-

GoFrame3

OpenTelemetryattributesPrometheuslabels

Content Menu

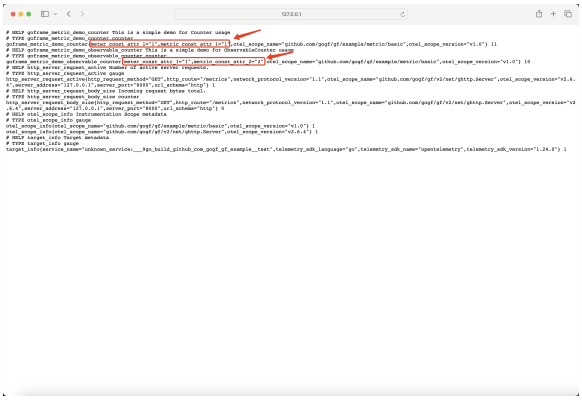
MeterMeterMeter

```
package main

import (
        "context"

        "go.opentelemetry.io/otel/exporters/prometheus"

        "github.com/gogf/gf/contrib/metric/otelmetric/v2"
        "github.com/gogf/gf/v2/frame/g"
        "github.com/gogf/gf/v2/os/gctx"
        "github.com/gogf/gf/v2/os/gmetric"
)

const (
        instrument        = "github.com/gogf/gf/example/metric/basic"
        instrumentVersion = "v1.0"
)

var (
        meter = gmetric.GetGlobalProvider().Meter(gmetric.MeterOption{
                Instrument:        instrument,
                InstrumentVersion: instrumentVersion,
                Attributes: gmetric.Attributes{
                        gmetric.NewAttribute("meter_const_attr_1", 1),
                },
        })
        counter = meter.MustCounter(
                "goframe.metric.demo.counter",
                gmetric.MetricOption{
                        Help: "This is a simple demo for Counter usage",
                        Unit: "bytes",
                        Attributes: gmetric.Attributes{
                                gmetric.NewAttribute
("metric_const_attr_1", 1),
                        },
                },
        )
        observableCounter = meter.MustObservableCounter(
                "goframe.metric.demo.observable_counter",
                gmetric.MetricOption{
                        Help: "This is a simple demo for ObservableCounter
usage",
                        Unit: "%",
                        Attributes: gmetric.Attributes{
                                gmetric.NewAttribute
("metric_const_attr_2", 2),
                        },
                },
        )
)

func main() {
        var ctx = gctx.New()
        // Callback for observable metrics.
        meter.MustRegisterCallback(func(ctx context.Context, obs gmetric.
Observer) error {
```

```
                obs.Observe(observableCounter, 10)
                return nil
        }, observableCounter)

        // Prometheus exporter to export metrics as Prometheus format.
        exporter, err := prometheus.New(
                prometheus.WithoutCounterSuffixes(),
                prometheus.WithoutUnits(),
        )
        if err != nil {
                g.Log().Fatal(ctx, err)
        }

        // OpenTelemetry provider.
        provider := otelmetric.MustProvider(
                otelmetric.WithReader(exporter),
        )
        provider.SetAsGlobal()
        defer provider.Shutdown(ctx)

        // Counter.
        counter.Inc(ctx)
        counter.Add(ctx, 10)

        // HTTP Server for metrics exporting.
        otelmetric.StartPrometheusMetricsServer(8000, "/metrics")
}
```

MeterMetricMeterOptionMetricOptionAttributes

http://127.0.0.1:8000/metrics Meter



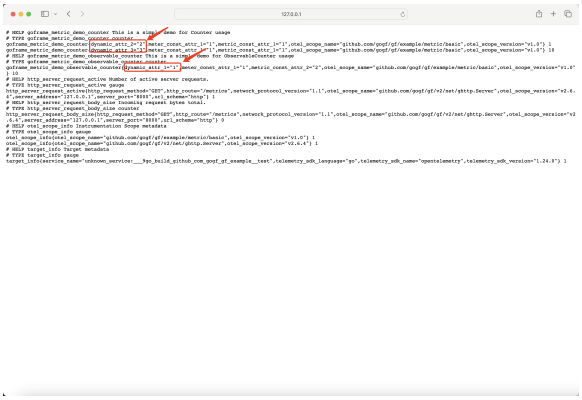
```
package main

import (
        "context"

        "go.opentelemetry.io/otel/exporters/prometheus"

        "github.com/gogf/gf/contrib/metric/otelmetric/v2"
        "github.com/gogf/gf/v2/frame/g"
        "github.com/gogf/gf/v2/os/gctx"
        "github.com/gogf/gf/v2/os/gmetric"
)

const (
```

```
        instrument        = "github.com/gogf/gf/example/metric/basic"
        instrumentVersion = "v1.0"
)

var (
        meter = gmetric.GetGlobalProvider().Meter(gmetric.MeterOption{
                Instrument:        instrument,
                InstrumentVersion: instrumentVersion,
                Attributes: gmetric.Attributes{
                        gmetric.NewAttribute("meter_const_attr_1", 1),
                },
        })
        counter = meter.MustCounter(
                "goframe.metric.demo.counter",
                gmetric.MetricOption{
                        Help: "This is a simple demo for Counter usage",
                        Unit: "bytes",
                        Attributes: gmetric.Attributes{
                                gmetric.NewAttribute
("metric_const_attr_1", 1),
                        },
                },
        )
        observableCounter = meter.MustObservableCounter(
                "goframe.metric.demo.observable_counter",
                gmetric.MetricOption{
                        Help: "This is a simple demo for ObservableCounter
usage",
                        Unit: "%",
                        Attributes: gmetric.Attributes{
                                gmetric.NewAttribute
("metric_const_attr_2", 2),
                        },
                },
        )
)

func main() {
        var ctx = gctx.New()
        // Callback for observable metrics.
        meter.MustRegisterCallback(func(ctx context.Context, obs gmetric.
Observer) error {
                obs.Observe(observableCounter, 10, gmetric.Option{
                        Attributes: gmetric.Attributes{
                                gmetric.NewAttribute("dynamic_attr_1", 1),
                        },
                })
                return nil
        }, observableCounter)

        // Prometheus exporter to export metrics as Prometheus format.
        exporter, err := prometheus.New(
                prometheus.WithoutCounterSuffixes(),
                prometheus.WithoutUnits(),
        )
        if err != nil {
                g.Log().Fatal(ctx, err)
        }

        // OpenTelemetry provider.
        provider := otelmetric.MustProvider(
                otelmetric.WithReader(exporter),
        )
        provider.SetAsGlobal()
        defer provider.Shutdown(ctx)

        // Counter.
        counter.Inc(ctx, gmetric.Option{
                Attributes: gmetric.Attributes{
                        gmetric.NewAttribute("dynamic_attr_2", 2),
                },
```

```
        })
        counter.Add(ctx, 10, gmetric.Option{
                Attributes: gmetric.Attributes{
                        gmetric.NewAttribute("dynamic_attr_3", 3),
                },
        })

        // HTTP Server for metrics exporting.
        otelmetric.StartPrometheusMetricsServer(8000, "/metrics")
}
```

OptionAttributes

http://127.0.0.1:8000/metrics



InstrumentInstrumentInstrument

```
package main

import (
        "context"

        "go.opentelemetry.io/otel/exporters/prometheus"

        "github.com/gogf/gf/contrib/metric/otelmetric/v2"
        "github.com/gogf/gf/v2/frame/g"
        "github.com/gogf/gf/v2/os/gctx"
        "github.com/gogf/gf/v2/os/gmetric"
)

const (
        instrument        = "github.com/gogf/gf/example/metric/basic"
        instrumentVersion = "v1.0"
)

var (
        meter = gmetric.GetGlobalProvider().Meter(gmetric.MeterOption{
                Instrument:        instrument,
                InstrumentVersion: instrumentVersion,
                Attributes: gmetric.Attributes{
                        gmetric.NewAttribute("meter_const_attr_1", 1),
                },
        })
        counter = meter.MustCounter(
                "goframe.metric.demo.counter",
                gmetric.MetricOption{
                        Help: "This is a simple demo for Counter usage",
                        Unit: "bytes",
```

```
                        Attributes: gmetric.Attributes{
                                gmetric.NewAttribute
("metric_const_attr_1", 1),
                        },
                },
        )
        observableCounter = meter.MustObservableCounter(
                "goframe.metric.demo.observable_counter",
                gmetric.MetricOption{
                        Help: "This is a simple demo for ObservableCounter
usage",
                        Unit: "%",
                        Attributes: gmetric.Attributes{
                                gmetric.NewAttribute
("metric_const_attr_2", 2),
                        },
                },
        )
)

func main() {
        var ctx = gctx.New()

        gmetric.SetGlobalAttributes(gmetric.Attributes{
                gmetric.NewAttribute("global_attr_1", 1),
        }, gmetric.SetGlobalAttributesOption{
                Instrument:        instrument,
                InstrumentVersion: instrumentVersion,
                InstrumentPattern: "",
        })

        // Callback for observable metrics.
        meter.MustRegisterCallback(func(ctx context.Context, obs gmetric.
Observer) error {
                obs.Observe(observableCounter, 10, gmetric.Option{
                        Attributes: gmetric.Attributes{
                                gmetric.NewAttribute("dynamic_attr_1", 1),
                        },
                })
                return nil
        }, observableCounter)

        // Prometheus exporter to export metrics as Prometheus format.
        exporter, err := prometheus.New(
                prometheus.WithoutCounterSuffixes(),
                prometheus.WithoutUnits(),
        )
        if err != nil {
                g.Log().Fatal(ctx, err)
        }

        // OpenTelemetry provider.
        provider := otelmetric.MustProvider(
                otelmetric.WithReader(exporter),
        )
        provider.SetAsGlobal()
        defer provider.Shutdown(ctx)

        // Counter.
        counter.Inc(ctx, gmetric.Option{
                Attributes: gmetric.Attributes{
                        gmetric.NewAttribute("dynamic_attr_2", 2),
                },
        })
        counter.Add(ctx, 10, gmetric.Option{
                Attributes: gmetric.Attributes{
                        gmetric.NewAttribute("dynamic_attr_3", 3),
                },
        })

        // HTTP Server for metrics exporting.
```

```
        otelmetric.StartPrometheusMetricsServer(8000, "/metrics")
}
```

gmetric.SetGlobalAttributesgmetric.SetGlobalAttributesOption

http://127.0.0.1:8000/metrics