2.3.4 MQTT

MQTT（ MQ Telemetry Transport）は、近年出てきた新しいプロトコルです。IoT の世界では標準的なプロトコルにする取り組みも出てきています。元々はIBM社が提唱していたプロトコルですが、現在はオープンソースとなって開発が進んでいます。

MQTTはパブリッシュ／サブスクライブ型と呼ばれる1対多の通信が可能なプロトコルです。3つの役割からなっており、それぞれブローカー(Broker)、パブリッシャー(Publisher)、サブスクライバー(Subscriber）と呼びます（図2.8)。

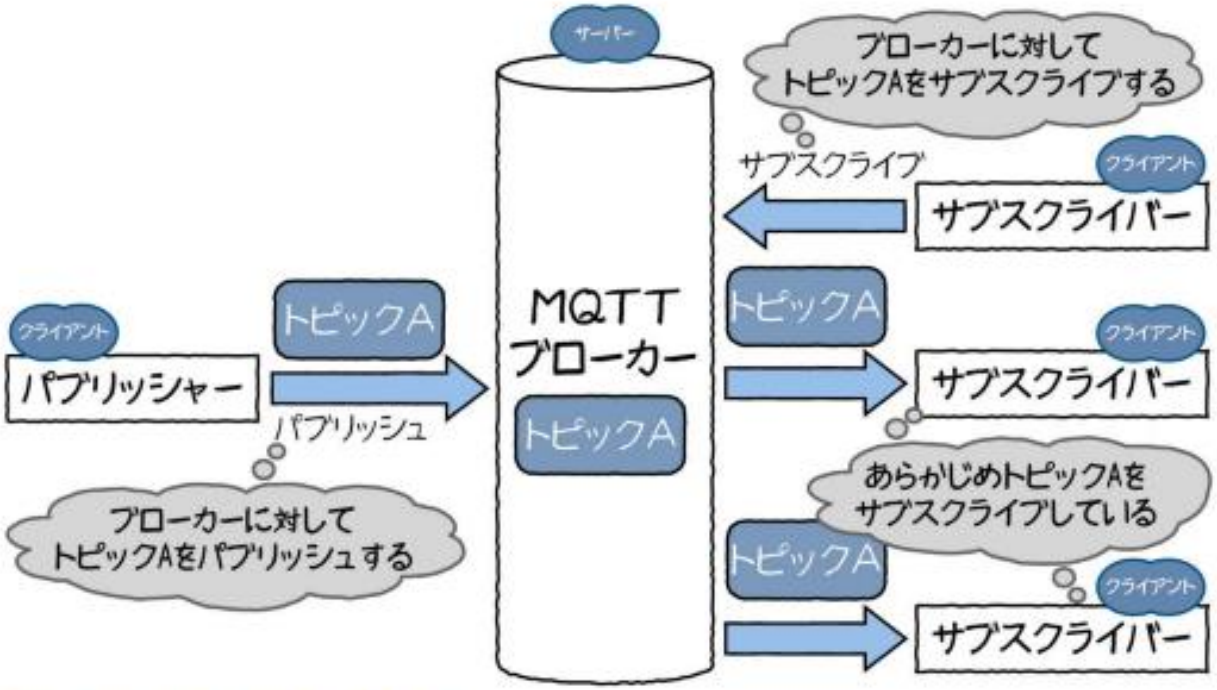

図2.8　MQTTによるメッセージ送受信

ブローカーはMQTTの通信を中継するサーバの役割を担っています。これに対してパブリッシャーとサブスクライバーには、クライアントの役割があります。パブリッシャーは、メッセージを送信するクライアントです。また、サブスクライバーはメッセージを受信するクライアントになります。MQTTでやり取りされるメッセージにはトピックと呼ばれるアドレスが付いており、各クライアントはこのトピックを宛先としてメッセージの送受信を行ないます。イ

メージとしては郵便の私書箱のようなイメージとなります。

MQTTの通信の仕組みをもう少し細かく見ていきます（図2.9）。まずブローカーは、各クライアントからの接続を待つ状態となっています。サブスクライバーは、ブローカーに対して接続を行ない、自分が購読したいトピック名を伝えます。これをサブスクライブと呼びます。

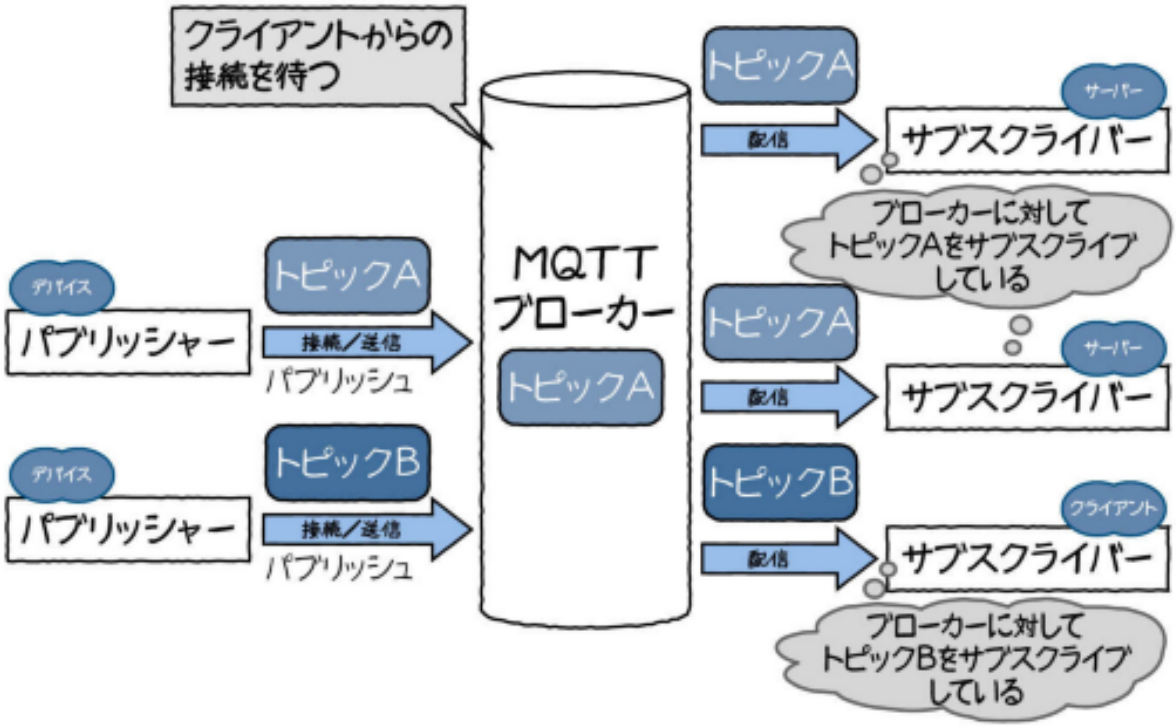

図2.9　MQTTの通信の仕組み

次にパブリッシャーはブローカーに接続を行ない、トピックを宛先にメッセージを送信します。これをパブリッシュと呼びます。

パブリッシャーがトピックをパブリッシュすると、パブリッシュされたトピックを購読しているサブスクライバーに対してブローカーからメッセージが配信されます。図2.9のように、トピックAをサブスクライブしている場合は、トピックAがパブリッシュされた場合のみ、メッセージがブローカーから配信されます。サブスクライバーとブローカーは、コネクションが常に確立状態となっています。また、パブリッシャーは、パブリッシュ時にコネクションを確立すれば良いですが、短期間にパブリッシュを繰り返す場合はコネクションを確立したままにします。ブローカーがメッセージを中継しているため、IPアドレスなどのネットワーク上の宛先をクライアント同士がお互いに知る必要があ

りません。

また、1つのトピックに対して複数のクライアントがサブスクライブすることができるため、パブリッシャーとサブスクライバーは1対多の関係となります。デバイスとサーバ間の通信において、デバイス側がパブリッシャーで、サーバ側がサブスクライバーとなります。

トピックは階層構造を取っています。「#」と「+」という記号を使い、複数のトピックを指定できます。たとえば、図2.10のように、「#」記号を使い「/Sensor/temperature/#」とした場合、先頭が「/Sensor/temperature/」に該当するトピックすべてを指定できます。また、記号「+」を使い、「/Sensor/+/room1」とした場合、先頭が「/Sensor/」でトピックのお尻が「/room1」に該当するトピックを指定できます。

図2.10　MQTTのトピックの例

このように、ブローカーを介したパブリッシュ／サブスクライブ型の通信により、MQTTはIoTサービスが複数のデバイスと通信することをサポートできます。また、MQTTは軽量なプロトコルを実現しています。そのため、ネットワークの帯域が狭く、信頼性が低い環境でも動作できます。また、メッセージ

サイズやプロトコルの仕組みが簡単なため、デバイスなどのCPUやメモリなどのハードウェアリソースが限られている環境でも動作可能です。まさにIoTにうってつけなプロトコルといえます。この他にもMQTTには特徴的な仕組みがあります。次にそれぞれについて説明していきます。

◉QoS

QoSは、Quality of Serviceの略です。ネットワーク分野では通信回線の品質保証を表わす言葉です。MQTTにおけるQoSは3つのレベルが存在します。「パブリッシャーとブローカー間」と「ブローカーとサブスクライバー間」それぞれでQoSのレベルを定義し、非同期で動作します。また、「パブリッシャーとブローカー間」でやり取りされているQoSよりも「ブローカーとサブスクライバー間」が小さいQoSを指定した場合、「ブローカーとサブスクライバー間」のQoSは指定されたQoSへダウングレードされます。「QoS 0」は最高1回（At most once）の送信です（図2.11）。「QoS 0」は、TCP/IP通信のベストエフォートに従って送信されます。メッセージは、ブローカーに対して1回到着するか、到着しない場合のどちらかとなります。

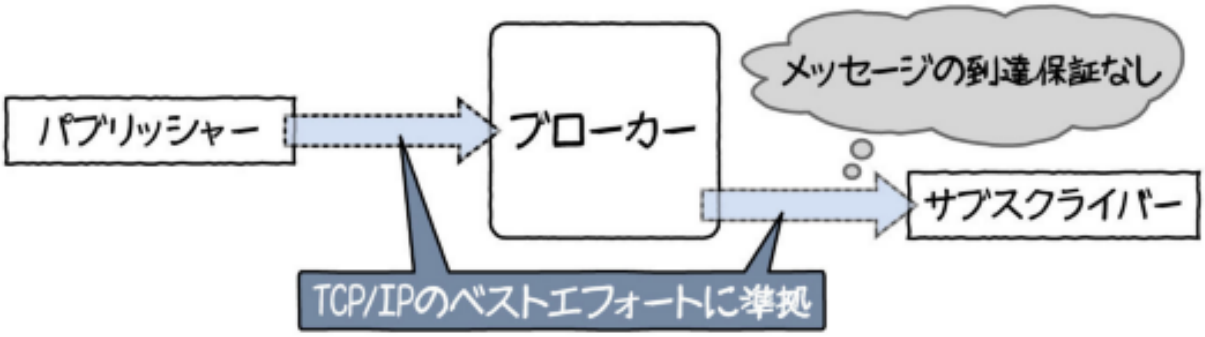

図2.11　QoS 0（At most once）

次に、「QoS 1」は最低1回（At least once）の送信となります（図2.12）。

ブローカーは、メッセージを受信するとパブリッシャーに対してPUBACKメッセージという応答を送信します。また、サブスクライバーから指定されたQoSに従いメッセージを送信します。パブリッシャーは障害の発生や、一定時間の経過後にPUBACKメッセージが確認できない場合、メッセージの再送を行なうことができます。もし、ブローカーがパブリッシャーからのメッセージを受け取った状態でPUBACKを返していなければ、ブローカーは二重にメッセー

ジを受信します。

最後に、「QoS 2」は正確に1回（Exactly once）の送信になります。「QoS 1」の送信に加えて、重複したメッセージが送信されないようにします（図2.13）。「QoS 2」で送信されるメッセージにはメッセージIDが含まれています。メッセージの受信後、ブローカーはメッセージを保存します。その後、PUBRECメッセージをパブリッシャーに送信します。パブリッシャーはPUBRELメッセージをブローカーに送信します。その後、ブローカーはパブリッシャーにPUBCOMPメッセージを送信します。また、ブローカーは受信したメッセージをサブスクライバーから指定されたQoSに従い配信します。

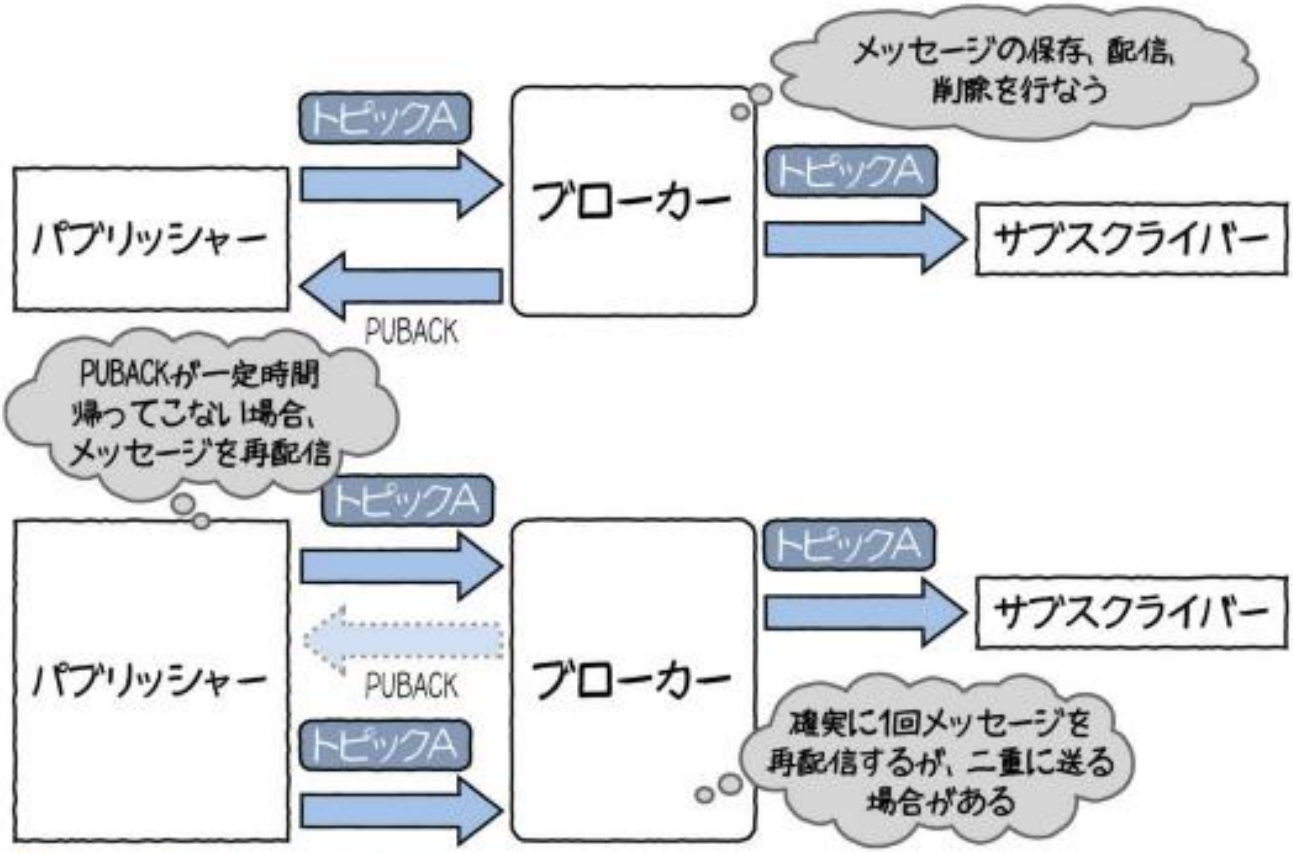

図2.12　QoS 1（At least once）

2.3.5 データフォーマット

ここまではデータを受信するための通信手順であるプロトコルを中心に解説してきました。実際にはこのプロトコルにデータを乗せてやり取りを行ないます。もちろん、これまで説明したように、IoTの世界でもそれは変わりません。このプロトコルに乗せるデータのフォーマットもまた重要になります。Webのプロトコルに乗せて利用するデータフォーマットの代表としてXMLとJSONがあります（図2.17）。

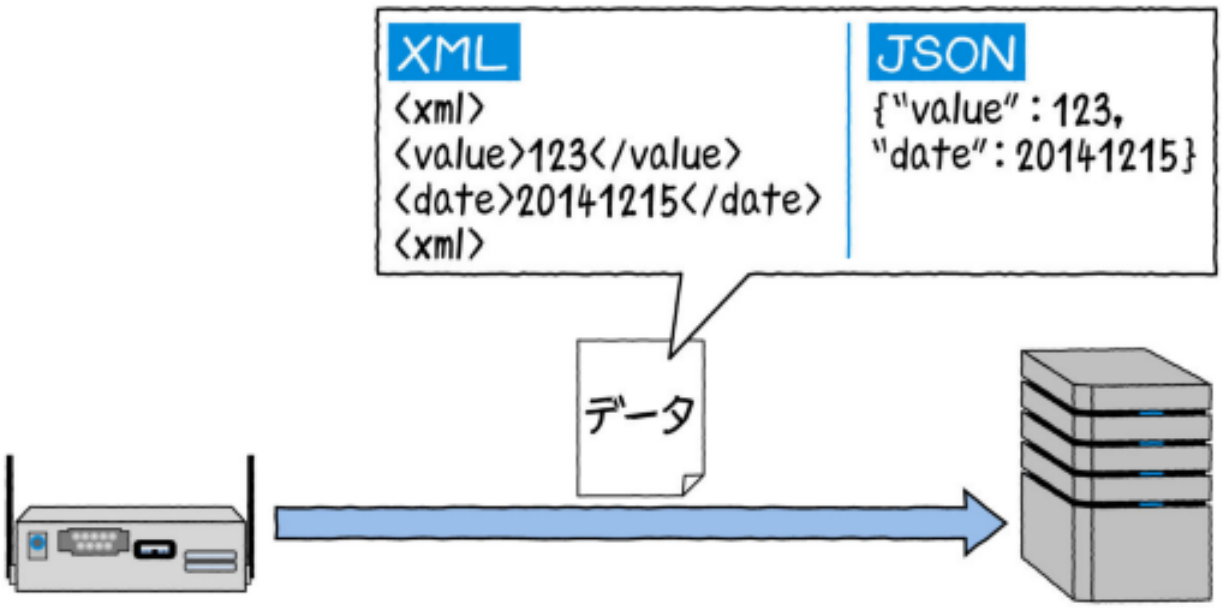

図2.17　代表的なフォーマット

IoTといった観点でみた場合もXMLとJSONを便利に利用することができますたとえば、デバイスからセンサの値が送られてくることを想定します。このとき、デバイスは単にセンサの値を送信するだけではありません。データを受信した時刻や、デバイスの機器の情報、ユーザの情報などを合わせて送信することになります。また、複数のセンサの値や機器の状態を通知することになるでしょう。こうなってくると、デバイスから送られてくるデータをうまく構造化してあげる必要があります。

図2.18は、2つのセンサ情報とデバイスのステータス、データを取得した時間、送信デバイス名などをXMLとJSONそれぞれで表現した例です。

XML

```
<xml>
 <info>
   <id>123</id>
   <name>RoomSensor</name>
   <date>20141121512322</date>
 </info>
 <data>
   <temperature>23.4</temperature>
   <humid>63</humid>
 </data>
<xml>
```

JSON

```
{"info":{"id":123,
         "name":"RoomSensor",
         "date":20141121512322
         },
"data"{"temperature":23.4,
       "humid":64}}
```

図2.18　センサ情報の表現例（XMLとJSON）

両者を比べてみると、XMLはJSONに比べて人間が読みやすい形式となっています。しかし、XMLは文字数が多く、データ量としては多くなってきます。これに対して、JSONはXMLよりも文字数が少なくデータ量が小さいです。

XMLとJSONは、どちらもプログラムから簡単に利用できるライブラリが各言語で実装されています。どちらのデータフォーマットを利用するのが良いか、一概には言えませんが、モバイル回線など低速な回線で通信をする場合はデータ量の少ないJSONのほうが適しているでしょう。

デバイスから送られてくるデータはWebとは違い、センサや画像、音声といった数値データが多くなります。このようなデータは、テキストよりもバイナリとして取り扱うほうが好ましいです。しかし、これまで紹介したXMLやJSONは、データをテキスト形式として取り扱います。

これらのフォーマットをIoTサービス上で取り扱う場合は、テキストデータから数値データやバイナリデータへと変換します。そのため、XML、JSONフォーマットの分解（パース）と分解された値をテキスト形式からバイナリ形式へと変換する作業が行なわれます。そのため、2段階の処理を行なうこととなります。

もし、受信するデータをバイナリ形式のまま受信できれば、より素早くデータを処理できるでしょう。そこで、考え出されたのがMessagePackと呼ばれるデータフォーマットです（図2.19）。

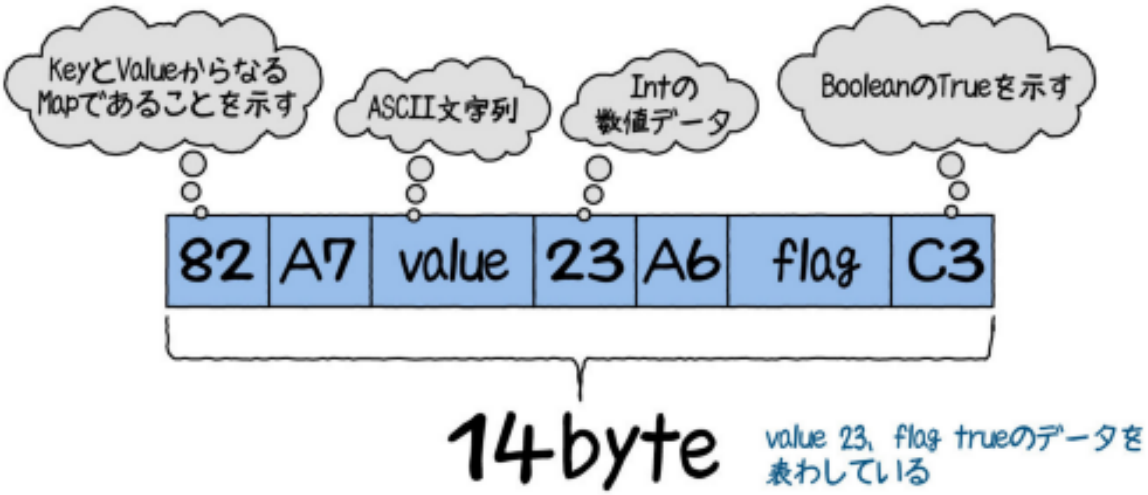

図2.19　MessagePackを利用したセンサデータの表現例とJSONとの比較

MessagePackは、JSONに似たデータフォーマットを持ちながら、そのデータはバイナリ形式のままとなります。そのため、人間が直接読むには適していませんが、コンピュータで処理を行ないやすいデータ形式となっています。

また、バイナリのままデータを送信するため、テキスト形式で送信するJSONに比べてよりコンパクトなデータサイズとなっています。MessagePackもXMLやJSONと同様、多くのプログラミング言語向けにライブラリが提供されています。また、近年では複数のOSS（オープンソースソフトウェア）で採用されている実績があります。

一概にどのフォーマットが良いとはいえませんが、データフォーマットは送信するデータの特性に合わせて、目的に応じたものを選択してください。

COLUMN

画像や音声、動画のデータの取り扱い

「センサデータ、テキストデータ」と「画像や音声、動画」のデータフォーマットには、大きな違いがあります。画像や音声、動画は、データ1つのサイズがセンサデータに比べて圧倒的に巨大なサイズになります。また、データ形式が文字列に変換しにくいバイナリデータとなるため、ここまで紹介したXMLやJSON形式で扱いにくくなります。

HTTPで画像データを送信する場合は、撮影時刻やデバイスの情報は

XMLやJSON形式で記述して、画像データは「multi-partフォームデータ」形式で送信できます。しかし、音声や動画の場合は、時系列に連続したデータとなります。そのため、データを送信する場合は、さらに工夫が必要です。

　たとえば、音声、動画は、細かく分割したファイルで送信する必要があります。HTTPプロトコルでこれを行なう場合は、細切れのデータが送られてくるたびにセッションが作成されます。そのため、WebSocketなどを有効に活用することで、IoTサービスへの負荷を減らすことができるでしょう。その場合は、MessagePackを利用したり、バイナリデータを扱う独自のフォーマットを定義したりする必要があるかもしれません。あるいは、IoTサービスで音声やデータ分析を行なうことを前提とし、データをすべて送信するのではなく、分析に利用する特徴のみをデバイス側で抽出して送信する、という工夫もできるかもしれません。音声や動画を使ったサービスを考えている場合は、本コラムで挙げた観点にも気をつけてください。