JMXReporterFactory

该类实现 MetricReporterFactory 接口，它是为 JMXReporter 提供的 MetricReporterFactory。在该类中有一个参数（表示常量 port，后面需要通过该字段从配置中获取对应的值）：

```
static final String ARG_PORT = "port";
```

重写了 MetricReporterFactory 中的 createMetricReporter 方法，该方法里面获取到 port 配置后，然后构造单个参数（参数为 port）的 JMXReporter 对象。

JMXReporter

上面通过 port 参数构造的对象，那么我们就来看下该构造参数的内部实现：

```
JMXReporter(@Nullable final String portsConfig) {
    //通过 ManagementFactory 获取到 MBeanServer
    this.mBeanServer = ManagementFactory.getPlatformMBeanServer();
    this.registeredMetrics = new HashMap<>();

    //如果 port 不为空
    if (portsConfig != null) {
        //解析传入的 port（传入的 port 可能是组合的，所以解析出来后可能有多
个）
        Iterator<Integer> ports =
NetUtils.getPortRangeFromString(portsConfig);

        JMXServer successfullyStartedServer = null;
        while (ports.hasNext() && successfullyStartedServer ==
null) {
            //构建 JMXServer 对象
            JMXServer server = new JMXServer();
            int port = ports.next();
            try {
                //根据解析出来的 port 开启 JMXServer
                server.start(port);
                LOG.info("Started JMX server on port " + port +
".");
                //将成功开启的 JMXServer 赋值给
successfullyStartedServer，所以，如果是多个端口的，在第一个端口就创建
```

```
JMXServer 成功的话，那么后面就不会再进入该逻辑执行了，因为在 while 循环里面已
经做了判断  successfullyStartedServer == null

                successfullyStartedServer = server;
            } catch (IOException ioe) { //assume port conflict
                LOG.debug("Could not start JMX server on port " +
port + ".", ioe);
                try {
                    server.stop();
                } catch (Exception e) {
                    LOG.debug("Could not stop JMX server.", e);
                }
            }
        }
        if (successfullyStartedServer == null) {
            throw new RuntimeException("Could not start JMX server
on any configured port. Ports: " + portsConfig);
        }
        //将 successfullyStartedServer 赋值给外部 JMXReporter 的属性
jmxServer
        this.jmxServer = successfullyStartedServer;
    } else {
        this.jmxServer = null;
    }
    LOG.info("Configured JMXReporter with {port:{}}", portsConfig);
}
```

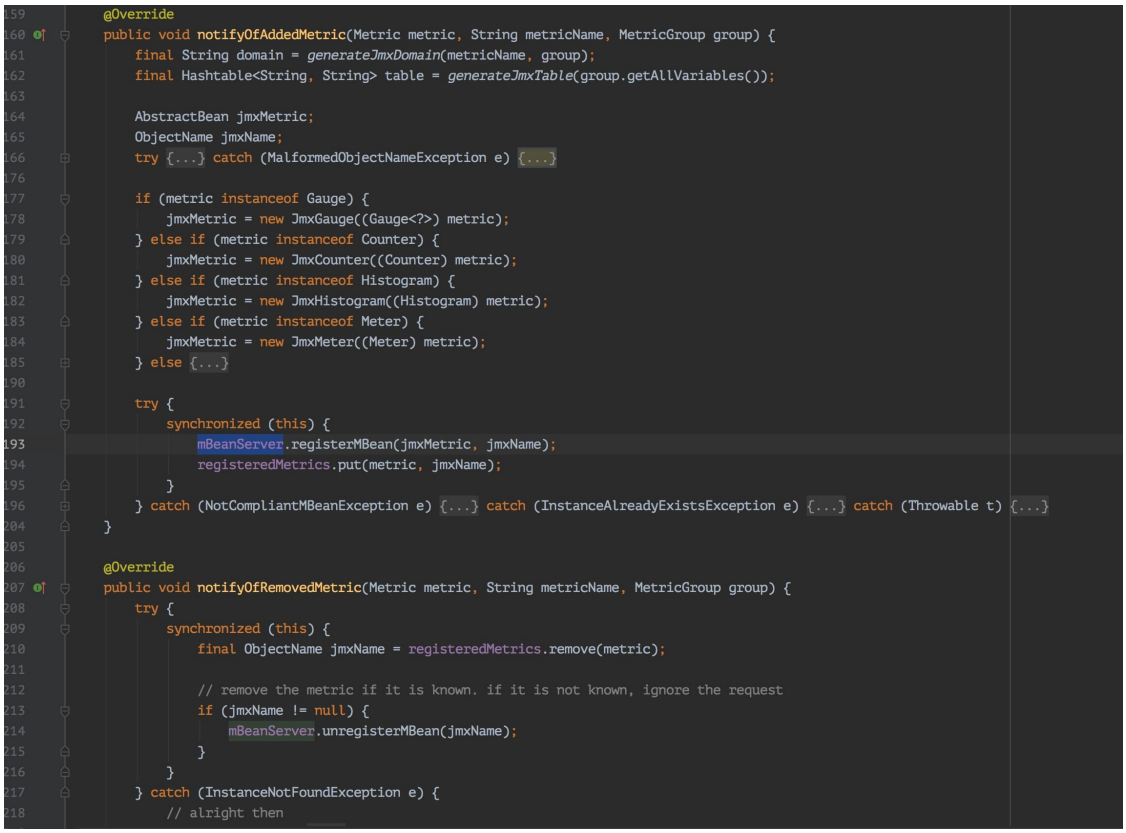
上面是 JMXReporter 中的构造方法，那么再来看下它的参数有哪些呢?

```
//管理 bean 注册和注销的服务器
private final MBeanServer mBeanServer;

//已经被添加到 MBeanServer 注册的 metric
private final Map<Metric, ObjectName> registeredMetrics;

//JMX 客户端连接的服务器端，允许更好的控制端口使用
@Nullable
private final JMXServer jmxServer;
```

在该类中又重写了 notifyOfAddedMetric 和 notifyOfRemovedMetric 方法。



```java
@Override
public void notifyOfAddedMetric(Metric metric, String metricName, MetricGroup group) {
    final String domain = generateJmxDomain(metricName, group);
    final Hashtable<String, String> table = generateJmxTable(group.getAllVariables());

    AbstractBean jmxMetric;
    ObjectName jmxName;
    try {...} catch (MalformedObjectNameException e) {...}

    if (metric instanceof Gauge) {
        jmxMetric = new JmxGauge((Gauge<?>) metric);
    } else if (metric instanceof Counter) {
        jmxMetric = new JmxCounter((Counter) metric);
    } else if (metric instanceof Histogram) {
        jmxMetric = new JmxHistogram((Histogram) metric);
    } else if (metric instanceof Meter) {
        jmxMetric = new JmxMeter((Meter) metric);
    } else {...}

    try {
        synchronized (this) {
            mBeanServer.registerMBean(jmxMetric, jmxName);
            registeredMetrics.put(metric, jmxName);
        }
    } catch (NotCompliantMBeanException e) {...} catch (InstanceAlreadyExistsException e) {...} catch (Throwable t) {...}
}

@Override
public void notifyOfRemovedMetric(Metric metric, String metricName, MetricGroup group) {
    try {
        synchronized (this) {
            final ObjectName jmxName = registeredMetrics.remove(metric);

            // remove the metric if it is known. if it is not known, ignore the request
            if (jmxName != null) {
                mBeanServer.unregisterMBean(jmxName);
            }
        }
    } catch (InstanceNotFoundException e) {
        // alright then
```

在 notifyOfAddedMetric 方法中你可以看到会根据 metric 来判断到底是哪种度量标准，然后创建对应的 JMX Bean 对象（比如 JmxGauge、JmxCounter、JmxHistogram、JmxMeter），最后将对应的 jmxMetric 注册到 mBeanServer 中，然后还会把 metric 和 对应的 jmxName 放到 registeredMetrics 中去。

MetricMBean

MetricMBean 只是一个接口，它有一个抽象实现类 AbstractBean，另外该 MBean 是所有 Metric 的公共接口，所以自然会有 JmxCounterMBean、JmxGaugeMBean、JmxHistogramMBean、JmxMeterMBean 都将继承自 MetricMBean 接口。另外 JmxCounter、JmxGauge、JmxHistogram、JmxMeter 分别继承自 AbstractBean 和实现其对应的 MBean 接口。

JMXServer

在 JMXServer 中有的属性有：

```
//注册器
private Registry rmiRegistry;

//JMX 连接器服务端地址
private JMXConnectorServer connector;

//端口
private int port;
```

启动注册器和 JMX 服务：

```
public void start(int port) throws IOException {
    if (rmiRegistry != null && connector != null) {
        LOG.debug("JMXServer is already running.");
        return;
    }
    //使用 LocateRegistry 来创建注册器
    startRmiRegistry(port);
    //根据 JMXConnectorServerFactory 来创建 JMXConnectorServer
    startJmxService(port);
    this.port = port;
}
```