-

gmetricgmetricNoopPerformergithub.com/gogf/gf/contrib/metric/otelmetric/v2OpenTelemetryotelmetrichttps://github.com/gogf/gf/tree/master/contrib/metric/otelmetric

```
package main

import (
        "go.opentelemetry.io/otel/sdk/metric"
        "go.opentelemetry.io/otel/sdk/metric/metricdata"

        "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"
)

var (
        meter = gmetric.GetGlobalProvider().Meter(gmetric.MeterOption{
                Instrument:        "github.com/gogf/gf/example/metric
/basic",
                InstrumentVersion: "v1.0",
        })
        counter = meter.MustCounter(
                "goframe.metric.demo.counter",
                gmetric.MetricOption{
                        Help: "This is a simple demo for Counter usage",
                        Unit: "bytes",
                },
        )
)

func main() {
        var (
                ctx    = gctx.New()
                reader = metric.NewManualReader()
        )

        provider := otelmetric.MustProvider(otelmetric.WithReader(reader))
        provider.SetAsGlobal()
        defer provider.Shutdown(ctx)

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

        var (
                rm  = metricdata.ResourceMetrics{}
                err = reader.Collect(ctx, &rm)
        )
        if err != nil {
                g.Log().Fatal(ctx, err)
        }
        g.DumpJson(rm)
}
```

gmetric.GetGlobalProvider()Meter/Instrument

Content Menu

```
meter = gmetric.GetGlobalProvider().Meter(gmetric.MeterOption{
    Instrument:        "github.com/gogf/gf/example/metric/basic",
    InstrumentVersion: "v1.0",
})
```

gmeter.MeterOption

```
// MeterOption holds the creation option for a Meter.
type MeterOption struct {
        // Instrument is the instrumentation name to bind this Metric to a
global MeterProvider.
        // This is an optional configuration for a metric.
        Instrument string

        // InstrumentVersion is the instrumentation version to bind this
Metric to a global MeterProvider.
        // This is an optional configuration for a metric.
        InstrumentVersion string

        // Attributes holds the constant key-value pair description
metadata for all metrics of Meter.
        // This is an optional configuration for a meter.
        Attributes Attributes
}
```

MeterMeter

```
// Meter hold the functions for kinds of Metric creating.
type Meter interface {
        // Counter creates and returns a new Counter.
        Counter(name string, option MetricOption) (Counter, error)

        // UpDownCounter creates and returns a new UpDownCounter.
        UpDownCounter(name string, option MetricOption) (UpDownCounter,
error)

        // Histogram creates and returns a new Histogram.
        Histogram(name string, option MetricOption) (Histogram, error)

        // ObservableCounter creates and returns a new ObservableCounter.
        ObservableCounter(name string, option MetricOption)
(ObservableCounter, error)

        // ObservableUpDownCounter creates and returns a new
ObservableUpDownCounter.
        ObservableUpDownCounter(name string, option MetricOption)
(ObservableUpDownCounter, error)

        // ObservableGauge creates and returns a new ObservableGauge.
        ObservableGauge(name string, option MetricOption)
(ObservableGauge, error)

        // MustCounter creates and returns a new Counter.
        // It panics if any error occurs.
        MustCounter(name string, option MetricOption) Counter

        // MustUpDownCounter creates and returns a new UpDownCounter.
        // It panics if any error occurs.
        MustUpDownCounter(name string, option MetricOption) UpDownCounter

        // MustHistogram creates and returns a new Histogram.
        // It panics if any error occurs.
        MustHistogram(name string, option MetricOption) Histogram

        // MustObservableCounter creates and returns a new
ObservableCounter.
        // It panics if any error occurs.
        MustObservableCounter(name string, option MetricOption)
ObservableCounter

        // MustObservableUpDownCounter creates and returns a new
ObservableUpDownCounter.
        // It panics if any error occurs.
        MustObservableUpDownCounter(name string, option MetricOption)
ObservableUpDownCounter

        // MustObservableGauge creates and returns a new ObservableGauge.
        // It panics if any error occurs.
        MustObservableGauge(name string, option MetricOption)
ObservableGauge

        // RegisterCallback registers callback on certain metrics.
        // A callback is bound to certain component and version, it is
called when the associated metrics are read.
        // Multiple callbacks on the same component and version will be
called by their registered sequence.
        RegisterCallback(callback Callback, canBeCallbackMetrics ...
ObservableMetric) error

        // MustRegisterCallback performs as RegisterCallback, but it
panics if any error occurs.
        MustRegisterCallback(callback Callback, canBeCallbackMetrics ...
ObservableMetric)
}
```

meter.MustCounterCounterMust*panicnameMetricOptionMetricOption

```
// MetricOption holds the basic options for creating a metric.
type MetricOption struct {
        // Help provides information about this Histogram.
        // This is an optional configuration for a metric.
        Help string

        // Unit is the unit for metric value.
        // This is an optional configuration for a metric.
        Unit string

        // Attributes holds the constant key-value pair description
metadata for this metric.
        // This is an optional configuration for a metric.
        Attributes Attributes

        // Buckets defines the buckets into which observations are counted.
        // For Histogram metric only.
        // A histogram metric uses default buckets if no explicit buckets
configured.
        Buckets []float64

        // Callback function for metric, which is called when metric value
changes.
        // For observable metric only.
        // If an observable metric has either Callback attribute nor
global callback configured, it does nothing.
        Callback MetricCallback
}
```

otelmetric.MustProvider

```
provider := otelmetric.MustProvider(otelmetric.WithReader(reader))
provider.SetAsGlobal()
defer provider.Shutdown(ctx)
```

GlobalProviderprovider.SetAsGlobal

maindefer provider.ShutDown

Counter

```
// Counter is a Metric that represents a single numerical value that can
ever
// goes up.
type Counter interface {
        Metric
        CounterPerformer
}

// CounterPerformer performs operations for Counter metric.
type CounterPerformer interface {
        // Inc increments the counter by 1. Use Add to increment it by
arbitrary
        // non-negative values.
        Inc(ctx context.Context, option ...Option)

        // Add adds the given value to the counter. It panics if the value
is < 0.
        Add(ctx context.Context, increment float64, option ...Option)
}
```

CounterIncAddMetric

```
// Metric models a single sample value with its metadata being exported.
type Metric interface {
        // Info returns the basic information of a Metric.
        Info() MetricInfo
}

// MetricInfo exports information of the Metric.
type MetricInfo interface {
        Key() string                // Key returns the unique string key
of the metric.
        Name() string               // Name returns the name of the metric.
        Help() string               // Help returns the help description
of the metric.
        Unit() string               // Unit returns the unit name of the
metric.
        Type() MetricType           // Type returns the type of the metric.
        Attributes() Attributes     // Attributes returns the constant
attribute slice of the metric.
        Instrument() InstrumentInfo // InstrumentInfo returns the
instrument info of the metric.
}

// InstrumentInfo exports the instrument information of a metric.
type InstrumentInfo interface {
        Name() string    // Name returns the instrument name of the metric.
        Version() string // Version returns the instrument version of the
metric.
}
```

OpenTelemetryMetricReaderOpenTelemetryManualReaderManualReaderOpenTelemetry

```
reader = metric.NewManualReader()
```

WithReaderProvider

```
provider := otelmetric.MustProvider(otelmetric.WithReader(reader))
```

Collect

```
var (
        rm  = metricdata.ResourceMetrics{}
        err = reader.Collect(ctx, &rm)
)
if err != nil {
        g.Log().Fatal(ctx, err)
}
g.DumpJson(rm)
```

```
...
    "ScopeMetrics": [
        {
            "Scope": {
                "Name": "github.com/gogf/gf/example/metric/basic",
                "Version": "v1.0",
                "SchemaURL": ""
            },
            "Metrics": [
                {
                    "Name": "goframe.metric.demo.counter",
                    "Description": "This is a simple demo for Counter
usage",
                    "Unit": "bytes",
                    "Data": {
                        "DataPoints": [
                            {
                                "Attributes": [],
                                "StartTime": "2024-03-25T10:13:19.326977
+08:00",
                                "Time": "2024-03-25T10:13:19.327144+08:00",
                                "Value": 11
                            }
                        ],
                        "Temporality": "CumulativeTemporality",
                        "IsMonotonic": true
                    }
                }
            ]
        },
...
```