今日から使える！
Zabbix監視設定ノウハウの共有

～ 進化に合わせた監視設定 ～

2023年11月17日

NTTコム エンジニアリング株式会社

アジェンダ

- はじめに
- 事例1: 複数データ集計に苦労した話
- 事例2: SNMPトラップ監視の汎用的な監視設定
- まとめ

はじめに

会社紹介

NTTコムエンジニアリング株式会社

- NTTコミュニケーションズの１００％子会社
- 2008年より、Zabbix社と連携したZabbix関連事業を開始
- ZABICOMソリューションとしてZabbixを用いた構築から運用までの様々なサービスを提供

(https://www.nttceng.com/)

(https://www.zabicom.com/zabbix/index.html)

自己紹介

吉田　剛太
(よしだ　こうた)

2014年頃～

Zabbixと出会い、Zabbixに関連した業務を開始する

2016年9月～

Zabbixを中心とした監視運用部隊にて数百～数千の監視設定を行う毎日を過ごす

(手動での作業を嘆き、Zabbix APIを利用したコードを書く趣味に目覚める)

2019年7月～

ZABICOMチームに合流。Zabbixを中心としたソリューション業務を開始する

2020年2月

Zabbix認定エキスパート資格試験に挑戦、一発合格する

そして2023年現在～

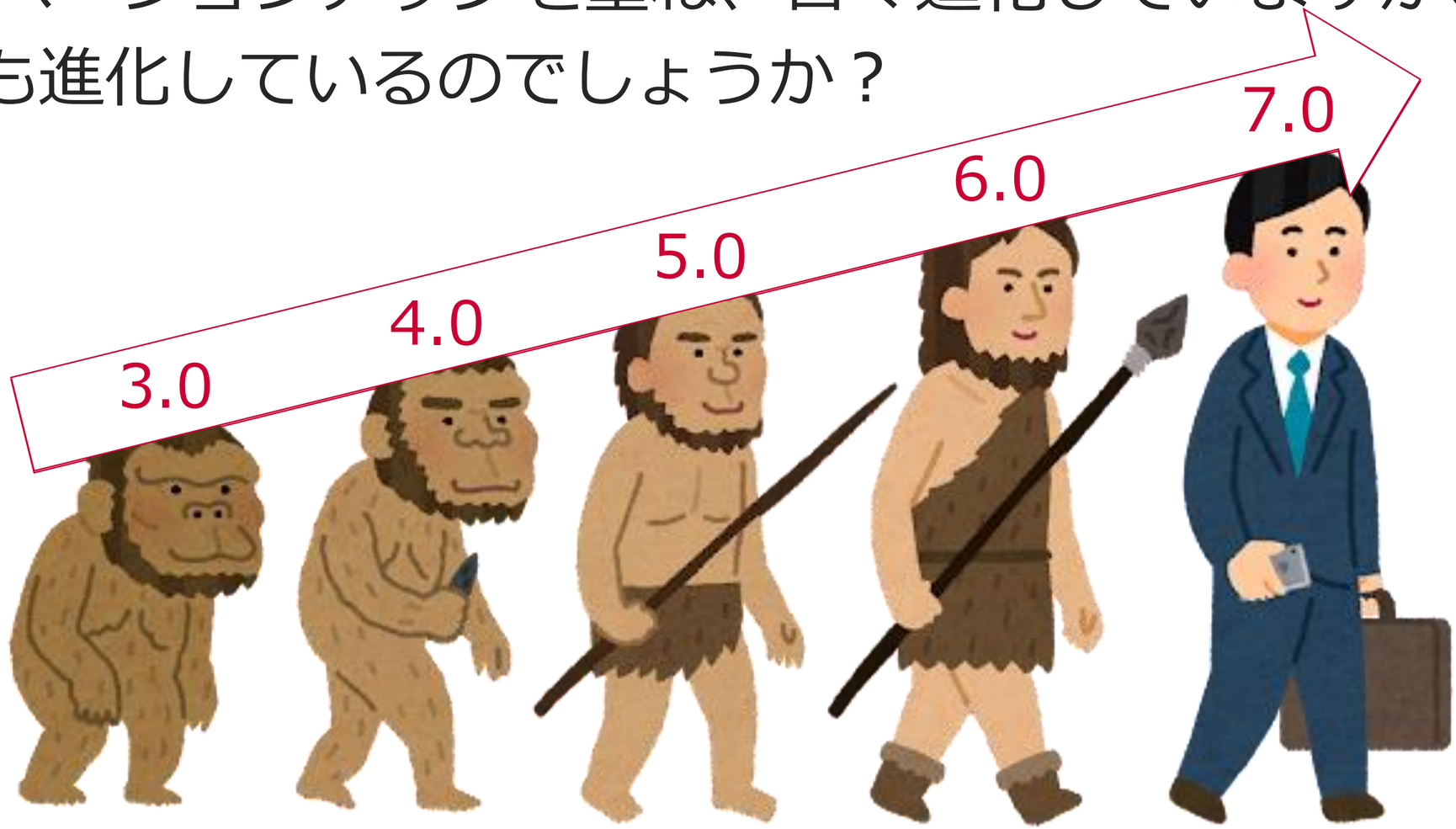
ZABICOMチームのサポートエンジニア、R&DエンジニアとしてZabbix業務に携わりながら、

2023年8月よりZabbix認定講師として認定ユーザートレーニングの講師業務を開始

サブタイトル(進化に合わせた監視設定)について

Zabbixの監視設定の進化についていこう

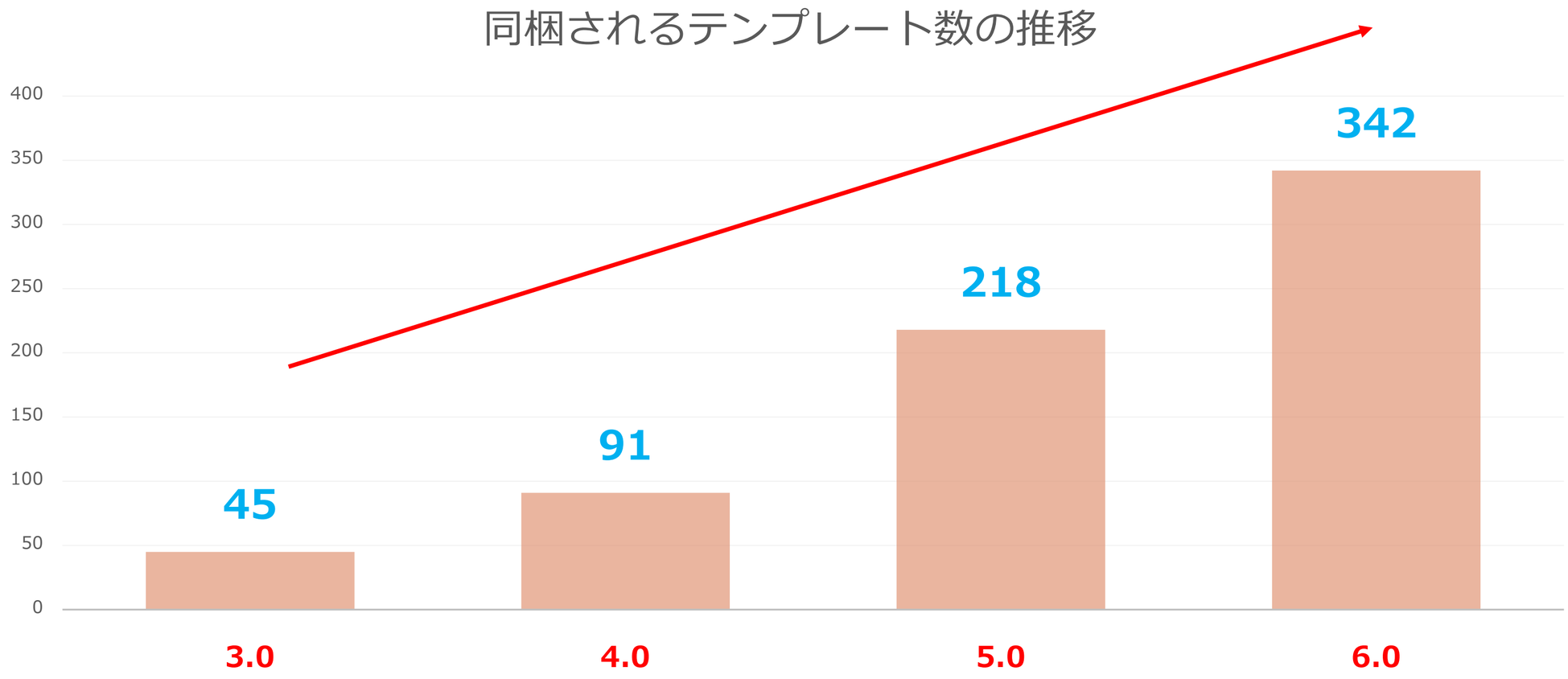
Zabbixはバージョンアップを重ね、日々進化していますが、
監視設定も進化しているのでしょうか？



※画像はイメージです。(深い意図はありません)

Zabbixの監視設定の進化についていこう

はい、Zabbixの監視設定も勿論一緒に進化しています！
インストール時に同梱されている監視テンプレート数も増え、
その**テンプレート内の監視設定方法も常に進化**をしています。



Zabbixの監視設定の進化についていこう

では、私たちの使っている監視テンプレートは・・・？

・**標準テンプレート**を3.0の時に作って**標準化**しているよ！
あれ？3.0って７年前・・・？もうすぐ7.0が出る・・・？

・使っている監視設定でも**困っていない**・・・

・監視設定の変更には**承認フローを通す必要**があって・・・

・**新しい機能**なんて使わなくても**困っていない**し・・・

・そもそも何が変わっているのか**よく知らない**し・・・

etc...

Zabbixの監視設定の進化についていこう

同じ悩みを持っている方に向けて、私が実際に悩んで解決してきた
監視設定の実例を踏まえた**ナレッジ、ノウハウの一部を共有**します。

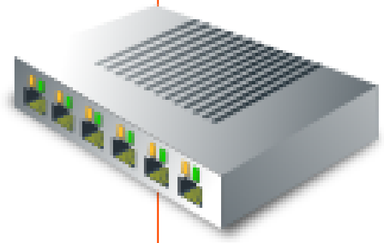
今回は
大体これらの機能を
使います。

事例1: 複数データ集計に苦労した話

経緯説明 (Zabbix 3.0時代)

Zabbix 3.0を利用されているお客様からこんなお問い合わせをいただきました。

受信トラフィックの全体平均値を出したいんだけど、どう設定したらいいの？



Interface 0/1 受信トラフィック

Interface 0/2 受信トラフィック

Interface 0/3 受信トラフィック

受信トラフィックの全体平均値

経緯説明 (Zabbix 3.0時代)

いただいたお問い合わせに対して、社内で議論を行いました。

Zabbix 3.0時代の社内会議

IFごとのアイテムはあるから**Zabbixアグリゲートで集計**したらいいんじゃないですかね？

確かに・・・でも**Zabbixアグリゲートはホストグループ単位**だから、ホスト毎にグループ作るのは妥当じゃないよね

アイテムのヒストリを取得して計算した結果を返す**Zabbix APIを使用した外部スクリプトを用意**したらいいんじゃないですかね？

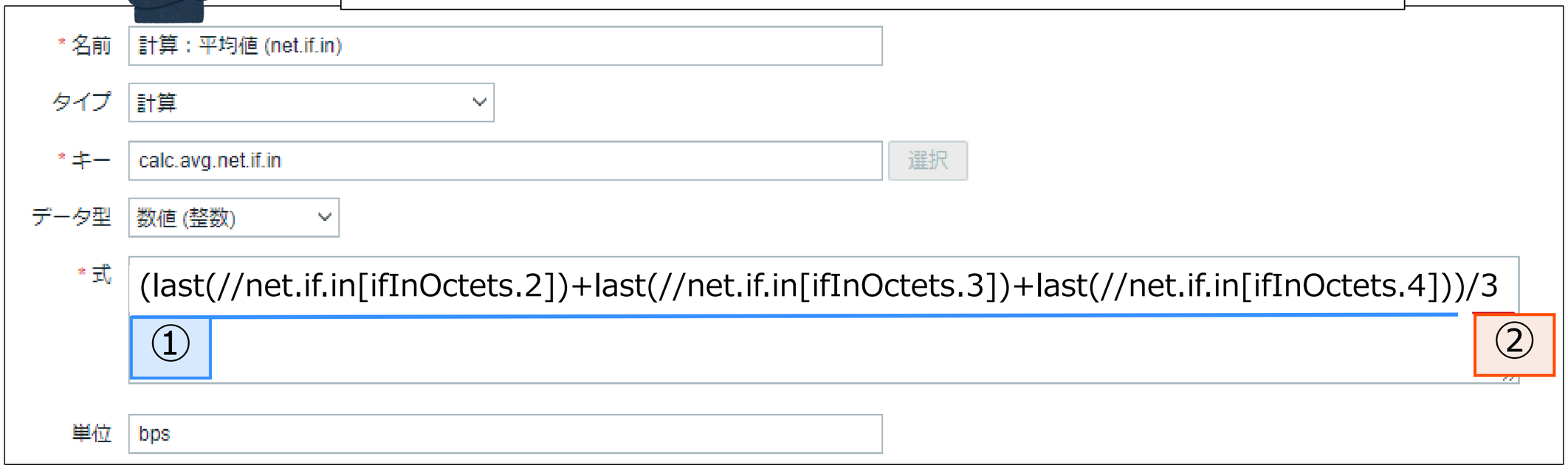
それが一番いいけど、お客様自身が設定をされる前提での相談だから、**標準機能の中で解決できるほうがいい**と思うんだよね

経緯説明 (Zabbix 3.0時代)

色々な方法、ご提案をした中で**最終的には計算アイテムでの設定が採用**されました。

(お客様のご要件、設定対象を絞られるという条件からこの設定が採用されました。)

ホスト単位での設定が可能だし、お客様自身での設定も可能。
ただ**メンテナンス性に難**があるのが課題・・・



① IF数に合わせて計算式の記載内容のメンテナンスが必要 (マクロ使用不可)

② 割り算する値もIF数に合わせてメンテナンスが必要

経緯の説明 (Zabbix 6.0時代)

時は流れ**2022年、Zabbix 6.0 LTSがリリース**され、自社の監視サーバーを更改(4.0→6.0)することになった際に、この事例を思い出し改めて挑戦してみようと考えました。

Zabbix 6.0時代の脳内会議

Zabbix 3.0で悩んだあの監視を自社NW監視でも使用したいな。
自社用だからスクリプトを書いてもいいけど、
せっかくだからZabbix 6.0までに追加された機能が使えないかな？

SNMP DiscoveryのLLDルールで使用されている**discovery[]キーと保存前処理うまく活用している監視設定**をZabbix社公式テンプレートなどで見たことがありますよ？

3.0時代になかった保存前処理の有効活用設定例か。
確かにそれを使えば要件を満たせるかもしれない。
まずは試してみよう！

保存前処理について

保存前処理について

保存前処理とは、**LTSでは4.0から使用**することが出来る機能です。
監視対象から**収集したデータを保存する前に加工処理**等を行うことが可能です。

■一部のデータを抽出して保存する

凄い大変なエラー (ErrorID: 10000) → 抽出 → ~~凄い大変なエラー (~~**ErrorID: 10000**~~)~~

■データを乗数で増やして保存する (byte → bitへの変換等に使用)

10000 → 8倍 → 80000

■データを置換して保存する

凄い大変なエラー (ErrorID: **10000**) → 変換 → 凄い大変なエラー (ErrorID: **Critical**)

保存前処理について

バージョンアップと共に増えていく保存前処理の設定

■テキスト
正規表現
前後文字列削除
末尾文字列削除
先頭文字列削除
置換

■構造化データ
XML Path
JSON Path
CSVからJSON

■計算
乗数

■変化
差分
1秒あたりの差分

■数値変換
論理値から10進数
8進数から10進数
16進数から10進数

■カスタムスクリプト
JavaScript

■バリデーション
値の範囲
正規表現と一致する
正規表現と一致しない
JSON内のエラーチェック
XML内のエラーチェック
正規表現使用時のエラーチェック
サポートしていない値のチェック

■絞り込み
変化がなければ破棄
指定秒内に変化がなければ破棄

■Prometheus
Prometheusパターン
PrometheusからJSON

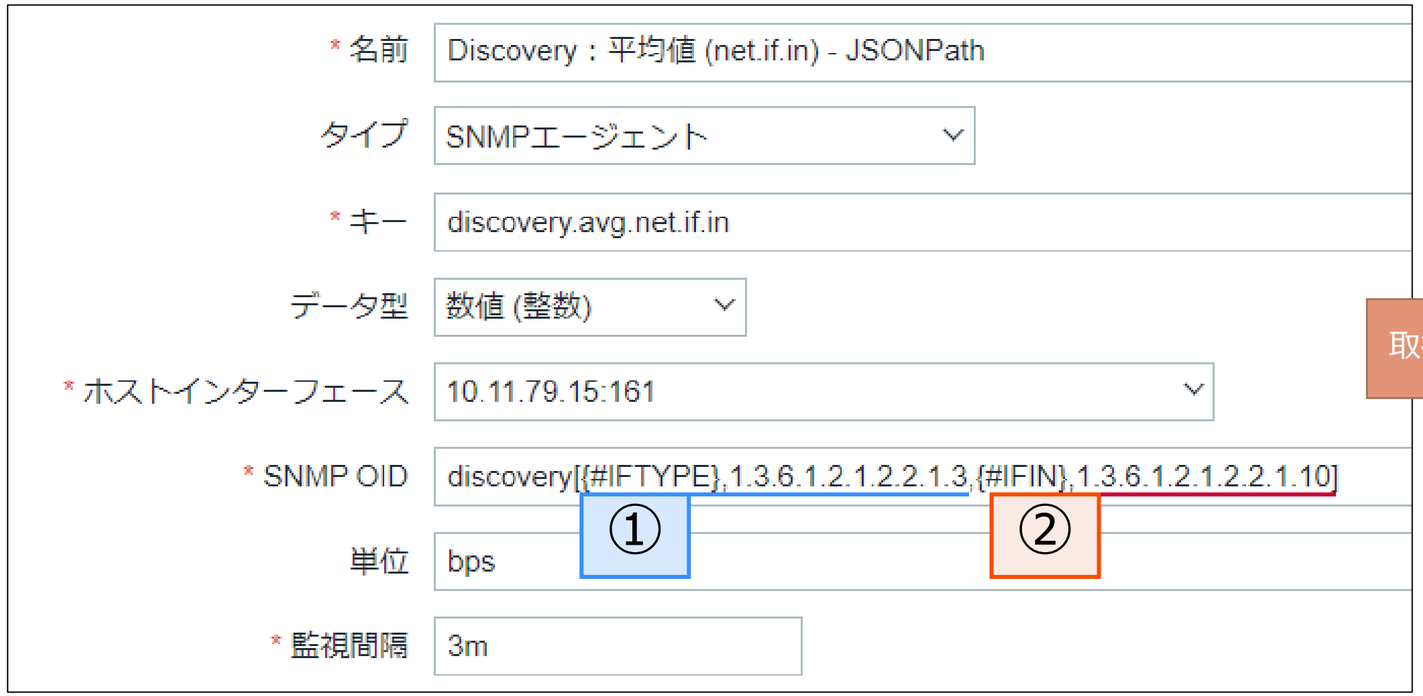
4.0 LTSで追加
5.0 LTSで追加
6.0 LTSで追加

Zabbixが保存前に行える処理がバージョンアップ毎に増えている！

【改善案1】discovery[]キー + JSONPath

【改善案1】discovery[]キー + JSONPath

discovery[]キーは、指定した**OIDのデータ**を取得し**JSON形式のデータに加工して保存**するため、
JSONデータをPath式を使用して処理が可能な、**JSONPathという保存前処理と連携**させてみました。



取得データ

```
[
    {
        "{#SNMPINDEX}":"1",
        "{#IFTYPE}":"24",
        "{#IFIN}":"673503672"
    },
    {
        "{#SNMPINDEX}":"2",
        "{#IFTYPE}":"6",
        "{#IFIN}":"1982022179"
    },
    {
        "{#SNMPINDEX}":"3",
        "{#IFTYPE}":"6",
        "{#IFIN}":"1269244555"
    },
    {
        "{#SNMPINDEX}":"4",
        "{#IFTYPE}":"6",
        "{#IFIN}":"1269244195"
    }
]
```

① IF-MIB::ifTypeでインターフェースのタイプのデータを取得

② IF-MIB::ifInOctetsでインターフェースの受信バイト数を取得

【改善案1】discovery[]キー + JSONPath

設定する保存前処理としては下記となります。

① LLDマクロ{#MACRO}が**JSONPathで書式エラー**になりやすいので、扱いやすいキー名に置換

② **JSONPath**で条件に合う**ifInOctetsの取得値の平均(avg)を算出** (例はifType:24を除外)

③ ifInOctetsの値はカウンター型のByteデータなので、**bps(bit per sec)に変換**

【改善案1】discovery[]キー + JSONPath

計算アイテムで集計した結果と比較すると、

discovery[]キー + JSONPathで集計した結果は、**ほぼ同一**であることが確認できます。

☐	snmp device15	計算：平均値 (net.if.in)	11s	6.31 Kbps	+538 bps
☐	snmp device15	Discovery：平均値 (net.if.in) - JSONPath	2m	6.3 Kbps	+536 bps

作り終わった直後の私

↓

計算アイテムの集計と同じデータを取得出来ているし、
機器のインターフェース構成に合わせて変更する必要がない監視設定ができた。
これは改善できたといえるんじゃないか！！

【改善案1】discovery[]キー + JSONPath

少し経って落ち着いた私の中の脳内会議

この設定も素晴らしいけど、元々思ってたより複雑な監視設定になっているな。(フィルターしなければ想定通りだけど)
保存前処理を節約した設定方法があるともっといい気がするな。

Zabbix 5.0からは保存前処理にJavaScriptが追加されましたよね？あれ使ってみたかったんですけど、うまく使えないですかね？

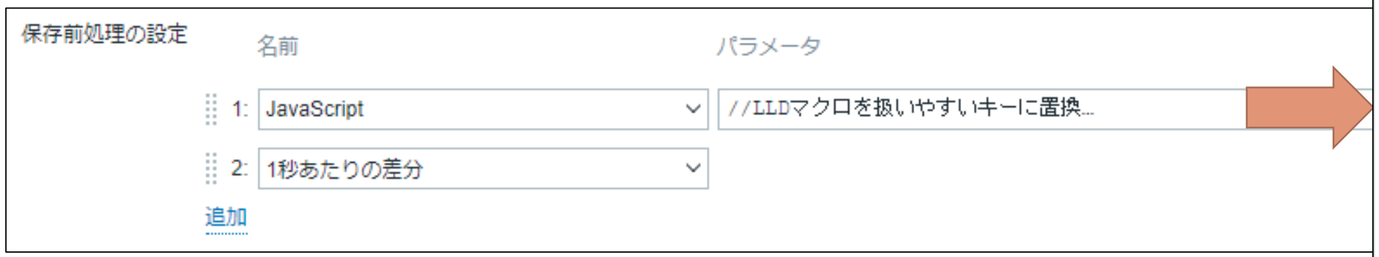
確かに・・・コードを書く必要があるけど、これなら標準機能の範囲だし**挑戦してみるのも面白い**かもしれないな・・・
一回書いて設定を試してみてもいいかもしれないな。

【改善案2】discovery[]キー + JavaScript

【改善案2】discovery[]キー + JavaScript

JavaScriptはその名の通り、**JavaScript形式でデータを処理できる保存前処理**です。

外部スクリプトの連携と異なり**監視設定上でコード**を記載することが可能なことが特徴です。



改善案1で実施していた**置換、乗数、JSONPathの処理をコード内で対応**することで、**保存前処理の数を削減**

```
function (value) {
  1  //LLDマクロを扱いやすいキーに置換
  2  var value = value.replace(/{#IFTYPE}/g, 'iftype');
  3  var value = value.replace(/{#IFIN}/g, 'ifin');
  4
  5  //JSONデータの処理
  6  resp = JSON.parse(value);
  7  var array = [];
  8  sum = 0;
  9  count = 0;
 10  for (var i = 0; i < resp.length; i++) {
 11      if (resp[i].iftype == '24') {
 12          continue;
 13      }
 14      else {
 15          count++
 16          sum += Number(resp[i].ifin)
 17      }
 18  }
 19
 20  //合計値をデータ数で割って平均値を算出 (乗数8倍も合わせて実施)
 21  result = sum / count * 8
 22
 23  return result
}
```

【改善案2】discovery[]キー + JavaScript

これまでの2つの手法(計算アイテム、JSONPath)の結果と比較すると、

discovery[]キー + JavaScriptの結果に関しても、**ほぼ同一**であることが確認できます。

ホスト	名前 ▲	最新のチェック時刻	最新の値	変化
snmp device15	Discovery：平均値 (net.if.in) - JavaScript	1m 56s	6.21 Kbps	-97 bps
snmp device15	Discovery：平均値 (net.if.in) - JSONPath	1m 56s	6.21 Kbps	-96 bps
snmp device15	計算：平均値 (net.if.in)	7s	6.21 Kbps	-98 bps

作り終わった直後の私

↓

JavaScriptの処理の中で、**複数の保存前処理でやっていた処理をまとめる**ことが出来たので、**保存前処理が5→2つに削減**できた！
JavaScriptを使用した監視設定事例も出来たし、すごく良い改善だったんじゃないか！？

【改善案2】discovery[]キー + JavaScript

少し経って落ち着いた私の中の脳内会議 (Part 2)

標準機能の範囲だし、この**コードを利用した監視設定には有用性がある**けど、他の人への**引継ぎは考えないといけない**な。
他に方法がないか、マニュアルを調べてみようかな？

例2

そのホストでnet.if.in[*]に一致するすべての item の最新値の合計

```
sum(last_foreach(/host/net.if.in[*]))
```

Zabbix Manual

※6.0公式マニュアル： 7.設定 -> 2.アイテム -> 7.計算アイテム -> 集計計算より抜粋

あれ、**Zabbix 6.0の集計関数のページに、似た設定例**がある。
6.0から追加された**foreach関数**を使うのか・・・
保存前処理に拘っていたけど、集計関数も試してみよう！

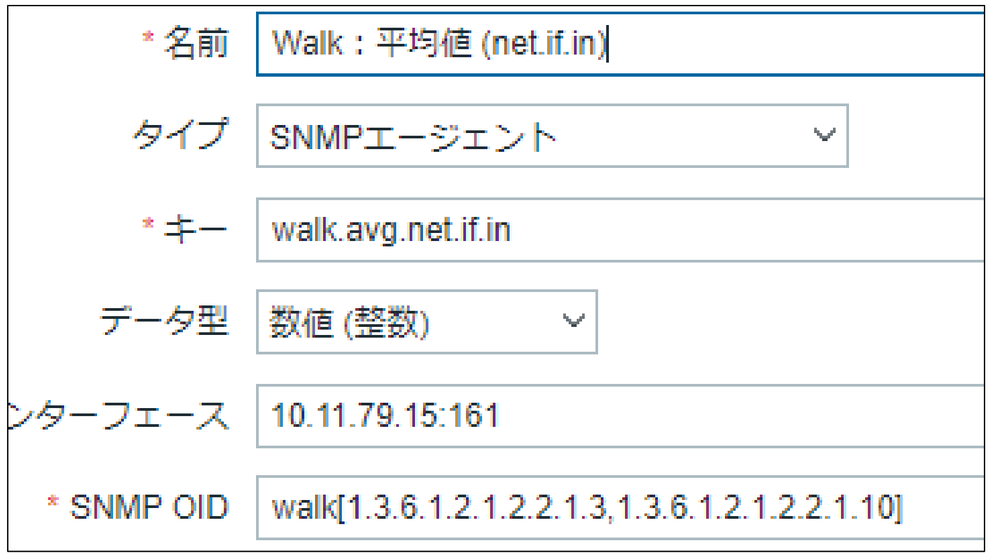
ということで、この事例の改善案の検討は集計関数に続きます。

集計関数に入る前に少し余談

集計関数に入る前に少し余談

ここまでdiscovery[]キーでJSONデータを取得し、保存前処理と連携させる方法を紹介してきました。

しかし、6.4以降では、discovery[]キーよりも**処理性能が向上**した**walk[]キー**が追加されており、
下記の参考例のように、**snmpwalkコマンドを打った時のようなデータ**が取得できます。

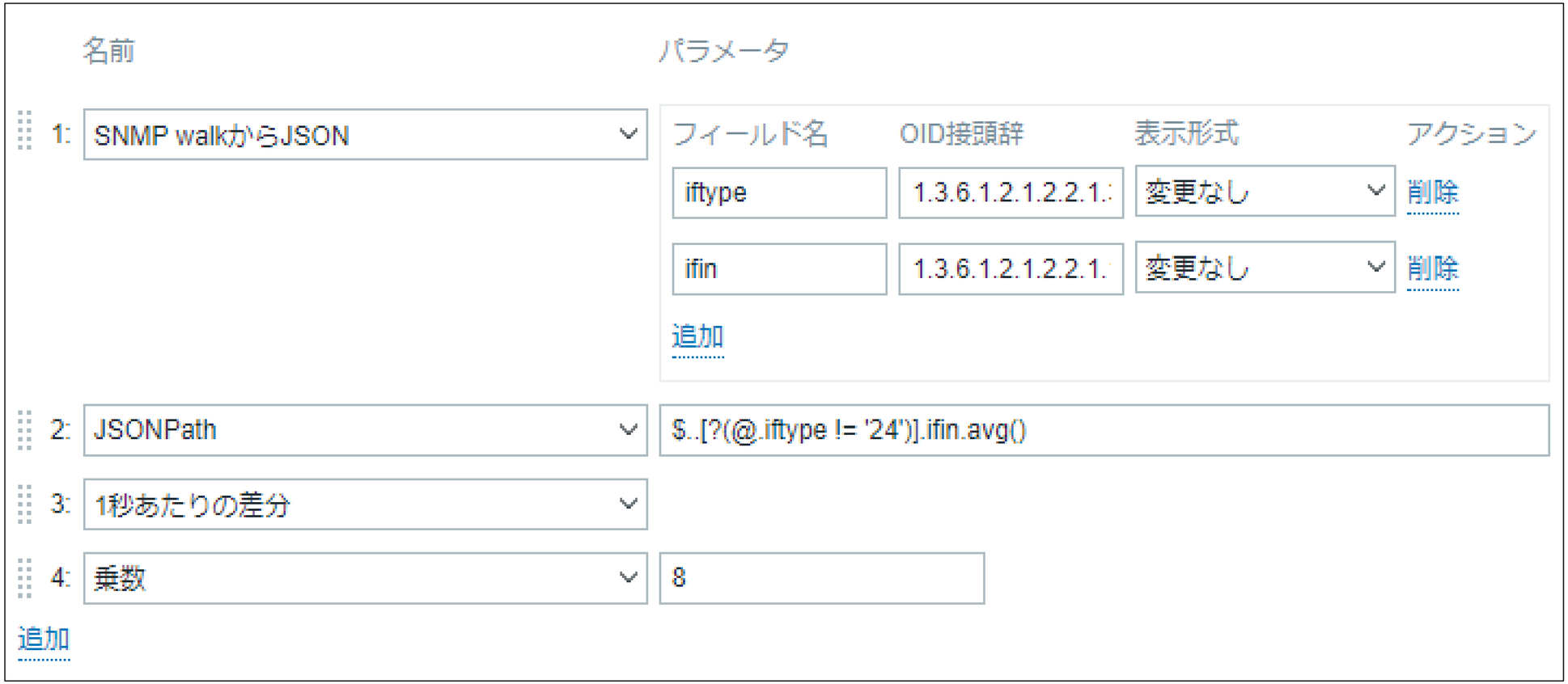



```
.1.3.6.1.2.1.2.2.1.10.1 = Counter32: 678674499
.1.3.6.1.2.1.2.2.1.10.2 = Counter32: 2021190926
.1.3.6.1.2.1.2.2.1.10.3 = Counter32: 1272431518
.1.3.6.1.2.1.2.2.1.10.4 = Counter32: 1272431158
.1.3.6.1.2.1.2.2.1.3.1 = INTEGER: 24
.1.3.6.1.2.1.2.2.1.3.2 = INTEGER: 6
.1.3.6.1.2.1.2.2.1.3.3 = INTEGER: 6
.1.3.6.1.2.1.2.2.1.3.4 = INTEGER: 6
```

集計関数に入る前に少し余談

6.4ではwalkコマンドの結果をJSONデータに加工できる、**「SNMP walkからJSON」**という保存前処理が追加されています。
現在作成中の**新バージョン用のテンプレート**では、**walkキーを使用した監視設定が採用**され、
既存テンプレートも置き換わっていることを確認していますので、今後こちらの監視設定が主流になってくると思われます。

そのため、もし**6.4以降**で今回のような連携を活用する場合は、**walkキーを使用**することを検討してみてください。

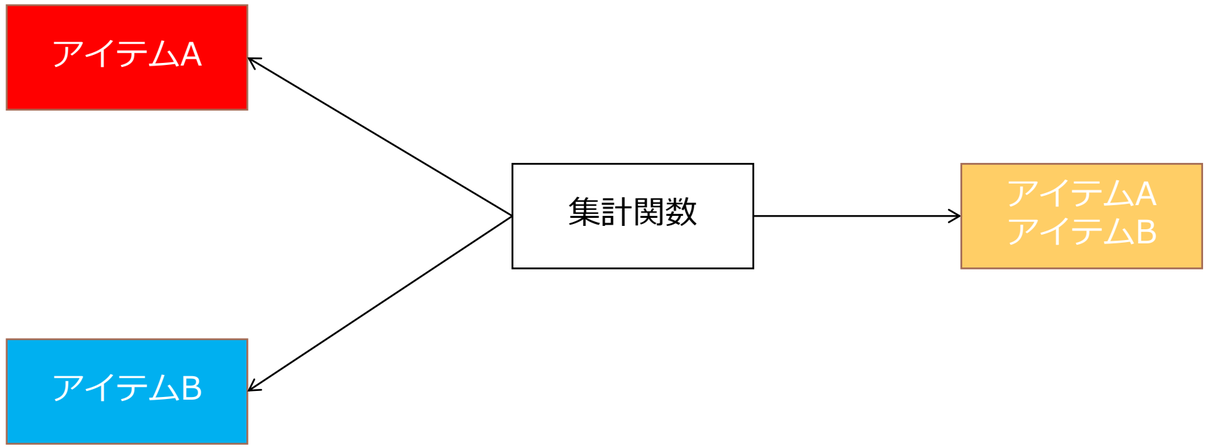


集計関数 - foreach関数について

集計関数 - foreach関数について

Zabbix 6.0 LTSから、**従来**の集計関数(**アイテム個別の履歴**を使った集計)とは別に、
複数のアイテム履歴から集計値を返却できる**foreach関数**が追加されています。

この関数を用いることで、以前のバージョンで集計しづらかった**アイテム間の集計を**
容易かつ高度に実施することが可能になっています。



※6.0公式マニュアル： 21.付録 -> 6.サポートされている関数 -> 1. 集計関数 -> 1. foreach関数より抜粋

関数	説明
avg_foreach	平均値を返します
bucket_rate_foreach	histogram_quantile() 関数での使用に適したペア (バケットの上限、レート値) を返します。ここでの"バケットの上限"は、<parameter number>パラメーター で定義された項目キー パラメーターの値です。
count_foreach	値の個数を返します
exists_foreach	現在有効になっているアイテムの数を返します
last_foreach	最後の値を返します
max_foreach	最大値を返します
min_foreach	最小値を返します
sum_foreach	合計値を返します

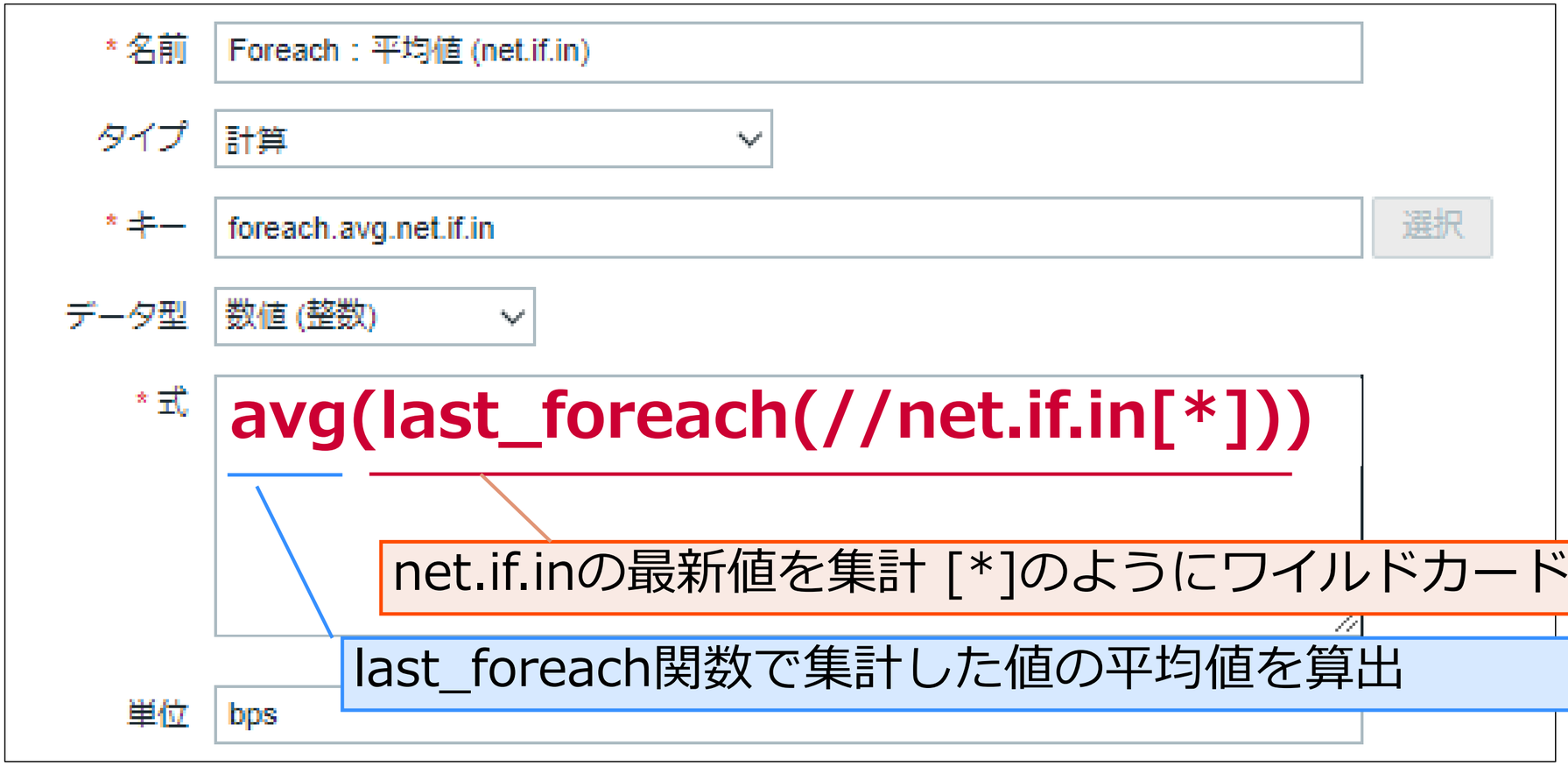
【改善案3】foreach関数

【改善案3】foreach関数

事例1の複数データの集計を集計関数を用いてやってみましょう。

今回、ホスト内の**net.if.inの最新値**の**平均値**を出したいので、

last_foreach関数を使ってnet.if.in最新値を集計し、その値を**avg関数で平均値**を算出してみます。



保存前処理の設定　追加

※保存前処理は設定不要
(各アイテムで実施済のため)

【改善案3】foreach関数

これまでの3つの手法(計算アイテム、JSONPath、JavaScript)の結果と比較すると、
foreach関数を使用した結果に関しても、**ほぼ同一**であることが確認できます。

ホスト	名前 ▲	最新のチェック時刻	最新の値	変化
snmp device15	Discovery：平均値 (net.if.in) - JavaScript	1m 56s	6.48 Kbps	+263 bps
snmp device15	Discovery：平均値 (net.if.in) - JSONPath	1m 56s	6.47 Kbps	+264 bps
snmp device15	Foreach：平均値 (net.if.in)	4s	6.47 Kbps	+266 bps
snmp device15	計算：平均値 (net.if.in)	7s	6.47 Kbps	+266 bps

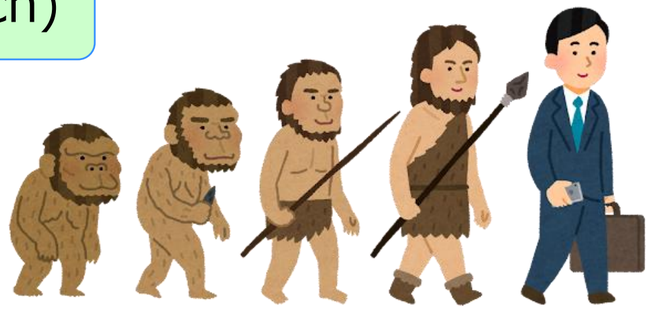
今回の要件は、**Zabbix 6.0 LTSではforeach関数**を活用することが最も簡単！
その他データ集計でも活用できると感じるので、チャレンジしてよかった！

事例1のまとめ

今回1つの監視要件に対して4つの手法をご紹介しました。

～3.0	4.0	5.0	6.0
計算	JSONPath※		計算(foreach)
		JavaScript	

※置換はZabbix 5.0以降で使用可能



各手法、**設定開始可能なバージョン**は上記の通りとなります。
そのため、**バージョンによる最適解**は**変動**します。
Zabbixの**進化により監視設定の幅**は広がっていきますので、
色んな監視設定方法にチャレンジすると**面白い**のでお勧めです。

事例2: SNMPトラップ監視の汎用的な監視設定

経緯説明 (今も昔も)

SNMPトラップの監視設定に関して、このような期待を要件定義時に受けたことはないでしょうか？

期待いただいているお客様

弊社の監視対象には複数のSNMP対応機器があって、**SNMPトラップの監視**を行いたいんです。

ただ、**どのようなSNMPトラップが飛ぶかは把握できていなくて**・・・

でも**御社には豊富な実績**があると思うので、
その実績を使った監視設定テンプレートの作成をお願いしたいんです。

個人的体感、案件中8割の要件確認時に言われるこの期待ですが、こんな悩みを抱えていないでしょうか。

期待に比例して悩みを抱える私

SNMPトラップって、**ベンダー毎に飛んでくるトラップ内容も違う**し、
ベンダーが一緒でも**機種やOSが違えばトラップ内容変わる**んだよな・・・

だから**過去作成したテンプレートを使いまわしづらい**から、
一から調査する必要があって、**実績を活用しづらい**んだよな・・・

経緯説明 (今も昔も)

このような悩みから、**初期導入時に使える**ベンダー、機種などに**依存しない汎用設定**が欲しいと考えました。

何かいい案がないか考える私の脳内会議

何かいい方法がないか、一般公開ナレッジ・ノウハウや、
公式マニュアルを見てみよう。いい案が思いつくかもしれない。

マクロ関数は、macroの値をカスタマイズする機能を提供します。

マクロは、必ずしも簡単に扱えない値を解決することがあります。値が長かったり、抽出したい特定の部分文字列が
含まれていたりする場合があります。このような場合に、マクロ関数が役に立ちます。

Zabbix Manual

※6.0公式マニュアル： 7.設定 -> 11.マクロ -> 1.マクロ関数より抜粋

そうだ、**マクロ関数**を使えば取得したトラップの**一部を切り出して
イベント名とかタグ名に使用**できるんじゃないか？
上手く行けば汎用的なトラップ監視設定ができるかもしれない！

ということで、マクロ関数を利用したSNMPトラップの汎用設定に挑戦してみたというのが事例2です。

マクロ関数について

マクロ関数について

マクロ関数とは、**マクロから引き渡されるデータ**を下記例のように**カスタマイズ**する関数です。

書式：{<macro>.<func>(<params>)}

■小数点以下に表示される桁数を制御する (LTSでは6.0以降で使用可能)

24.3413523 ➡ {{ITEM.VALUE}.fmtnum(2)} ➡ **24.34**~~1352~~

24.3413523 ➡ {{ITEM.VALUE}.fmtnum(0)} ➡ **24**~~.341352~~

■データのうち正規表現に一致する一部データのみを抽出する

凄い大変なエラー (ErrorID: **10000**)

⬇

{{ITEM.VALUE}.regsub(“.*ErrorID:¥s([0-9]+)”, ¥1)} ➡ **10000**

【改善案】マクロ関数によるデータ活用

【改善案】マクロ関数によるデータ活用

※注意

今回、ZabbixのSourceに同梱されている**zabbix_trap_receiver.plを使用**します。
その他ツール(SNMPTT、Zabbix Japan提供のZabbix用SNMPトラップフォーマッタツールなど)を使う場合は、設定を調整する必要があるので、その点はご注意ください。

■zabbix_trap_receiver.plの入手方法

・ZabbixのSourceを取得
URL: https://www.zabbix.com/download_sources

・Source入手し解凍後、 misc/snmptrap/ディレクトリ配下からファイルを取得可能

・設定方法
URL: https://www.zabbix.com/documentation/6.0/en/manual/config/items/itemtypes/snmptrap#configuring-perl-trap-receiver

【改善案】マクロ関数によるデータ活用

■実際に収集されるトラップサンプルと抽出して使用したい値 (OID表記)

```
2023/09/14 15:32:44 ZBXTRAP 127.0.0.1
PDU INFO:
  requestid                504716776
  receivedfrom             UDP: [127.0.0.1]:54001->[127.0.0.1]:162
  messageid                0
  community                public
  transactionid            1
  errorstatus              0
  version                  1
  notificationtype         TRAP
  errorindex               0
VARBINDS:
  .1.3.6.1.2.1.1.3.0                type=67 value=Timeticks: (0) 0:00:00.00
  .1.3.6.1.6.3.1.1.4.1.0            type=6  value=OID: .1.3.6.1.4.1.8072.2.3.0.1
  .1.3.6.1.4.1.8072.2.3.2.1         type=2  value=INTEGER: 60
```

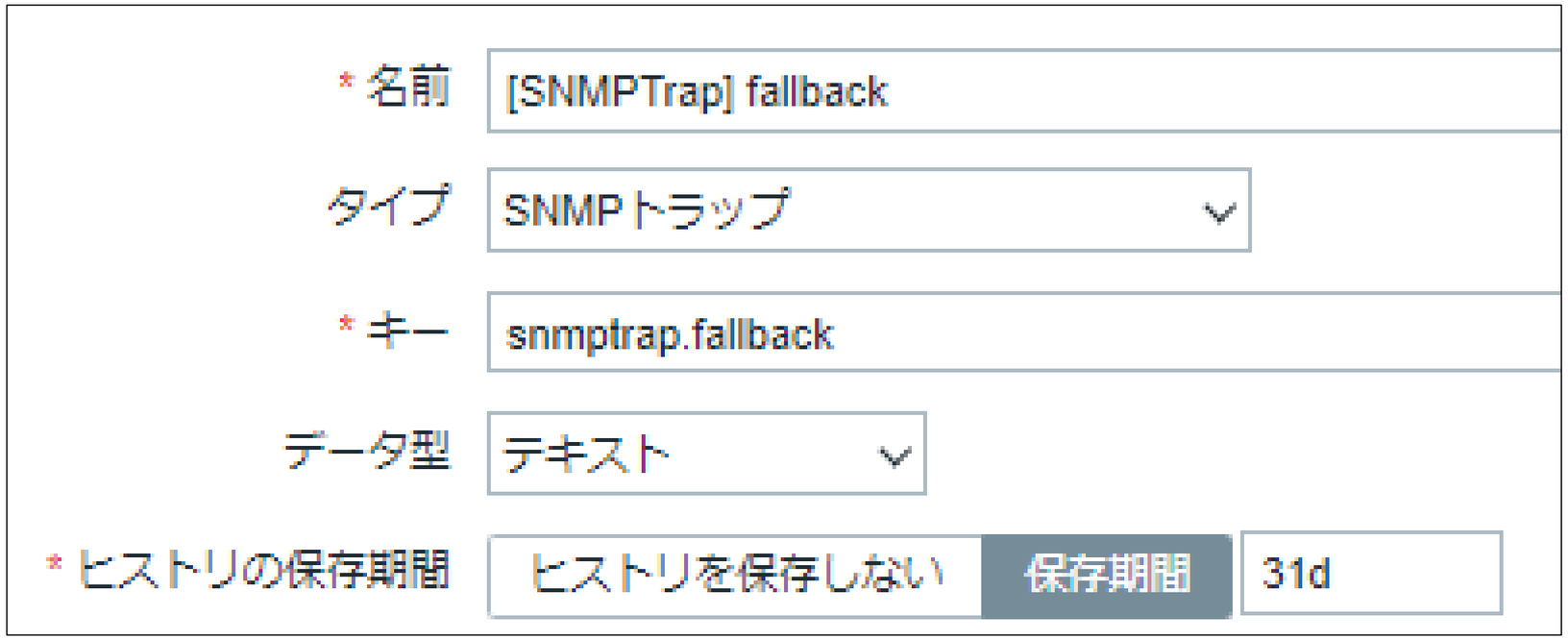
snmpTrapOID (.1.3.6.1.6.3.1.1.4.1.0)
の**value値をマクロ関数で抽出**して使用する

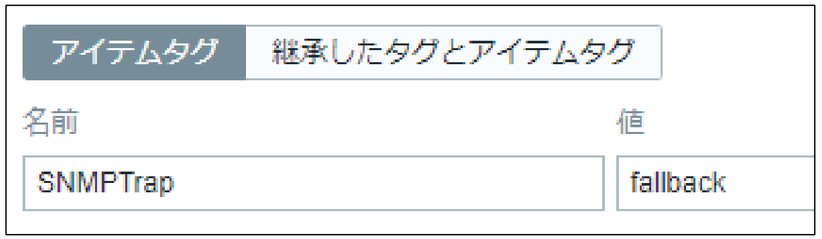
【改善案】マクロ関数によるデータ活用

■アイテム設定

シンプルなsnmp.fallback (snmptrap[regex]に一致しないトラップが全て対象)のアイテムを用意。

アイテムタグはお好みで設定。





【改善案】マクロ関数によるデータ活用

■トリガー設定

イベント名とタグ値にマクロ関数を設定。その他は一般的な全検知の監視設定を用意。

深刻度	名前 ▲	運用データ	条件式	ステータス	タグ
警告	[SNMPTrap] fallback		find(/Common_SNMP-TRP_General/snmptrap.fallback,,"like")=1	有効	snmpTrapOID: {{ITEM...

* 名前: [SNMPTrap] fallback

イベント名: [SNMPTrap] fallback (snmpTrapOID: {{ITEM.VALUE}.regsub("1\.3\.6\.1\.6\.3\.1\.1\.4\.1.*value=OID:\s+(.*)(\s|$)", \1)})

運用データ:

深刻度: 未分類 | 情報 | 警告 | 軽度の障害 | 重度の障害 | 致命的な障害

* 条件式: find(/Common_SNMP-TRP_General/snmptrap.fallback,,"like")=1 追加

条件式ビルダー

正常イベントの生成: 条件式 | 復旧条件式 | なし

障害イベント生成モード: 単一 | 複数

正常時のイベントクローズ: すべての障害 | タグの値が一致したすべての障害

* クローズに利用するタグ名: snmpTrapOID

手動クローズを許可

トリガータグ | 継承したタグとトリガータグ

名前	値
snmpTrapOID	{{ITEM.VALUE}.regsub("1\.3\.6\.1\.6\.3\.1\.1\.4\.1.*value=OID:\s+(.*)(\s|$)", \1)}

追加

【改善案】マクロ関数によるデータ活用

マクロ関数は下記のような設定を用意。

```
{{ITEM.VALUE}.regsub("1¥.3¥.6¥.1¥.6¥.3¥.1¥.1¥.4¥.1.*value=OID:¥s+(.*)(¥s|$)", ¥1)}
```

①　②　③

① マクロ関数が処理する対象マクロとして**{ITEM.VALUE}**を指定

② {ITEM.VALUE}からデータを抽出するための**正規表現**を記載

③ 正規表現の**1つ目の()内のデータを抽出**するように指定

【改善案】マクロ関数によるデータ活用

■ **実際に収集されるトラップサンプルと抽出して使用したい値 (OID表記)**

2023/09/14 15:32:44 ZBXTRAP 127.0.0.1

～省略～

VARBINDS:
.1.3.6.1.2.1.1.3.0 type=67 value=Timeticks: (0) 0:00:00.00
.1.3.6.1.6.3.1.1.4.1.0 type=6 value=OID: **.1.3.6.1.4.1.8072.2.3.0.1**
.1.3.6.1.4.1.8072.2.3.2.1 type=2 value=INTEGER: 60

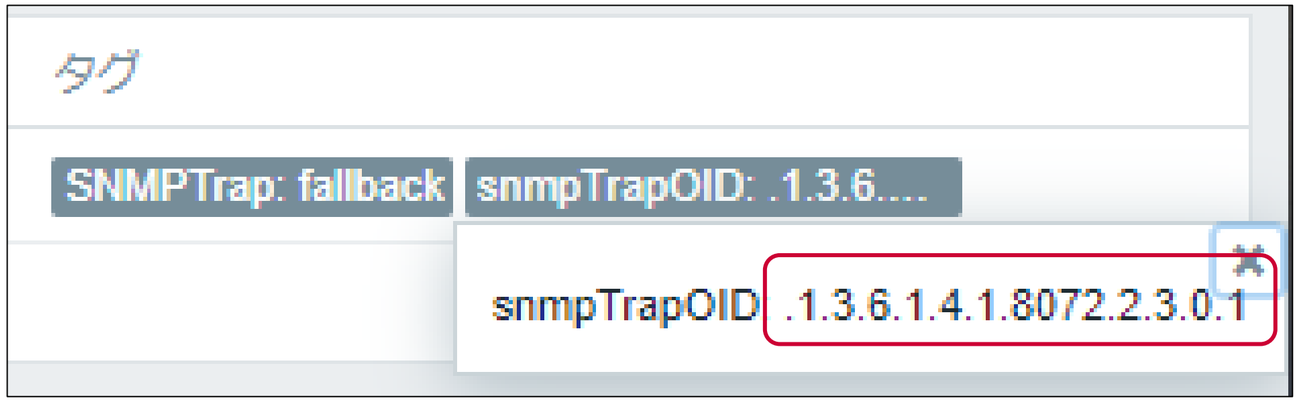
{{ITEM.VALUE}.regsub("1¥.3¥.6¥.1¥.6¥.3¥.1¥.1¥.4¥.1.*value=OID:¥s+**(.*)**(¥s|$)", ¥1)}

※¥1 = regsubの正規表現内の一つ目の()内のデータを取得。(今回であれば**.1.3.6.1.4.1.8072.2.3.0.1**)

【改善案】マクロ関数によるデータ活用

実際にトラップを出力してみると、下記のようにイベント名、タグ名にOIDが記載されます。

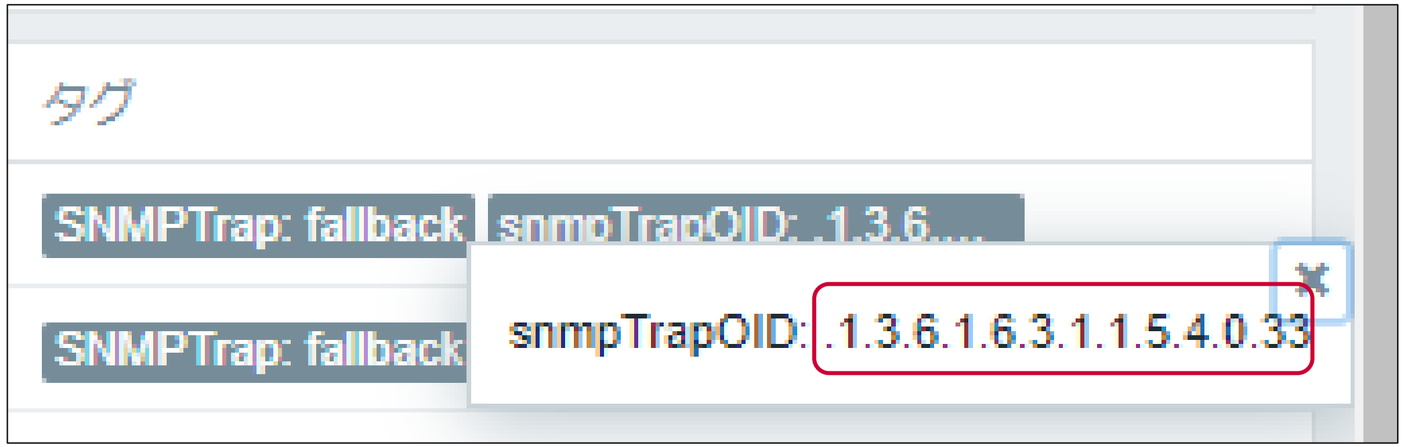
深刻度	復旧時刻	ステータス	情報	ホスト	障害
警告		障害		snmp trap	[SNMPTrap] fallback (snmpTrapOID: .1.3.6.1.4.1.8072.2.3.0.1)



【改善案】マクロ関数によるデータ活用

その他トラップを出力すると、違うOIDとして表示されることが確認できる

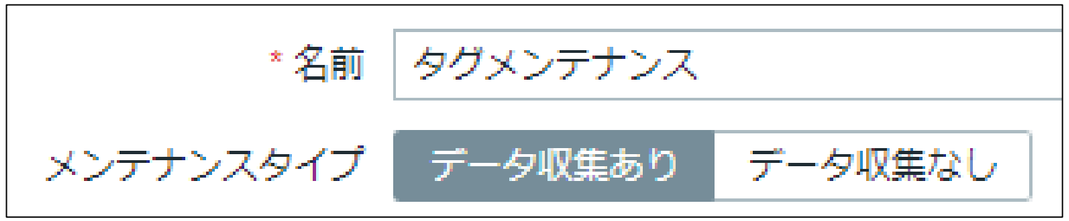
時間		深刻度	復旧時刻	ステータス	情報	ホスト	障害
16:12:10		警告		障害		snmp trap	[SNMPTrap] fallback (snmpTrapOID: .1.3.6.1.6.3.1.1.5.4.0.33)
16:08:17		警告		障害		snmp trap	[SNMPTrap] fallback (snmpTrapOID: .1.3.6.1.4.1.8072.2.3.0.1)

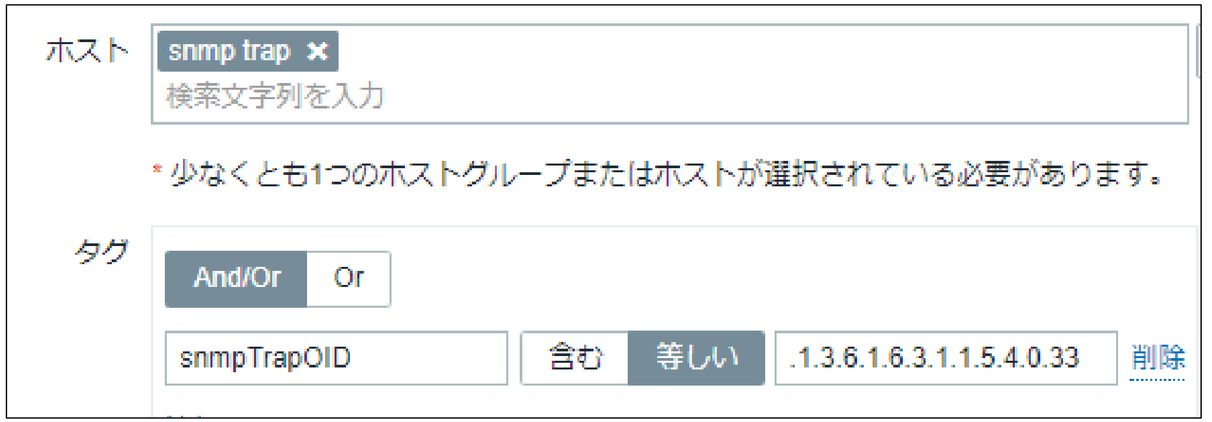


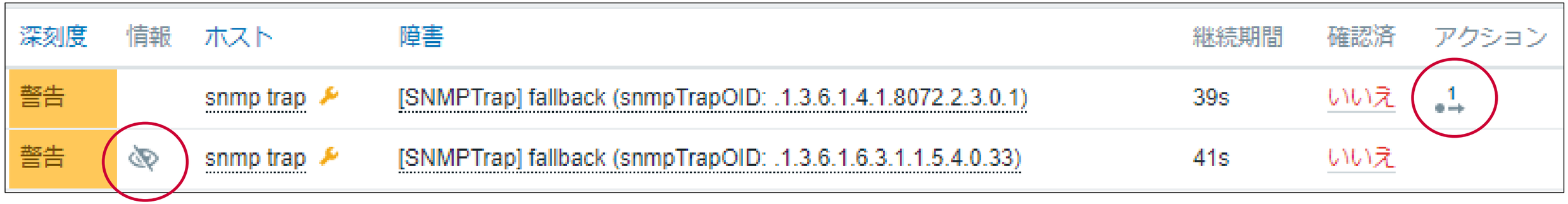
【改善案】マクロ関数によるデータ活用

タグは**障害画面でのフィルタリングの活用**は勿論、**メンテナンス時の条件設定としても活用**することが可能。

snmpTrapOIDが一致するトラップだけをメンテナンスとして**アラート抑止**が可能。





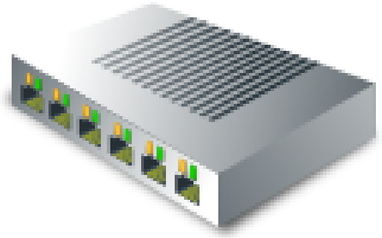


※.1.3.6.1.6.3.1.1.5.4.0.33のsnmpTrapOIDを持つトラップをメンテナンスに設定

【改善案】マクロ関数によるデータ活用

今回の設定ではsnmptrap.fallbackを使用したアイテム設定としていますので、
もし**個別のSNMPトラップに関する監視設定**を行いたい場合は、正規表現をで保存先アイテムを分岐できる
snmptrap[regex]と連携すると良いと思います。

分岐させたアイテム側に**専用トリガーを用意**し、**個別設定(障害条件設定、深刻度変更など)**が可能です。
また、**アイテムのヒストリ保存期間を"ヒストリを保存しない"**とすることで**取得データの除外も可能**です。



正規表現に一致しない
トラップ

正規表現1に一致する
トラップ

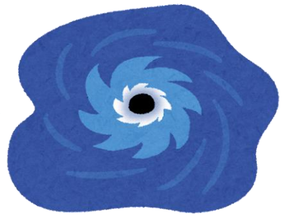
正規表現2に一致する
トラップ

snmptrap.fallback → 汎用トリガー

snmptrap[正規表現1] → **専用**トリガー

snmptrap[正規表現2]

※ヒストリの保存期間を"ヒストリを保存しない"に設定



ヒストリ保存されないため、消える

事例2のまとめ？

これで飛んでくるトラップ内容が分からなくても、**イベント名、タグで判別できる汎用的なSNMPトラップ設定**が出来た！

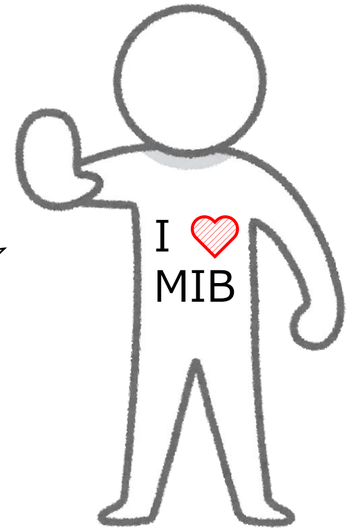
実際にお客様にも導入して満足いただけたし、良い監視設定が出来た！

これで事例2に対する監視設定の改善案は終了です！！

といいたいですが・・・この例を見せると**多くの方にいただく意見**があります。

【補足】マクロ関数によるデータ活用 (MIB)

ちょっと待って！！
うちの環境は**MIBファイルでの変換を有効**にしているから、
MIBファイルを使った設定例を教えて欲しい！



※以下の理由から個人的には**MIBファイルを使用しないSNMPトラップ設定を推奨**しています。

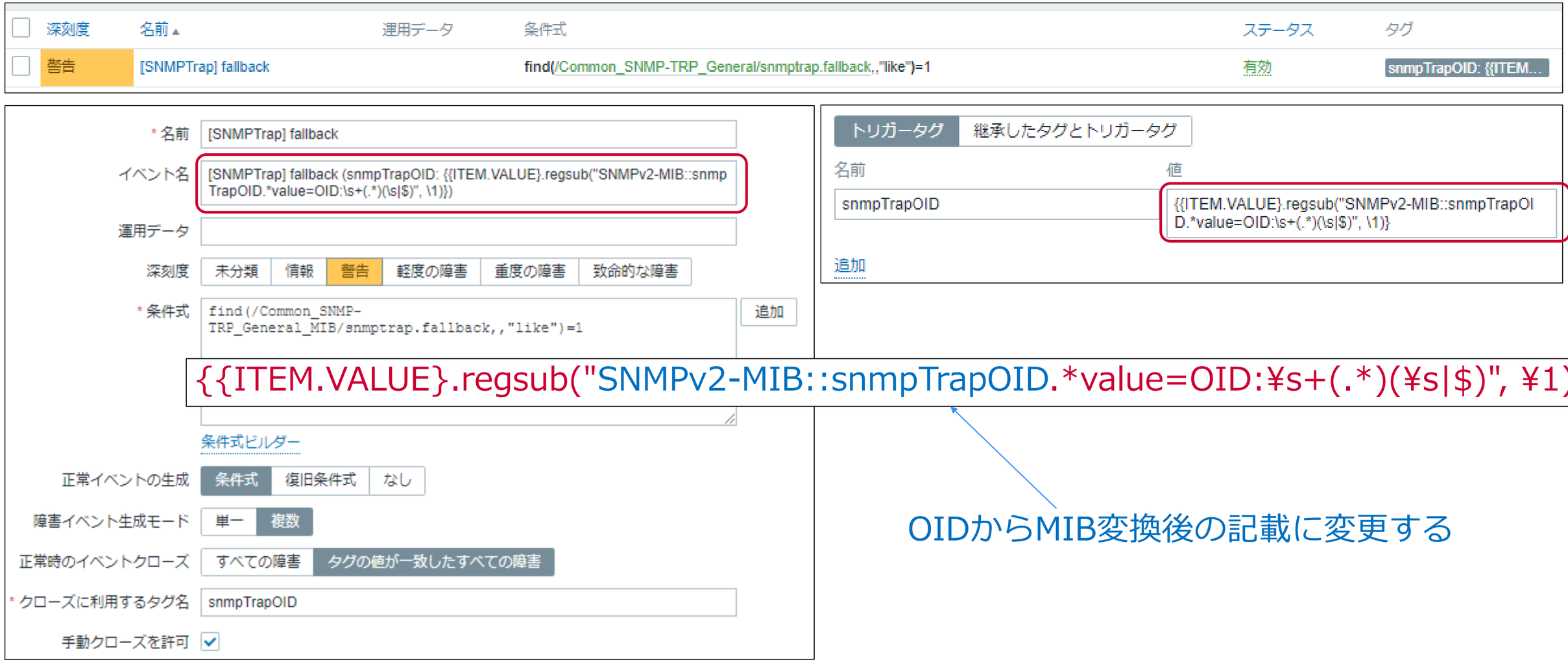
- MIBファイルを**使用しないほうが処理性能が良い**
- MIBファイルの**配置自体が難易度が高く**、エラーなく運用することが**難しい**
  (エラー発生させたまま運用している場合、処理性能に影響がある)
- **全てのトラップに対応するMIB**をロードしておかないと、**不完全な変換処理**となってしまう

とはいえMIBファイルを使用した環境が大半なので、そちらの例は**次のページから紹介**します。

【補足】マクロ関数によるデータ活用（MIB）

■トリガー設定　※アイテムは変わらないので割愛

OID表記の時との違いは、マクロ関数の正規表現をMIB変換後の文字列に合わせるだけです。



{{ITEM.VALUE}.regsub("SNMPv2-MIB::snmpTrapOID.*value=OID:¥s+(.*)(¥s|$)", ¥1)}

OIDからMIB変換後の記載に変更する

【補足】マクロ関数によるデータ活用 (MIB)

■実際に収集されるトラップサンプルと抽出して使用したい値 (MIB変換後の表記)

2023/09/14 16:52:15 ZBXTRAP 127.0.0.1
PDU INFO:

～～省略～～

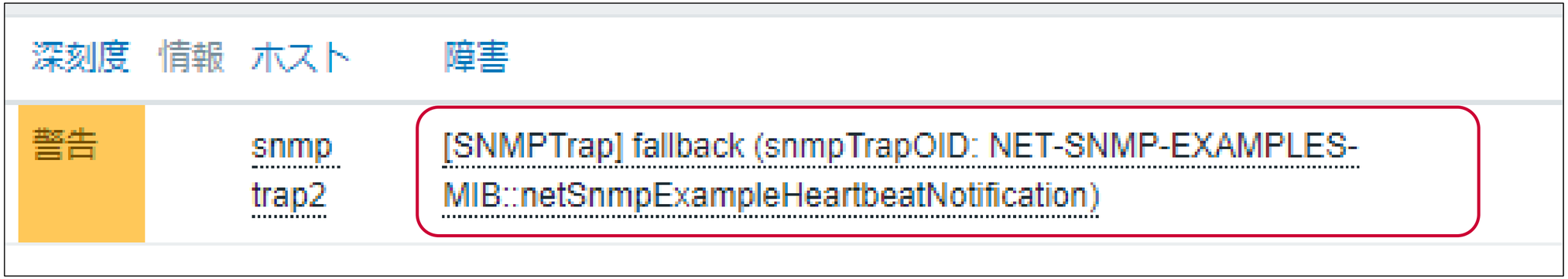
VARBINDS:
DISMAN-EVENT-MIB::sysUpTimeInstance type=67 value=Timeticks: (0) 0:00:00.00
SNMPv2-MIB::snmpTrapOID.0 type=6 value=OID: **NET-SNMP-EXAMPLES-MIB::netSnmpExampleHeartbeatNotification**
NET-SNMP-EXAMPLES-MIB::netSnmpExampleHeartbeatRate type=2 value=INTEGER: 60

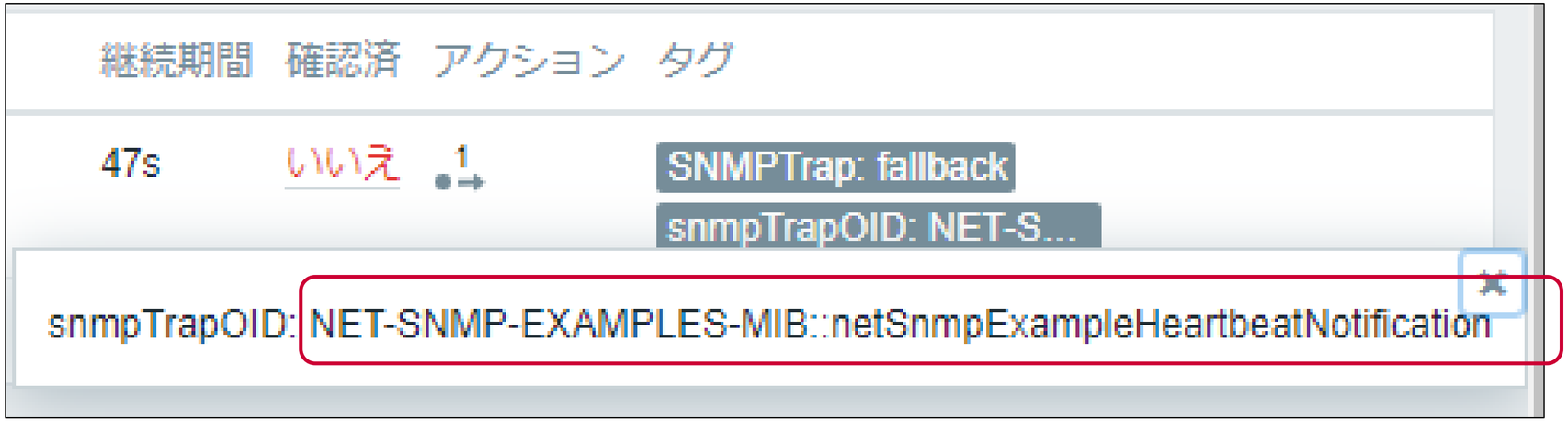
{{ITEM.VALUE}.regsub("SNMPv2-MIB::snmpTrapOID.*value=OID:¥s+**(.*)**(¥s|$)", ¥1)}

※¥1 = regsubの正規表現内の一つ目の()内のデータを取得。
(今回であれば**NET-SNMP-EXAMPLES-MIB::netSnmpExampleHeartbeatNotification**)

【補足】マクロ関数によるデータ活用 (MIB)

実際にトラップを出力してみると、下記のようにイベント名、タグ名にOID(MIB変換後)が記載されます。





まとめ

まとめ

Zabbixは常に進化するソフトウェアです。
監視設定に関しても振り返ってみると、より良い監視設定が必ずできます。

そこを探しているとき、また気付けて改善案を出せることが、
Zabbix監視設定の楽しいところであり、醍醐味だと思います。
是非皆さんにもこの機会にチャレンジしてみていただければと考えています。

また、個人的に色々な監視設定方法に関する共有、議論を出来たら嬉しいので、
お会いする機会などあれば是非お声がけください。

一緒にZabbixの監視設定を楽しみましょう!!

Let‘s enjoy Zabbix monitoring setting together!!

ご清聴ありがとうございました