MQTTとAMQPと.NET

terurou

2015/02/07

自己紹介

自己紹介/所属等

- デンキヤギ株式会社
  - 代表取締役
  - 社員, フリーランス, 協業等 募集してます
- DSTokai管理人
  - 東海地方のメタコミュニティ
  - IT勉強会カレンダーっぽいやつの東海地方版
- NGK(名古屋合同懇親会)
  - エンジニア向けクロスコミュニティ忘年会
  - ここ数年は100人前後の参加者

自己紹介/技術領域

- フロントエンドアーキテクト(自称)
  - JavaScript(Haxe), WPFあたりがわりと得意
  - 周辺にGUIできる人が居なかった結果...
  - 普通のアーキテクト歴も5年ぐらいはある
- 大量データ×フロントエンド
  - 100万件を操作する高速なExcelみたいなヤツとか...
  - フロントエンドに限らず、バックエンドも最適化

はじめに

今日のネタについて

- 最近、お仕事でMQTTとAMQPについて調べる機会があったので、まとめてきました
- .NETの勉強会なので、.NETにも少し絡めますが、大半はMQTTとAMQPの話です

予備知識

MQTT, AMQPの雑な説明

- 通信プロトコルの一種
  - HTTP, FTP等と同じアプリケーション層のプロトコル
- メッセージ転送 (MQ)
- IoT/M2Mでの利用も想定されている
  - デバイス - サーバ間で小さなデータを送受信
  - 世間的にはMQTTの方がIoT向けっぽいけど、Microsoft的にはAMQP (後述)

MQ : Message Queue

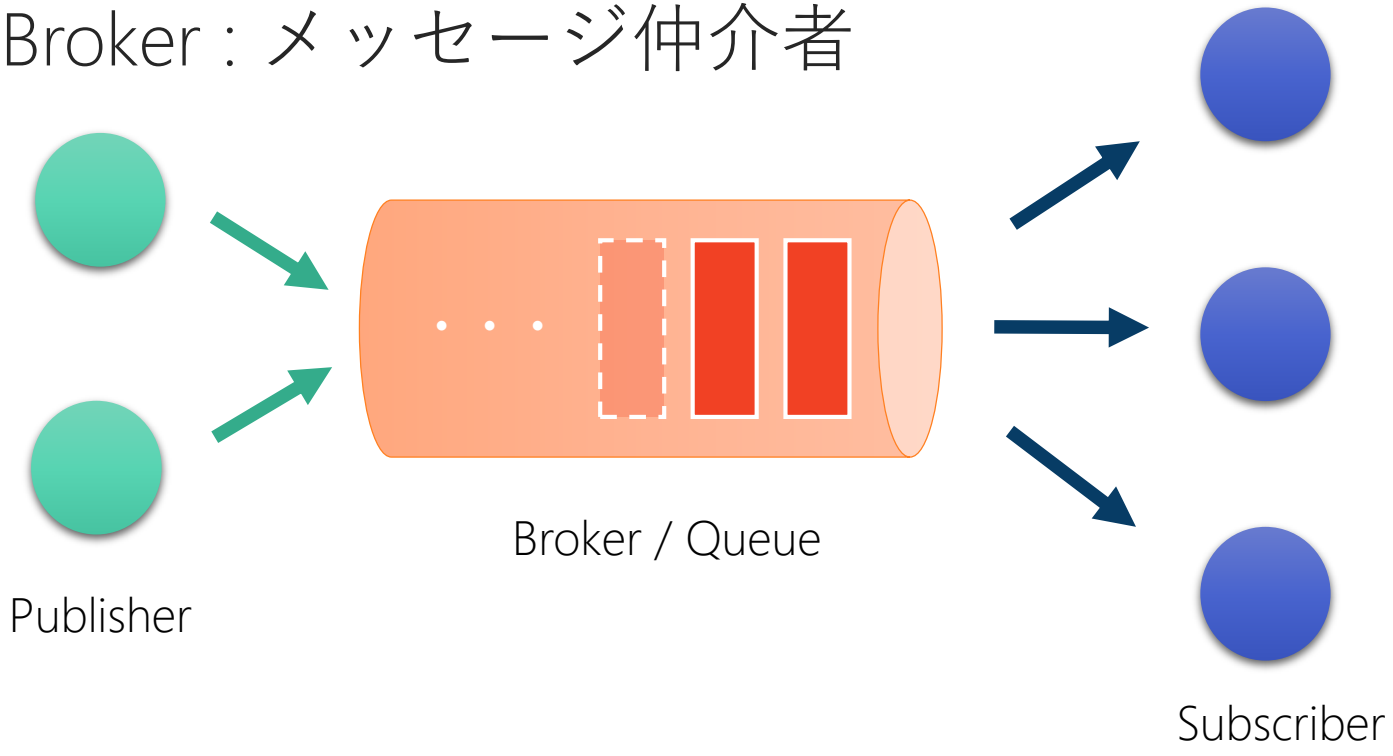
- メッセージ指向
  - Message Passing + Queue
    - 複数オブジェクト間でのメッセージ受け渡し
    - 受け取ったメッセージをQueueに格納
- 非同期・疎結合
  - 送信側はメッセージ送信までを考えればよい
  - 受信側はQueueの消費だけを考えればよい
  - 分散システム・非同期処理がシンプルに書ける

MQの移り変わり

- 10-15年ぐらい前
  - MSMQ
  - JMS
- わりと最近
  - AMQP
  - MQTT
  - STOMP
  - ZeroMQ
  - etc…

Publish/Subscribeモデル

- Publisher : メッセージ発信者
- Subscriber : メッセージ購読者
- Broker : メッセージ仲介者



Publish/Subscribeモデルの補足

- Brokerは存在する・しないパターンがある
  - Brokerモデル vs Brokerlessモデル
  - このスライドではBrokerモデルを前提に解説
- ReactiveExtensionsもPublish/Subscribeモデル

MQ利用例

- メッセージ配信基盤
  - 1:1, 1:N, 配信順序, エラー再送, ...全部MQ任せ
  - 応用で資産や設定情報の配布とか
- Job Queue
  - 時間のかかる処理をWorkerに処理依頼
  - 一時的な処理殺到時の負荷分散、キューイング
  - ログ収集サービス、SlideShareライクサービスなど
    - メッセージはとりあえず待ち行列に貯める
    - 待ち行列が増えすぎたら一時的にWorkerを増やす

MQTT

MQTT : MQ Telemetry Transport

- 軽量・省電力なプロトコル
- 1対Nに特化したメッセージ配信
- 低速・不安定なネットワークでの利用を想定
- 2010年に仕様公開・ロイヤリティフリー
  - プロトコル自体は1999年から存在し、枯れている
- IBMが主導的な立場
  - 仕様策定、商用製品の提供

軽量・省電力なプロトコル

- 軽量プロトコル
  - ヘッダサイズが最小で2byte
  - シンプルなプロトコルシーケンス
- HTTPとの比較 (IBMの資料による)
  - トラフィック10-100分の1 → スループット10-100倍
  - 消費電力 10分の1以下

1対Nに特化したメッセージ配信

- Publish/Subscribeモデル
- TopicベースでPublish/Subscribe
  - “/”区切りの階層構造（ファイルパスっぽい）
  - Subscribe時にワイルドカードで指定可
    - /japan/tokyo/device1/cpu
    - /japan/tokyo/+/meory ←部分一致
    - /japan/tokyo/device1/# ←以下全て

低速・不安定なネットワークでの利用を想定

- QoS : Quality of Service
- Durable subscribe
- LWT : Last Will &Testament，遺言状
  - 単にWillと表記するケースも
- Retain : Retained Publication

QoS : メッセージ配信の品質レベル

- QoS 0 : 最高1回
  - メッセージが確実に届く保証はない
  - 配信失敗しても再送を行わない
- QoS 1 : 最低1回
  - 必ずメッセージは届くが、重複する可能性がある
- QoS 2 : 必ず1回
  - 必ずメッセージが届き、重複もしない
- 当然、QoS 0の方が処理は軽い

Durable subscribe

- メッセージ再配信機能
  - 意図せずにSubscriberの通信が断絶
  - その後、当該Subscriberが再接続
  - 切断～再接続までに発生したメッセージを再送
  - QoS 1, 2のメッセージを再配信
- 明示的なdisconnect, unsubscribeを行った場合は再配信されない

LWT

- ClientはBroker接続時にLWTを設定可能
- Clientが通信を切断時にメッセージを配信する
  - 通信死活監視を行っていて、意図しない切断時にもちゃんとメッセージが配信される
- LWTにもQoSを設定可能

Retain

- ClientがSubscribeする際に、最後に配信されたメッセージを取得できる
- Brokerは最後のメッセージのみは必ず保存する
- 逆説的に、エラー再接続ではないケースでは古いメッセージを取得することはできない

主なMQTT Broker

- IBM MessageSight (商用,ハードウェア)
- IBM WebSphere MQ Telemetry (商用)
- 時雨堂 Akane (商用)
- Mosquitto (OSS)
- RabbitMQ (OSS, Plugin使用)
- etc ...

MQTT over WebSocket

- WebSocket経由でMQTT接続するライブラリも存在する
  - Paho JavaScript Client
- MQTT Brokerの前にWebSocket Gatewayを配置
  - Lighttpdでブリッジできることは確認した
  - たぶんNginx等のロードバランサーでもOKなはず

MQTTの弱いところ

- シンプルな仕様のため、できないことは多い
  - Job Queueは作れない
    - Subscriberが接続してくる前にPublisherが送信したメッセージは最後の一つしか残らない(Retain)
  - 1対1 メッセージ配信が考慮されていない
  - 認証の仕様が明確には定められていない
    - Brokerごとの実装に依存
    - 認証機能をサポートしていないBrokerも多い
- OSSのBroker実装の層が薄い

MQTTと.NET

- MQTT .NET Client
  - Paho(M2Mqtt), WinRTにも対応
  - MQTTDotNet
  - nmqtt
- MQTT .NET Broker
  - GnatMQ
    - 完成度がまだ低いので、素直にMosquittoかRabbitMQか商用Brokerを使った方が良い

M2Mqtt, GnatMQの実装が微妙っぽい話

- ソケットが同期通信で実装されている
  - パフォーマンス上の問題を抱えていそう…
- WinRT版で無理やり同期処理…
  - Task<T>.Result で、非同期処理を同期待ち合わせ…
  - おそらく後付けで #ifdef したせい

AMQP

AMQP : Advanced Message Queuing Protocol

- Enterpriseに対応できるプロトコル
  - 高信頼性
  - 高度なメッセージ配信
  - アクセス制御
- 相互運用性が高い
- 仕様がオープン(OASIS)
- ワーキンググループのメンバーが豪華

高信頼性

- メッセージ永続化
  - Publisherが送信したメッセージをBrokerが永続化
  - 仮にBrokerが死んでも、再起動で復元できる
- トランザクション
  - Subscriber側のメッセージを受け取る一連の処理をトランザクションを使って記述できる
    - tx.select, tx.commit, tx.rollback
  - メッセージは受け取れたが、後続処理でエラー発生というケースでも、メッセージがロストしない

高度なメッセージ配信

- Exchange (Direct, Fanout, Topic, Header)
- Priority Queue
  - 優先度の高いメッセージを他より先に配信
- QoS, Prefetching
  - メッセージを受け取れるSubscriber数を制限
  - 分散JobQueueを作るときに必須
- immediate
  - Subscriberが存在していないときに配信できない

Exchange

- Direct
  - 特定のSubscriberのみに配信, 1:1
- Fanout
  - 全てのSubscriberに配信, 1:N (1:All)
- Topic
  - 特定のTopicのSubscriberのみに配信, 1:N
- Header
  - Header(KeyValue)の組み合わせで配信先を特定, 1:N

アクセス制御

- “virtual host” + ACL
  - ACL : userごとのRead/Write権限
  - virtual hostごとにACLを設定
    - いわゆるマルチテナント的な分割もできる

相互運用性

- 実装面の仕様が明確化されている
  - メッセージフォーマット仕様が厳密
    - 仕様書にTypeSystemの章とかある
  - セキュリティ・アクセス制御等も明確に仕様化
  - 複数プログラミング言語、ベンダー間の相互運用に問題が生じないようになっている
- ライブラリAPIについては規定なし
  - JMSでは逆にAPIを定義するが、実装は未定義

ワーキンググループのメンバーが豪華

- テクノロジ系
  - Cisco, Red Hat, VMware, Microsoft, ...
- 金融(証券)系
  - Bank of America, JP Morgan, Goldman Sachs, Barclays, CREADIT SUISSE, ...
- 金融系が沢山いる＝金融系で使われている
  - トレーディングシステムでのメッセージング基盤
  - 以前はJMSだったらしい

主なAMQP Broker

- RabbitMQ
  - AMQPといえばRabbitMQ
- Apache ActiveMQ, Apollo
- Apache Qpid (, Red Hat Enteprise MSG)
- IBM MQ Light
- Microsoft Azure Service Bus & Event Hubs
  - Service Bus for Windows Server

AMQPの弱いところ

- 高機能な反面、仕様が複雑
- “高機能・高信頼性”と”性能”がトレードオフ
  - MQTT等のシンプル系プロトコルとの比較
    - 要件によっては、NATSやKafka等も視野にいれる
- トレードオフとはいえ…
  - 金融系で実績があることは重要
  - RabbitMQは完成度が高く、クラスタ構成が前提
    - MQTT Brokerでは商用とOSSの差が激しい

AMQPと.NET

- AMQP .NET Client
  - Broker製品ごとにライブラリが存在
  - RabbitMQ .NET Client (WCFもいける)
  - Qpid WCF
  - Azure SDK, ...
- AMQP .NET Broker
  - 少なくともOSSは無いっぽい
  - 素直にRabbitMQかService Busを使いましょう

AMQPとAzure Event Hubs

- Azure Event HubsはIoT用サービスらしい
  - AMQPとHTTPプロトコルをサポートしている
- IoTデバイスのバッテリーに優しくない気が...
  - AMQPプロトコルは重い (MQTTとの比較)
  - なんでMQTTサポートしてないんだろ...

まとめ

まとめ

- MQTTとAMQPはMQのプロトコル
  - それぞれの特徴は全然違う
  - MQTT : シンプル・省電力
  - AMQP : 高機能・高信頼性
- .NETからはどちらのプロトコルも使える
  - Microsoft的にはAMQP推し
  - MQTTの.NET Clientの品質は検証が必要
    - MosquittoのBindingを作るのもありかも…