第十八讲
Node-RED（II）
Lecture 18
Node-RED (II)

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit for the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for remedy including deletion have you had any concern.

上下文栏
Context Tab/Sidebar

- 在讲上下文栏之前，我们先考虑一个例子。对于一个温控系统，假设我们想知道前后两次采样时，温度的变化。如果在Node-RED实现该功能，显然上次采样的温度值是不能基于消息的，因为直观上，基于消息的上下文是基于当前消息的。因此，我们需要合适的存储机制与合适的存储范围来解决该问题。用我们接下来要讲的上下文机制，便可以较好地解决这个问题。

  Before we address the context tab or sidebar, we first consider an example to motivate the seeking for a proper solution. For a temperature control system, if we intend to know the temperature change between the subsequent two sampling steps, and require to implement it in Node-RED. First, it is obvious that to associate the last temperature value to the message is not very appropriate, due to the fact that in terms of message context, it always refers to the current message. Therefore, we need appropriate storage mechanism and storage scope to solve the problem. Actually, the context mechanism we discuss below can solve the problem well.

上下文栏
Context Tab/Sidebar

- 尽管Node-RED给上下文的定义是独立于流的存储信息的地方，实际上在Node-RED是存于内存或文件系统中的。由于这些独立于流的信息在存储时的组织是分层的，因此，当决定存储消息时，需要首先决定存储的作用域或可见域，或者说存储的分级。

  The definition of context by Node-RED is a place to store information outside the flow of messages, but actually the information is got stored in memory or local file system. However, since the storage of these flow-independent information is arranged in a hierarchical way, the first thing to decide is the scope or visibility of the stored information, or alternatively, their levels in the hierarchy.

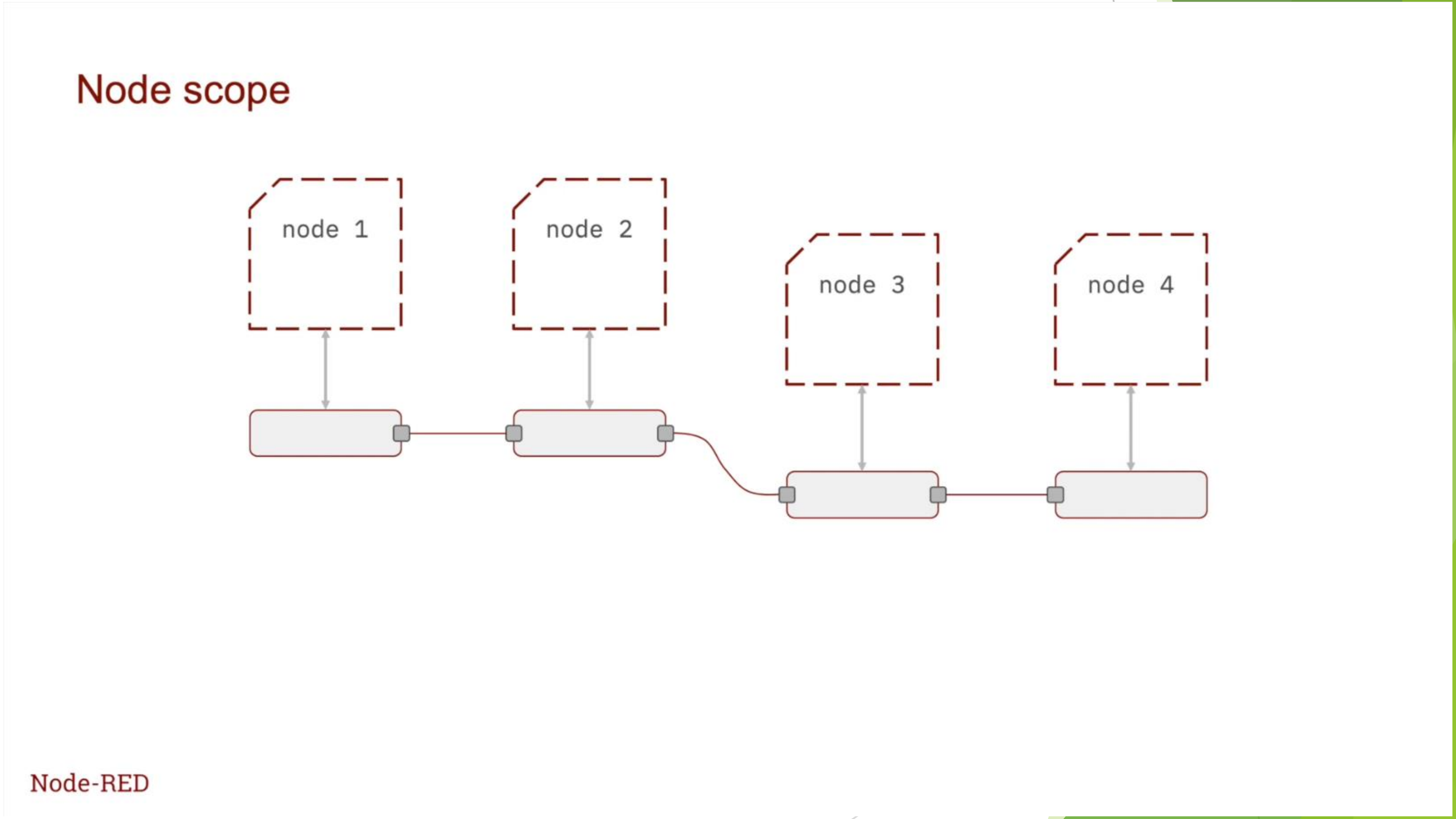
- 在Node-RED中，存储的作用域有三类：
  - 节点级别
  - 流级别
  - 全局

- There are three types of scope Node-RED in terms of context
  - Node scope
  - Flow scope
  - Global scope

上下文栏
Context Tab/Sidebar

- 节点或消息级别的上下文中的值，仅对设置该值的节点可见。

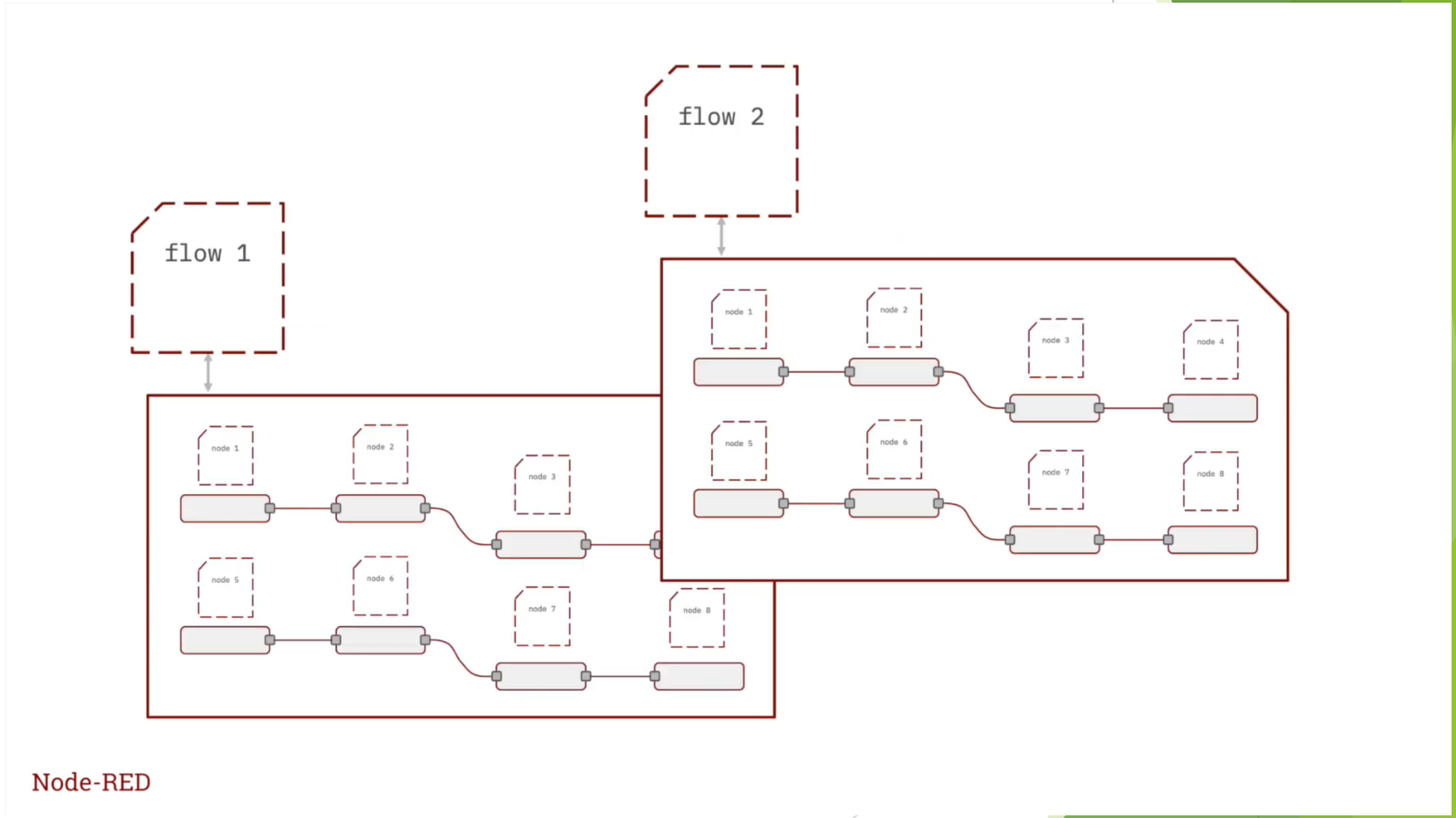
  The value in the node or message scope is only visible to the node that sets the value.



上下文栏
Context Tab/Sidebar

- 流级别的上下文中的值，对设置该值的流中的所有节点可见。

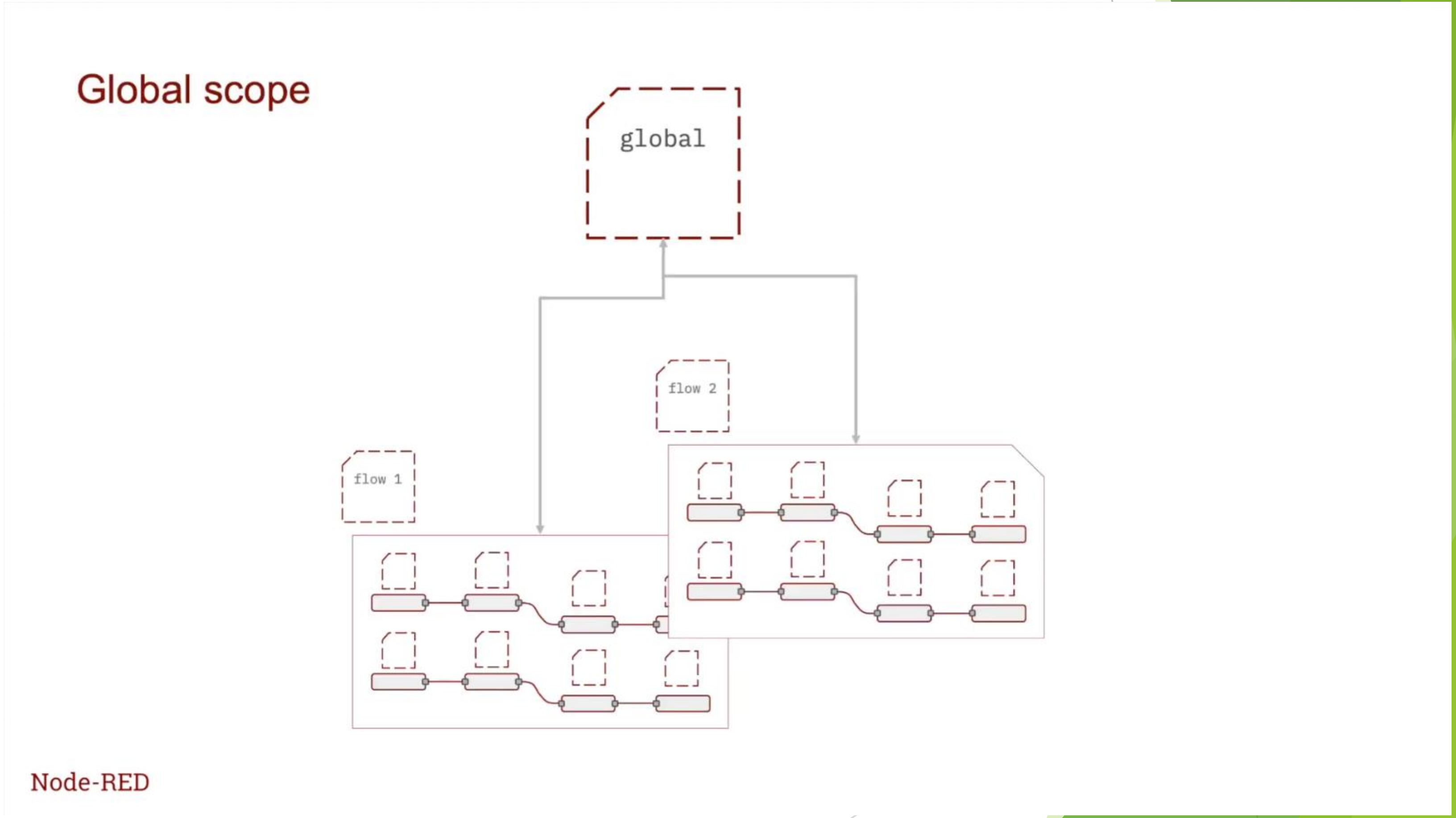
  The value in the flow scope is visible to all nodes in the flow where the context value is got set.



上下文栏
Context Tab/Sidebar

- 全局级别的上下文中的值，对所有流中的所有节点可见。

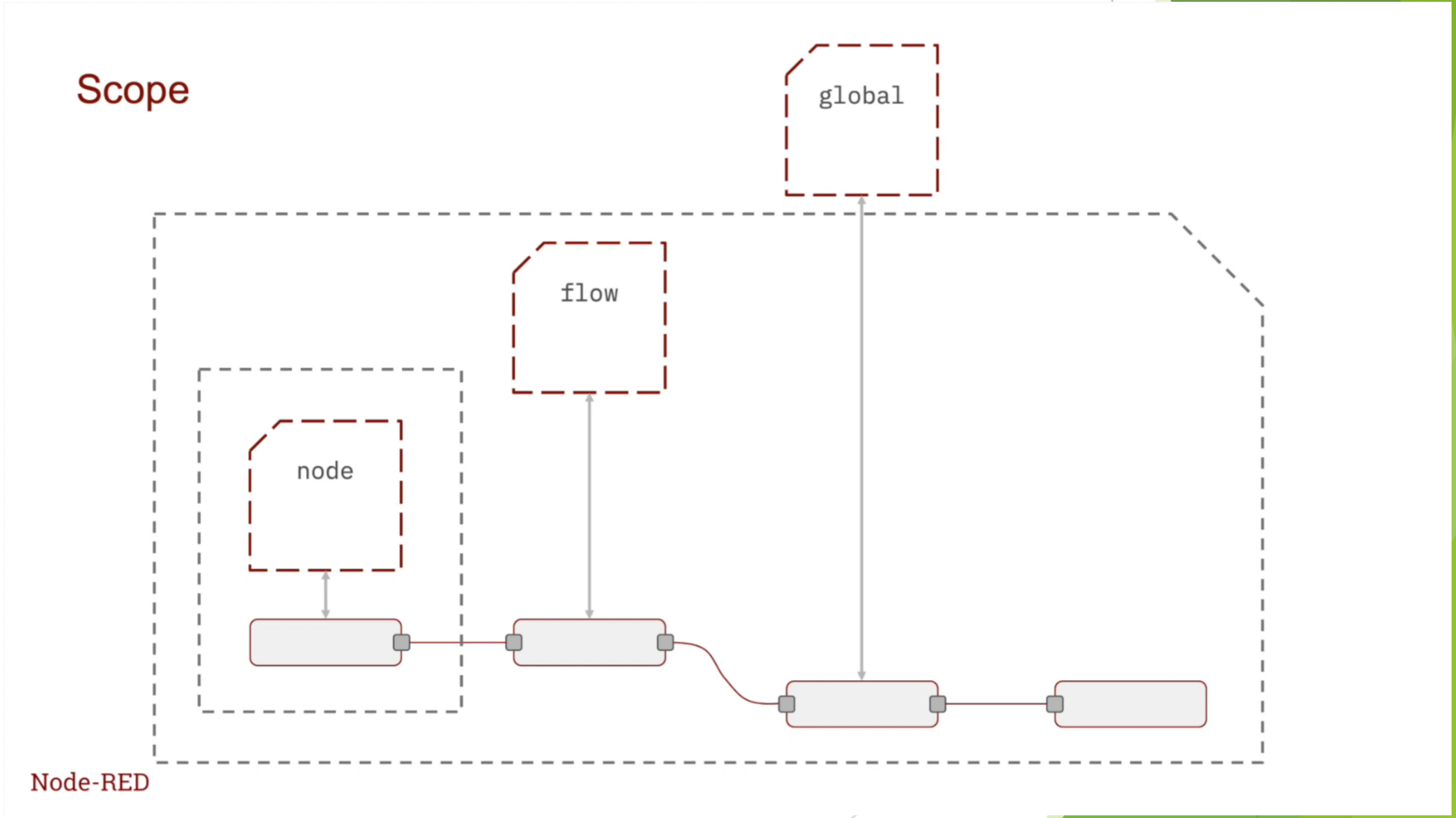
  The value in the global scope is visible to all nodes in all flows.



上下文栏
Context Tab/Sidebar

- 右图展示了不同级别的作用域的比较。

  The right figure shows the comparison of different scopes.

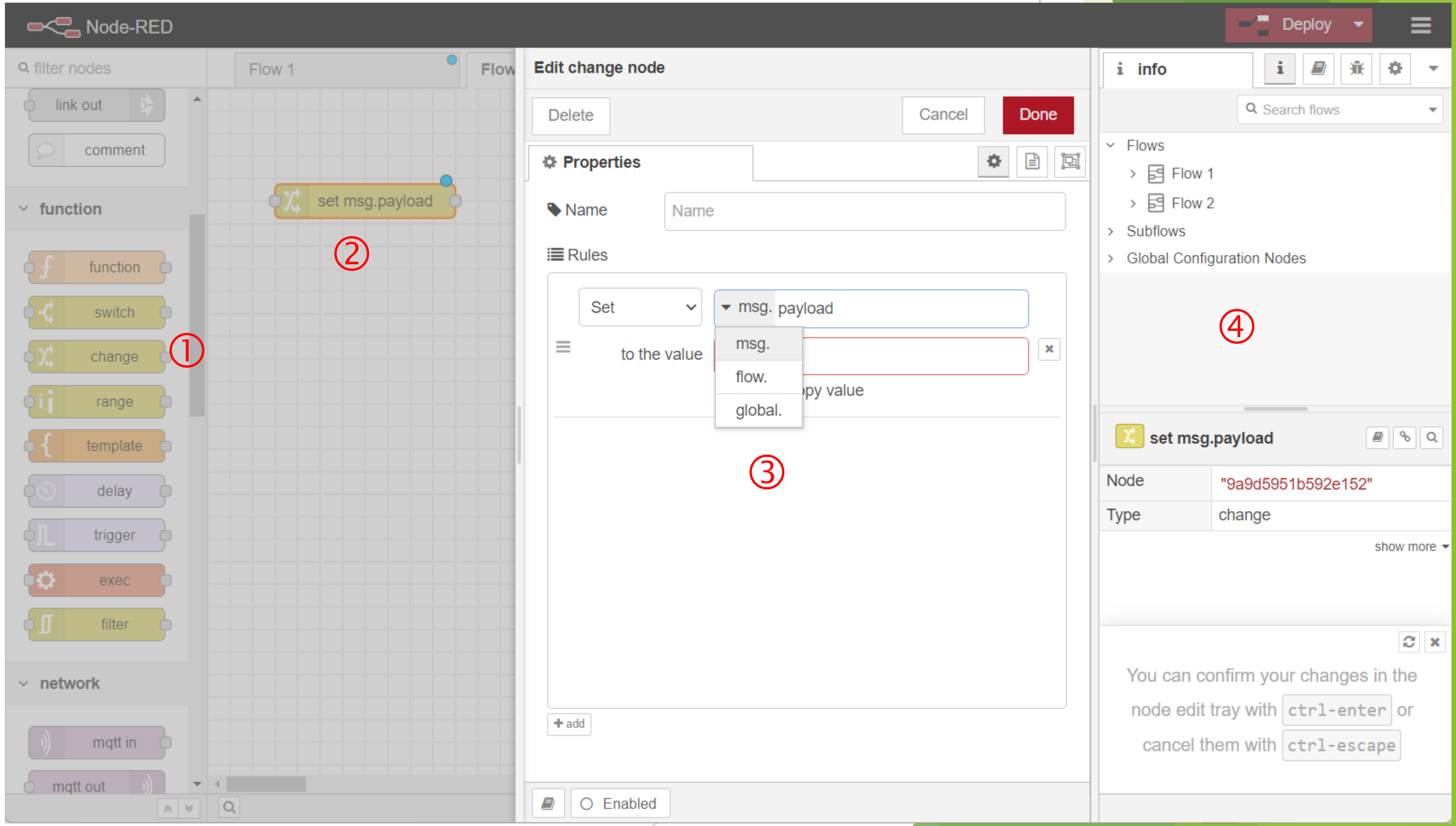


上下文栏
Context Tab/Sidebar

如图所示，当将变换节点拖至工作区并打开之后，在编辑对话框可以看出，可以将当前消息关联的值设置成合适的作用域的上下文值。As indicated in the figure below, after dragging the change node into the workspace, users can double-click it to open the editing dialog. To set context value is straightforward, just set some value associated the current message into some value of some variable in an appropriate scope.

- 下面我们来逐步实现前面所讲的例子。虽然可以直接通过编程来操作上下文变量，但一种较为直接的方法是通过变换节点。

  We now try to implement the example aforementioned step by step. Although it is possible to manipulate the context values via programming, a simple way is to do it via the change node.



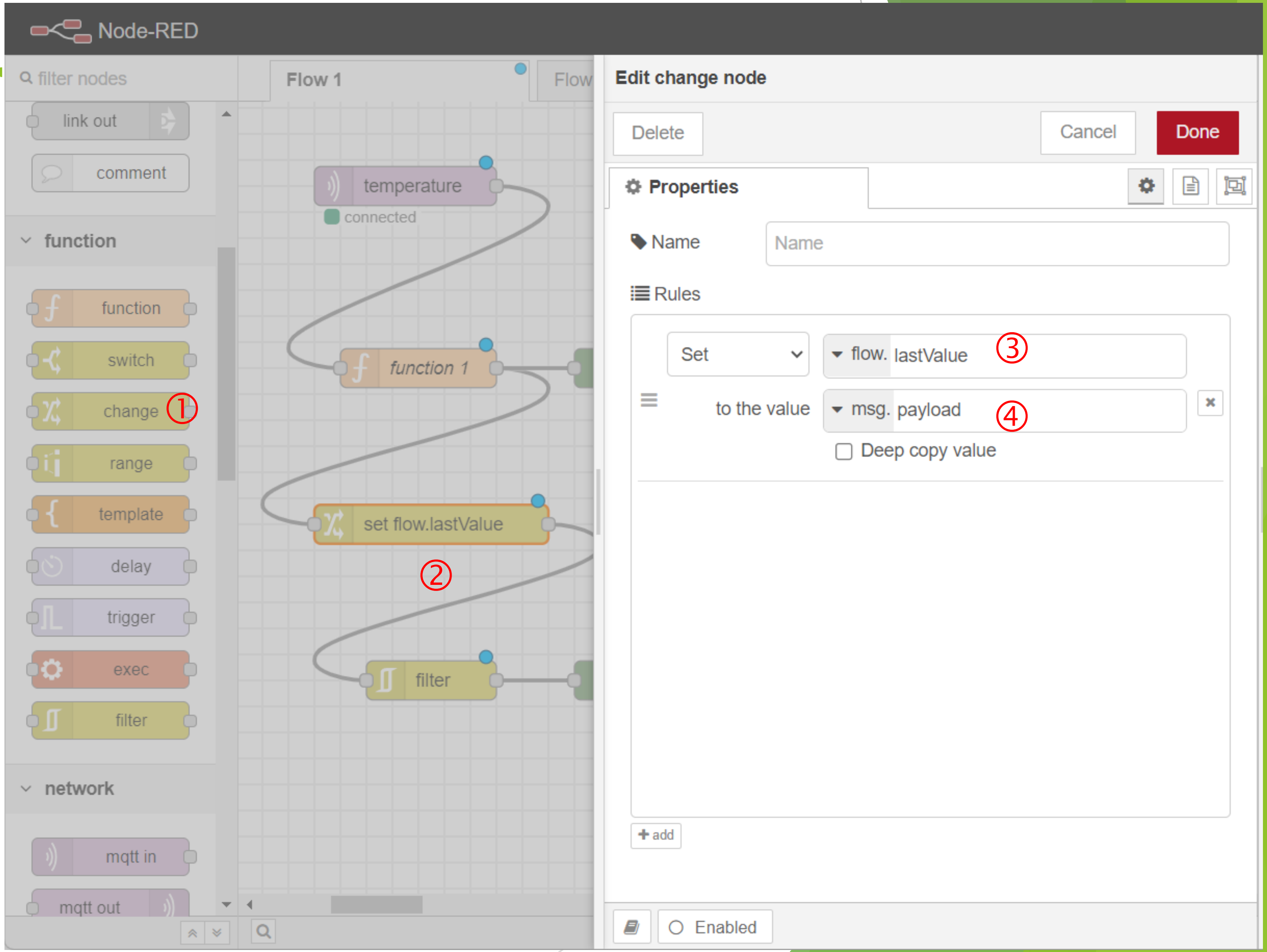
Node	"9a9d5951b592e152"
Type	change

You can confirm your changes in the node edit tray with ctrl-enter or cancel them with ctrl-escape

上下文栏
Context Tab/Sidebar

- 针对我们的例子，我们需要在合适的步骤，将当前消息的载荷，实际上是温度值，设置成上一时刻的温度值，以便当新的发布值到来时，进行比较，计算温度的变化。

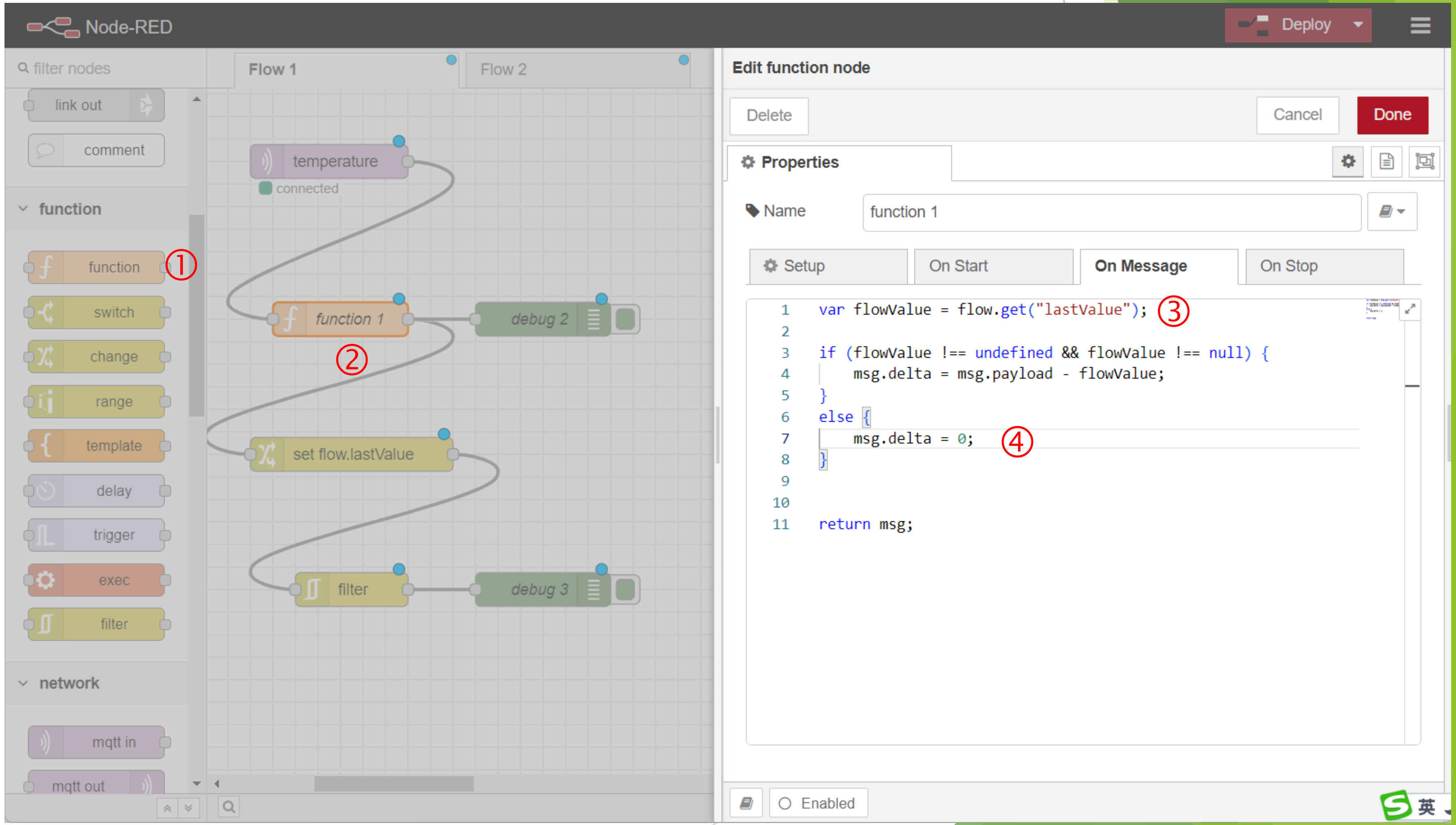
  For our example, we need to assign the payload of the current message as the value of the last temperature variable or key of message object. In this case, when a new publishing value arrives, we can do the comparison and calculate the change of the temperature.



上下文栏
Context Tab/Sidebar

- 在变换节点之前，我们添加一函数节点，其主要作用比较当前温度值与上一温度值并记录。注意，如果是第一个消息，则位于下游的变换节点尚未起作用，则将delta赋值为0。

  We insert an function node before the change node, where to compare the current temperature with previous one. Note for the very first time, the downstream change node has not taken effect, just assign 0 to delta this time.

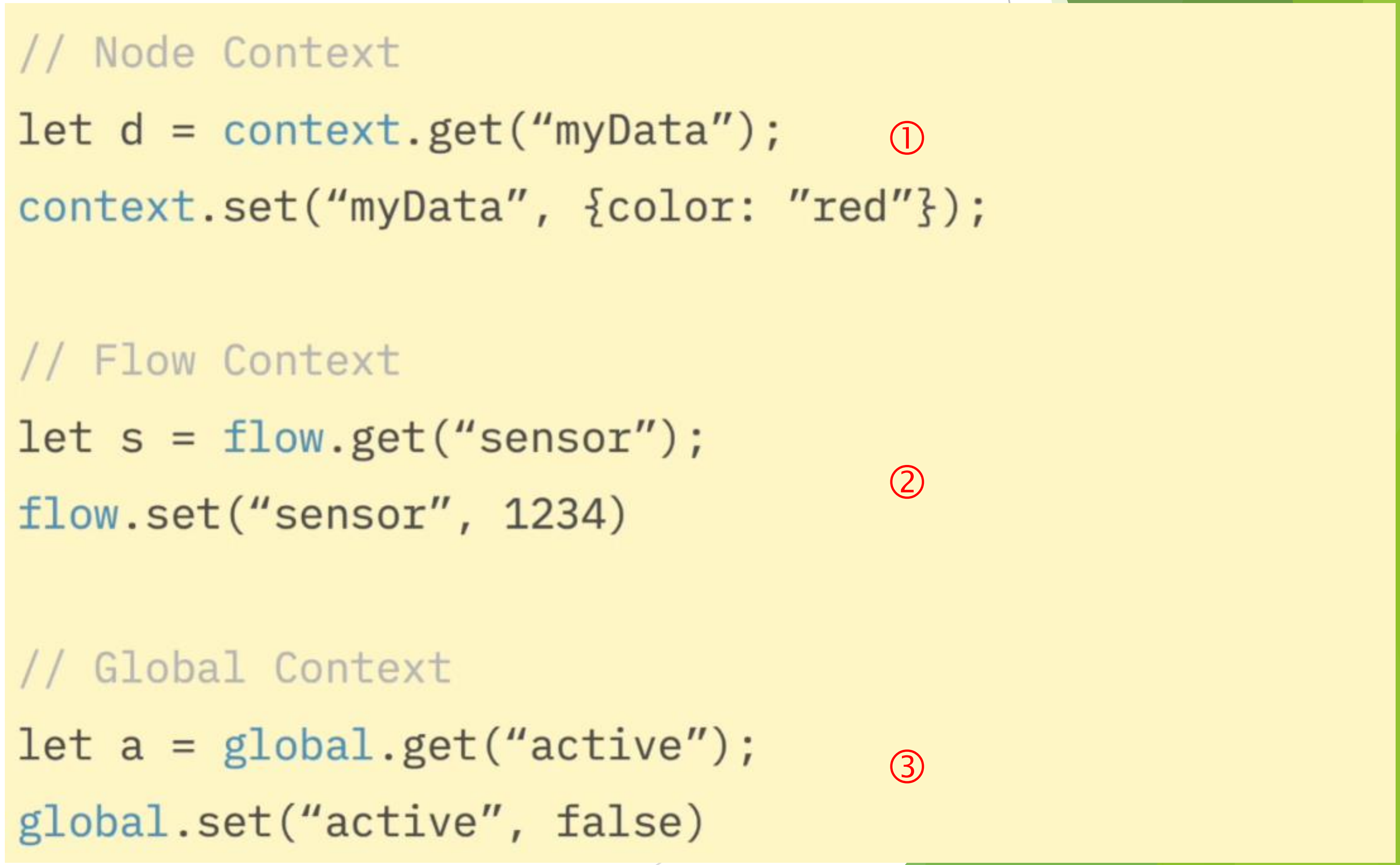


```
var flowValue = flow.get("lastValue");

if (flowValue !== undefined && flowValue !== null) {
    msg.delta = msg.payload - flowValue;
}
else {
    msg.delta = 0;
}


return msg;
```

上下文栏
Context Tab/Sidebar

- 在上面例子中，我们注意到除了通过变换节点，更为灵活的方式是通过编程，下面例子展示了通过编程，在不同的作用域获取或设置值的情况。

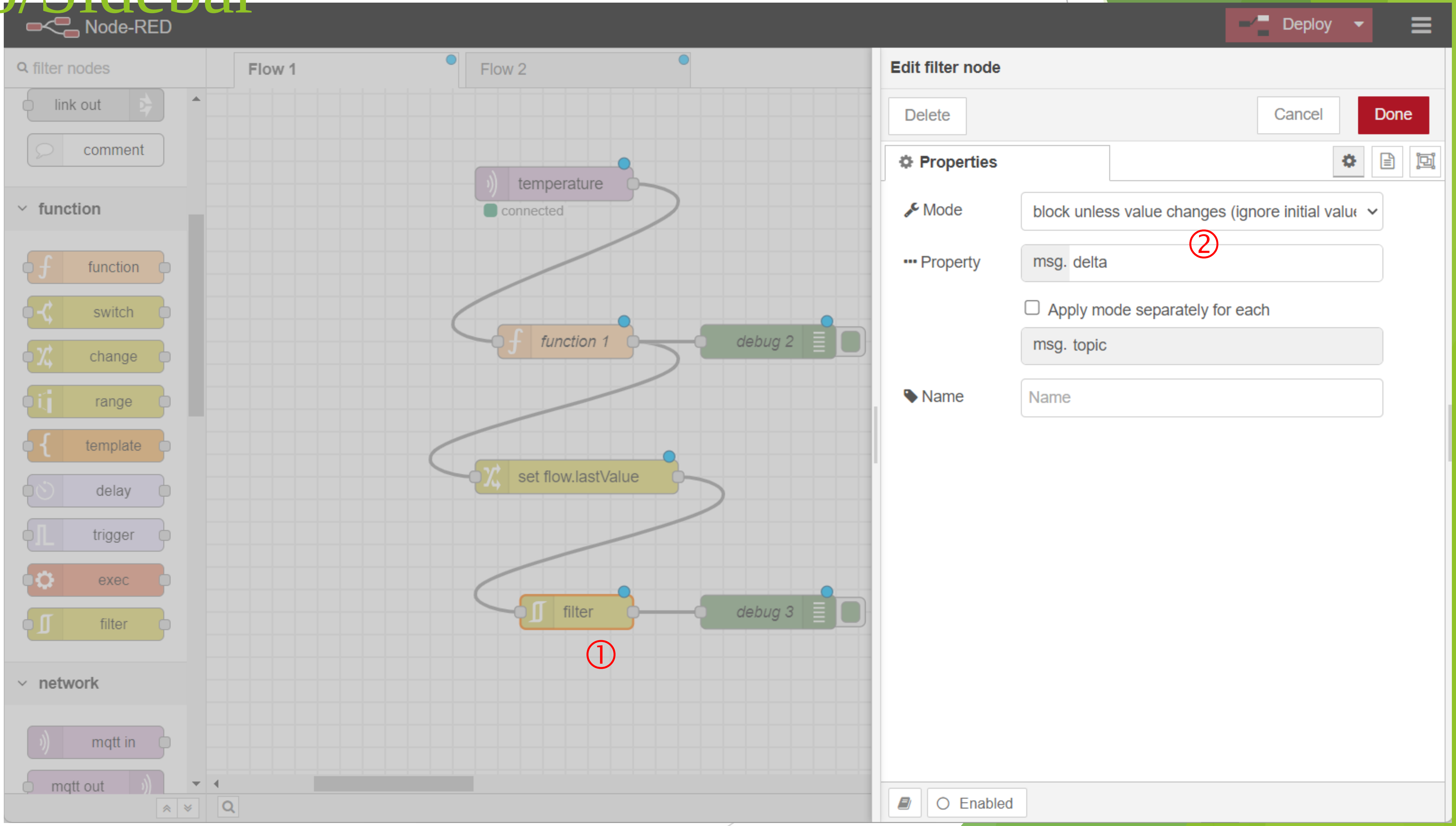
  In the above example, we notice besides using change node, a more flexible way is via programming. The right example demonstrates how to get and set context values in different scopes.



```
// Node Context
let d = context.get("myData");        ①
context.set("myData", {color: "red"});


// Flow Context
let s = flow.get("sensor");
                                      ②
flow.set("sensor", 1234)


// Global Context
let a = global.get("active");         ③
global.set("active", false)
```

上下文栏
Context Tab/Sidebar

- 另外，为使系统更加完善，我们填加了一个新的过滤节点。当温度变化值本身没有变化时，则阻断消息向下游传递。
- In addition, to make the system more comprehensive, we add an extra filter node. When there is no change of delta value itself, it will stop the message propagating downstream.

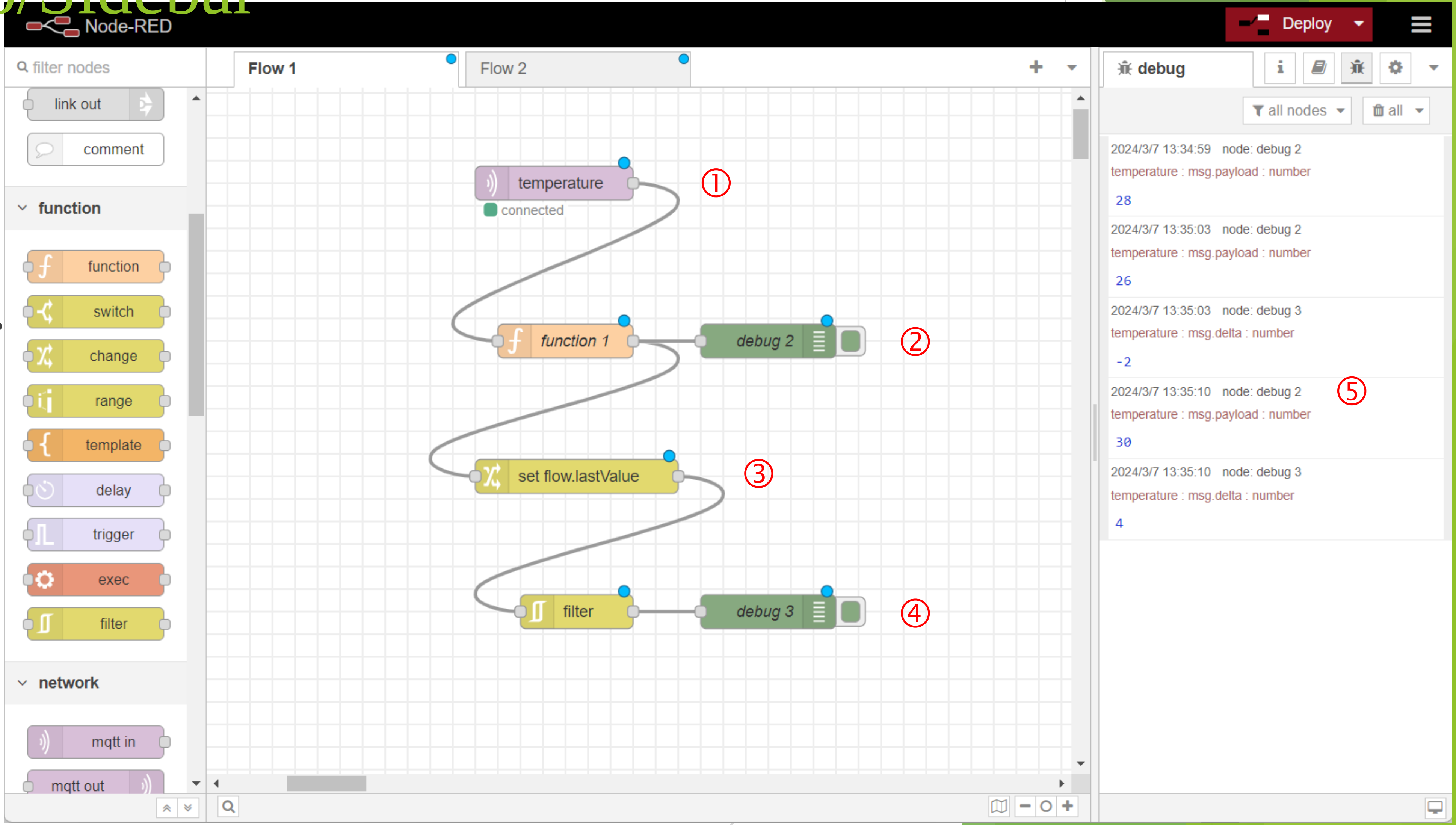


Mode: block unless value changes (ignore initial value)
Property: msg. delta
☐ Apply mode separately for each
msg. topic
Name: Name

上下文栏
Context Tab/Sidebar

- 基于上面讲述，整个系统流程如右所示。我们可以重复之前的测试，来验证系统的功能。相关输出信息如调试栏所示，证实了系统设计的正确性。

  Based on the description above, the overall flow is as the right figure. We can repeat the previous test to verify the system. The output form the debug tab shows the correctness of the system design.

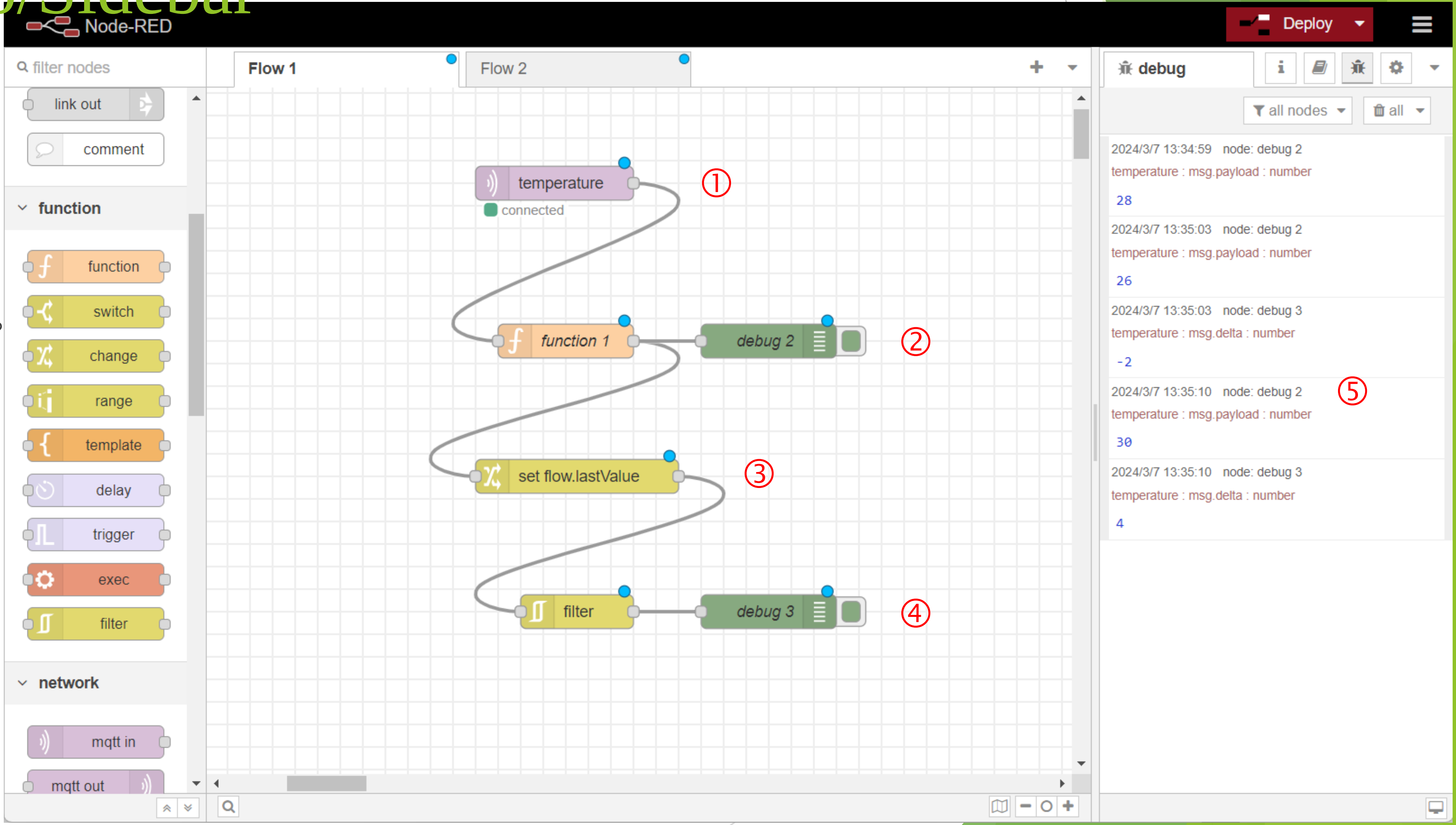


```
2024/3/7 13:34:59   node: debug 2
temperature : msg.payload : number
28
2024/3/7 13:35:03   node: debug 2
temperature : msg.payload : number
26
2024/3/7 13:35:03   node: debug 3
temperature : msg.delta : number
-2
2024/3/7 13:35:10   node: debug 2
temperature : msg.payload : number
30
2024/3/7 13:35:10   node: debug 3
temperature : msg.delta : number
4
```

上下文栏
Context Tab/Sidebar

- 基于上面讲述，整个系统流程如右所示。我们可以重复之前的测试，来验证系统的功能。相关输出信息如调试栏所示，证实了系统设计的正确性。

  Based on the description above, the overall flow is as the right figure. We can repeat the previous test to verify the system. The output form the debug tab shows the correctness of the system design.

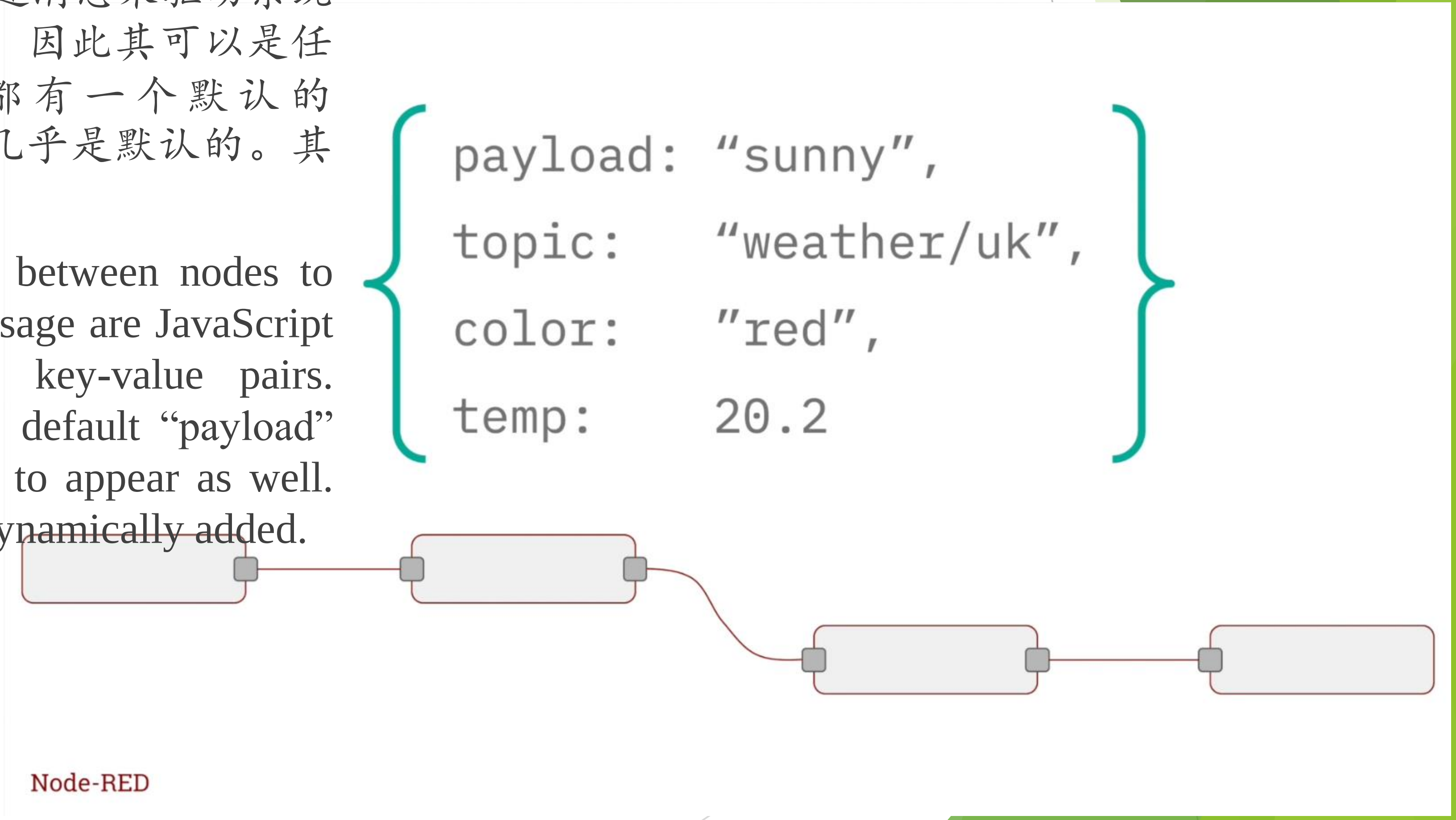


```
2024/3/7 13:34:59   node: debug 2
temperature : msg.payload : number
 28
2024/3/7 13:35:03   node: debug 2
temperature : msg.payload : number
 26
2024/3/7 13:35:03   node: debug 3
temperature : msg.delta : number
 -2
2024/3/7 13:35:10   node: debug 2
temperature : msg.payload : number
 30
2024/3/7 13:35:10   node: debug 3
temperature : msg.delta : number
 4
```

消息
Message

- 在Node-RED里，通过在节点间传递消息来驱动系统工作。由于消息是JavaScript对象，因此其可以是任意合法的键值对。通常消息都有一个默认的“payload”键，当然“topic”也几乎是默认的。其他键值可视逻辑情况进行添加。

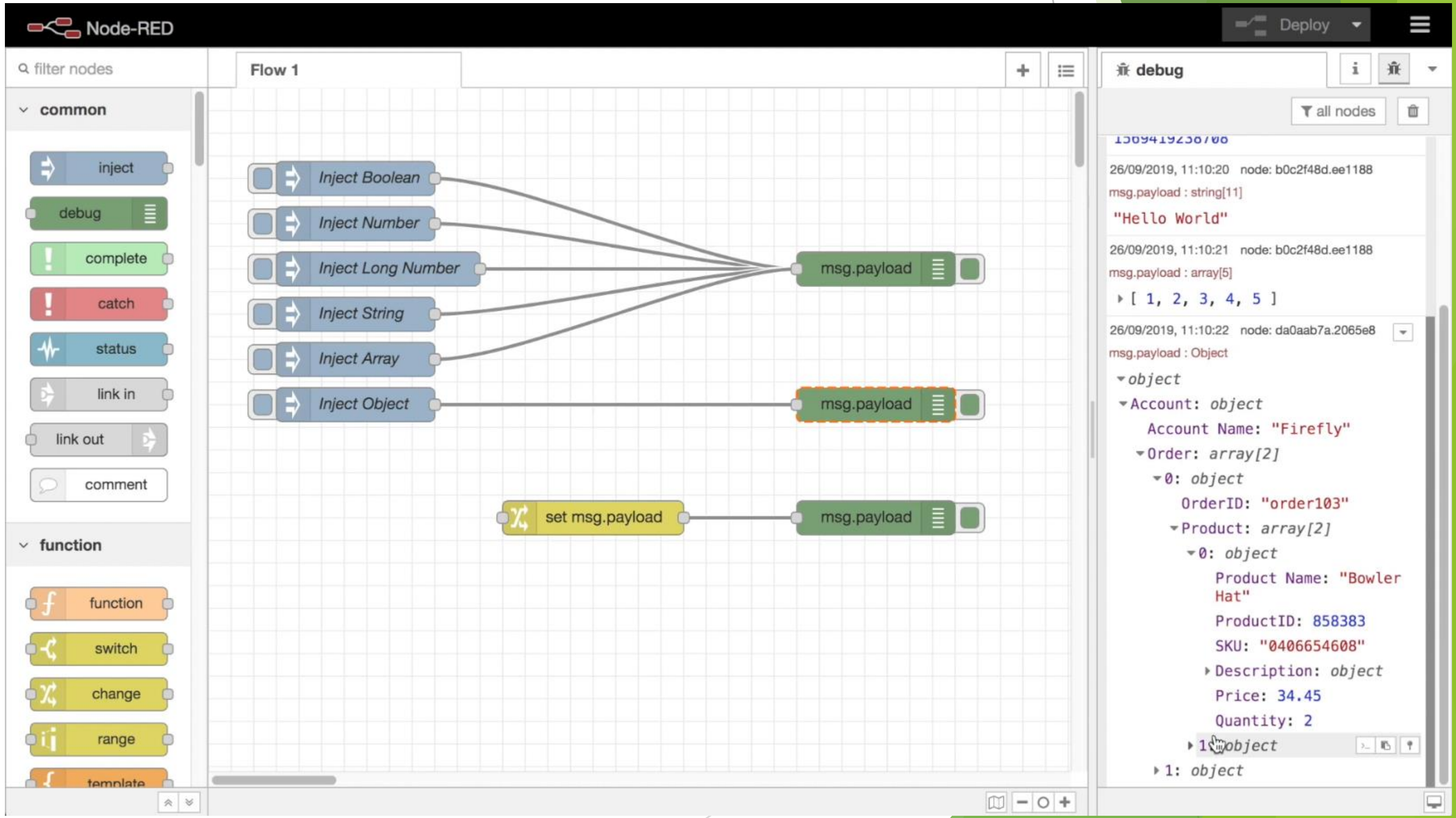
  In Node-RED, message are passed between nodes to drive the system to work. Since message are JavaScript object, it consists of any valid key-value pairs. Generally, all message contains the default “payload” key, although the “topic” key tends to appear as well. For other key-value pairs, it can be dynamically added.


```
{
  payload: “sunny”,
  topic:   “weather/uk”,
  color:   ″red″,
  temp:    20.2
}
```

消息
Message

- 通常查看消息的结构的最简单的方式，是通过调试工具栏来查看。用户通过单击被解析的JavaScript对象（可能是嵌套的），可以展开查看具体消息。

  The most convenient way to inspect the structure of a message is via the debug sidebar. By clicking the parsed object (maybe contains nested structures), the user can check the details of the message.

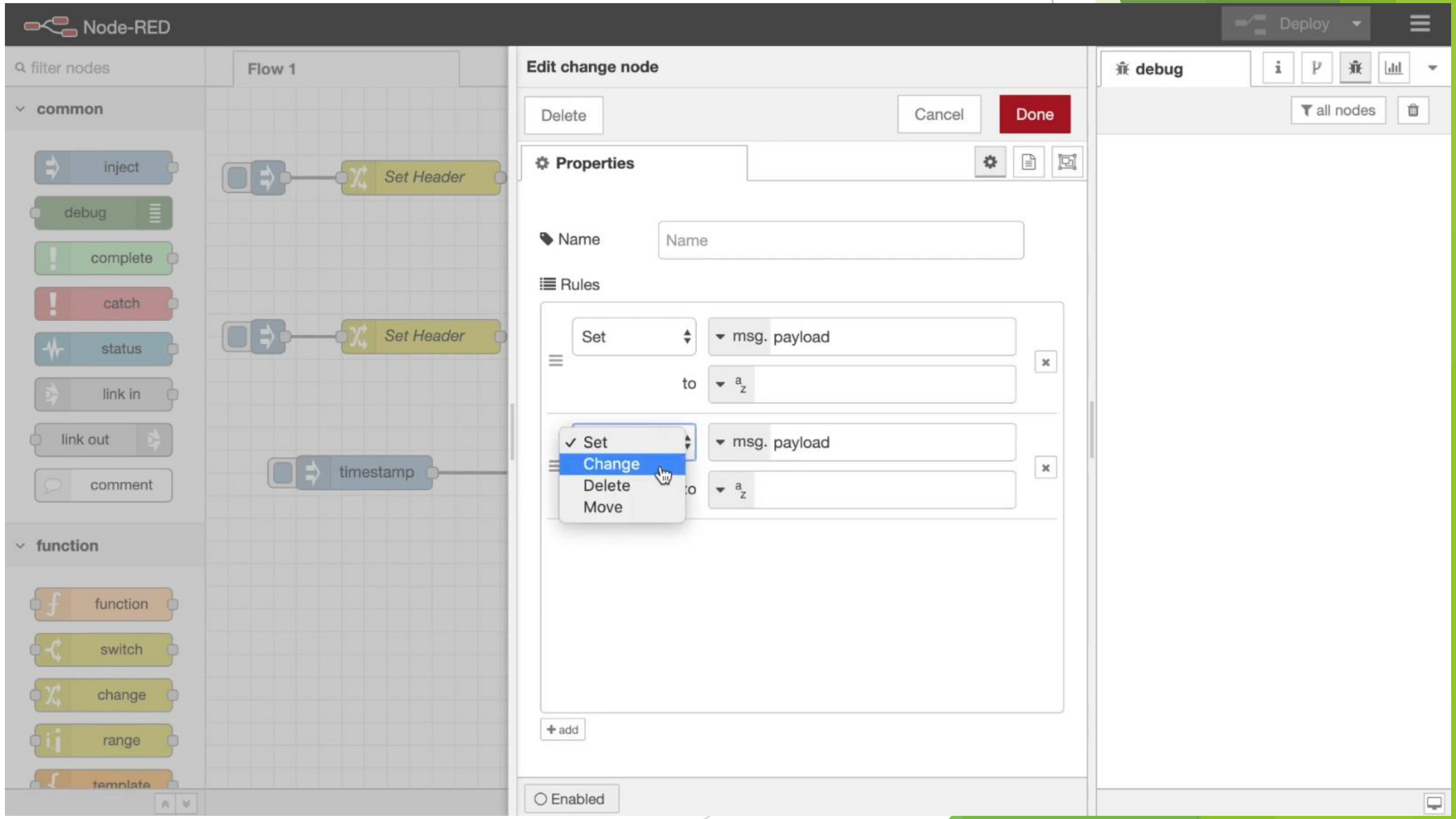


```
26/09/2019, 11:10:20   node: b0c2f48d.ee1188
msg.payload : string[11]
"Hello World"
26/09/2019, 11:10:21   node: b0c2f48d.ee1188
msg.payload : array[5]
▸[ 1, 2, 3, 4, 5 ]
26/09/2019, 11:10:22   node: da0aab7a.2065e8
msg.payload : Object
▾object
 ▾Account: object
    Account Name: "Firefly"
   ▾Order: array[2]
     ▾0: object
        OrderID: "order103"
       ▾Product: array[2]
         ▾0: object
            Product Name: "Bowler
            Hat"
            ProductID: 858383
            SKU: "0406654608"
           ▸Description: object
            Price: 34.45
            Quantity: 2
         ▸1: object
     ▸1: object
```

消息
Message

- 尽管可以通过编程的方式更改消息，但更为直接的方式是通过“变换”节点。其可以同时对消息的多个属性或键值对进行增删改，比较直观地反映流的节点对消息的加工情况。

  Although it is possible to modify the message via programming, however, a more direct way is to use the “change” node. It can simultaneously manipulate multiple key-value pairs or properties of the message.

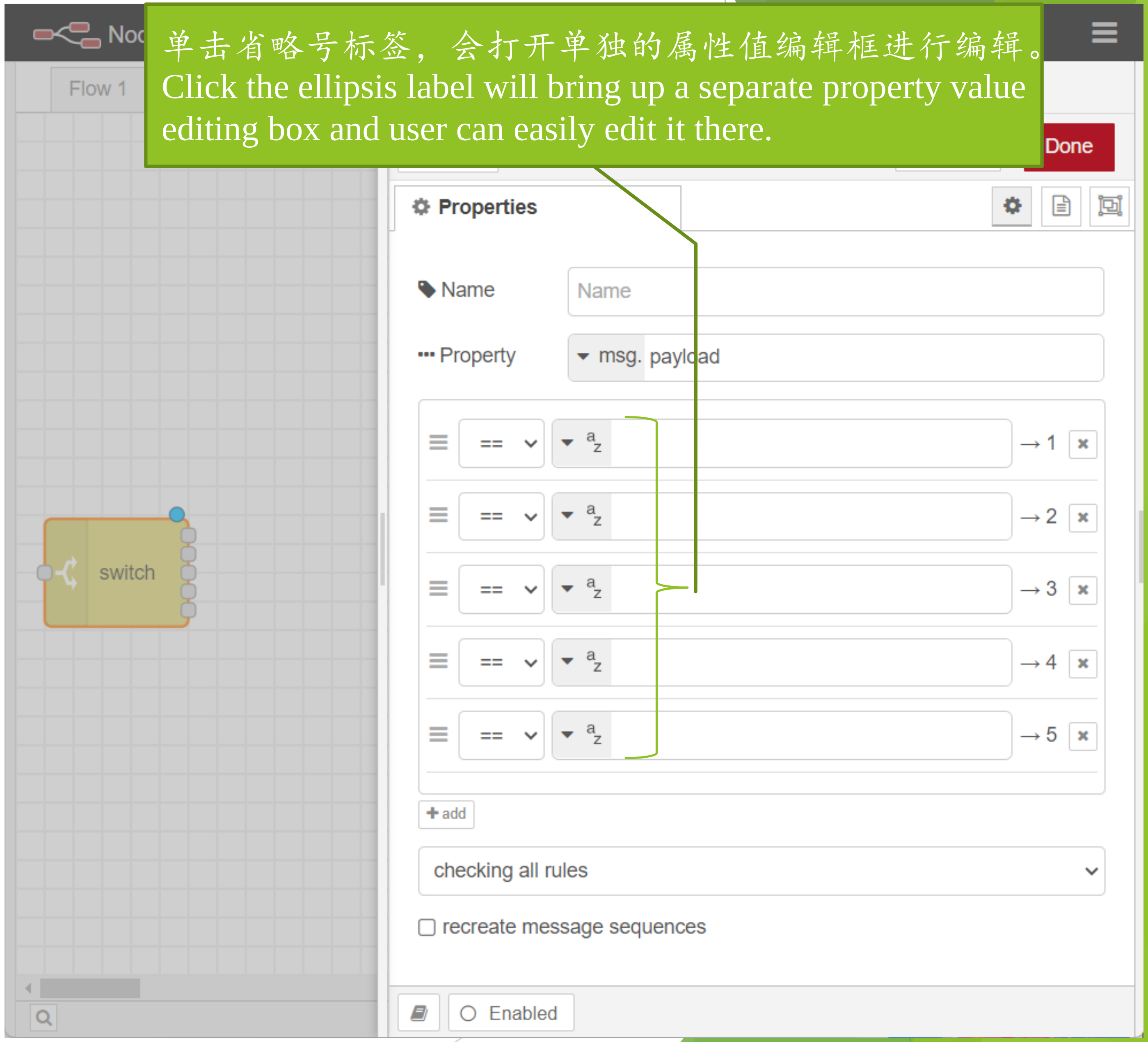


消息

Message

- 通常情况下，当前节点的消息可以传递给多个下游节点，此时，接收消息的所有节点得到的都是同一消息的拷贝。但某些情况下，需要根据不同的情况下游节点互斥地得到消息，此时需要用到交换节点。

  Generally, the current node can pass the message simultaneously to many downstream nodes. However, all the sent out messages are of the same copy. However, in some scenarios, downstream nodes should receive the message exclusively. In this case, one can use the switch node to dispatch the message properly.

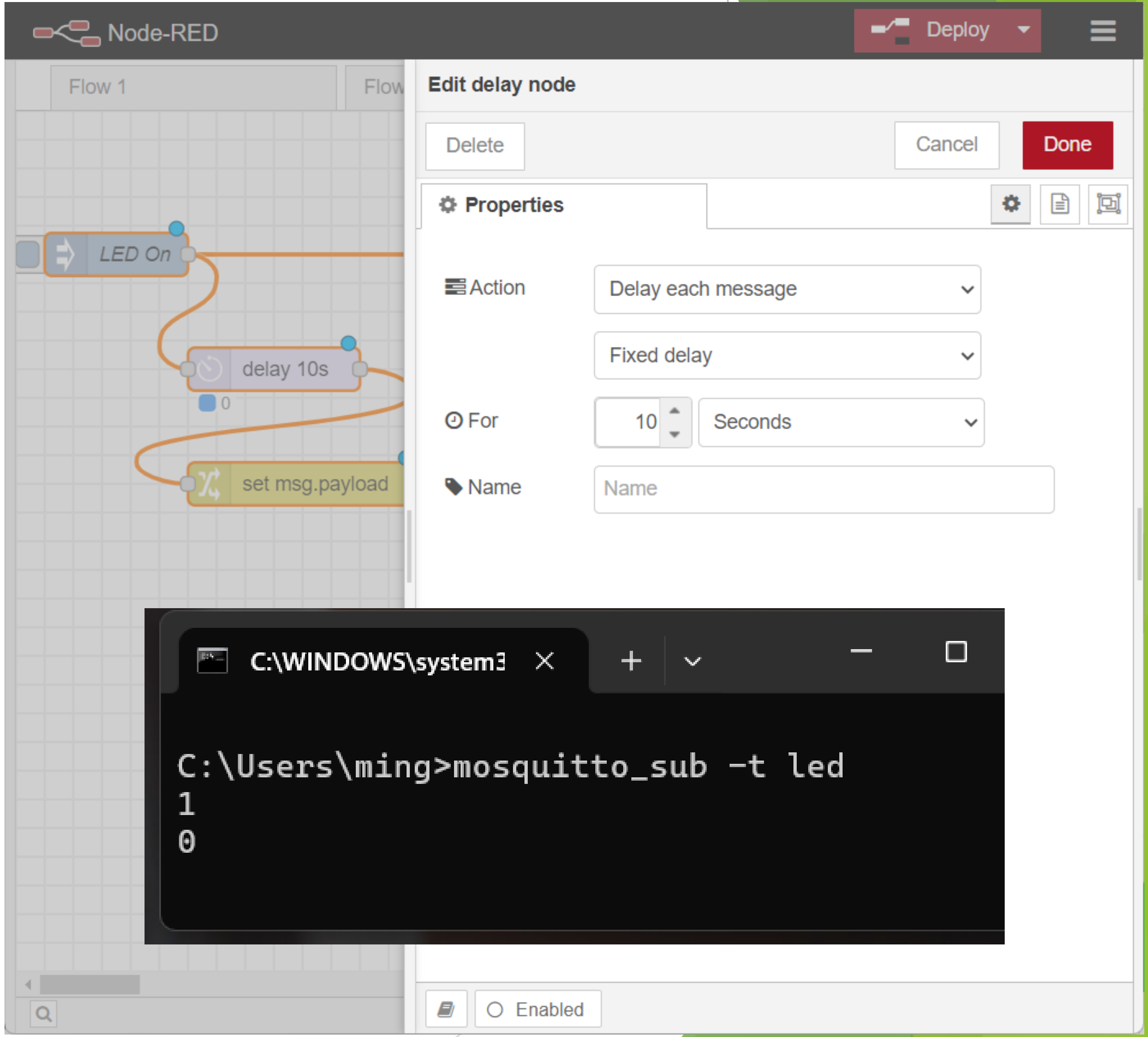


单击省略号标签，会打开单独的属性值编辑框进行编辑。
Click the ellipsis label will bring up a separate property value editing box and user can easily edit it there.

消息
Message

- 实际上，Node-RED提供了丰富的节点供对消息流进行加工。例如，对于延时控制的例子，例如，灯亮10秒后自动熄灭。通常，此种情况下相关消息的绝大多数属性都相同，但仅载荷的值变化。那么可以用延时节点与变换节点达到目的。

  Actually, Node-RED provides rich nodes for manipulating the message flow. For example, consider the delay-control scenario, such as turn off the light after 10 seconds after lit it up. In this condition, the relevant messages share almost all the properties, just the payloads varies in opposite cases. Now, the user can use the delay node and change node for this purpose.



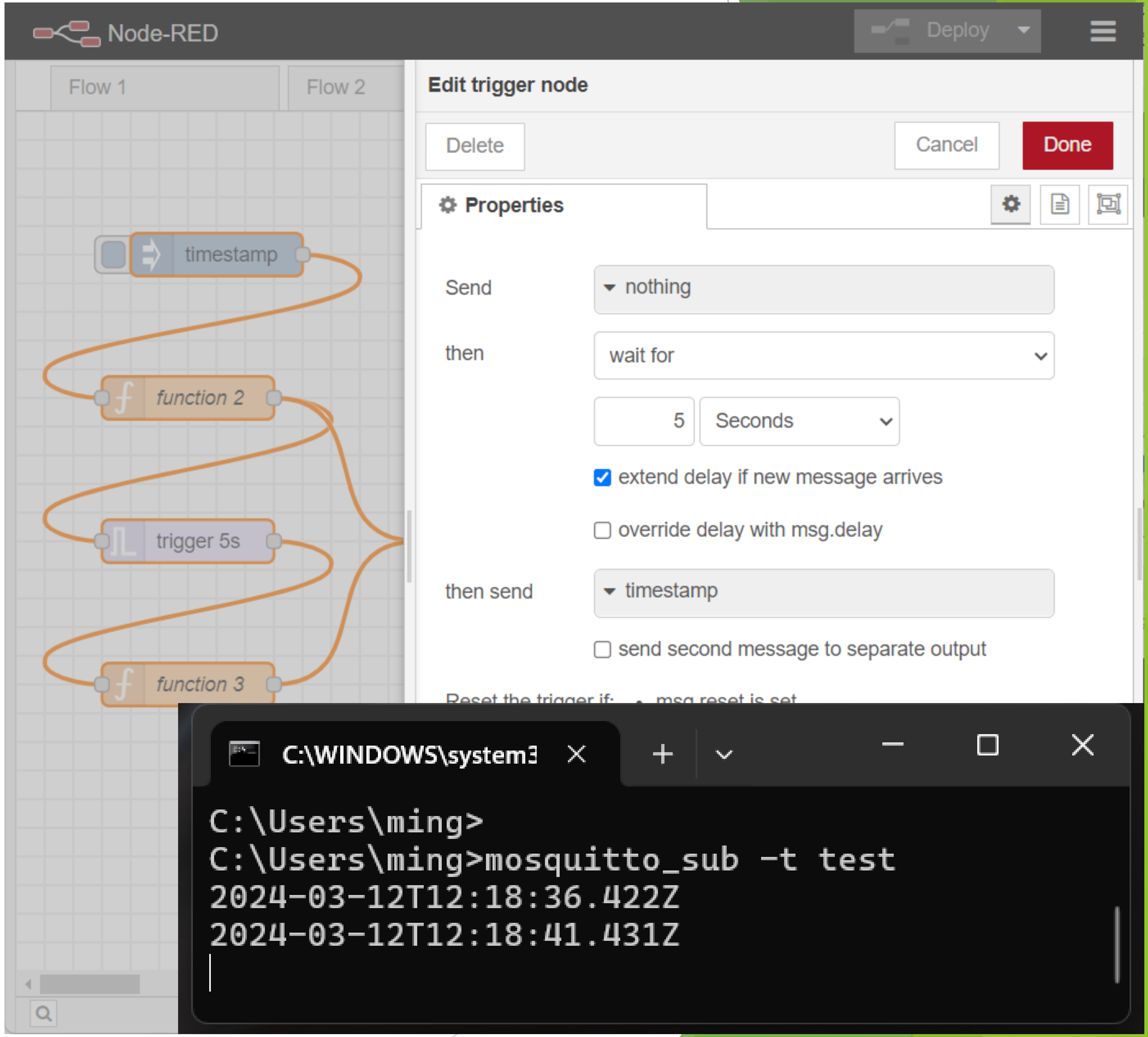
Action: Delay each message
Fixed delay
For: 10 Seconds
Name: Name

```
C:\Users\ming>mosquitto_sub -t led
1
0
```

消息 Message

- 上面的功能也可以通过另外的节点实现，例如触发节点。实际上，触发节点有更为灵活的配置方式，例如，可以选择部署时是否发送消息，还是等到消息时再发送。同时，可以决定当被触发时，具体发送什么。这里选择发送时间戳。

  The function in the last slide can also be fulfilled by other types of nodes, for example trigger node. In fact, trigger node can be more versatile in terms of configuration. For example, users can choose whether to send out message upon deployment, or postpone to the arrival of upstream message. At the same time, the user can decide the content to be sent out when triggered.



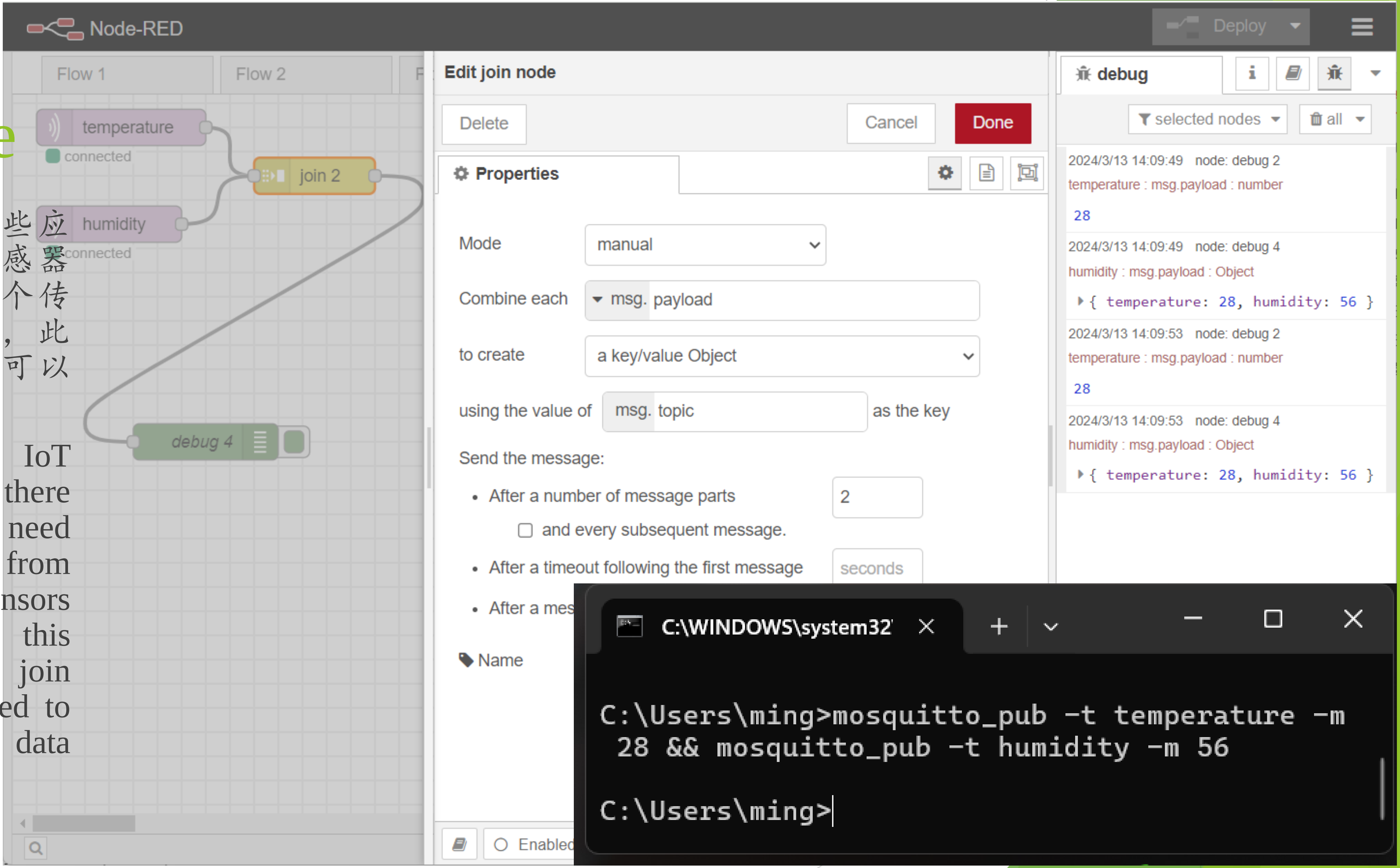
Send: nothing
then: wait for
5 Seconds
[✓] extend delay if new message arrives
[ ] override delay with msg.delay
then send: timestamp
[ ] send second message to separate output
Reset the trigger if: • msg.reset is set

```
C:\Users\ming>
C:\Users\ming>mosquitto_sub -t test
2024-03-12T12:18:36.422Z
2024-03-12T12:18:41.431Z
```

消息
Message

- 在物联网的某些应用中，会有传感器融合，即将多个传感器数据合并，此通过合并节点可以完成相关功能。

  In some IoT applications, there are might be a need to merge data from multiple sensors together. In this scenario, the join node can be used to combine data together.



Mode: manual
Combine each: msg. payload
to create: a key/value Object
using the value of msg. topic as the key
Send the message:
- After a number of message parts 2
  - [ ] and every subsequent message.
- After a timeout following the first message seconds
- After a mes
Name

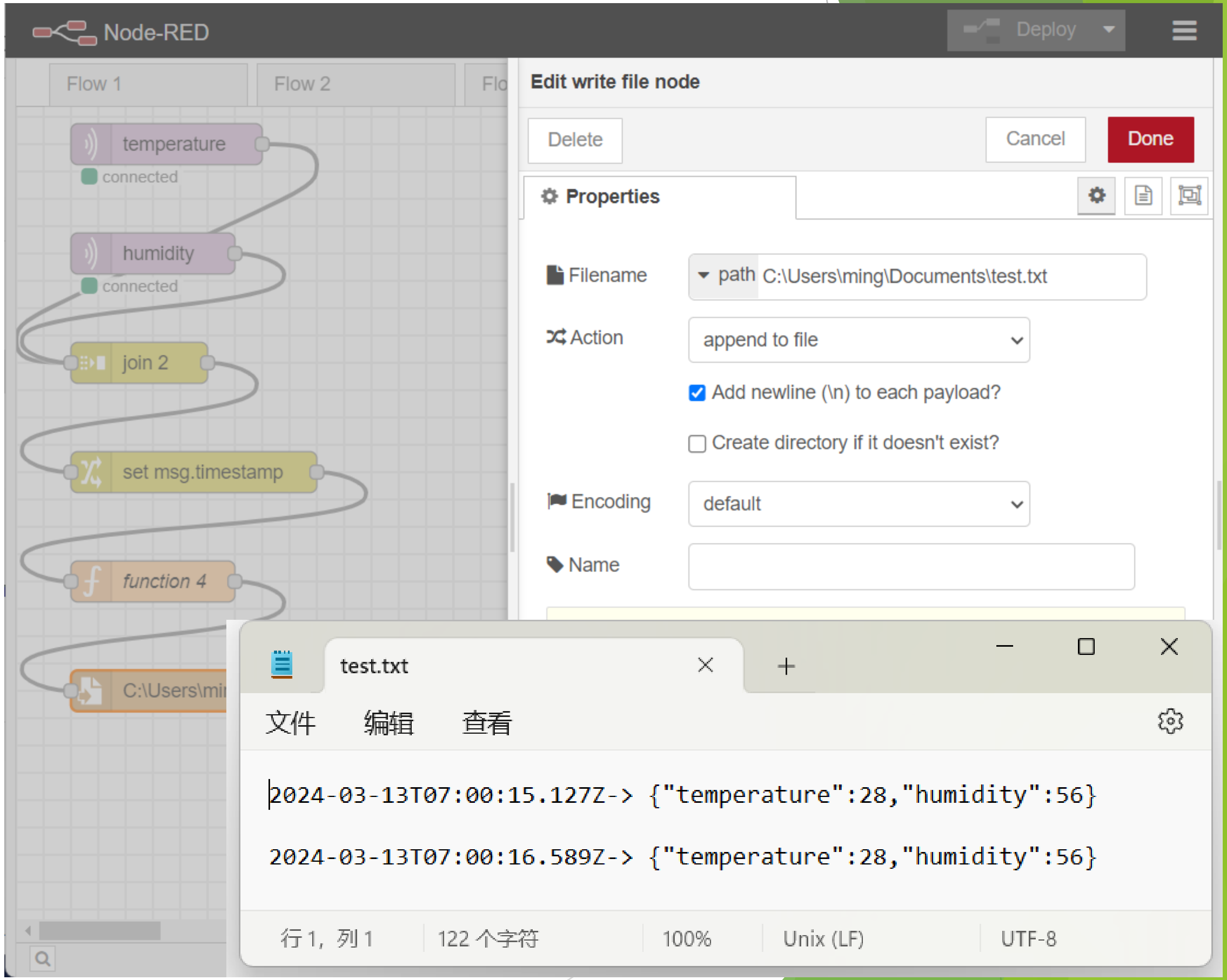
```
2024/3/13 14:09:49 node: debug 2
temperature : msg.payload : number
28
2024/3/13 14:09:49 node: debug 4
humidity : msg.payload : Object
▸{ temperature: 28, humidity: 56 }
2024/3/13 14:09:53 node: debug 2
temperature : msg.payload : number
28
2024/3/13 14:09:53 node: debug 4
humidity : msg.payload : Object
▸{ temperature: 28, humidity: 56 }
```

```
C:\WINDOWS\system32

C:\Users\ming>mosquitto_pub -t temperature -m
 28 && mosquitto_pub -t humidity -m 56

C:\Users\ming>
```

消息
Message

- 通常情况下，消息的逻辑处理是在内存中进行的，但有些情况下，需要将消息持久化到非易失性介质，如硬盘，这可以通过写节点完成。当然，相反的操作可以通过读节点完成。

  Generally, the processing of message is backed up by the memory. However, in some cases, it is necessary to store the message in non-volatile media, for example, hard disk. This can be fulfilled by the write node. In addition, the reverse operation can be fulfilled by the read node.



Filename: path C:\Users\ming\Documents\test.txt
Action: append to file
[✓] Add newline (\n) to each payload?
[ ] Create directory if it doesn't exist?
Encoding: default
Name:

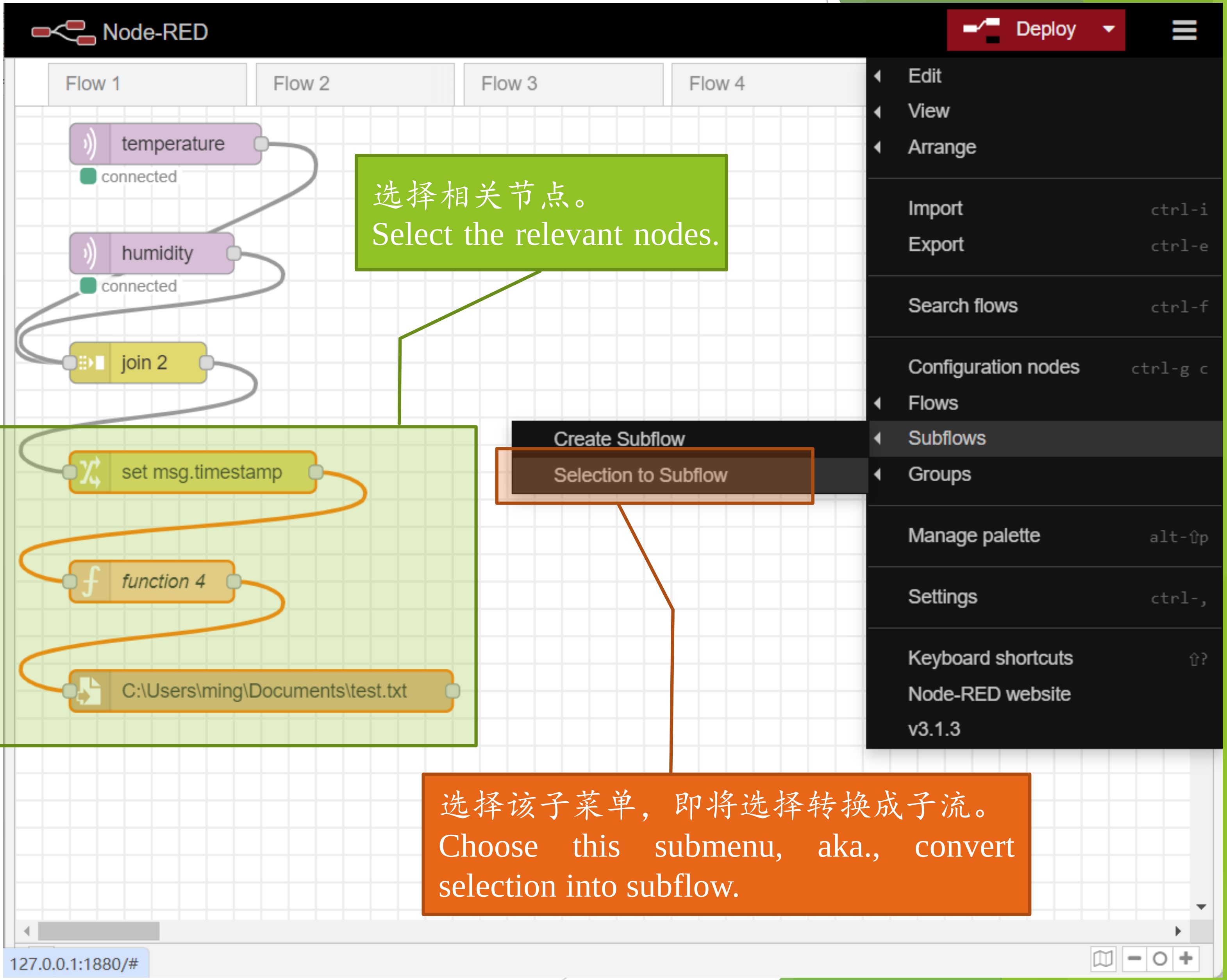
```
2024-03-13T07:00:15.127Z-> {"temperature":28,"humidity":56}
2024-03-13T07:00:16.589Z-> {"temperature":28,"humidity":56}
```

行 1，列 1 | 122 个字符 | 100% | Unix (LF) | UTF-8

消息
Message

- 实际上，持久化是一个非常常见的工作，可能在许多场合都用的到。在上例中，持久化之前还添加了时间戳。因此，可以将相关节点选中，创建一个子流，重复使用。

  Actually, persistency is a quite common operation, which is needed in many scenarios. In the example above, timestamp is also added before storage. Therefore, to select all the relevant nodes and based on them to create a sub-flow can facilitate the repeating use later.



选择相关节点。
Select the relevant nodes.

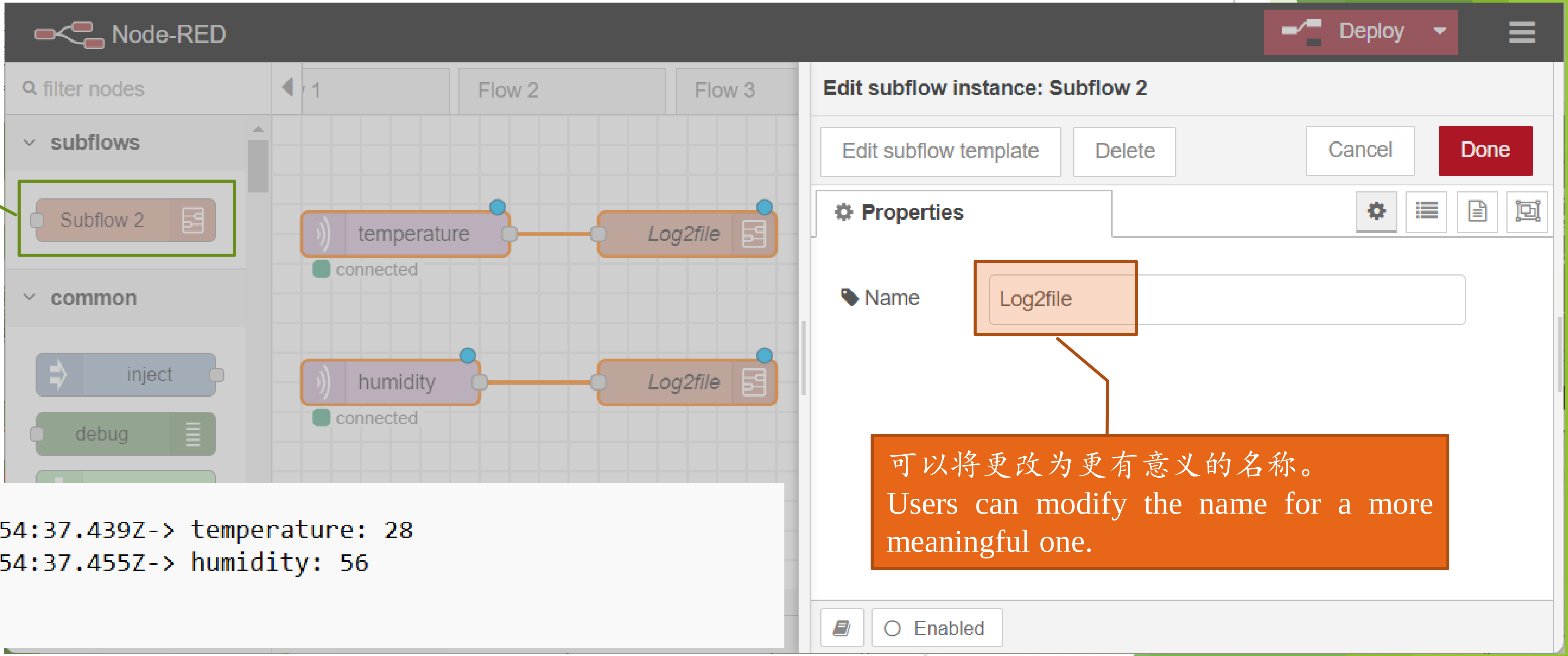
选择该子菜单，即将选择转换成子流。
Choose this submenu, aka., convert selection into subflow.

消息
Message

- 当创建子流后，在调色板中会有子流的节点。当需要时，只需要拖到工作区即可使用。

  There will be a new subflow node in the palette after creation. Users can simply drag as many the node into the workspace when needed.

选择相关节点。
Select the relevant nodes.



可以将更改为更有意义的名称。
Users can modify the name for a more meaningful one.

```
2024-03-13T07:54:37.439Z-> temperature: 28
2024-03-13T07:54:37.455Z-> humidity: 56
```