docomo business

ZABBIX '25 CONFERENCE JAPAN

生成型AIを用いたLLDテンプレート設計支援

令和7年 11月7日

NTTドコモビジネスエンジニアリング株式会社
田中　武信

自己紹介

田中　武信

所属

NTTドコモビジネスエンジニアリング株式会社
スマートオペレーションサービス部

出身

東京都豊島区

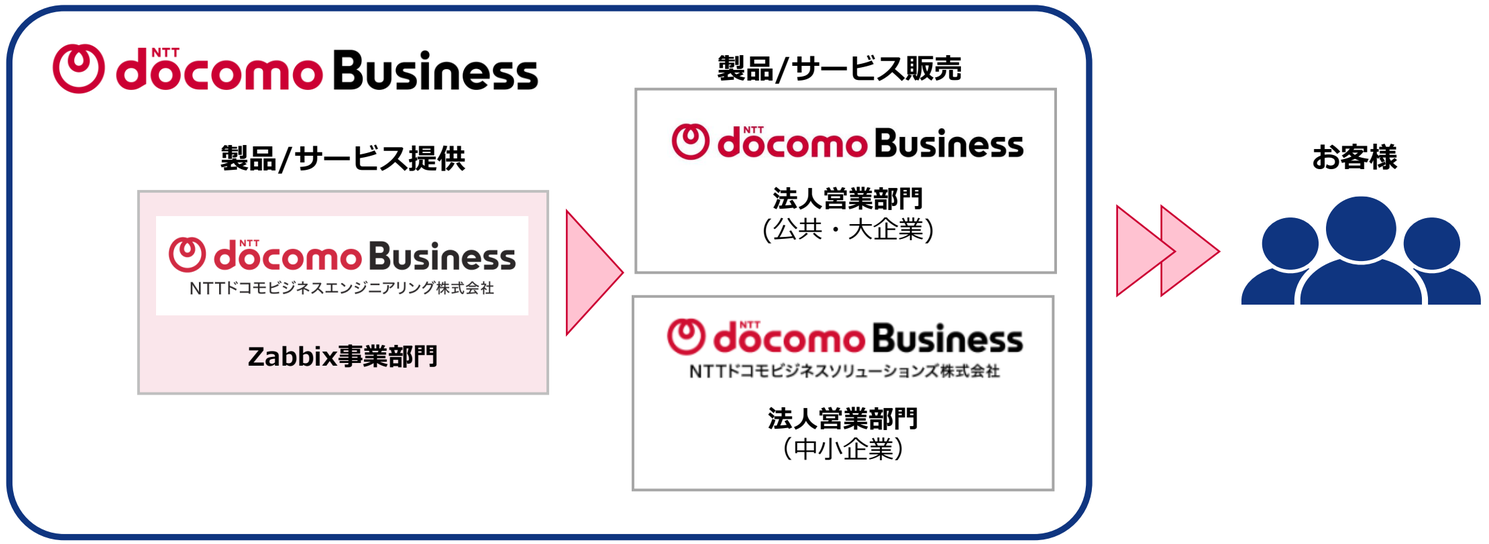
経歴

2000～08年	5社の運用現場を転々
2008年	NTTコムテクノロジーに入社
2011年	Zabbix関連部署に所属
2018年	Zabbix Summit(Riga)登壇
	Zabbix Conference Japan(Tokyo)登壇
2019年	Zabbix Conference Japan(Tokyo)登壇
2021年	Zabbix Conference Japan(Tokyo)登壇
2024年	Zabbix Summit(Riga)登壇

自社紹介

NTTドコモビジネスエンジニアリング株式会社（旧NTTコム エンジニアリング）

- 2007年よりZabbix関連事業に携わり、ZABICOMソリューションとして提供
- NTTドコモビジネスのサービスにおける、設計・構築・保守・運用を担う
- NTTドコモビジネスまたはNTTドコモビジネスソリューションズが販売を担当



【公式X】

https://x.com/NTTdBEngineer

NTTグループの社名変更

NTT株式会社

NTT

総合ICT事業

NTTドコモ

docomo

https://www.docomo.ne.jp/

NTTドコモビジネス

docomo Business

https://www.ntt.com/index.html

NTTドコモソリューションズ

docomo Solutions

https://www.nttcom.co.jp/

地域通信事業

NTT東日本

NTT東日本

https://www.ntt-east.co.jp/

NTT西日本

NTT西日本

https://www.ntt-west.co.jp/

グローバルソリューション事業

NTTデータグループ

NTT DATA

https://www.nttdata.com/global/ja/

その他

NTTアーバンソリューションズ

NTTアーバンソリューションズ

https://www.ntt-us.com/index.html

NTTアノードエナジー

NTTアノードエナジー

https://www.ntt-ae.co.jp/

その他グループ会社

docomo Business

NTTドコモビジネスエンジニアリング株式会社

エヌ・ティ・ティ・コミュニケーションズ株式会社※

NTT Communications

NTTドコモビジネス株式会社

docomo Business

はじめに

はじめに

本セッションは、**生成AIを用いて監視テンプレートを作成する事例**の紹介となります。

紹介する内容は、**2025年7月に行ったLLDテンプレート作成の実証試験**を基にしています。
（弊社で本内容に基づくサービスを提供している訳ではありません）

生成AIは進展が早いため、**現在では改善されている問題が含まれている場合**があります。

生成型AIを使っていますか？

①生成型AIって何ですか？

②生成型AIは時々使ってます（週1回くらい）

③生成型AIを毎日使ってます！

④生成型AIが無いと仕事が回りません！

⑤生成型AIは友達です

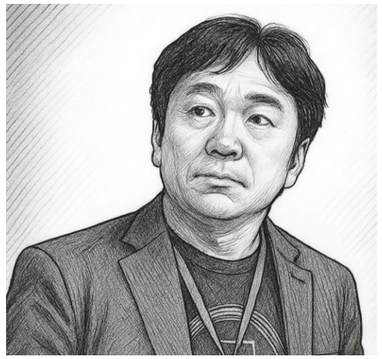
実証試験の目的

・**人とAIが協力して設定を作成することを目的としています**
　⇒　人の補助機能としてAIを使う検証

・**AIで全工程を自動化することを目的としたものではありません**
　⇒　人とAIの得手不得手を明確化する検証

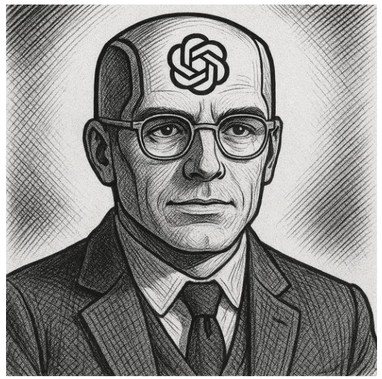
・**実際の運用で使用できる品質のテンプレート作成を目的とします**
　⇒　人の知見を踏まえたデータをAIに生成させる検証

使用する知能



マネジメント兼オペレータ

役割　　：目標設定と評価 / 実機操作を担当
自己紹介　：文系の元サーバエンジニア（現：万屋）
　　プログラミングとAIは専門ではない。
Zabbix歴　：Ver.1.4、Zabbix 6.0プロフェッショナル資格



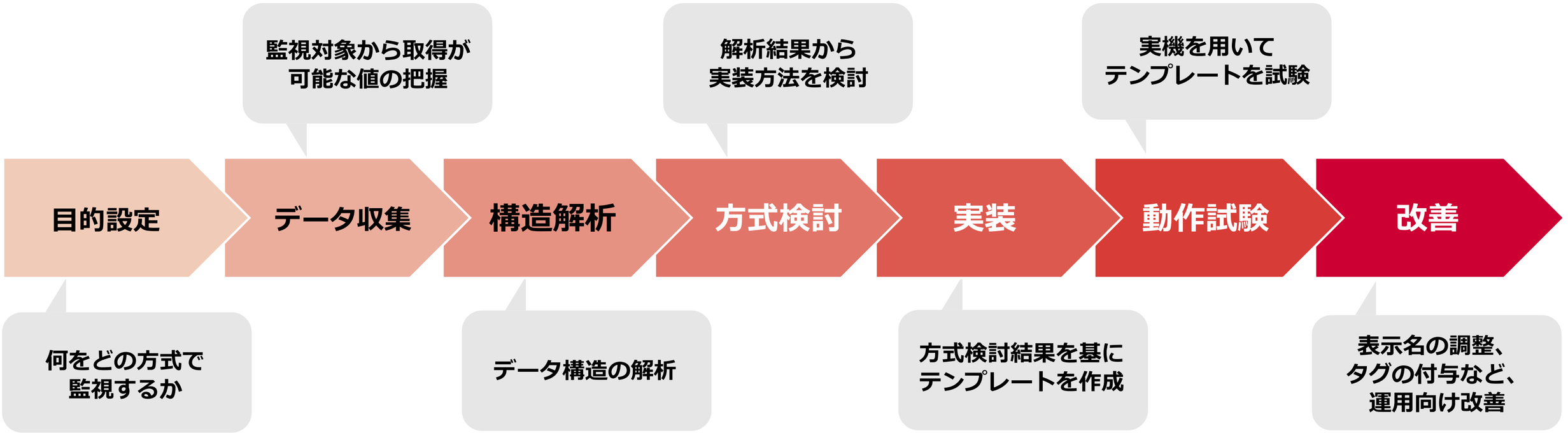
ChatGPT 5o （¥3,376/月）

役割　　：分析と設計を担当
自己紹介　：知識と会話で助けるAI。
　　創造が得意、感情理解は少し苦手です。
Zabbix歴　：Ver.7.0 （マニュアルを明示的に参照）、無資格

Gemini 2.5 Flush （無料）

役割　　：ChatGPTのアウトプット検証を担当
自己紹介　：Google生まれの知識探求AIです。
　　長文の一貫性や厳密な計算は苦手です。
Zabbix歴　：　Ver.7.0 （マニュアルを明示的に参照）、無資格

監視テンプレート作成のフロー

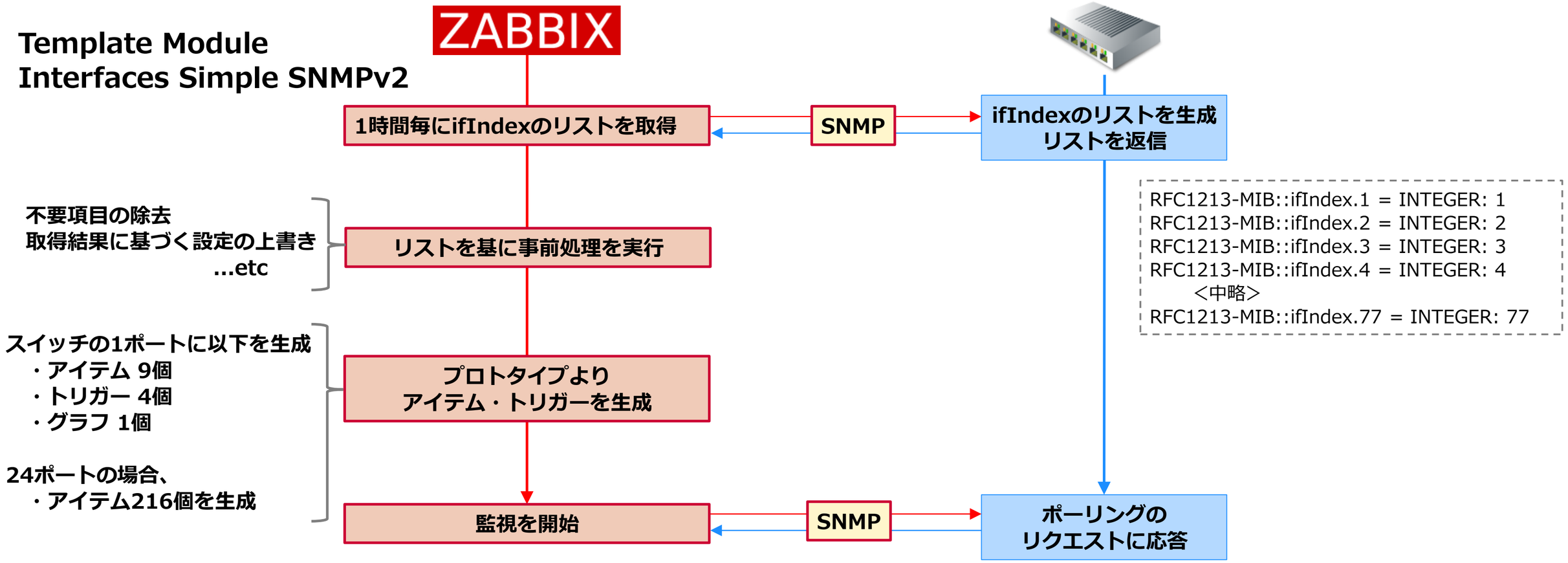


☞ これらの工程において、どの様にタスクを分担できるかを検証

閑話：LLD（Low-Level Discovery）とは

LLDとは、対象から監視可能な項目を取得して**アイテム/トリガー/グラフを自動生成**する機能。

Zabbixの標準テンプレートには本機能を活用した設定が多数含まれている。



経緯と前提知識

経緯

Cisco製品と**Allied Telesis製品**でポート毎のトラフィック量を取りたいです

SNMPを用いた監視で対応できます

トラフィックと同時に、光ケーブル接続の**受光レベル**を取得したいです

SNMPでデータが取れれば実現できそう。
実際に試してみるか・・・

実施目的

光接続されたネットワークにおいて、**受光レベルの品質低下は様々な要因**で発生します。

1. 物理的要因
- 光ファイバの曲げ（曲率過大）
- ファイバ接続部の劣化／汚れ
- 光ファイバ断線・破損

2. 光信号要因
- 伝送距離の増加による光パワー不足（減衰増大）
- 発光源、受光器の劣化
- 分散（色分散・偏波モード分散）

3. 環境要因
- 温度変動による機器性能への影響
- 振動/衝撃によるコネクタ接合部の変動

これらは、最終的にはTCP/IP通信における**エラーなどの形で通信障害**として現れますが、
受光レベルを監視することにより、**通信障害に達する前に異常を検知**することができます。

閑話：ネットワーク機器の光ケーブル接続方法

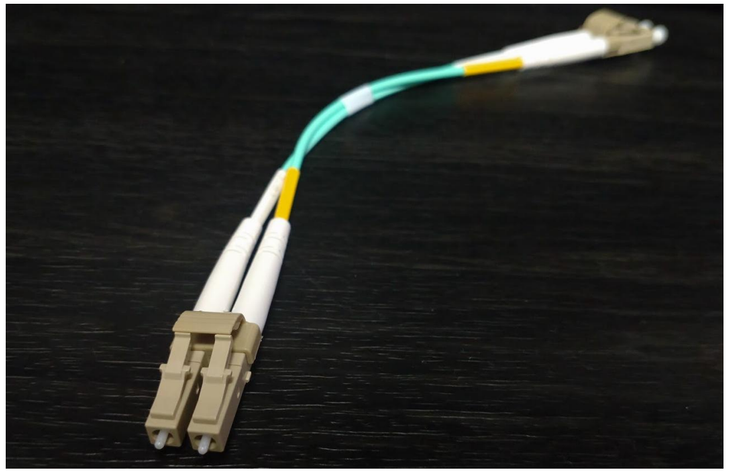
ネットワーク接続における光ケーブルの使用は、**1990年代の100BASE-FX**に遡ります。
1998年には1Gbpsの帯域を実現した1000BASE-SX / LX、
2002年以降には10Gbps、40Gbps、100Gbpsなどの帯域に対応しています。

ネットワーク機器で使用される光トランシーバは、着脱可能なモジュール化されており、
現在は**SFP（Small Form-factor Pluggable）**が使用されています。

SFP（1Gbps）、SFP+（10Gbps～）の二つの規格が広く使われています。

ネットワークスイッチのSFPポート

SFP / SFP+モジュール



光ケーブルと端子

閑話：光接続のモニタリング

モジュールの状態を把握する規格としてDDM (Digital Diagnostic Monitoring)があります。
この中で、光関連の監視はDOM (Digital Optical Monitoring)として定義されています。

光通信の「受光レベル（Rx Power）」は、光信号の強さ（光パワー）を示す値であり、
通常は **dBm（デシベルミリワット）**単位で表示されます。

dBmは、**1mW（ミリワット）を基準とした対数表現**の電力単位です。

$$\mathrm{dBm} = 10 \times \log_{10}(\mathrm{mW})$$

光パワー (mW)	対応する dBm
10 mW	+10 dBm
1 mW	0 dBm
0.1 mW	-10 dBm
0.01 mW	-20 dBm

Powered by Gemini

Cisco Catalyst 2960Lの場合

実機の用意

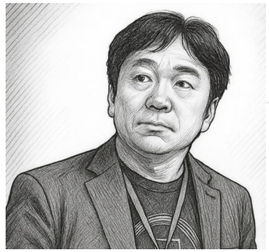
Cisco製品の検証には以下の機材を使用

＜L2スイッチ＞
品名：Catalyst 2960L-48-TS

＜SFPモジュール＞
品名：GCL-SX-MMD

Catalyst 2960L
（**WS-C2960L-48TS-JP**）
Cisco Systems

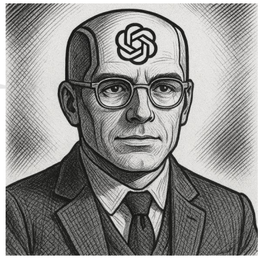
監視するOIDの調査



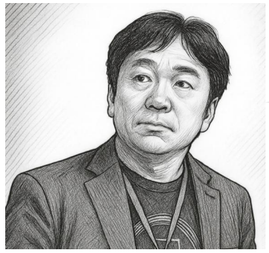
Catalyst2960のSFPの受光レベルに関するOIDを調べて

Catalyst 2960 シリーズ（2960/2960-S）は、DOM 対応 SFP を使用しても**受光（Rx）・送信（Tx）レベルの SNMP OID をサポートしていません**。Reddit でも「SFP モジュールの電力値は返さず、温度のみ対応」と報告されています。CISCO-ENTITY-SENSOR-MIB（1.3.6.1.4.1.9.9.91）では ASR1000 など他機種で温度・電圧・光パワーを取得できますが、**2960 では非対応**です。

結論：2960 シリーズでは受光/送信パワー値は取得不可、温度や電圧のみ可能。

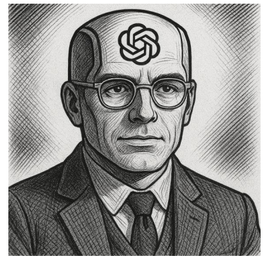


監視するOIDの調査



Catalyst2960のSFPの受光レベルに関するOIDを調べて

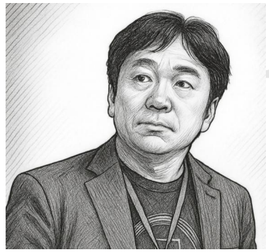
Catalyst 2960 シリーズ（2960/2960-S）は、DOM 対応 SFP を使用しても **受光（Rx）・送信（Tx）レベルの SNMP OID をサポートしていません**。Reddit でも「SFP モジュールの電力値は返さず、温度のみ対応」と報告されています。
CISCO-ENTITY-SENSOR-MIB（1.3.6.1.4.1.9.9.91）では ASR1000 など他機種で温度・電圧・光パワーを取得できますが、2960 では非対応です。
結論：2960 シリーズでは受光/送信パワー値は取得不可、温度や電圧のみ可能。



Catalyst 2960のSFP受光レベルのOIDは**CISCO-ENTITY-SENSOR-MIBで定義**されます。
1. 対象SFPの**entPhysicalIndex（物理エンティティインデックス）を特定**します。
2. センサーインデックスの特定と値の取得:CISCO-ENTITY-SENSOR-MIB (`1.3.6.1.4.1.9.9.91...`) を使用します。cessSensorTypeでreceiverPower（受光レベル）に対応するセンサーを探し、**最終的な値はcessSensorsValue (1.3.6.1.4.1.9.9.91.1.1.1.1.4) から取得**します。

⚠ **注意点:**
機種やIOSバージョンにより、DDM機能のサポート状況が異なります。
SNMP Walkで探索するのが最も確実です。

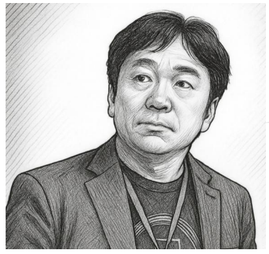
実機の値を取ってみる



snmpwalk -v2c -c public 192.168.1.5 1.3.6.1.4.1.9.9.91

SNMPv2-SMI::enterprises.9.9.91.1.1.1.1.1.1016 = INTEGER: 8
SNMPv2-SMI::enterprises.9.9.91.1.1.1.1.1.1017 = INTEGER: 4
～中略～
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1018.1 = INTEGER: 150
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1018.2 = INTEGER: 120
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1018.3 = INTEGER: 20
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1018.4 = INTEGER: 10
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1019.1 = INTEGER: 0
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1019.2 = INTEGER: -30
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1019.3 = INTEGER: -95
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1019.4 = INTEGER: -135
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1020.1 = INTEGER: 30
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1020.2 = INTEGER: 0
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1020.3 = INTEGER: -170
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.4.1020.4 = INTEGER: -210
～後略～
SNMPv2-SMI::enterprises.9.9.91.1.2.1.1.6.1020.4 = INTEGER: 1
（全部で270行）

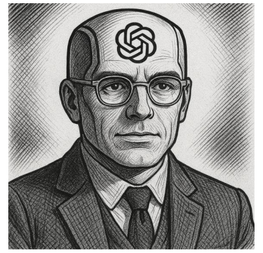
実機の値を見せてみる

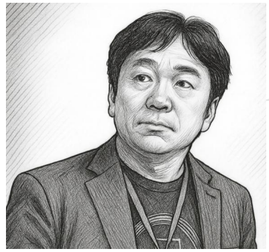


Catalyst2960Lでは、以下のような返りがあります

SNMPv2-SMI::enterprises.9.9.91.1.1.1.1.1.1016 = INTEGER: 8
SNMPv2-SMI::enterprises.9.9.91.1.1.1.1.1.1017 = INTEGER: 4
～以下略（270行分）～

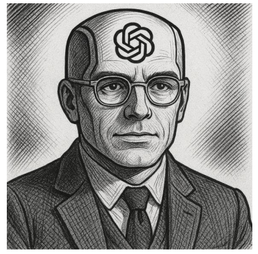
なるほど、Catalyst 2960L では CISCO-ENTITY-SENSOR-MIB (1.3.6.1.4.1.9.9.91) を通じて SFP の DOM 情報が取得できていますね 👍





・・・構造を分析して

分析結果 1/3



1. OID の構造

コードをコピーする

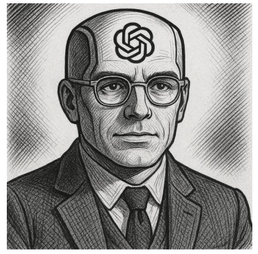
```
.1.3.6.1.4.1.9.9.91.1.1.1.1.x.YYYY
```

- `x` ... フィールドの種類
- `YYYY` ... entPhysicalIndex（物理エントリID。ポートごとやセンサーごとに異なる）

主なフィールドは次の通りです：

サフィックス (x)	意味 (entSensorTable)
1	entSensorType （単位の種類。例：dBm, mV, °C などを表す整数）
2	entSensorScale（倍率。例：units, milli, micro など）
3	entSensorPrecision（小数点以下の桁数）
4	entSensorValue（センサーの実測値）
5	entSensorOperStatus（状態: 1=ok, 2=unavailable, 3=nonoperational）
6	entSensorValueTimeStamp
7	entSensorValueUpdateRate

分析結果 2/3



2. Catalyst2960L の返り値例と解釈

例） `1019` と `1020` はそれぞれ SFP の Rx Power センサーを表している可能性が高いです。

makefile　　コードをコピーする

```
SNMPv2-SMI::enterprises.9.9.91.1.1.1.1.1.1019 = INTEGER: 14   ← entSensorType (dBm)
SNMPv2-SMI::enterprises.9.9.91.1.1.1.1.2.1019 = INTEGER: 9    ← entSensorScale (10^-3 = milli)
SNMPv2-SMI::enterprises.9.9.91.1.1.1.1.3.1019 = INTEGER: 1    ← entSensorPrecision (小数1桁)
SNMPv2-SMI::enterprises.9.9.91.1.1.1.1.4.1019 = INTEGER: -53  ← entSensorValue
SNMPv2-SMI::enterprises.9.9.91.1.1.1.1.5.1019 = INTEGER: 1    ← entSensorOperStatus (ok)
```

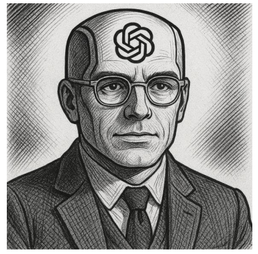
これを解釈すると：

- `entSensorType=14` → dBm（光レベル単位）
- `entSensorScale=9` → milli (10^-3)
- `entSensorValue=-53`
- `entSensorPrecision=1`

したがって、**受光レベル = -5.3 dBm** になります。

同様に `1020` の場合は `-57` → **-5.7 dBm** です。

分析結果 3/3



3. まとめ

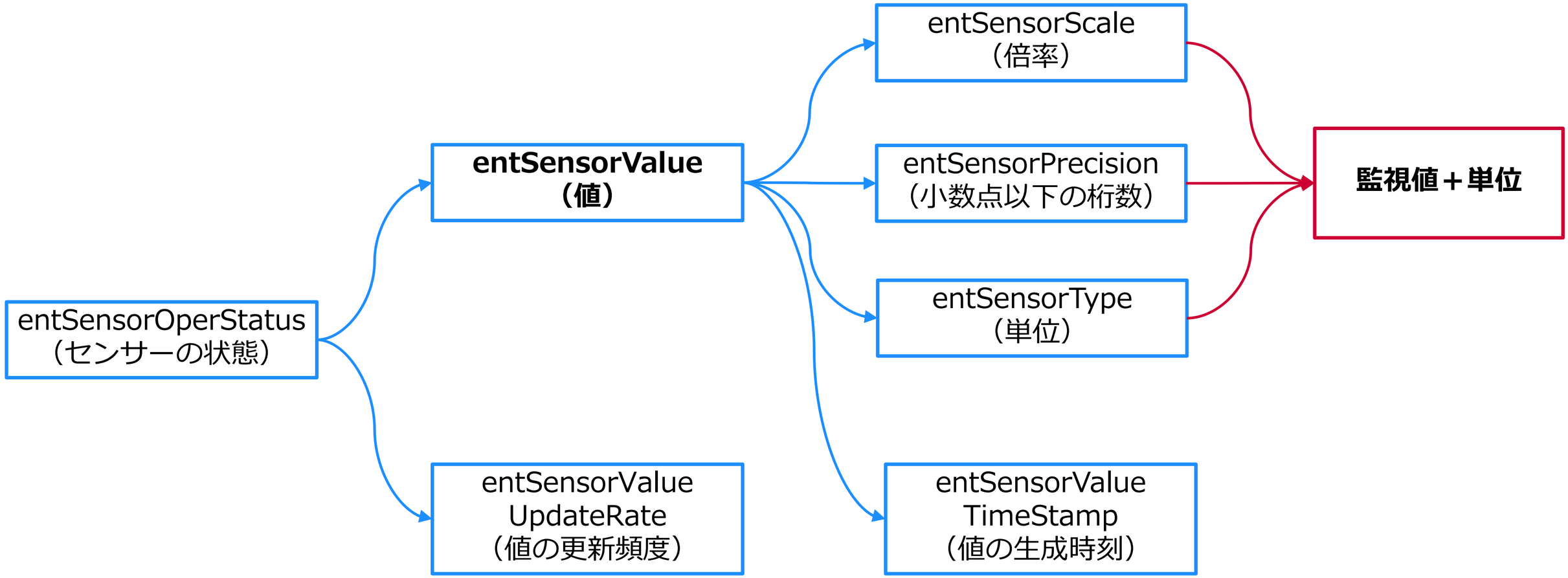
Catalyst 2960L の SFP 受光レベル (Rx Power) OID は以下の通りです：

- **ベース OID**: `1.3.6.1.4.1.9.9.91.1.1.1.1.4.{index}`
  - `{index}` = entPhysicalIndex（ポートごとの番号）
  - 値は整数（例：-53）
- 単位は `entSensorType` + `entSensorScale` + `entSensorPrecision` を組み合わせて解釈する
  - 今回のケースでは → dBm, milli, precision=1 → -5.3 dBm

👉 つまり、Catalyst 2960L では **CISCO-ENTITY-SENSOR-MIB の entSensorValue OID** を使えば、SFP の受光レベルを取得可能です。

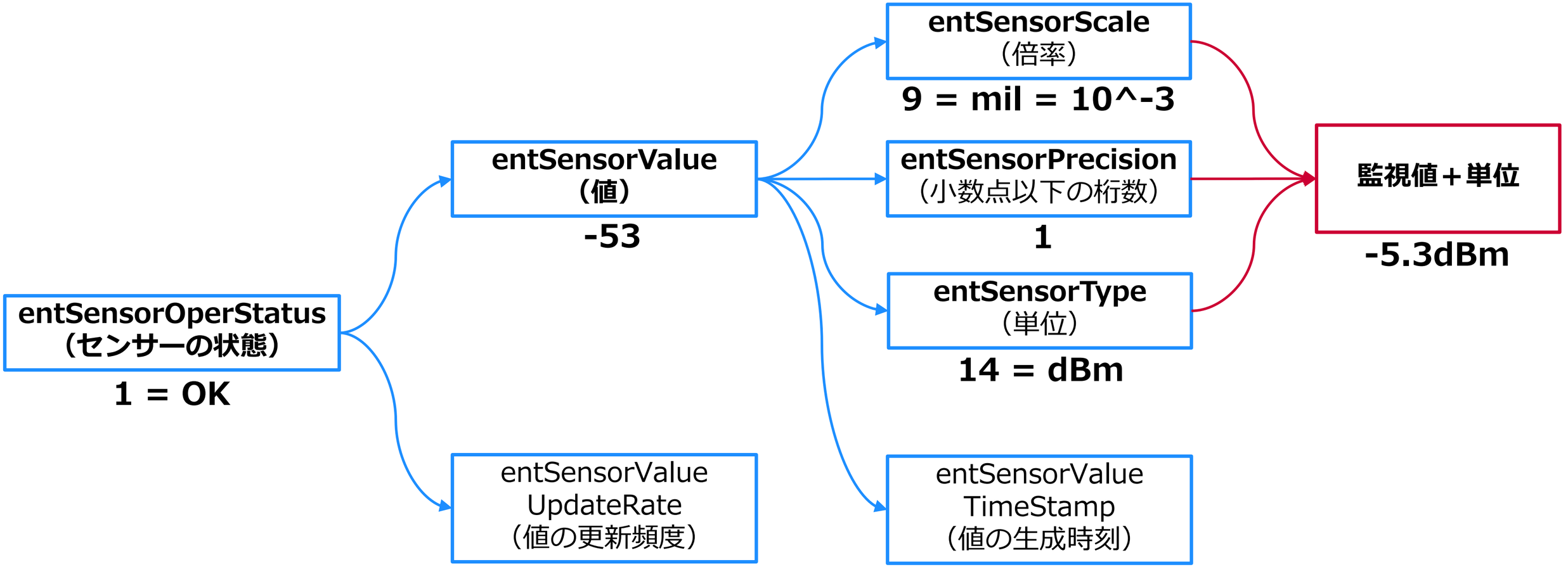
センサーとOIDの情報構造

分析結果から図示した構造は以下の様になります

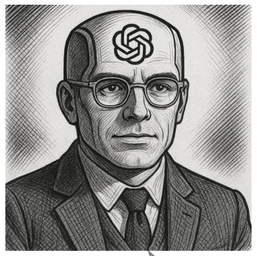


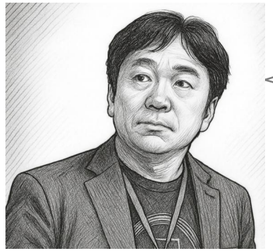
センサー情報とOIDの構造（受光レベル）

分析結果から図示した構造は以下の様になります



実装（LLDルール）





作り方（Zabbix 6/7/8 共通イメージ）

1) ディスカバリルール（SNMP agent）

名前: `ENTITY-SENSOR dBm sensors`

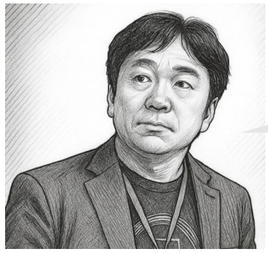
キー: `sensor.dbm.discovery`

SNMP OID: `1.3.6.1.4.1.9.9.91.1.1.1.1.1`（=entSensorType のテーブルwalk）

- `{#TYPE}` … `1.3.6.1.4.1.9.9.91.1.1.1.1.1.{#SNMPINDEX}`
- `{#SCALE}` … `1.3.6.1.4.1.9.9.91.1.1.1.1.2.{#SNMPINDEX}`
- `{#PREC}` … `1.3.6.1.4.1.9.9.91.1.1.1.1.3.{#SNMPINDEX}`
- `{#NAME}` … `1.3.6.1.2.1.47.1.1.1.1.7.{#SNMPINDEX}`（entPhysicalName）

discovery[{#TYPE},.1.3.6.1.4.1.9.9.91.1.1.1.1.1,{#SCALE},.1.3.6.1.4.1.9.9.91.1.1.1.1.2,{#PREC},.1.3.6.1.4.1.9.9.91.1.1.1.1.3,{#NAME},.1.3.6.1.2.1.47.1.1.1.1.7]

実装（LLDルール：自動生成対象のフィルタリング）



SFPを含む全ての物理ポートのアイテムが生成されてしまいました。
SFPのみを対象にしたいです

解決方法（フィルタで SFP のみを残す）

1. LLDルールのフィルタ条件追加

「ディスカバリルール → フィルタ」タブで、以下の条件を指定します。

- **フィルタA**

  `{#NAME} MATCHES "Gi0/4[9-9]|Gi0/5[0-2]"`

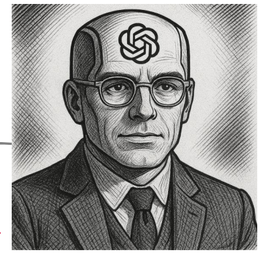
  → つまり Gi0/49, Gi0/50, Gi0/51, Gi0/52 のみ残す

  （2960L-48TS-LL など uplinkポート番号は固定）
- **フィルタB**（任意・さらに精度を上げる場合）

  `{#NAME} MATCHES "(Temperature|Voltage|Current|Power)"`

  → SFPセンサー系だけ拾う（ifNameとか不要なポート名を除外）

この二つを「AND」で組み合わせると、SFPセンサーのみ抽出できます。



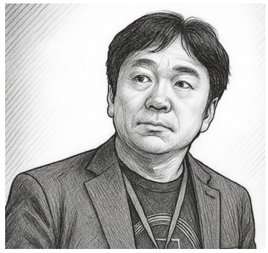
2. もっと汎用的にする方法

もし将来 24ポート版や他モデルも混ぜる場合は、次のようにします。

- `{#NAME} MATCHES "Module Temperature|Bias Current|Transmit Power|Receive Power|Supply Voltage"`

  （= SFPモジュールに特有のセンサー名を拾う）
- これだけで “GigabitEthernetX raw/status” 系は除外されます。

実装（LLDルール：自動生成対象のフィルタリング）



ディスカバリルール　保存前処理　LLDマクロ　フィルター 1　オーバーライド 8

フィルター　ラベル　マクロ　正規表現　アクション

A　{#NAME}　一致する　(?:Module Temperature Sensor|Supply Voltage Sensor　削除

追加

更新　複製　テスト　削除　キャンセル

(?:Module Temperature Sensor|Supply Voltage Sensor|Bias Current Sensor|Transmit Power Sensor|Receive Power Sensor)

アイテム

Catalyst2960L-48TS-LL　有効　SNMP　アイテム 2562　トリガー 538　グラフ 472　ディスカバリルール 14　Webシナリオ

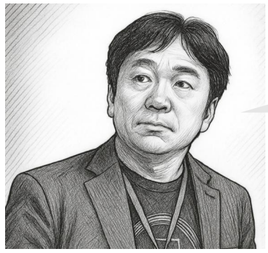
Gi0/49の情報

	トリガー	キー	監視間隔	ヒストリ	トレンド	タイプ	ステータス
SFP discovery	Gi0/49 Receive Power Sensor (raw data): Gi0/49 Receive Power Sensor (RX dBm)		sensor.rx-power.value[1062]	31d	365d	依存アイテム	有効
SFP discovery	Gi0/49 Transmit Power Sensor (raw data): Gi0/49 Transmit Power Sensor (TX dBm)		sensor.tx-power.value[1061]	31d	365d	依存アイテム	有効
SFP discovery	Gi0/51 Receive Power Sensor (raw data): Gi0/51 Receive Power Sensor (RX dBm)		sensor.rx-power.value[1069]	31d	365d	依存アイテム	有効
SFP discovery	Gi0/51 Transmit Power Sensor (raw data): Gi0/51 Transmit Power Sensor (TX dBm)		sensor.tx-power.value[1068]	31d	365d	依存アイテム	有効

Gi0/51の情報

監視データ取得　ヒストリとトレンドを削除　コピー　一括更新　削除

実装（LLDルール：値に合わせた単位の付与）



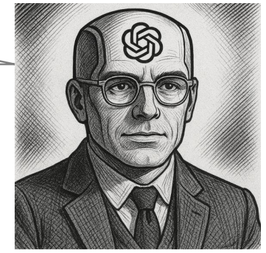
アイテムには、取得した値に合致する単位を表示させる方法が必要です。

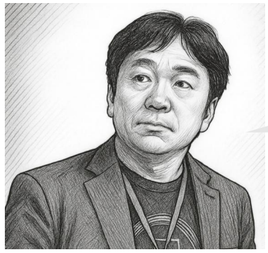
今回やりたいこと（センサー種別ごとに倍率・単位を振り分け）

あなたの LLD は `{#TYPE}` を持っていますよね（8=温度, 4=電圧, 5=電流, 14=dBm）。

依存アイテム `sensor.value[{#SNMPINDEX}]` を1個だけ用意しておき、

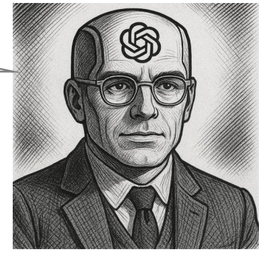
オーバーライドで TYPE ごとに**前処理（乗数）と単位**を上書きします。



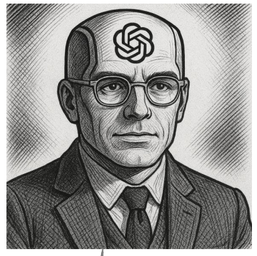


Zabbix 7.0ではオーバーライドで単位を設定できないんだけど、これは仕様？

- Zabbix 7.0 のオーバーライドでは「単位」は変更不可 → 仕様
- 解決法は **プロトタイプ分割** or **単位なし＋名前で区別**
- Zabbix 8.0 系では改善される可能性がありますが、7.0 LTS では仕様のままです。



実装（LLDルール：アイテムプロトタイプ）



(4) Tx/Rx Power (dBm) 用

- 名前: `{#NAME} Optical Power`
- キー: `sensor.power.value[{#SNMPINDEX}]`
- データ型: 数値(浮動小数)
- 単位: `dBm`
- 前処理: 乗数 `0.1`（raw が -63 → -6.3 dBm）

アイテムのプロトタイプ

アイテムのプロトタイプ　タグ 1　保存前処理 1

* 名前	{#NAME} (RX dBm)
タイプ	依存アイテム
* キー	sensor.rx-power.value[{#SNMPINDEX}] 選択
データ型	数値 (浮動小数)
* マスターアイテム	Template_Network_Cisco_Catalyst2960L: {#NAME} (raw data) ✕ 選択
単位	dBm
* ヒストリ	保存しない / 保存期間 31d
* トレンド	保存しない / 保存期間 365d
値のマッピング	検索文字列を入力 選択
説明	
自動生成時のステータス有効	☐

アイテムのプロトタイプ　タグ 1　保存前処理 1

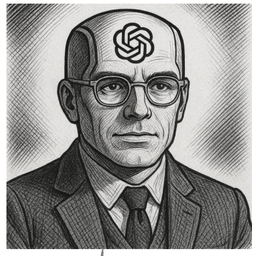
保存前処理の設定　名前　パラメータ

1: 乗数　0.1

追加

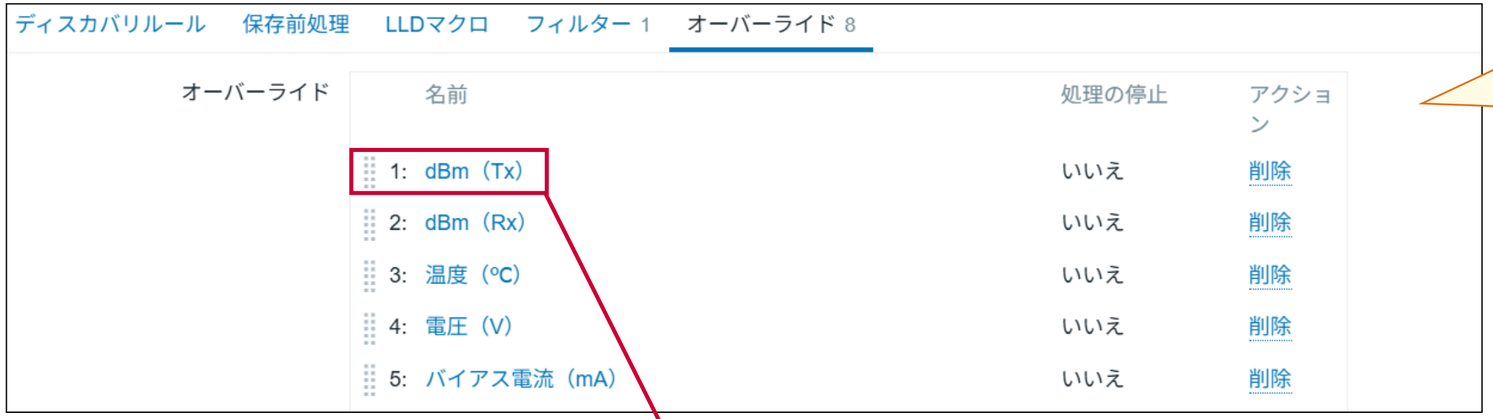
データ型　数値 (浮動小数)

実装（LLDルール：オーバーライド）

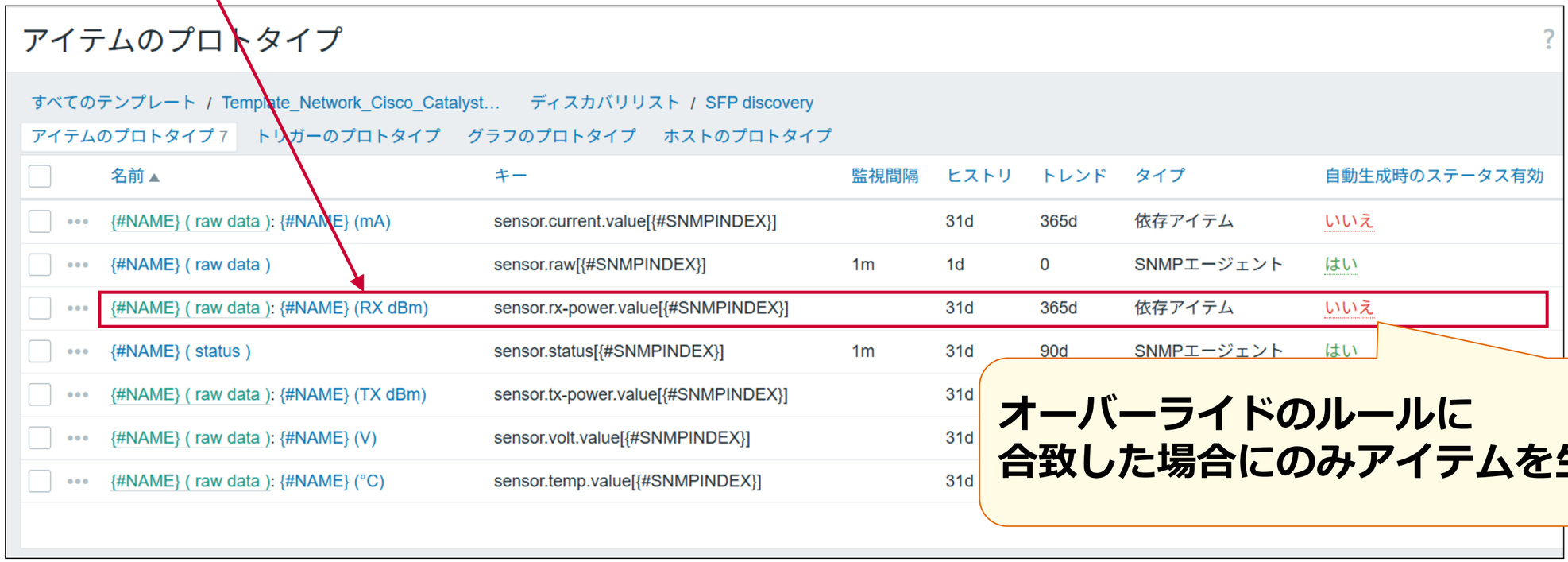


1. **プロトタイプ既定を無効**（raw/status だけ有効）← これはOK
2. **フィルター文字列を厳密に**
   - `{#TYPE}` を使うなら：`一致する` に `^14$` `^8$` `^4$` `^5$` を使用
   - `{#NAME}` を使うなら：
     - Tx: `^.*Transmit Power Sensor$`
     - Rx: `^.*Receive Power Sensor$`
     - 温度: `^.*Module Temperature Sensor$`
     - 電圧: `^.*Supply Voltage Sensor$`
     - 電流: `^.*Bias Current Sensor$`
3. **オーバーライドの並び順と停止**
   - それぞれの上書きで「処理の停止」に切り替える（マッチしたら以降の上書きを止める）
4. **“実行内容 → アイテムのプロトタイプ一致する” の名前が完全一致**か確認
   - プロトタイプ名の小文字/大文字・スペースが一致していないと効きません
5. **LLDの再生成**
   - ホスト → ディスカバリ → 生成済みを削除 → ルール無効→有効 → 今すぐ検出

実装（LLDルール：オーバーライド）

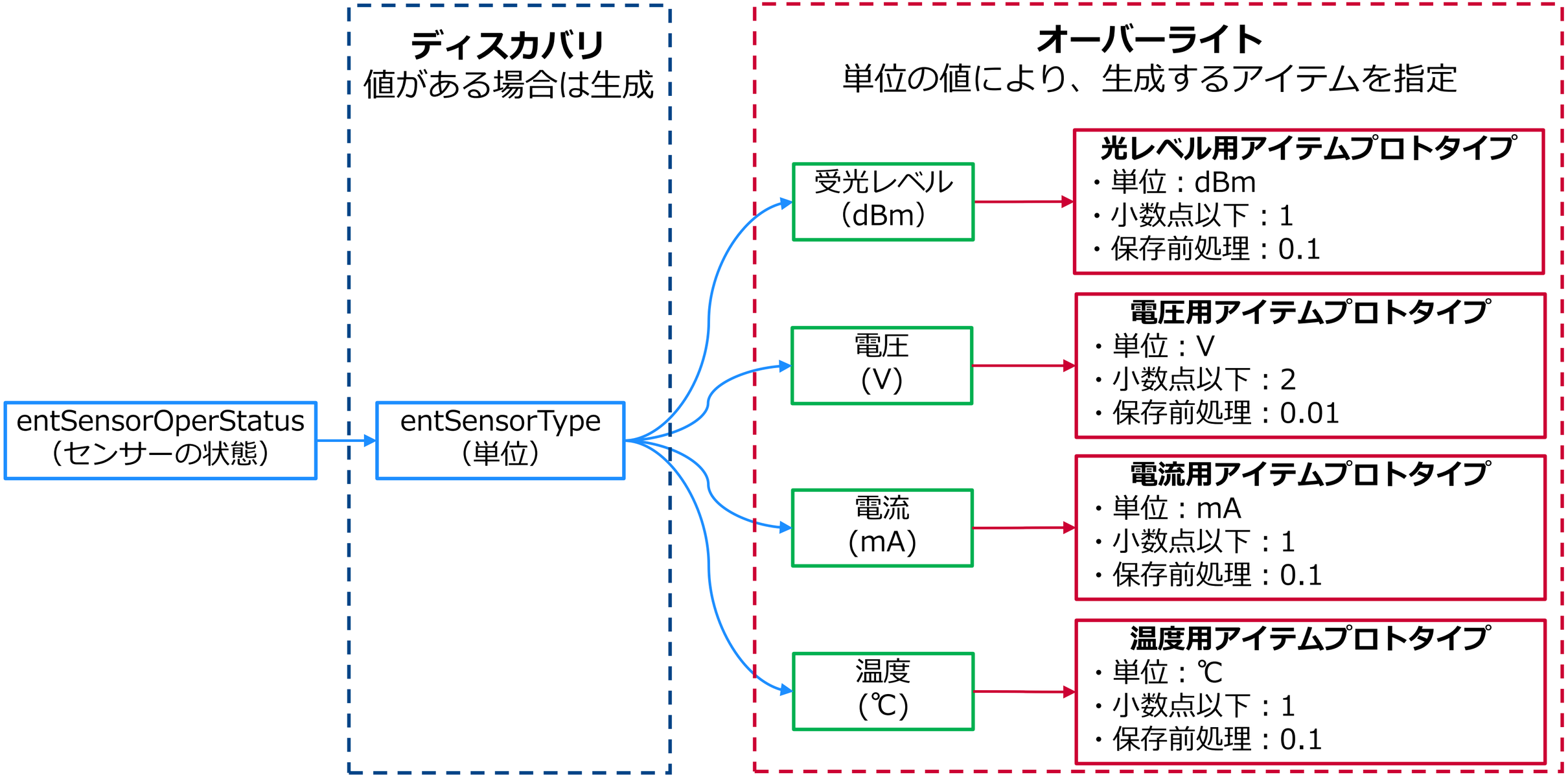


単位の種類毎にオーバーライドのルールを設定



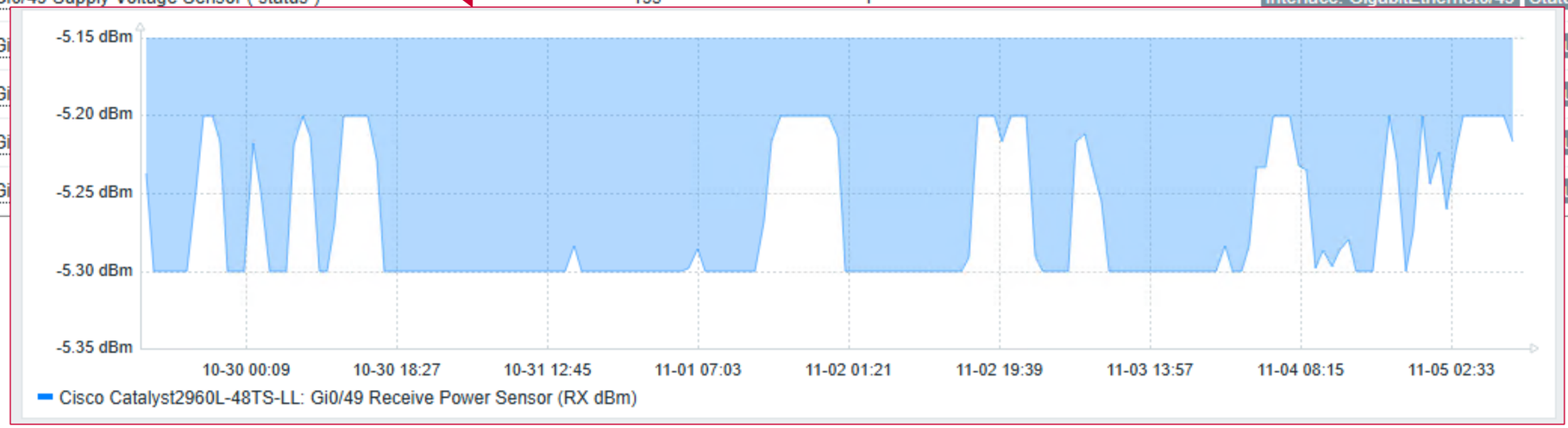
オーバーライドのルールに合致した場合にのみアイテムを生成

LLDの構造

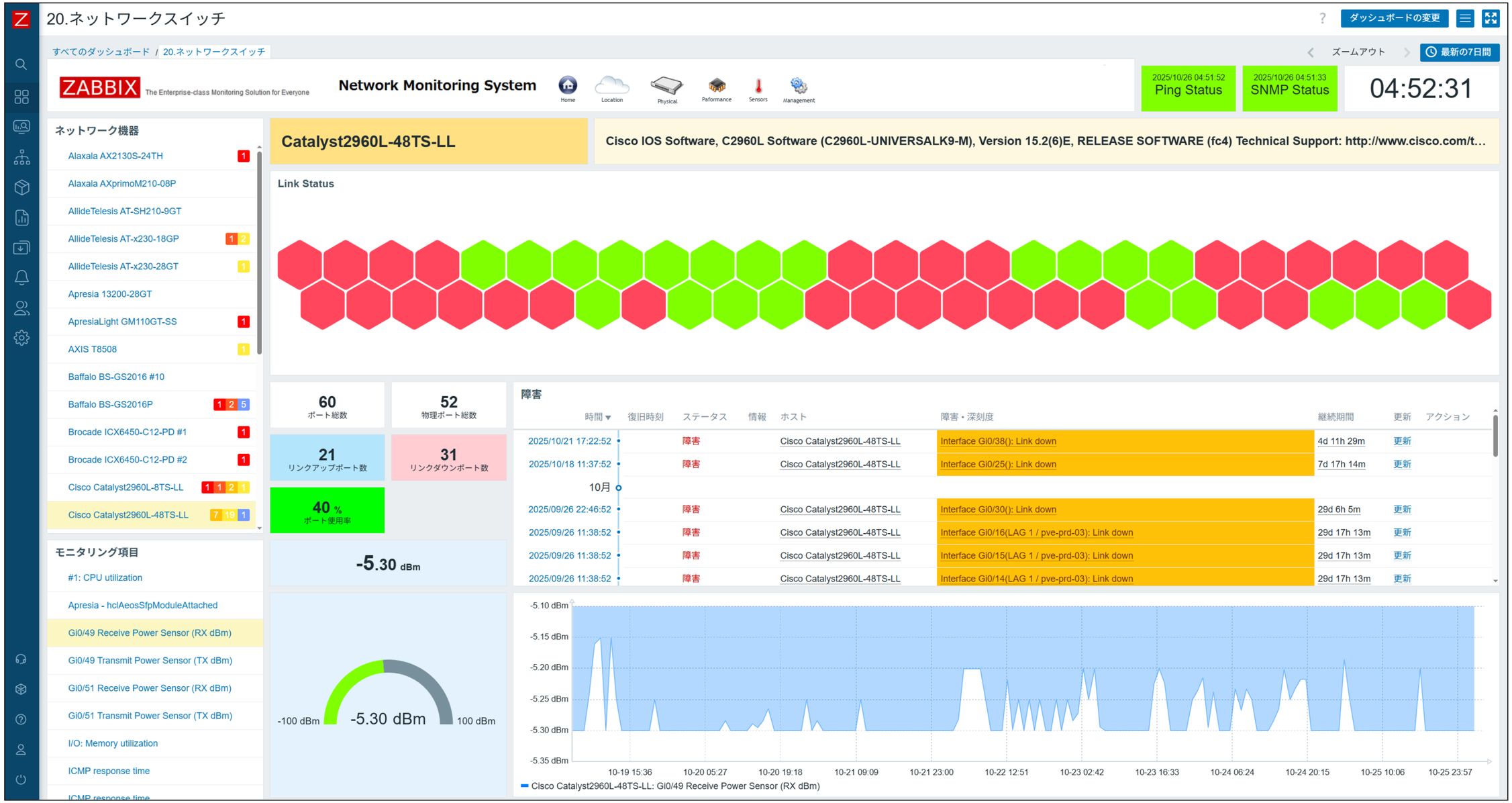


動作試験

ホスト	名前 ▲	最新のチェック時刻	最新の値	変化	タグ
Cisco Catalyst2960L-48TS-LL	Gi0/49 Bias Current Sensor (raw data)	18s	28		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Bias Current Sensor (status)	18s	1		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Bias Current Sensor (V)	18s	0.28 V		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Module Temperature Sensor (raw data)	18s	474		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Module Temperature Sensor (status)	18s	1		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Module Temperature Sensor (°C)	18s	47.4 °C		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Receive Power Sensor (raw data)	18s	-53		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Receive Power Sensor (RX dBm)	18s	-5.3 dBm		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Receive Power Sensor (status)	19s	1		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Supply Voltage Sensor (raw data)	18s	337		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi0/49 Supply Voltage Sensor (status)	19s	1		Interface: GigabitEthernet0/49 Status: SFP
Cisco Catalyst2960L-48TS-LL	Gi				us: SFP
Cisco Catalyst2960L-48TS-LL	Gi				us: SFP
Cisco Catalyst2960L-48TS-LL	Gi				us: SFP
Cisco Catalyst2960L-48TS-LL	Gi				us: SFP



ダッシュボード事例



Allied Telesis AT-x230 の場合

実機の用意(Allied Telesis)

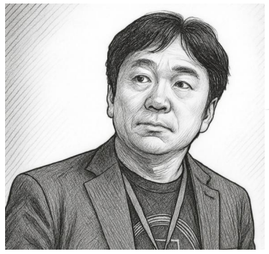
Allied Telesis製品の検証には以下の機材を使用

＜L2スイッチ＞
品名：AT-x230-28GT

＜SFPモジュール＞
品名：GCL-SX-MMD

AT-x230-28GT
Allied Telesis

実機の値を取ってみる

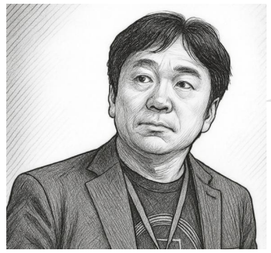


snmpwalk -v2c -c public 192.168.1.6 .1.3.6.1.4.1.207.8.4.4.3.28.1

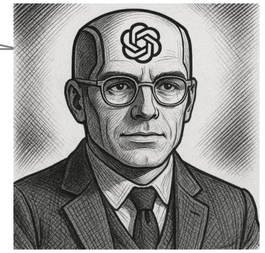
```
.1.3.6.1.4.1.207.8.4.4.3.28.1.1.1.3.5025.1 = INTEGER: 46468 0.001 degree C
.1.3.6.1.4.1.207.8.4.4.3.28.1.1.1.3.5027.1 = INTEGER: 29132 0.001 degree C
～中略～
.1.3.6.1.4.1.207.8.4.4.3.28.1.3.1.9.5025.1 = INTEGER: 1000 0.001 mA
.1.3.6.1.4.1.207.8.4.4.3.28.1.3.1.9.5027.1 = INTEGER: 2000 0.001 mA
.1.3.6.1.4.1.207.8.4.4.3.28.1.4.1.3.5025.1 = INTEGER: 2295 0.0001 mW
.1.3.6.1.4.1.207.8.4.4.3.28.1.4.1.3.5027.1 = INTEGER: 3166 0.0001 mW
～後略～
.1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.7.5027.1 = STRING: "Low"
.1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.8.5025.1 = INTEGER: 10000 0.0001 mW
.1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.8.5027.1 = INTEGER: 10000 0.0001 mW
.1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.9.5025.1 = INTEGER: 199 0.0001 mW
.1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.9.5027.1 = INTEGER: 200 0.0001 mW
.1.3.6.1.4.1.207.8.4.4.3.28.1.6.1.3.5025.1 = STRING: "Rx Up"
.1.3.6.1.4.1.207.8.4.4.3.28.1.6.1.3.5027.1 = STRING: "Rx Down"
```

（全部で72行）

分析結果



構造を分析して

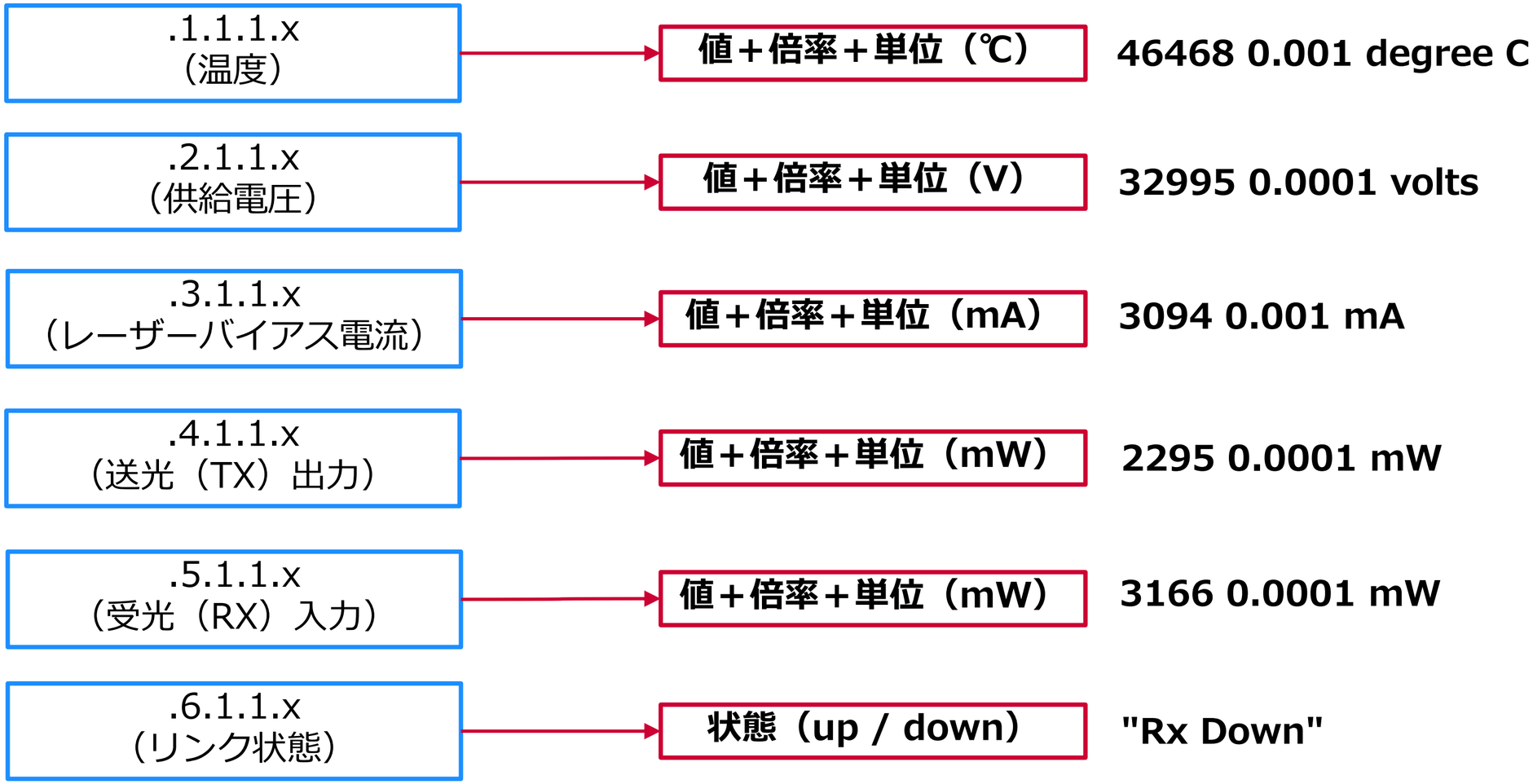


今回の配列を見ると **ベースツリーは** `.1.3.6.1.4.1.207.8.4.4.3.28.1` で、下位はだいたい次の表構造です（単位スケールはベンダ表記）：

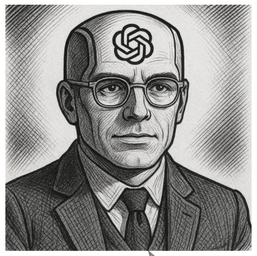
ブロック	意味	現在値列	閾値（例）	スケール
`.1.1.1.x`	温度	`.3`	`.5` 高アラーム / `.6` 低アラーム / `.8` 高警告 / `.9` 低警告	**0.001 °C**
`.2.1.1.x`	供給電圧	`.3`	`.5` HighAlarm / `.6` LowAlarm / `.8` HighWarn / `.9` LowWarn	**0.0001 V**
`.3.1.1.x`	レーザーバイアス電流	`.3`	`.5` HighAlarm / `.6` LowAlarm / `.8` HighWarn / `.9` LowWarn	**0.001 mA**
`.4.1.1.x`	送光（TX）出力	`.3`	`.5` HighAlarm / `.6` LowAlarm / `.8` HighWarn / `.9` LowWarn	**0.0001 mW**
`.5.1.1.x`	受光（RX）入力	`.3`	`.5` HighAlarm / `.6` LowAlarm / `.8` HighWarn / `.9` LowWarn	**0.0001 mW**
`.6.1.1.x`	リンク状態	`.3`（“Rx Up/Down”）	—	文字列

センサーとOIDの情報構造

分析結果から図示した構造は以下の様になります。
.1.3.6.1.4.1.207.8.4.4.3.28.1 をベースとして、センサー種類毎にOIDが用意されています。



実装（LLDルール）



1) ディスカバリ・ルール（SNMP LLD）

- ルール名：`SFP DOM discovery`
- タイプ：`SNMPエージェント`
- キー：`allied.sfp.discovery`
- **SNMP OID（表走査）**：

`discovery[{#SFP_STATE},.1.3.6.1.4.1.207.8.4.4.3.28.1.6.1.3]`

→ 同一行の状態文字列（`Rx Up/Down`）を `{#SFP_STATE}` に入れる。

※ インデックスは自動で `{#SNMPINDEX}` に入ります（例：`5025.1`）。

- LLDマクロ（**タイプ＝SNMP OID** で追加）

bash　コードをコピーする

```
{#TEMP_NOW}    = .1.3.6.1.4.1.207.8.4.4.3.28.1.1.1.3
{#VCC_NOW}     = .1.3.6.1.4.1.207.8.4.4.3.28.1.2.1.3
{#BIAS_NOW}    = .1.3.6.1.4.1.207.8.4.4.3.28.1.3.1.3
{#TXP_NOW}     = .1.3.6.1.4.1.207.8.4.4.3.28.1.4.1.3
{#RXP_NOW}     = .1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.3
{#RXLEV_STR}   = .1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.4   ; 例 "Low" / "-"
```

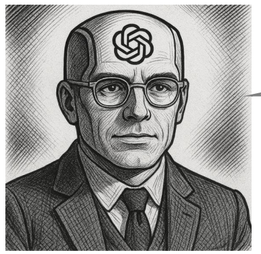
（必要に応じて閾値列 `.5/.6/.8/.9` も追加可）

ディスカバリルール　保存前処理　LLDマクロ　フィルター 1　オーバーライド

* 名前	SFP discovery
タイプ	SNMPエージェント
* キー	allied.sfp.discovery
* SNMP OID	discovery[{#SFP_STATE},.1.3.6.1.4.1.207.8.4.4.3.28.1.6.1.3,{#TEMP_NOW},.1.3.6.
* 監視間隔	1h
監視間隔のカスタマイズ	タイプ：例外設定 / 定期設定　監視間隔：50s　期間：1-7,00:00-24:00　アクション：削除 追加
* 存在しなくなったデバイスの削除	未監視 / すぐに / 以降　7d
* 存在しなくなったデバイスの無効化	未監視 / すぐに / 以降
説明	
有効	☑

更新　複製　テスト　削除　キャンセル

実装（LLDルール：自動生成対象のフィルタリング）

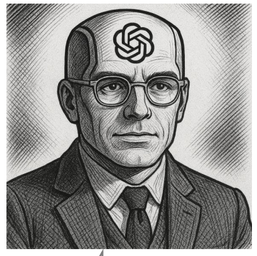


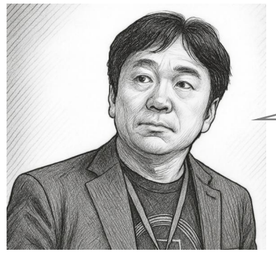
- フィルター（任意）
  - **空スロット排除の簡易策：** `{#TEMP_NOW}` が `0` でない など
    → ラベルA：`{#TEMP_NOW}`「一致しない」`^0$`
  - もしくは `{#SFP_STATE}` が `Rx Down` でも監視したいならフィルターなしでOK

SNMP OID入力内容（前ページ）

discovery[{#SFP_STATE},.1.3.6.1.4.1.207.8.4.4.3.28.1.6.1.3,{#TEMP_NOW},.1.3.6.1.4.1.207.8.4.4.3.28.1.1.1.3,{#VCC_NOW},.1.3.6.1.4.1.207.8.4.4.3.28.1.2.1.3,{#BIAS_NOW},.1.3.6.1.4.1.207.8.4.4.3.28.1.3.1.3,{#TXP_NOW},.1.3.6.1.4.1.207.8.4.4.3.28.1.4.1.3,{#RXP_NOW},.1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.3,{#RXLEV_STR},.1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.4]

実装（LLDルール：アイテムプロトタイプ）





(B) TX/RX 光レベル

- TX optical power (mW)
  - OID: `.28.1.4.1.3.{#SNMPINDEX}`
  - 単位: `mW`
  - 前処理: 乗数 `0.0001`
- RX optical power (mW)
  - OID: `.28.1.5.1.3.{#SNMPINDEX}`
  - 単位: `mW`
  - 前処理: 乗数 `0.0001`
- RX optical level (state)（文字列）
  - OID: `.28.1.5.1.4.{#SNMPINDEX}`
  - 情報タイプ: テキスト（例 `"Low"`, `"-"` など）

> 使い勝手向上のため **dBm への換算アイテム（依存アイテム）**も用意します。

アイテムのプロトタイプ

アイテムのプロトタイプ　タグ 1　保存前処理 1

* 名前: {#SNMPINDEX} : RX optical power (mW)
タイプ: SNMPエージェント
* キー: allied.sfp.rx.mw[{#SNMPINDEX}]　選択
データ型: 数値 (浮動小数)
* SNMP OID: .1.3.6.1.4.1.207.8.4.4.3.28.1.5.1.3.{#SNMPINDEX}
単位: mW
* 監視間隔: 1m
監視間隔のカスタマイズ

タイプ	監視間隔	期間	アクション
例外設定 / 定期設定	50s	1-7,00:00-24:00	削除

追加

アイテムのプロトタイプ　タグ 1　保存前処理 1

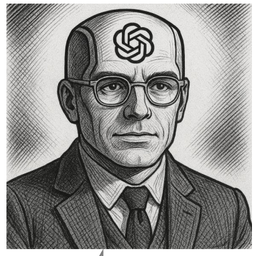
保存前処理の設定

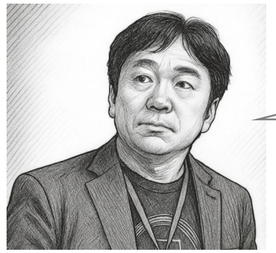
	名前	パラメータ
1:	乗数	0.0001

追加

データ型: 数値 (浮動小数)

実装（LLDルール：依存アイテム）





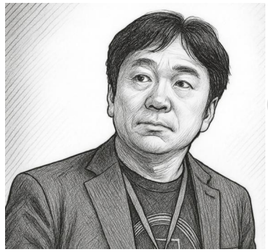
(C) dBm（依存アイテム）

- **TX optical power (dBm)**
  - タイプ: `依存アイテム`（親：TX mW）
  - 単位: `dBm`
  - 前処理: `JavaScript: 10*Math.log10(value)`
- **RX optical power (dBm)**
  - タイプ: `依存アイテム`（親：RX mW）
  - 単位: `dBm`
  - 前処理: `JavaScript: 10*Math.log10(value)`

> mW が 0 のときに `-Inf` になるため、直前に「**しきい値外は破棄**（≤0 を破棄）」または「**JS内で0は-99に置換**」などを入れておくと実運用で安定します。
> 例：JS
> `if (value<=0) { return 1e-10; } else { return 10*Math.log10(value); }`

if (value<=0) { return 1e-10; } else { return 10*Math.log10(value); }

この変換式は正しいか？



mWをdBmに変換するこのコードは正しいのか？（文系サーバエンジニアには辛いところ）

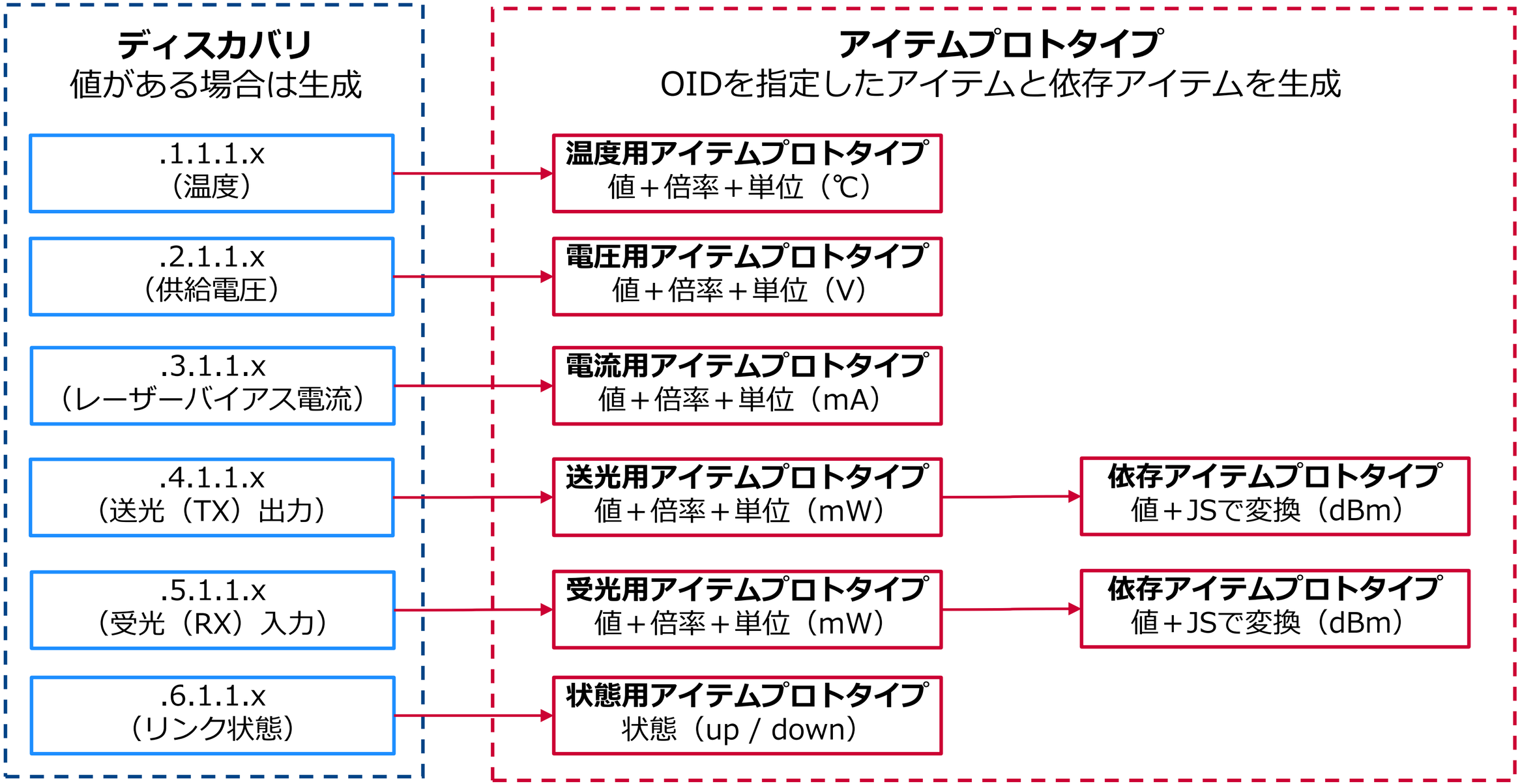
これは何をやっている式？
if (value<=0) { return 1e-10; } else { return 10*Math.log10(value); }

💻 式の動作

この式は、プログラミングで一般的に見られる**条件分岐**（`if-else` 文）を利用しています。

1. **条件判定:**
   - まず、入力された `value` が **0 以下**（`value <= 0`）であるかをチェックします。
2. `value <= 0` **の場合 (真数 0 以下の処理):**
   - 条件が真の場合、1×10^{-10} (または 0.0000000001) という**非常に小さい正の定数**を返します。
   - これは、多くの場合、デシベル計算などで**真数がゼロや負の値**になると**対数が定義できなくなる**（計算エラーになる）のを防ぐための**ガード処理**です。この非常に小さい値を返すことで、エラーを回避しつつ、「ほぼゼロ」であることを示します。
3. `value > 0` **の場合 (真数 正の値の処理):**
   - 条件が偽の場合、`value` の**デシベル値 (dB)** を計算して返します。
   - 計算式は $10 \times \log_{10}(\text{value})$ で、これは**電力やエネルギー**などの物理量における**デシベル**を求める標準的な式です。

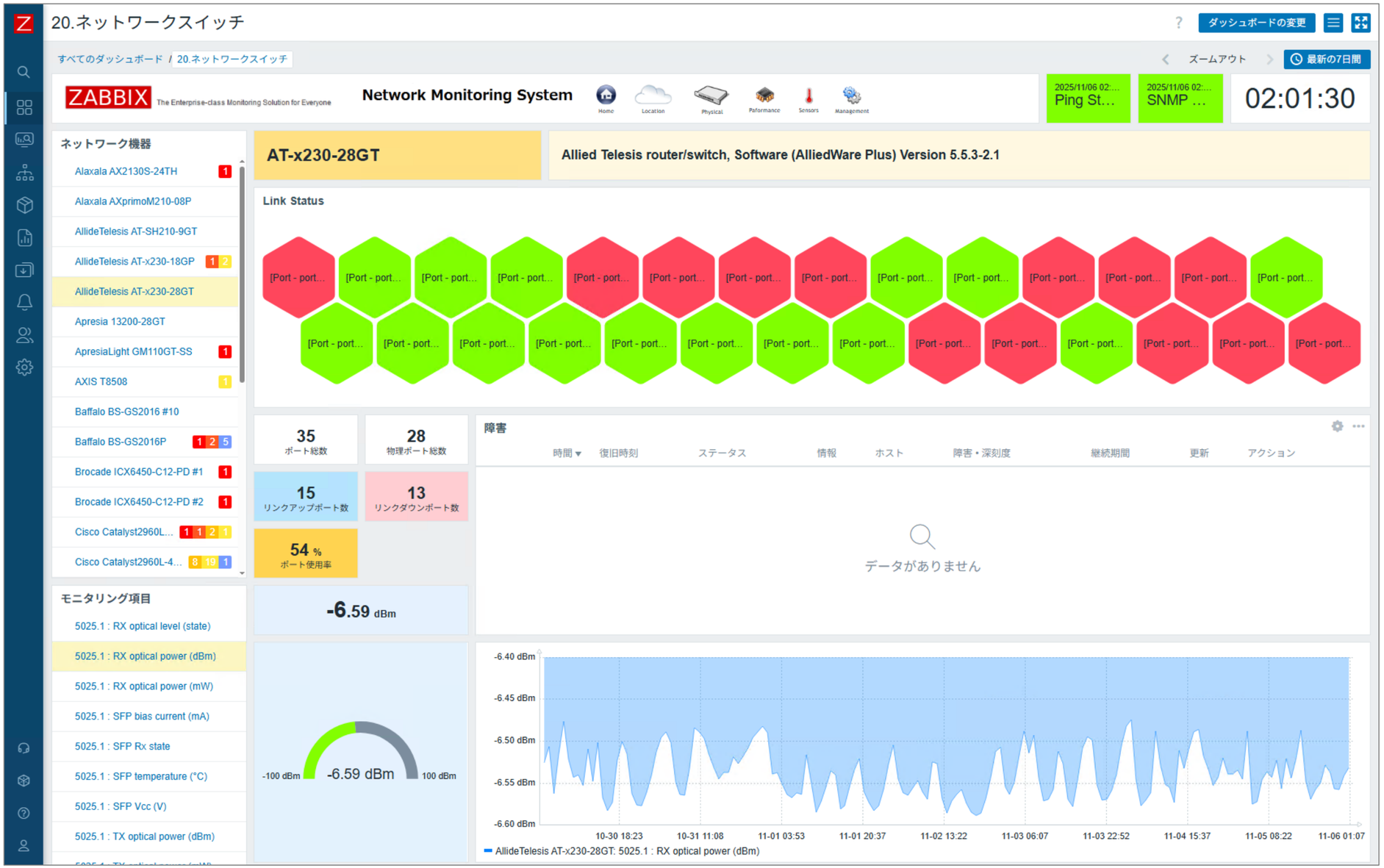
LLDの構造



動作試験

ホスト	名前 ▲	最新のチェック時刻	最新の値	変化	タグ
AllideTelesis AT-x230-28GT	5025.1 : RX optical level (state)	20s	-		Status: SFP
AllideTelesis AT-x230-28GT	5025.1 : RX optical power (dBm)	20s	-6.5916 dBm	+0.01588 dBm	Status: SFP
AllideTelesis AT-x230-28GT	5025.1 : RX optical power (mW)	20s	0.2192 mW	+0.0008 mW	Status: SFP
AllideTelesis AT-x230-28GT	5025.1 : SFP bias current (mA)	20s	3.068 mA	+0.014 mA	Status: SFP
AllideTelesis AT-x230-28GT	5025.1 : SFP Rx state	20s	2192		Status: SFP
AllideTelesis AT-x230-28GT	5025.1 : SFP temperature (°C)	20s	44.507 °C	-0.055 °C	Status: SFP
AllideTelesis AT-x230-28GT	5025.1 : SFP Vcc (V)	20s	3.3064 V	+0.0078 V	Status: SFP
AllideTelesis AT-x230-28GT	5025.1 : TX optical power (dBm)	20s	-6.3922 dBm	+0.0361 dBm	Status: SFP
AllideTelesis AT-x230-28GT	5025.1 : TX optical power (mW)	20s	0.2295 mW	+0.0019 mW	Status: SFP

ダッシュボード事例

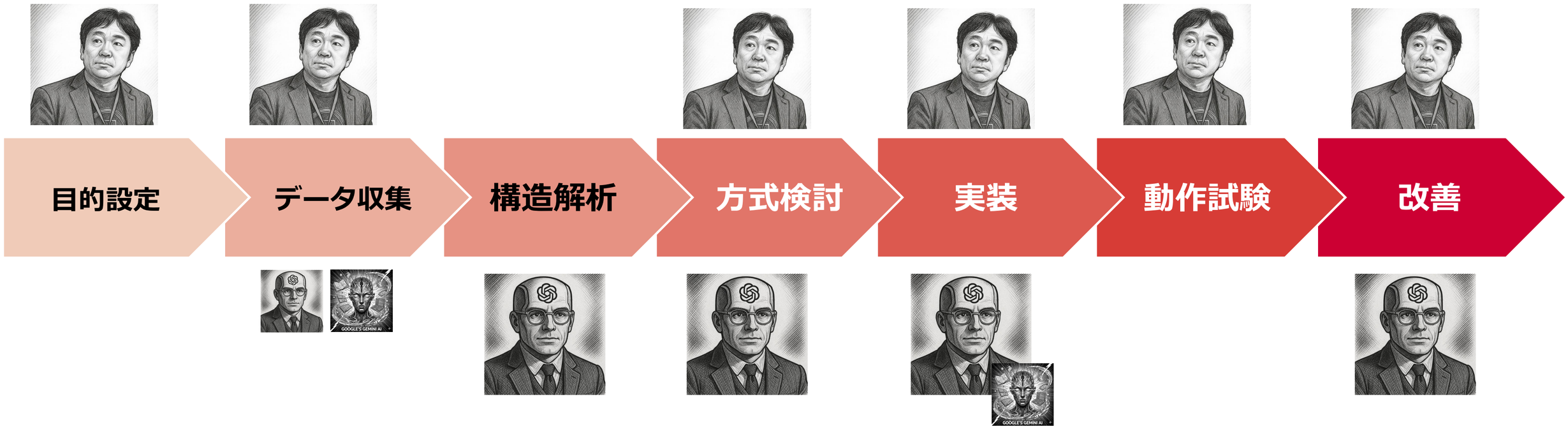


結論

監視テンプレート作成における分担

監視テンプレートの作成のフローにおける分担結果は以下の様になりました。

実機操作・設定作業を人が行う構成上、生成AIが**単独**で力を発揮するタスクは実機からの
大量なデータを読み込む**データ構造の解析**となります。
以後の工程において**設定の生成やエラー分析**などでは**十分な支援**を得ることができました。



生成AIの支援を得る場合の注意点 ①

①情報を省略する傾向

設定上必須となるパラメータを省略して回答をする場合がある。

・回答をそのまま入力しても設定が通らない
・複雑な内容や情報量が多い場合には、省略が顕著に表れる

⇒　省略されている個所を**補完できる知識**が必要

②正規表現や関数の扱い間違い

Zabbixの仕様や制限を考慮しない回答をする場合がある。

・正規表現として使用できないパターンや、異常なネストを繰り返す
・やり取りの回数が増えると、修正案として仕様や制限を無視した回答が増える

⇒　回答を鵜呑みにせず、動作を確かめながら使う**慎重さ**が必要
⇒　**公式マニュアルを参照**して仕様や制限に従っているかを確認する。

生成AIの支援を得る場合の注意点 ②

③多段の依存関係を考慮しない

3層以上の依存関係がある場合、通貫して整合が取れない回答をする場合がある。

・データ型が文字列のアイテムに対して、数値比較を行うトリガー条件式を提示する
・生成AIが指定したフィルタ条件に干渉する文字列の設定を提示する

⇒　Zabbixのデータ構造を理解し、**矛盾点を発見**できる知識が必要
⇒　公式マニュアル・公式研修により得られる知識と組合わせることで回避が可能

④誤情報をスルーする

過去の回答と事実関係が矛盾した場合、撤回はせずに回答を変えてくる。

⇒　**どの時点での情報**を基にした回答であるかの見極めが必要
⇒　**煽り耐性が低い人**には向かないかもしれない

生成AIの支援を得る場合の利点

①最新の情報を提供

Zabbixの新しい機能を用いた設定を提案してくるケースが多い。（学習データに依る）

- 使用する**Zabbixのバージョンを明示**した上で回答を求めると精度が上がる
- **モダンな設定**方法を取り入れやすい

②苦手な知識や分野の補完

Zabbixの設定は複雑化が進み、多様な要素に対する知見が求められることから、
不得手な箇所を補完する動きが期待できる。

- 正規表現の生成やJS作成などの**非プログラマ向けの支援**
- 実データからの**パターン解析**と解析結果を基に**実装方法を提案**がシームレスに行える
- 監視とは**異なる分野の知識**に基づく設計やコード生成が可能

生成AI活用により期待される効果

生成AIを活用することにより、エンジニア層毎に異なる利点があります。
現在の生成AIの水準では、**高度な設定の自動生成には至りません**が、
中堅・熟練層と**協業による高度な設定の作成の支援**としての活用は期待できます。

①若手エンジニア層
・ステップ・バイ・ステップで実装方法が提案されるため、複雑な設定の**自己解決**に寄与
・コーディングなどの不得手な分野に対する**実装可能な回答**の提示

②中堅エンジニア層
・分析や検討時間の短縮による**生産性の向上**

③熟練エンジニア層
・最新の情報や仕様に基づいた設計が提案されることによる**自己学習の負担を低減**