项目依赖了 influxdb 的依赖：

```
<dependency>
    <groupId>org.influxdb</groupId>
    <artifactId>influxdb-java</artifactId>
    <version>2.14</version>
</dependency>
```

MeasurementInfo

这个类是 metric 的数据结构，里面含有两个字段 name 和 tags，通常

MetricInfoProvider

一个通用接口，为度量标准提供自定义信息，它支持传入一个自定义度量信息类型的类，另外可以通过方法获取 metric 信息：

```
MetricInfo getMetricInfo(String metricName, MetricGroup group);
```

MetricMapper

通过该类将 MeasurementInfo 和 Gauge、Counter、Histogram、Meter 组装成 InfluxDB 中的 Point 数据，Point 结构如下（主要就是构造 metric name、fields、tags 和 timestamp）：

```
private String measurement;
private Map<String, String> tags;
private Long time;
private TimeUnit precision;
private Map<String, Object> fields;
```

整个 MetricMapper 代码如下：

```
class MetricMapper {

    static Point map(MeasurementInfo info, Instant timestamp,
```

```
Gauge<?> gauge) {
        Point.Builder builder = builder(info, timestamp);
        Object value = gauge.getValue();
        if (value instanceof Number) {
            builder.addField("value", (Number) value);
        } else {
            builder.addField("value", String.valueOf(value));
        }
        return builder.build();
    }

    static Point map(MeasurementInfo info, Instant timestamp,
Counter counter) {
        return builder(info, timestamp)
            .addField("count", counter.getCount())
            .build();
    }

    static Point map(MeasurementInfo info, Instant timestamp,
Histogram histogram) {
        HistogramStatistics statistics = histogram.getStatistics();
        return builder(info, timestamp)
            .addField("count", statistics.size())
            .addField("min", statistics.getMin())
            .addField("max", statistics.getMax())
            .addField("mean", statistics.getMean())
            .addField("stddev", statistics.getStdDev())
            .addField("p50", statistics.getQuantile(.50))
            .addField("p75", statistics.getQuantile(.75))
            .addField("p95", statistics.getQuantile(.95))
            .addField("p98", statistics.getQuantile(.98))
            .addField("p99", statistics.getQuantile(.99))
            .addField("p999", statistics.getQuantile(.999))
            .build();
    }

    static Point map(MeasurementInfo info, Instant timestamp, Meter
meter) {
        return builder(info, timestamp)
            .addField("count", meter.getCount())
            .addField("rate", meter.getRate())
            .build();
    }

    private static Point.Builder builder(MeasurementInfo info,
Instant timestamp) {
        return Point.measurement(info.getName())
            .tag(info.getTags())
```

```
                .time(timestamp.toEpochMilli(), TimeUnit.MILLISECONDS);
        }
    }
```

MeasurementInfoProvider

MeasurementInfoProvider 类实现了 MetricInfoProvider 接口，传入的值就是 MeasurementInfo。

该类可以通过 metric name 和 MetricGroup 获取到对应的 MeasurementInfo、Tags 和 ScopedName 信息。

AbstractReporter

该类是一个抽象类，实现了接口 MetricReporter，该类和 org.apache.flink.metrics.reporter.AbstractReporter 是一样的，但是有广义的度量信息。

InfluxdbReporter

该 reporter 可以将 metric 数据导入 InfluxDB，该类继承自 AbstractReporter 抽象类，实现了 Scheduled 接口，有下面三个属性：

```
private String database;
private String retentionPolicy;
private InfluxDB influxDB;
```

在 open 方法中获取到配置文件中的 InfluxDB 设置，然后初始化 InfluxDB 相关的配置，构造 InfluxDB 客户端：

```
public void open(MetricConfig config) {
    //获取到 host 和 port
    String host = getString(config, HOST);
    int port = getInteger(config, PORT);
    //判断 host 和 port 是否合法
    if (!isValidHost(host) || !isValidPort(port)) {
        throw new IllegalArgumentException("Invalid host/port
configuration. Host: " + host + " Port: " + port);
    }
    //获取到 InfluxDB database
    String database = getString(config, DB);
    if (database == null) {
        throw new IllegalArgumentException("'" + DB.key() + "'
configuration option is not set");
    }
    String url = String.format("http://%s:%d", host, port);
    //获取到 InfluxDB username 和 password
    String username = getString(config, USERNAME);
    String password = getString(config, PASSWORD);

    this.database = database;
    //InfluxDB 保留政策
    this.retentionPolicy = getString(config, RETENTION_POLICY);
    if (username != null && password != null) {
        //如果有用户名和密码，根据 url 和 用户名密码来创建连接
        influxDB = InfluxDBFactory.connect(url, username,
password);
    } else {
        //否则就根据 url 连接
        influxDB = InfluxDBFactory.connect(url);
    }

    log.info("Configured InfluxDBReporter with {host:{}, port:{},
db:{}, and retentionPolicy:{}}", host, port, database,
retentionPolicy);
}
```

然后在 report 方法中调用一个内部 buildReport 方法来构造 BatchPoints，将一批 Point 放在该对象中，BatchPoints 对象的属性如下：

```
private String database;
private String retentionPolicy;
private Map<String, String> tags;
private List<Point> points;
private ConsistencyLevel consistency;
private TimeUnit precision;
```

通过 buildReport 方法返回的 BatchPoints 如果不为空，则会通过 write 方法将 BatchPoints 写入 InfluxDB：

```
if (report != null) {
    influxDB.write(report);
}
```

InfluxdbReporterOptions

InfluxDB 相关的配置，比如 host、port、username、password、db、retentionPolicy

注意

如果你要使用 InfluxDB reporter，你必须复制 /opt/flink-metrics-influxdb-1.9.0.jar 到 flink 的 lib 目录下。

参数：

```
host: InfluxDB服务器主机
port: (可选) InfluxDB 服务器端口，默认为 8086
db: 用于存储指标的 InfluxDB 数据库
username: (可选) 用于身份验证的 InfluxDB 用户名
password: (可选) InfluxDB 用户名用于身份验证的密码
```

配置示例：

```
metrics.reporter.influxdb.class:
org.apache.flink.metrics.influxdb.InfluxdbReporter
metrics.reporter.influxdb.host: localhost
metrics.reporter.influxdb.port: 8086
metrics.reporter.influxdb.db: flink
metrics.reporter.influxdb.username: zhisheng
metrics.reporter.influxdb.password: zhisheng
```