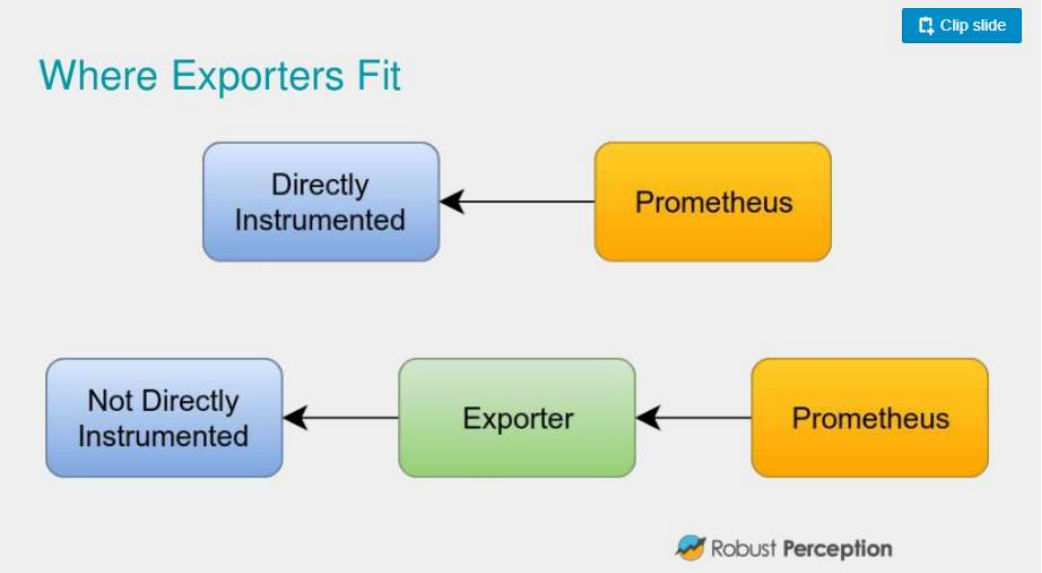
当我们希望写一个 exporter 的时候-PromCon 2016
https://www.youtube.com/watch?v=KXq5ibSj2qA&

Ideals
充分利用 Prometheus 的方法是直接检测代码。
在应用程序和库中使用客户端库可提供全面的监视
　　-查看整个软件堆栈
不过，这并非总是可能的



Architecture:
　　我们希望 exporters 可以看起来尽可能像直接检测的应用程序。这意味着 Prometheus 负责 调度(Scheduling) 及 服务发现(Service discovery)。意味着 Exporter 不应有任何逻辑来确定团队的位置。Prometheus 从控制台(Console) 或 ec2 订单（ec2 order）中找出适合你的使用方式。同样，Prometheus 决定何时进行刮擦(Scrapes),及返回无状态的数据。因为视图（view）使用水平监控（horizontal monitoring）。
也意味着 Exporter 仅与一个应用实例进行对话，例外：SNMP 和 blackbox Exporter 不应该使用推送网关(pushgateway)，而且不应该使查询崩溃。

Metric naming – Units

Bads: no unit
request_latency

Better: non-base unit
request_latency_millisecons

Good: standar base units
request_latency_seconds

Prometheus 已对基本单位（base units）例如（Seconds）秒和(Bytes)字节 进行了标准化。
在可行的地方转换。始终在使用的时候放入单元（unit）。

Metric naming – Namespaces
Bad: what sort of request? 哪种请求
request_latency_seconds

#Bad: where in the stack? 在堆栈的哪
http_request_latency_seconds

Good: Ah, when it enters the HTTP server 噢！当它进入 Http 服务时
http_server_request_latency_seconds

只有模糊地熟悉您的系统的人应该能够很好地猜测指标的含义，
把它当成库（libraries），而不是应用程序（applications）。

Metric naming – Namespaces
Bad: Latency calculated on client 客户端上的延迟
http_server_request_latency_seconds

Good: Prometheus Histogram/ Summary-style counters 直方图/摘要式计数器
http_server_request_latency_seconds_count
http_server_request_latency_seconds_sum

如果数据可用于计算 Prometheus 中的延迟，将它公开。

Metric naming – Ratios
Ratio 比例：是基本单位占所发送均值的百分比
最常见的一种是在内存中使用了的磁盘，得到可用磁盘空间 除于 总磁盘空间。
Bad: Non-base units, incorrect units
http_server_requests_failed_percentage
http_server_requests_failed_percentage # Actually a ratio

Better: Base units
http_server_requests_failed_ratio

Best: Raw data as Counters (or Gaugs, depending) 约定了的计数器 total
http_server_requests_total
http_server_reqeusts_failed_total

Metric naming – Suffixes
Bad: Counter without suffix 计数器没有后缀
http_server_requests
Good: Correct counter suffix 正确的计数器后缀

http_server_requests_total

Bad: Using Counter/Summary/Histogram suffixes on a Gauge 在量规上使用计数器/摘要/直方图后缀
http_server_queued_requests_count
http_server_queued_requests_total
Good: Gauge has no suffix 量规没有后缀
http_server_queued_requests

Metric naming – Mixed types
有时会遇到一系列指标，其中一些是 Counters，另外一些则是 Gauges。例如在 SHOW GLOBAL STATUS 中，MySQL 具有
threads_created
threads_running

你可以浏览 497 个条目并手动对其进行分类，然后添加_total，也可以将它们保持原样并将其显示为 Untyped。这是你必须权衡的。

Metrics – General
在创建 Metric 时不要做数学。在 Prometheus 方面进行处理。
不要将 Metric 的标签放在名称中 如_by_type。标签以某种方式聚合是毫无意义的

snake_case 是标准的，其他的，在 Exporter 中不能转换。

process_和 scrape_前缀是保留的。

Labels
如果有有用的标签，则最好将它们提取出来。
不要仅仅因为看起来相关或“类似标签”而添加标签
再举一个例子
innodb_buffer_pool_pages_data

Innodb_Buffer_Pool_Pages_Dirty
Innodb_Buffer_Pool_Pages_Flushed
Innodb_Buffer_Pool_Pages_Free
Innodb_Buffer_Pool_Pages_Latched
Innodb_Buffer_Pool_Pages_Misc

Labels -Partition
具有 Metric 的时间序列不应重叠。
各项 Metric 的总和或平均值应该有意义
my_metric{l="a"}
my_metric{l="b"}
my_metric{l="c"}
my_metric{l="d+a"} # not a partition
my_metric{l="total"} # Breaks aggregation: Remove;
my_metric{} # Breaks aggregation: Just no

Labels – The Table Exception
有时数据不应该成为标签，但这是在 PromQL 终端上使用的唯一明智的方法。

比如硬件温度传感器。
不能求和或求平均温度（焦耳就是想要的）

它们是相当不标准的，并且有数百种类型。因此将（“Adapter”）适配器类型放入度量标准将无法使用可用的仪表板
因此，拥有一个带有（“Adapter”）适配器和（“sensor”）传感器标签的度量标准是有意义的

Labels - Failure and Caches
最好为(total)总数和(failures)失败设置单独的计数器，而不是一个(result)结果标签。在 PromQL 中更易于使用

缓存也是相似的，导出匹配和总计作为计数器

不要通过 成功或失败 标签来区分（latency metrics）延迟指标
当延迟由于故障超时而增加时，它将引起混乱，但是无法在仅仅通过图表发现它（是的，人们会这样做，而在紧急情况下会忘记细微之处）

Labels – General Advice
不要让所有指标都返回静态标签,根据您的工作使用机器角色方法或目标标签

所有时间序列应具有一致的标签名称

为了清楚起见并减少碰撞的机会，请避免使用 le, quantile, instance, job, type, id, cluster, datacenter, az, zone, region, service, team, env, group tec.作为标签名称。

避免使用没用一直设置的时间序列。

如果不确定，请不要使用标签

Metrics to drop
有一些指标并不如是很有用的，比如 Min 和 Max 在统计上是没用的，而且并没有说明在什么阶段是有用的

（Machine metrics）机器的指标， 如 filesystem, memory disk, -除非 Node exporter

对于 JMX, jmx_exporter 已经覆盖了 process（进程）和 JVM stats. 所以可以删除

具有(high cardinality)高基数或 (churn)流失率的指标-太昂贵
仔细考虑是否保留分位数。

Exporters – Failed scrapes
有时无法与应用程序对话。怎么处理呢?

您可以返回 500，这意味着 up 将为 0。简单

如果您的 Exporters 在这种情况下仍然可以提供一些有用的指标（例如流程统计信息），则您可以使用 myexpoter_up 指标，用户必须检查该指标 0/1

拥有 myexporter_scrape_duration_seconds 很有用。

Exporter – other tips
保持 config 最小化和简单化。最好带有应用程序终结点的标记

不要询问或尝试公开时间戳。这几乎不不会用到。

始终在其他客户端库中使用 ConstMetrics 及其等效项

Example

```
from prometheus_client import start_http_server
from prometheus_client.core import GaugeMetricFamily, REGISTRY

class MyCollector(object):
  def collect(self):
    # Your code here
    yield GaugeMetricFamily('my_gauge', 'Help text', value=42)

REGISTRY.register(MyCollector())
start_http_server(1234)
while True: time.sleep(1)
```