```
//
c := NewHelloTraceClient(conn)
ctx, cancel := context.WithTimeout(context.Background(), time.Second)
defer cancel()
// 发起RPC调用
r, err := c.SayHello(ctx, &TraceRequest{Name: defaultName})
if err != nil {
    log.Fatalf("could not greet: %v", err)
}
log.Printf("RPC Client receive: %s", r.GetMessage())
}
```

通过 OpenTelemetry 上报应用数据

最近更新时间：2023-11-22 10:06:03

OpenTelemetry 是工具、API 和 SDK 的集合。使用它来检测、生成、收集和导出遥测数据（指标、日志和跟踪），以帮助您分析软件的性能和行为。本文将介绍如何使用 OpenTelemetry 上报 Go 应用数据。

步骤一：获取接入点和 Token

进入 [应用性能监控控制台](#) [应用监控](#) > [应用列表](#) 页面，单击 [接入应用](#)，在接入应用时选择 Go 语言与 OpenTelemetry -agent 的数据采集方式。

在选择接入方式步骤获取您的接入点和 Token，如下图所示：

选择接入方式 > 2 数据接入

JAEGER Jaeger Skywalking Skywalking OpenTelemetry OpenTelemetry

grpc-go 拦截器上报方式 agent 接入方式

步骤 1: 获取接入点和 Token

- 接入点: [Redacted]
- Token: [Redacted]

ⓘ 上报方式说明

- 内网上报：使用此上报方式，您的服务需运行在腾讯云 VPC。通过 VPC 直接联通，在避免外网通信的安全风险同时，可以节省上报流量开销。
- 外网上报：当您的服务部署在本地或非腾讯云 VPC 内，可以通过此方式上报数据。请注意外网通信存在安全风险，同时也会造成一定上报流量费用。

步骤二：上报 Go 应用数据

1. 引入 opentelemetry-sdk 依赖，进行 sdk 埋点上报，首先对 trace 进行构造：

```
import (
    "context"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/attribute"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc"
    "go.opentelemetry.io/otel/propagation"
    "go.opentelemetry.io/otel/sdk/resource"
    sdktrace "go.opentelemetry.io/otel/sdk/trace"
    "log"
    _ "os"
)
// Init configures an OpenTelemetry exporter and trace provider
func Init(ctx context.Context) *sdktrace.TracerProvider {
    //New otlp exporter
    opts := []otlptracegrpc.Option{
        // 配置上报地址，如 config.yaml 里已配置，此处可忽略
        otlptracegrpc.WithEndpoint("{接入点信息}"), otlptracegrpc.WithInsecure(),
    }
    exporter, err := otlptracegrpc.New(ctx,opts...)
    if err != nil {
        log.Fatal(err)
    }
    //在 Resource 下设置上报 Token，也可以直接配置环境变量来设置 token:
    OTEL_RESOURCE_ATTRIBUTES=token=xxxxxxx 如 config.yaml 里已配置，此处可忽略
    r,err := resource.New(ctx,[]resource.Option{
        resource.WithAttributes(attribute.KeyValue{Key: "token",Value:
attribute.StringValue("{上报token}"),}),
    }...)

    if err != nil{
        log.Fatal(err)
    }
    //创建一个新的TracerProvider
    tp := sdktrace.NewTracerProvider(
        sdktrace.WithSampler(sdktrace.AlwaysSample()),
        sdktrace.WithBatcher(exporter),
        sdktrace.WithResource(r),
    )
    otel.SetTracerProvider(tp)
```

```
otel.SetTextMapPropagator(propagation.NewCompositeTextMapPropagator(propagation.TraceContext{}, propagation.Baggage{}))
return tp
}
```

2. 以 grpc 为例，服务端代码中，首先进行 trace 的初始化，并在创建服务时设置拦截器，再根据自身的服务编写服务代码即可。

```
func main() {

tp := trace.Init() //初始化工作， Init方法即为上述的构造方法
defer func() {
    if err := tp.Shutdown(context.Background()); err != nil {
        log.Printf("Error shutting down tracer provider: %v", err)
    }
}()

host := os.Getenv("grpc1")

lis, err := net.Listen("tcp", host+":7778")
if err != nil {
    log.Fatalf("failed to listen: %v", err)
}

s := grpc.NewServer(
    grpc.UnaryInterceptor(otelgrpc.UnaryServerInterceptor()), //设置拦截器进行埋点
    grpc.StreamInterceptor(otelgrpc.StreamServerInterceptor()),
)

api.RegisterHelloServiceServer(s, &server{}) //注册服务，具体服务代码可自行更改
reflection.Register(s)
if err := s.Serve(lis); err != nil {
    log.Fatalf("failed to serve: %v", err)
}
}
```

3. 以 grpc 为例，客户端代码中，依然首先进行 trace 的初始化，并建立链接。

```
func main() {
tp := trace.Init() //初始化
fmt.Println("tp create success")
defer func() {
    if err := tp.Shutdown(context.Background()); err != nil {
        log.Printf("Error shutting down tracer provider: %v", err)
    }
}()
}
```

```

    }
  }()
  fmt.Println("aaa")
  conn, err := grpc.DialContext(context.Background(), "localhost:7778",
  grpc.WithTransportCredentials(insecure.NewCredentials()), grpc.WithBlock()) //建立
  连接
  fmt.Println("pro")
  if err != nil {
    log.Fatalf("did not connect: %v", err)
  }
  defer conn.Close()
  fmt.Println("conn")

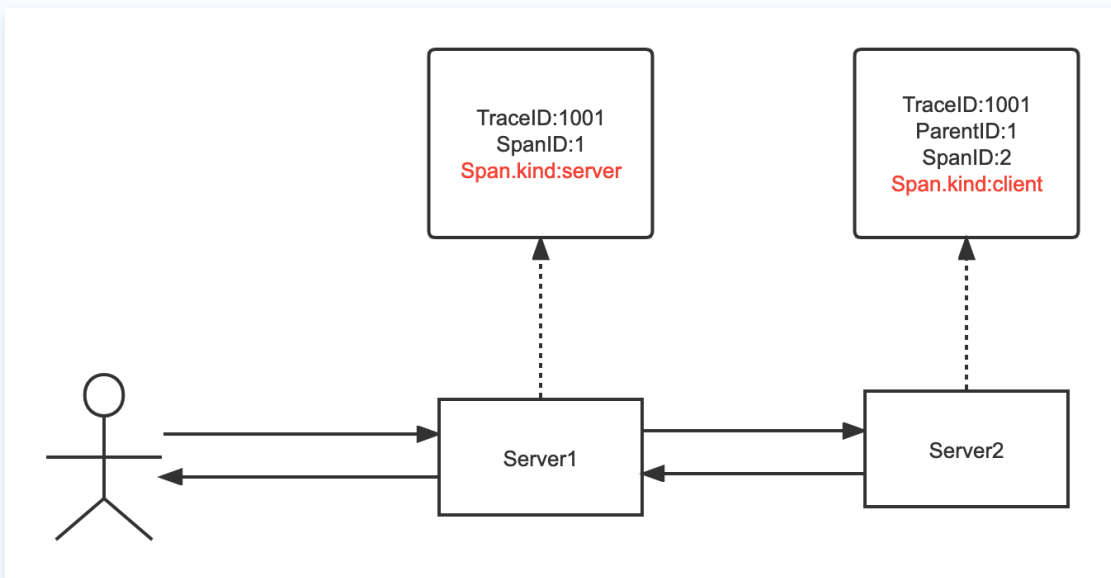
  c := api.NewHelloServiceClient(conn) //客户端代码

  for {
    callSayHelloClientStream(c)
    time.Sleep(100 * time.Millisecond)
  }
}

```

说明

若需要上报服务端数据，则需要指明字段 `span.kind = server`



步骤三：启动您的应用

查看应用数据

登录 [应用性能监控控制台](#)，在应用列表中即可查看性能数据。

通过 opentelemetry – grpc-go 拦截器上报

最近更新时间：2023-11-22 10:06:03

本文将为您介绍如何通过 opentelemetry – grpc-go 上报 Go 应用数据。

操作步骤

步骤1: 获取接入点和 Token

进入应用监控 > 应用列表，单击接入应用页面，在接入应用时选择 GO 语言与 opentelemetry – grpc-go 的数据采集方式。

在选择接入方式步骤获取您的接入点和 Token，如下图所示：



步骤2: 修改接入点信息

说明

我们采用 grpc-go 拦截器方式上报数据，需要修改接入点信息为：ap-guangzhou.apm.tencentcs.com:14268/api/traces，可以直接将数据上报到 collector，无需使用 agent。

步骤3: 引入依赖

需要引入 opentelemetry 的 SDK 埋点依赖。

- 依赖路径: "go.opentelemetry.io/otel/sdk/trace"

步骤4: trace 初始化

```
// Init配置OpenTelemetry
func Init() *sdktrace.TracerProvider {
    if ctx == nil {
        ctx = context.Background()
    }
    //创建新的exporter, 设置基本的endpoint
    opts := []otlptracegrpc.Option{
        otlptracegrpc.WithEndpoint("<接入点>"),
        otlptracegrpc.WithInsecure(),
    }
    exporter, err := otlptracegrpc.New(ctx, opts...)
    if err != nil {
        log.Fatal(err)
    }

    //设置Token, 也可以设置环境变量: OTEL_RESOURCE_ATTRIBUTES=token=xxxxxxxxx
    r, err := resource.New(ctx, []resource.Option{
        //设置Token值
        resource.WithAttributes(attribute.KeyValue{
            Key: "token", Value: attribute.StringValue("<Token>"),
        }),
        //设置服务名
        resource.WithAttributes(attribute.KeyValue{
            Key: "service.name", Value: attribute.StringValue("audotanggrpcdemo"),
        }),
    }...)
    if err != nil {
        log.Fatal(err)
    }

    //创建新的TracerProvider
    tp := sdktrace.NewTracerProvider(
        sdktrace.WithSampler(sdktrace.AlwaysSample()),
        sdktrace.WithBatcher(exporter),
        sdktrace.WithResource(r),
    )
    otel.SetTracerProvider(tp)
```

```
otel.SetTextMapPropagator(propagation.NewCompositeTextMapPropagator(propagation.TraceContext{}, propagation.Baggage{}))
return tp
}
```

步骤5: 选择上报端类型上报应用数据

服务端

1. 初始化 TracerProvider。

```
//初始化trace
tp := trace.Init()
defer func() {
    if err := tp.Shutdown(context.Background()); err != nil {
        log.Printf("Error shutting down tracer provider: %v", err)
    }
}()
//指定host
host := os.Getenv("grpc1")

lis, err := net.Listen("tcp", host+":7778")
if err != nil {
    log.Fatalf("failed to listen: %v", err)
}
```

2. 配置拦截器

```
s := grpc.NewServer(
    grpc.UnaryInterceptor(otelgrpc.UnaryServerInterceptor()),
    grpc.StreamInterceptor(otelgrpc.StreamServerInterceptor()),
)
```

3. 启动 server 服务

```
//在gRPC服务器处注册我们的服务
api.RegisterHelloServiceServer(s, &server{})
reflection.Register(s)
if err := s.Serve(lis); err != nil {
    log.Fatalf("failed to serve: %v", err)
}
```

客户端配置

1. 初始化 TracerProvider。

```
//初始化trace
tp := trace.Init()
defer func() {
    if err := tp.Shutdown(context.Background()); err != nil {
        log.Printf("Error shutting down tracer provider: %v", err)
    }
}()
}()
```

2. 建立连接，配置拦截器。

```
//向服务端建立连接，配置拦截器
conn, err := grpc.DialContext(context.Background(), "localhost:7778",
    grpc.WithTransportCredentials(insecure.NewCredentials()), grpc.WithBlock())
```

3. 进行 GRPC 调用。

```
c := api.NewHelloServiceClient(conn)
for {
    callSayHelloClientStream(c)
    time.Sleep(100 * time.Millisecond)
}
```

接入 JAVA 应用 通过 OpenTelemetry 增强探针上报

最近更新时间：2023-05-18 17:43:56

Java Agent 基于字节码增强技术研发，支持自动埋点完成数据上报，Java Agent 包含(并二次分发) opentelemetry-java-instrumentation CNCF 的开源代码，遵循 Apache License 2.0 协议，在 Java Agent 包中对 opentelemetry License 进行了引用。

说明

OpenTelemetry 是工具、API 和 SDK 的集合。使用它来检测、生成、收集和导出遥测数据（指标、日志和跟踪），以帮助您分析软件的性能和行为。OpenTelemetry 社区活跃，技术更迭迅速，广泛兼容主流编程语言、组件与框架，为云原生微服务以及容器架构的链路追踪能力广受欢迎。通过对 Java 字节码的增强技术 OpenTelemetry-java-instrumentation 可以实现自动埋点上报数据，且腾讯云 APM 基于 OpenTelemetry-java-instrumentation 进行二次开发，可以让您拿到更完善的调用链数据及其对应的行号信息。

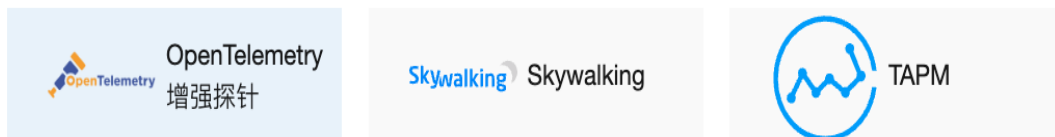
本文将通过相关操作介绍如何在腾讯云 APM 使用 OpenTelemetry-java-instrumentation 上报 Java 应用数据。

步骤一：获取接入点和 Token

进入 [应用性能监控控制台](#) [应用监控](#) > [应用列表](#) 页面，单击 [接入应用](#)，在接入应用时选择 Java 语言与 OpenTelemetry 的数据采集方式。

在选择接入方式步骤获取您的接入点和 Token，如下图所示：

1 选择接入方式 > 2 数据接入



步骤 1：获取接入点和 Token

- 接入点：*****
- Token：*****

ⓘ 上报方式说明

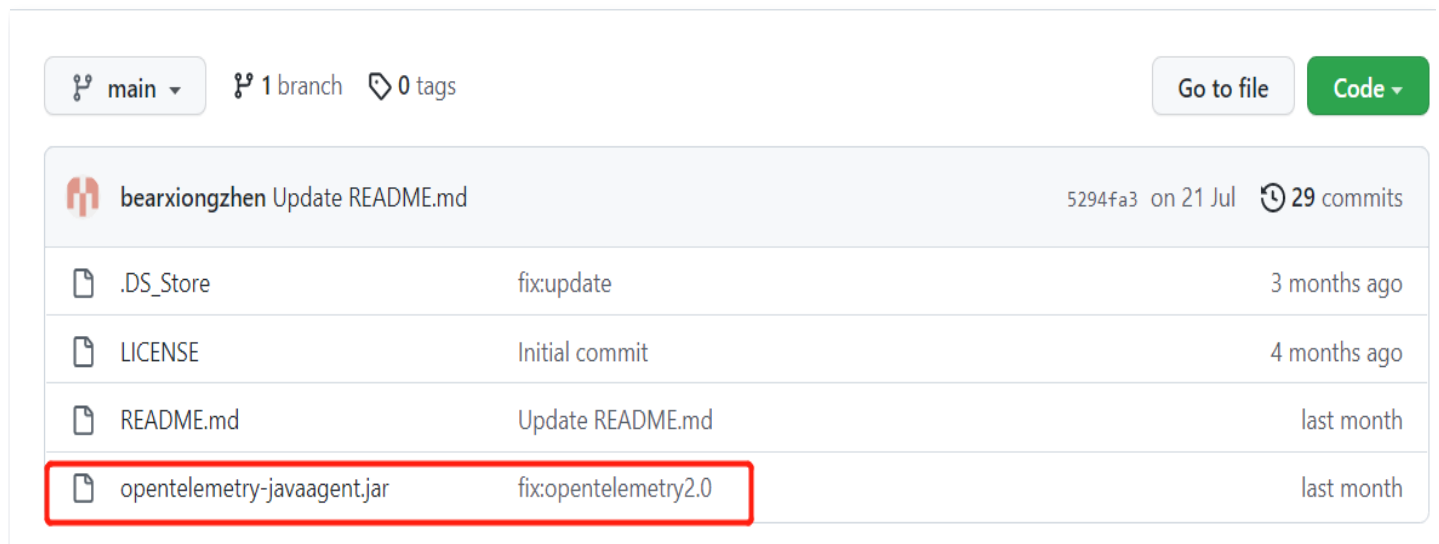
- 内网上报：使用此上报方式，您的服务需运行在腾讯云 VPC 。通过 VPC 直接联通，在避免外网通信的安全风险同时，可以节省上报流量开销。
- 外网上报：当您的服务部署在本地或非腾讯云 VPC 内，可以通过此方式上报数据。请注意外网通信存在安全风险，同时也会造成一定上报流量费用。

步骤二：下载 opentelemetry-javaagent.jar

ⓘ 说明

OpenTelemetry-java-instrumentation 支持数十种框架自动埋点能力。更多信息，请参见 [OpenTelemetry 官方文档](#)。

下载 Java agent: [opentelemetry-javaagent.jar](#)。



main 1 branch 0 tags Go to file Code

bearxiongzhen Update README.md 5294fa3 on 21 Jul 29 commits

.DS_Store	fix:update	3 months ago
LICENSE	Initial commit	4 months ago
README.md	Update README.md	last month
opentelemetry-javaagent.jar	fix:opentelemetry2.0	last month

步骤三：修改上报参数

通过修改 Java 启动的 VM 参数上报链路数据。

```
-javaagent:/path/to/opentelemetry-javaagent.jar //请将路径修改为您文件下载的实际地址。
-Dotel.resource.attributes=service.name=<appName>,token=<token>
//service.name: 服务名,如果是spring cloud/dubbo服务,最好与其服务名保持一致
-Dotel.exporter.otlp.endpoint=<接入点>
```

说明

- 如果您选择直接上报数据，请将< token >替换成从前提条件中获取的 Token，将<接入点>替换成对应地域的接入点。替换对应参数值时，“<>”符号需删去，仅保留文本。
- 如果您选择使用 OpenTelemetry Collector 转发，则需删除-Dotel.exporter.otlp.headers=Authentication=< token >并修改<接入点>为您本地部署的服务地址。

步骤四：启动您的应用

查看应用数据

登录 [应用性能监控控制台](#)，在应用列表中即可查看性能数据。

通过 Skywalking 协议上报

最近更新时间：2023-04-26 15:52:08

本文将为您介绍如何使用 Skywalking 协议上报 Java 应用数据。

前提条件

- 打开 [SkyWalking](#) 下载页面，下载 SkyWalking8.5.0 以上的（包含8.5.0）版本，并将解压后的 Agent 文件夹放至 Java 进程有访问权限的目录。
- 插件均放置在/plugins 目录中。在启动阶段将新的插件放进该目录，即可令插件生效。将插件从该目录删除，即可令其失效。另外，日志文件默认输出到/logs 目录中。

接入步骤

步骤1：获取接入点和 Token

进入 [应用性能监控控制台](#) 应用监控 > 应用列表页面，单击接入应用，在接入应用时选择 Java 语言与 SkyWalking 的数据采集方式。

在选择接入方式步骤获取您的接入点和 Token，如下图所示：



选择接入方式 > 2 数据接入

OpenTelemetry 增强探针 | Skywalking Skywalking | TAPM

步骤 1：获取接入点和 Token

- 接入点：[Redacted]
- Token：[Redacted]

步骤2：下载 Skywalking

- 若您已经使用了 SkyWalking，可跳过本步骤。

- 若您还未使用 SkyWalking，建议 [下载最新版本](#)，下载方式参见 [前提条件](#)。

步骤3：配置相应参数及名称

Skywalking Agent 支持多种方式完成参数配置，不同配置方式之间可以彼此组合，下面给出配置方式及其示例。

方法1：使用 agent.config 文件配置

打开 agent/config/agent.config 文件，配置接入点、Token 和自定义服务名称。

```
collector.backend_service=<接入点>
agent.authentication=<Token>
agent.service_name=<上报的服务名称>
```

❗ 说明

修改完 agent.config 需要把配置项前反注释符号 # 去掉。否则更改的信息将无法生效。

方法2：Java VM Options

启动 Java 应用程序时，在命令行中添加相应以 -DSkywalking 开头的参数,以下给出方法一的等效范例。

```
java -javaagent:<步骤一目录绝对路径>/skywalking-agent.jar
-Dskywalking.collector.backend_service=<接入点>
-Dskywalking.agent.authentication=<Token>
-Dskywalking.agent.service_name=<上报的服务名称> 要启动的程序
```

方法3：设置相应环境变量

可以在系统中设置相应环境变量来完成 skywalking 客户端的配置，以下为 Linux 命令的示例。

```
export SW_AGENT_AUTHENTICATION=<Token> (等效于agent.authentication=Token)
export SW_AGENT_NAME=<上报的服务名称> (等效于agent.service_name=<上报的服务名称>)
export SW_AGENT_COLLECTOR_BACKEND_SERVICES=<接入点>(等效于
collector.backend_service=接入点)
```

❗ 说明

以上三种方式读取优先级关系为：服务器配置->环境变量->配置文件。优先级高的配置会将优先级的低的配置覆盖。

步骤4：选择相应方法指定插件路径

根据应用的运行环境，选择相应的方法来指定 SkyWalking Agent 的路径。

- Linux Tomcat 7/Tomcat 8

在 `tomcat/bin/catalina.sh` 第一行添加以下内容：

```
CATALINA_OPTS="$CATALINA_OPTS -javaagent:<skywalking-agent-path>"; export CATALINA_OPTS
```

- Jetty

在 `{JETTY_HOME}/start.ini` 配置文件中添加以下内容：

```
--exec # 去掉前面的井号取消注释。  
-javaagent:<步骤一目录绝对路径>
```

- JAR File 或 Spring Boot

在应用程序的启动命令行中添加 `-javaagent` 参数（`-javaagent` 参数一定要放在 `-jar` 参数之前），参数内容如下：

```
java -javaagent:<skywalking-agent-path> -jar yourApp.jar
```

- IDEA

在 IDEA 中运行时，可在 Configuration 中配置应用程序的 VM option，添加 `-javaagent` 参数，参数配置如上一致。

步骤5：重新启动应用

完成上述部署步骤后，参见 [Skywalking 官网指导](#) 重新启动应用即可。

通过 TAPM 上报

最近更新时间：2023-04-26 15:52:08

本文将介绍以下两种方式自动安装自研探针：

- [通过修改配置文件安装](#)
- [通过添加 JVM 参数安装，无需修改配置文件](#)

前提条件

- 在安装探针前，需要先确保本地浏览器时间与服务器时区、时间都一致。若有多个服务器，则要保证本地浏览器、多个服务器的时区、时间都一致。否则，可能会影响数据的准确性，例如拓扑不正确等。
- 下载 [自研SDK](#)。

操作步骤

通过修改配置文件安装

Linux/Mac

1. 执行以下命令，解压 Agent 安装文件包到您的应用服务器的根目录。示例如下：

```
unzip tapm-agent-java-x.x.x.zip -d /path/to/appserver/
```

ⓘ 说明

“ /path/to/appserver ” 为示例路径，请用户根据自身不同的环境修改正确的目录。
例如：应用服务器的根目录为： /path/to/tomcat，则解压后tapm所处目录为：
/path/to/tomcat。

2. 修改解压 tapm 目录下的 tapm.properties 文件。

需修改文件中的 license_key、app_name 和 collector.addresses 两个配置项，否则探针无法进行数据采集也无法启动探针。其他配置项，可根据实际需要进行配置。

- **license_key**：填写在 [应用性能监控控制台](#) 接入应用时获取的 token。与您的应用性能监控账号关联。探针采集到的数据，会上传到该 LicenseKey 绑定的账号下。
- **app_name**：自定义应用名称，建议配置为应用的业务名称。
- **collector.addresses**：填写在 [应用性能监控控制台](#) 接入应用时获取的接入点。例如 tapm.ap-guangzhou.api.tencentyun.com:80。

3. 在 tapm 目录下执行以下命令自动安装探针。

```
cd /path/to/appserver/tapm
java -jar tapm-agent-java.jar install
```

4. 启动或重启应用服务器。
5. 登录应用性能监控控制台查看性能数据。
重启5分钟后，当您的 Java 应用服务有 HTTP 请求进入，性能数据将发送到应用性能监控系统。

Windows

1. 打开 `tapm-agent-java-x.x.x.zip`。
2. 拷贝 `tapm` 目录到您的应用服务器的根目录。

说明

例如：应用服务器的根目录为： `/path/to/tomcat`，则解压后 `tapm` 所处目录为：
`/path/to/tomcat`。

3. 修改放在服务器解压的 `tapm` 目录下 `tapm.properties` 文件。
修改文件中的 `license_key`、`app_name` 和 `collector.addresses` 配置项，否则探针无法进行数据采集也无法启动探针。对于其他配置项，请根据实际需要进行配置。配置说明如下：
 - **license_key**: 填写在 [应用性能监控控制台](#) 接入应用时获取的 token。与您的应用性能监控账号关联。探针采集到的数据，会上传到该 LicenseKey 绑定的账号下。
 - **app_name**: 自定义应用名称，建议配置为应用的业务名称。
 - **collector.addresses**: 填写在 [应用性能监控控制台](#) 接入应用时获取的接入点。例如 `tapm.ap-guangzhou.api.tencentyun.com:80`。
4. 在命令行窗口执行：（唤起控制台：Windows键+ R，然后输入 `cmd`）。

```
cd tapm
java -jar tapm-agent-java.jar install
```

5. 启动或重启您的应用服务器。
6. 登录应用性能监控控制台查看性能数据。
7. 重启5分钟后，当您的 Java 应用服务有 HTTP 请求进入，性能数据将发送到应用性能监控系统。

说明

如果在几分钟之内，无任何应用性能数据，请确保以下信息是否正确：

- 请按照以上步骤重新查看是否安装正确、目录是否正确、启动脚本是否正确。
- 请检查 `tapm.properties` 中的 `license_key` 是否与您在应用性能监控控制台的 token 一致。

通过添加 JVM 参数安装

当应用采用 `Jar` 方式部署时，可采取以下方式进行探针安装：

1. 解压 `tapm-agent-java-x.x.x.zip`。
2. 在命令行窗口执行：（唤起控制台：Windows键+ R，然后输入cmd）

使用示例：

```
java -javaagent:/path/to/appserver/tapm/tapm-java-agent.jar -Dtapm.app_name={APP_NAME} -Dtapm.collector.addresses={COLLECTOR_ADDRESSES} -Dtapm.license_key={LICENSE_KEY} -jar application.jar
```

说明

其中 `application.jar` 为您所要监控的应用。

仅需执行上述语句，即可开启探针。

配置 `JAVA_OPTS`，需在 `-javaagent` 后加入以下三个参数，中间以空格分隔：

```
-Dtapm.app_name=${APP_NAME}  
-Dtapm.license_key=${LICENSE_KEY}  
-Dtapm.collector.addresses=${COLLECTOR_ADDRESSES}
```

参数说明如下：

- `-Dtapm.app_name`：自定义应用名称，建议配置为应用的业务名称。
- `-Dtapm.license_key`：填写在 [应用性能监控控制台](#) 接入应用时获取的 token。与您的应用性能监控账号关联。探针采集到的数据，会上传到该 LicenseKey 绑定的账号下。
- `-Dtapm.collector.addresses`：填写在 [应用性能监控控制台](#) 接入应用时获取的接入点。例如 `tapm.ap-guangzhou.api.tencentyun.com:80`。Agent Collector 是指服务器的地址和端口号，Agent Collector 在高可用部署模式下，请务必将同一机房内所有的 Agent Collector 服务器地址和端口号都配置进来，以英文逗号分隔。

上述参数为可选参数，当在启动时配置以上参数，将会替换配置文件中相对应的参数。因此，当启动参数未配置上述参数时，系统将会自动获取配置文件(`tapm.properties`)中的参数。

查看监控数据

登录 [应用性能监控控制台](#) 即可查看性能数据。

接入 Python 应用 通过 Jaeger 协议上报

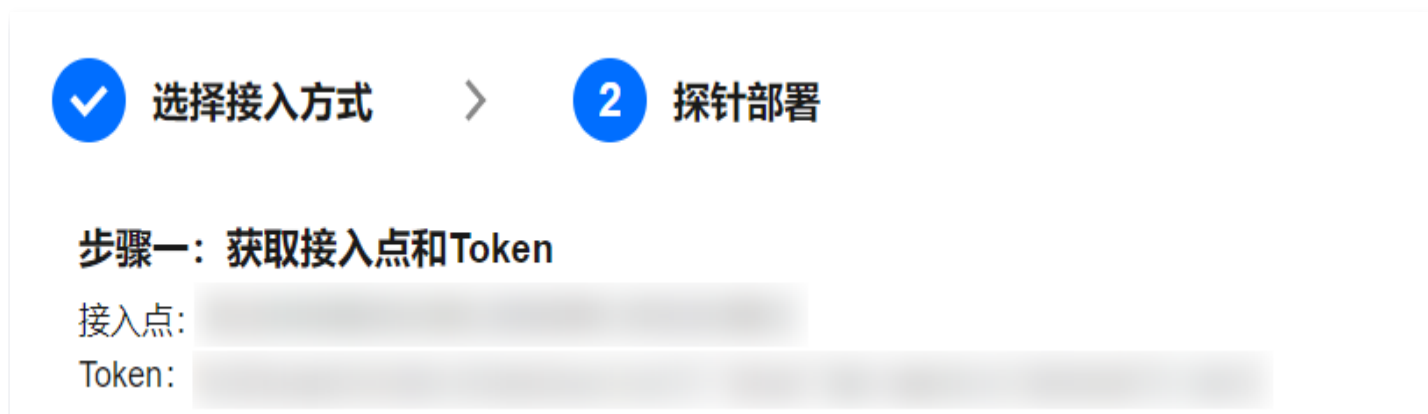
最近更新时间：2023-11-22 10:06:03

本文将为您介绍如何使用 Jaeger 协议上报 Python 应用数据。

操作步骤

步骤1：获取接入点和 Token

进入 [应用性能监控控制台](#) [应用监控](#) > [应用列表](#) 页面，单击 [接入应用](#)，在接入应用时选择 Python 语言与 Jaeger 协议的数据采集方式。在选择接入方式步骤获取您的接入点和 Token，如下图所示：



步骤2：安装 Jaeger Agent

1. 下载官方 [Jaeger Agent](#)。
2. 执行下列命令启动 Agent。

```
nohup ./jaeger-agent --reporter.grpc.host-port={{接入点}} --jaeger.tags=token={{token}}
```

步骤3：通过 Jaeger 上报数据

1. 执行下列命令安装 jaeger_client 包。

```
pip install jaeger_client
```

2. 创建如下 Python 文件和 Tracer 对象，跟踪所有的 Request。

```
from jaeger_client import Config
import time
from os import getenv

# 配置jaeger代理的地址，默认本机localhost
JAEGER_HOST = getenv('JAEGER_HOST', 'localhost')
SERVICE_NAME = getenv('JAEGER_HOST', 'my_service_test')

def build_your_span(tracer):
    with tracer.start_span('yourTestSpan') as span:
        span.log_kv({'event': 'test your message', 'life': 42})
        span.set_tag("span.kind", "server")
    return span

def build_your_tracer():
    my_config = Config(
        config={
            'sampler': {
                'type': 'const',
                'param': 1,
            },
            'local_agent': {
                'reporting_host': JAEGER_HOST,
                'reporting_port': 6831,
            },
            'logging': True,
        },
        service_name=SERVICE_NAME,
        validate=True
    )
    tracer = my_config.initialize_tracer()

    return tracer

if __name__ == "__main__":
    tracer = build_your_tracer()
    span = build_your_span(tracer)
    time.sleep(2)
    tracer.close()
```

! 说明

目前 Jaeger 支持 Flask、Django 和 Grpc 等框架进行上报，更多请参见：

- [jaeger-client-python](#)
- [OpenTracing API Contributions](#)

接入 PHP 应用 通过 OpenTelemetry 协议上报

最近更新时间：2024-03-28 10:32:01

📌 说明：

在接入 PHP 应用场景下，推荐您选择 OpenTelemetry 协议上报。

本文主要介绍使用 OpenTelemetry PHP SDK 为应用手动埋点（即在代码中创建 Span，并为 Span 设置属性、事件、状态等），实现自定义的 PHP 应用链路追踪。

示例 Demo

示例代码 Github 仓库地址：[tencentcloud-opentelemetry-demo-php](https://github.com/tencentcloud-opentelemetry-demo-php)

前提条件

请确保已安装以下软件：

- [PHP 7.4+](#)
- [PECL](#)
- [composer](#)

在开始接入之前，请确保 shell 中有可用的以下两个工具：

```
php -v  
composer -v
```

📌 说明：

OpenTelemetry 的自动埋点能力需要 PHP 8.0+ 版本来实现数据上报，而手动埋点上报能力可以在 PHP 7.4+ 版本实现数据上报。如需更多其他自定义操作，可参考 [OpenTelemetry 官方文档](#)。

操作流程

步骤1：获取接入点和 Token

1. 登录 [应用性能监控](#) 控制台。
2. 在左侧菜单栏中选择应用监控，单击应用列表 > 接入应用。
3. 在右侧弹出的数据接入抽屉框中，单击 PHP 语言。
4. 在接入 PHP 应用页面，选择您所要接入的地域以及业务系统。

5. 选择接入协议类型为 OpenTelemetry。
6. 上报方式选择您所想要的上报方式，获取您的接入点和 Token。

说明：

- 内网上报：使用此上报方式，您的服务需运行在腾讯云 VPC。通过 VPC 直接联通，在避免外网通信的安全风险同时，可以节省上报流量开销。
- 外网上报：当您的服务部署在本地或非腾讯云 VPC 内，可以通过此方式上报数据。请注意外网通信存在安全风险，同时也会造成一定上报流量费用。

步骤2：上报 PHP 应用数据

创建应用

1. 初始化应用

```
mkdir <project-name> && cd <project-name>

composer init \
--no-interaction \
--stability beta \
--require slim/slim:"^4" \
--require slim/psr7:"^1"
composer update
```

2. 编写业务代码

在<project-name>目录下创建一个 index.php 文件，添加如下内容。以下内容将使用一个扔骰子游戏（返回1-6之间的一个随机数）模拟业务代码。

```
<?php
use Psr\Http\Message\ResponseInterface as Response;
use Psr\Http\Message\ServerRequestInterface as Request;
use Slim\Factory\AppFactory;

require __DIR__ . '/vendor/autoload.php';

$app = AppFactory::create();

$app->get('/rolldice', function (Request $request, Response $response) {
    $result = random_int(1,6);
    $response->getBody()->write(strval($result));
    return $response;
});
```

```
});  
  
$app->run();
```

此时应用已经编写完成，执行以下命令即可运行应用：

```
php -S localhost:8080
```

在浏览器中访问以下链接：

```
http://localhost:8080/rolldice
```

构建 OpenTelemetry PHP 扩展（需 PHP 8.0+，仅使用手动埋点可以跳过改步骤）

说明：

如果已经构建过 OpenTelemetry PHP 扩展，或者只使用手动埋点，可跳过当前步骤。

1. 下载构建 OpenTelemetry PHP extension 所需要的工具：

- macOS

```
brew install gcc make autoconf
```

- Linux (apt)

```
sudo apt-get install gcc make autoconf
```

2. 使用 PECL 构建 OpenTelemetry PHP 扩展：

```
pecl install opentelemetry-1.0.0beta6
```

注意，构建成功时输出内容的最后几行如下（路径可能不完全一致）：

```
Build process completed successfully  
Installing '/opt/homebrew/Cellar/php/8.2.8/pecl/2020829/opentelemetry.so'  
install ok: channel://pecl.php.net/opentelemetry-1.0.0beta6  
Extension opentelemetry enabled in php.ini
```

3. 启用 OpenTelemetry PHP 扩展。

说明：

如果上一步输出了 `Extension opentelemetry enabled in php.ini`，表明已经启用，请跳过当前步骤。

在 `php.ini` 文件中添加如下内容：

```
[opentelemetry]
extension=opentelemetry.so
```

`php.ini` 文件可能存在的位置：

OS	PATH
Linux	<code>/etc/php.ini</code> <code>/usr/bin/php5/bin/php.ini</code> <code>/etc/php/php.ini</code> <code>/etc/php5/apache2/php.ini</code>
Mac OSX	<code>/private/etc/php.ini</code>
Windows (with XAMPP installed)	<code>C:/xampp/php/php.ini</code>

4. 验证是否构建并启用成功。

- 方法一：

```
php -m | grep opentelemetry
```

预期输出：

```
opentelemetry
```

- 方法二：

```
php --ri opentelemetry
```

预期输出：

```
opentelemetry
opentelemetry support => enabled
extension version => 1.0.0beta6
```


导入 OpenTelemetry PHP SDK 以及 OpenTelemetry gRPC Explorer 所需依赖

1. 下载 PHP HTTP 客户端库，用于链路数据上报。

```
composer require guzzlehttp/guzzle
```

2. 下载 OpenTelemetry PHP SDK。

```
composer require \
open-telemetry/sdk \
open-telemetry/exporter-otlp
```

3. 下载使用 gRPC 上报数据时所需依赖。

```
pecl install grpc # 如果之前已经下载过grpc，可以跳过这一步
composer require open-telemetry/transport-grpc
```

编写 OpenTelemetry 初始化工具类

在 `index.php` 文件所在目录中创建 `opentelemetry_util.php` 文件。并在文件中添加如下代码：

```
<?php
// 包含设置应用名、Trace导出方式、Trace上报接入点，并创建全局TraceProvide

use OpenTelemetry\API\Globals;
use OpenTelemetry\API\Trace\Propagation\TraceContextPropagator;
use OpenTelemetry\Contrib\Otlp\SpanExporter;
use OpenTelemetry\SDK\Common\Attribute\Attributes;
use OpenTelemetry\SDK\Common\Export\Stream\StreamTransportFactory;
use OpenTelemetry\SDK\Resource\ResourceInfo;
use OpenTelemetry\SDK\Resource\ResourceInfoFactory;
use OpenTelemetry\SDK\Sdk;
use OpenTelemetry\SDK\Trace\Sampler\AlwaysOnSampler;
use OpenTelemetry\SDK\Trace\Sampler\ParentBased;
use OpenTelemetry\SDK\Trace\SpanProcessor\SimpleSpanProcessor;
use OpenTelemetry\SDK\Trace\SpanProcessor\BatchSpanProcessorBuilder;
use OpenTelemetry\SDK\Trace\TracerProvider;
use OpenTelemetry\SemConv\ResourceAttributes;
use OpenTelemetry\Contrib\Grpc\GrpcTransportFactory;
use OpenTelemetry\Contrib\Otlp\OtlpUtil;
use OpenTelemetry\API\Signals;
```

```
// OpenTelemetry 初始化配置 (需要在PHP应用初始化时就进行OpenTelemetry初始化配置)
function initOpenTelemetry()
{
    // 1. 设置 OpenTelemetry 资源信息
    $resource = ResourceInfoFactory::emptyResource()-
    >merge(ResourceInfo::create(Attributes::create([
        ResourceAttributes::SERVICE_NAME => '<your-service-name>', // 应用名, 必填, 如 php-opentelemetry-demo
        ResourceAttributes::HOST_NAME => '<your-host-name>' // 主机名, 选填
        'token' => '<your-token>' // 替换成步骤1中获得的 Token
    ])));

    // 2. 创建将 Span 输出到控制台的 SpanExporter
    // $spanExporter = new SpanExporter(
    // (new StreamTransportFactory()->create('php://stdout', 'application/json')
    // );

    // 2. 创建通过 gRPC 上报 Span 的 SpanExporter
    $transport = (new GrpcTransportFactory()->create('<grpc-endpoint>' .
    OtlpUtil::method(Signals::TRACE)); # 替换成步骤1中获得的接入点信息
    $spanExporter = new SpanExporter($transport);

    // 3. 创建全局的 TracerProvider, 用于创建 tracer
    $tracerProvider = TracerProvider::builder()
    ->addSpanProcessor(
    (new BatchSpanProcessorBuilder($spanExporter)->build()
    )
    ->setResource($resource)
    ->setSampler(new ParentBased(new AlwaysOnSampler()))
    ->build();

    Sdk::builder()
    ->setTracerProvider($tracerProvider)
    ->setPropagator(TraceContextPropagator::getInstance())
    ->setAutoShutdown(true) // PHP 程序退出后自动关闭 tracerProvider, 保证链路数据都被上报
    ->buildAndRegisterGlobal(); // 将 tracerProvider 添加到全局
}
?>
```

修改应用代码, 使用 OpenTelemetry API 创建 Span

1. 在 index.php 文件中导入所需包:

```
<?php

use OpenTelemetry\API\Globals;
use OpenTelemetry\API\Trace\StatusCode;
use OpenTelemetry\API\Trace\SpanKind;
use OpenTelemetry\SDK\Common\Attribute\Attributes;
use OpenTelemetry\SDK\Trace\TracerProvider;

use Psr\Http\Message\ResponseInterface as Response;
use Psr\Http\Message\ServerRequestInterface as Request;
use Slim\Factory\AppFactory;

require __DIR__ . '/opentelemetry_util.php';
```

2. 调用 `initOpenTelemetry` 方法完成初始化, 需要在 PHP 应用初始化时就进行 OpenTelemetry 初始化配置:

```
// OpenTelemetry 初始化, 包含设置应用名、Trace导出方式、Trace上报接入点, 并创建全局TraceProvider
initOpenTelemetry();
```

3. 在 `rolldice` 接口中创建 Span。

```
/**
 * 1. 接口功能: 模拟扔骰子, 返回一个1-6之间的随机正整数
 * 并演示如何创建Span、设置属性、事件、带有属性的事件
 */
$app->get('/rolldice', function (Request $request, Response $response) {
    // 获取 tracer
    $tracer = \OpenTelemetry\API\Globals::tracerProvider()->getTracer('my-tracer');
    // 创建 Span; 设置span kind, 不设置默认为KIND_INTERNAL
    $span = $tracer->spanBuilder("/rolldice")-
    >setSpanKind(SpanKind::KIND_SERVER)->startSpan();
    // 为 Span 设置属性
    $span->setAttribute("http.method", "GET");
    // 为 Span 设置事件
    $span->addEvent("Init");
    // 设置带有属性的事件
    $eventAttributes = Attributes::create([
        "key1" => "value",
        "key2" => 3.14159,
```

```
]);  
  
// 业务代码  
$result = random_int(1,6);  
$response->getBody()->write(strval($result));  
  
$span->addEvent("End");  
// 销毁 Span  
$span->end();  
  
return $response;  
});
```

4. 创建嵌套 Span。

新建一个 `rolltwodices` 接口，模拟扔两个骰子，返回两个1-6之间的随机正整数。以下代码演示如何创建嵌套的 Span：

```
$app->get('/rolltwodices', function (Request $request, Response $response) {  
    // 获取 tracer  
    $tracer = \OpenTelemetry\API\Globals::tracerProvider()->getTracer('my-tracer');  
    // 创建 Span  
    $parentSpan = $tracer->spanBuilder("/rolltwodices/parent")-  
>setSpanKind(SpanKind::KIND_SERVER)->startSpan();  
    $scope = $parentSpan->activate();  
  
    $value1 = random_int(1,6);  
  
    $childSpan = $tracer->spanBuilder("/rolltwodices/parent/child")->startSpan();  
  
    // 业务代码  
    $value2 = random_int(1,6);  
    $result = "dice1: " . $value1 . ", dice2: " . $value2;  
  
    // 销毁 Span  
    $childSpan->end();  
    $parentSpan->end();  
    $scope->detach();  
  
    $response->getBody()->write(strval($result));  
    return $response;  
});
```

5. 使用 Span 记录代码中发生的异常。

新建 `error` 接口，模拟接口发生异常。以下代码演示如何在代码发生异常时使用 Span 记录状态：

```
$app->get('/error', function (Request $request, Response $response) {  
    // 获取 tracer  
    $tracer = \OpenTelemetry\API\Globals::tracerProvider()->getTracer('my-tracer');  
    // 创建 Span  
    $span3 = $tracer->spanBuilder("/error")->setSpanKind(SpanKind::KIND_SERVER)-  
>startSpan();  
    try {  
        // 模拟代码发生异常  
        throw new \Exception('exception!');  
    } catch (\Throwable $t) {  
        // 设置Span状态为error  
        $span3->setStatus(\OpenTelemetry\API\Trace\StatusCode::STATUS_ERROR,  
            "expcetion in span3!");  
        // 记录异常栈轨迹  
        $span3->recordException($t, ['exception.escaped' => true]);  
    } finally {  
        $span3->end();  
        $response->getBody()->write("error");  
        return $response;  
    }  
});
```

运行应用

1. 执行以下命令：

```
php -S localhost:8080
```

2. 在浏览器中访问以下链接：

```
http://localhost:8080/rolldice  
http://localhost:8080/rolltwodices  
http://localhost:8080/error
```

每次访问页面，OpenTelemetry 会创建链路数据，并将链路数据上报至 APM。

接入 Node.js 应用 通过 Jaeger 原始 SDK 上报

最近更新时间：2023-11-22 10:06:03

本文将为您介绍如何使用 Jaeger 原始 SDK 上报 Node.js 应用数据。

操作步骤

步骤1: 获取接入点和 Token

进入 [应用性能监控控制台](#)，进入 [应用监控 > 应用列表](#) 页面，单击 [接入应用](#)，在接入应用时选择 Node.js 语言与 Jaeger 的数据采集方式。在选择接入方式步骤获取您的接入点和 Token，如下图所示：



步骤2: 安装依赖

使用 npm 安装依赖

```
$ npm i jaeger-client
```

步骤3: 引入 SDK 并且进行数据上报

1. 引入SDK，示例如下：

```
const initTracer = require('jaeger-client').initTracer;

// jaeger 配置
const config = {
  serviceName: 'service-name', // 服务名称，根据业务自行修改
  sampler: {
```

```
    type: 'const',
    param: 1,
  },
  reporter: {
    logSpans: true,
    collectorEndpoint: 'http://ap-guangzhou.apm.tencentcs.com:14268/api/traces',
    // 接入点, 此前在应用性能监控获取的接入点多了 api/traces
  },
};

const options = {
  tags: {
    token: 'Vds*****CrKck' // 业务申请的 token
  },
};
```

❗ 说明

Node 使用 API 直接进行数据上报，因此不需要启动 Jaeger agent。接入点选择自己对应的网络环境，并且在后面加入 `/api/traces` 后缀即可。

2. 进行数据上报，示例如下：

```
// 初始化 tracer 实例对象
const tracer = initTracer(config, options);

// 初始化 span 实例对象
const span = tracer.startSpan('spanStart');

// 当前服务为 server
span.setTag('span.kind', 'server');

// 设置标签（可选，支持多个）
span.setTag('tagName', 'tagValue');

// 设置事件（可选，支持多个）
span.log({ event: 'timestamp', value: Date.now() });

// 标记Span结束
span.finish();
```