ZeroMQ

1. Čo je ZeroMQ?

ZeroMQ (tiež známy ako ØMQ, 0MQ, ZMQ) je vysoko výkonná knižnica asynchrónnych správ zameraná na použitie v distribuovaných alebo súbežných aplikáciách. Poskytuje front správ, ale na rozdiel od middlewaru orientovaného na správy môže systém ZeroMQ bežať bez vyhradeného sprostredkovateľa správ.

ZeroMQ podporuje bežné vzory správ (publisher/subscriber, request/reply, client/server a ďalšie) cez rôzne prenosy (TCP, medziprocesové, multicastové, WebSocket a ďalšie), vďaka čomu je zasielanie správ medzi procesmi jednoduché. Stručne povedané, ZMQ vám umožňuje posielať správy (binárne údaje, serializované údaje, jednoduché reťazce atď.) cez sieť rôznymi metódami, ako je TCP alebo multicast, ako aj medzi procesmi. ZeroMQ poskytuje celý rad jazykových rozhraní API, ktoré bežia na väčšine operačných systémov a umožňujú bezproblémovú komunikáciu medzi všetkými druhmi programov. Poskytuje tiež zbierku vzorov, ako napríklad

request-response a publish-subscribe, ktoré vám pomôžu pri vytváraní a štruktúrovaní vašej siete.

Filozofia ZeroMQ začína ***nulou***. Nula je pre nulového brokera (ZeroMQ je bez brokera), nulovú latenciu, nulové náklady (je zadarmo) a nulovú správu. Všeobecnejšie povedané, ***nula*** sa vzťahuje na kultúru minimalizmu, ktorá preniká do projektu. Výkon pridávame skôr odstránením zložitosti, než odhalením nových funkcií.

2. Ako funguje ZeroMQ

ZeroMQ je asynchrónna knižnica sieťových správ známa svojim vysokým výkonom. Poskytuje sockety, ktoré prenášajú atómové správy cez rôzne prenosy, ako sú medziprocesové, TCP a multicast. Zásuvky N-to-N môžete prepojiť so vzormi ako fan-out, pub-sub, request-response a pod. Je velmi rýchly a jeho asynchrónny I/O model vám poskytuje škálovateľné viacjadrové aplikácie, vytvorené ako úlohy asynchrónneho spracovania správ. Má množstvo jazykových rozhraní API a beží na väčšine operačných systémov. Poskytuje vám bohaté vzory pre interakciu s programami a ponúka škálovateľnosť. Veľmi nízka latencia, nízka réžia, má filozofiu riadiť sa udalosťami a neblokovať

Sockets

ZMQ obsahuje niekoľko rôznych socketov, z ktorých každý má svoje vlastné vlastnosti a prípady použitia. Sockety možno kombinovať mnohými rôznymi spôsobmi, aj keď existuje veľa kombinácií socketov, ktoré sú jednoducho nekompatibilné. Pochopenie základov socketov je kľúčom k návrhu ZeroMQ systému.

Vysvetlime si niektoré z väčšiny typov zásuviek a niektoré z ich typických kombinácií, aj keď existuje veľa rôznych kombinácií zásuviek.

ZeroMQ prichádza s 5 základnými vzormi : synchrónny request/response, asynchrónny request/response, publish/subscribe, push/ pull a exkluzívny pár.

Sockets REQ a REP

Socket REQ je skratka pre „request“. Toto je jedna z najzákladnejších zásuviek, ktorá sa zvyčajne spája so zásuvkou REP alebo „reply“. Zásuvku REQ, ako aj zásuvku REP je však možné kombinovať s mnohými inými typmi zásuviek.

Kombinácia zásuvky REQ/REP funguje takto:

Zásuvky REQ a REP sa inicializujú v samostatných programoch. Funkciu „connect()“ voláme na zásuvke REQ a „bind()“ na zásuvke REP.

Zásuvka REP je inicializovaná a zablokovaná a čaká na doručenie správy.

Zásuvka REQ posiela cez sieť nejakú svojvoľnú „požiadavku“.

REP prijme túto správu a urobí s ňou všetko, čo potrebuje, a soket REP môže poslať odpoveď/potvrdenie späť do soketu REQ.

Majte na pamäti, že zásuvky REQ/REP sú synchrónne zásuvky. To znamená, že sa môžu naraz rozprávať len s jedným rovesníkom. Z tohto dôvodu je prípad použitia kombinácie zásuviek REP/REQ veľmi úzky a zvyčajne ju nenájdete v aplikácii v reálnom svete. REQ/REP by ste mali skutočne používať iba ako vzdelávací nástroj, ktorý vám pomôže pochopiť základy ZeroMQ

Socket ROUTER a DEALER

Zásuvky DEALER-ROUTER možno považovať za neblokujúcu asynchrónnu verziu zásuviek REQ-REP. Zásuvka DEALER funguje ako zásuvka REQ a ROUTER funguje ako zásuvka REP. Niečo, čo treba mať na pamäti pri vlastnom skúmaní ZMQ, si všimnete v starších blogoch/príspevkoch výraz XREQ a XREP. Toto sú staré názvy pre zásuvky DEALER a ROUTER.

Odosielanie a prijímanie zo zásuvky ROUTER sa trochu líši od iných zásuviek. Tieto rozdiely umožňujú, aby bol asynchrónny. Najdôležitejšia vec, ktorú treba poznamenať, je, že zásuvka ROUTER očakáva, že všetky prichádzajúce správy budú mať pred samotnou správou priradený rámec identity. To umožňuje ROUTERU vedieť, odkiaľ správa prišla. Pri odosielaní správy na ROUTER sa prvý rámec správy odstráni a použije sa na identifikáciu klienta, ktorému sa má poslať odpoveď.

Zásuvky DEALER sú určené na „rozdávanie“ správ pripojeným pracovníkom. Pri odosielaní správy cez zásuvku DEALER sa správy odosielajú spôsobom „round-robin“.Na získanie hlbšieho pochopenia týchto zásuviek, ako aj iných zásuviek zahŕňajúcich štruktúru žiadosť/odpoveď, vrelo odporúčam sekciu Mechanizmy žiadosti a odpovede v príručke ZeroMQ.

Zásuvky PUB a SUB

PUB/SUB znamená, ako ste možno uhádli, „publish“ a „subscribe“. Spôsob, akým táto kombinácia zásuviek funguje, môže byť pre niektorých jasne zrejmý, ale rád by som to stručne prebehol v prípade, že nepoznáte model publikovania/prihlásenia na odber.Základnou myšlienkou modelu publish/subscribe je, že PUB soket vytlačí správy a všetky pridružené SUB sokety tieto správy dostanú. Táto komunikácia je striktne jednosmerná, SUB zásuvky neposielajú žiadne odpovede ani potvrdenia.

PÁROVÉ zásuvky

Zásuvky PAIR sú typu zásuviek, ktoré je možné používať iba medzi sebou. PAIR zásuvky umožňujú obojsmernú komunikáciu a môžu byť pripojené iba k jednému peeru naraz.Jedna veľmi dôležitá vec, ktorú treba poznamenať, je, že PAIR zásuvky nemožno použiť v sieti. Môžu byť použité iba na zasielanie správ medzi procesnými vláknami.

PUSH/PULL zásuvky

Zásuvky PUSH/PULL predstavujú jednosmernú kombináciu zásuviek, ktorá vám umožňuje distribuovať správy mnohým pracovníkom. PUSH socket bude distribuovať odoslané správy svojim PULL klientom rovnomerne.Tento vzor je užitočný v prípadoch, keď robíte veľké množstvo malých úloh.

3. Ako pracovať s ZeroMQ

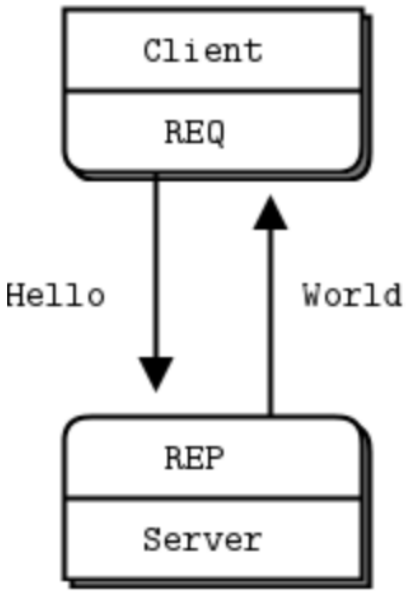
Ako sme uz povedali ZeroMQ prichádza s 5 základnými vzormi, z ktorých každý je dodávaný s vlastným párom zásuviek (niektoré sa dajú kombinovať, existujú však aj nekompatibilné páry). V tejto kapitole si ukážeme architektúru, ukážeme príklad kódu a vysvetlíme ho. Nebudeme pokrývať všetky možné kombinácie socketov, ale len tie najbežnejšie.

Kontexty

Pred začatím akejkoľvek ZeroMQ expedície si musíte vytvoriť kontext. Kontexty pomáhajú spravovať všetky vytvorené zásuvky, ako aj počet vlákien, ktoré ZeroMQ používa v zákulisí. Vytvorte si ho pri inicializácii procesu a zničite ho po ukončení procesu. Kontexty môžu byť zdieľané medzi vláknami a v skutočnosti sú to jediné objekty ZeroMQ, ktoré to dokážu bezpečne.

Synchrónny request/response

Najprv sa pozrieme na synchrónny vzor žiadosti/odpovede, aby sme vytvorili povinnú aplikáciu „Hello World“.



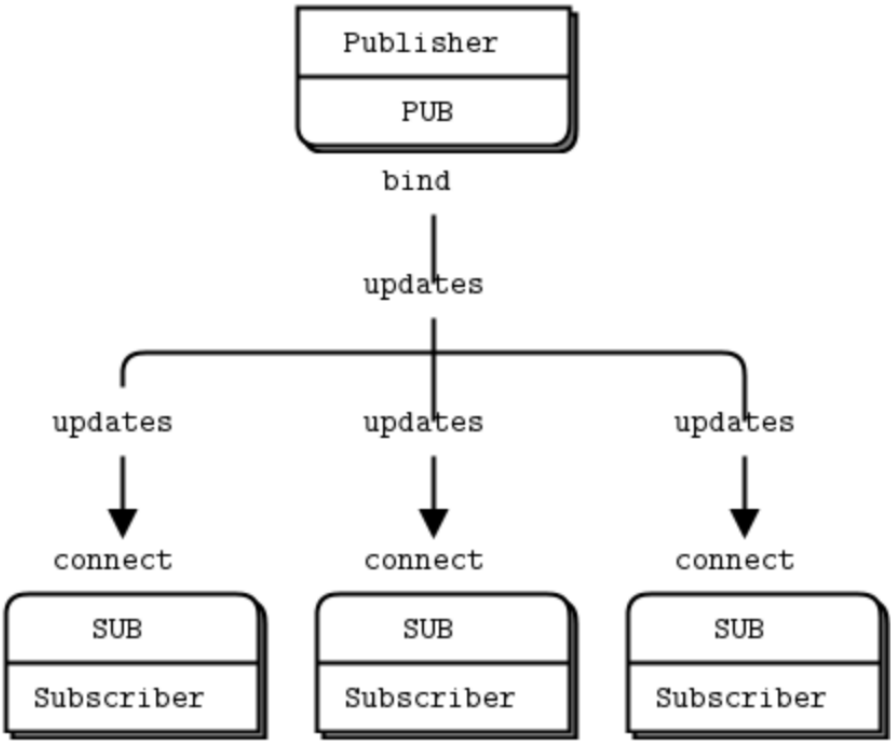

```
using System;

using NetMQ;

namespace HelloWorld {
 class Program {
  static void Main(string[] args) {
   string connection = "inproc://HelloWorld";
   using (NetMQContext ctx = NetMQContext.Create()) {
    using (var server = ctx.CreateResponseSocket()) {
     server.Bind(connection);
     using (var client = ctx.CreateRequestSocket()) {
      client.Connect(connection);
```

```
        client.Send("Hello");

        string fromClientMessage = server.ReceiveString();
        Console.WriteLine("From Client: {0}", fromClientMessage);
        server.Send("Hi Back");

        string fromServerMessage = client.ReceiveString();
        Console.WriteLine("From Server: {0}", fromServerMessage);

        Console.ReadLine();
      }
     }
    }
   }
  }
}
```

Najprv vytvoríme reťazec *connection* a kontext. Potom vytvoríme soket *response* a naviažeme ho pomocou pripájacieho reťazca, po ktorom nasleduje soket *request* a pripojíme ho. Podobne ako bežné TCP sokety, jeden koniec sa musí viazať a druhý sa musí pripojiť. Teoreticky by nemalo záležať na tom, kto čo robí, ale v praxi to tak celkom nie je. Vo všeobecnosti chcete spojiť najstabilnejšie časti vašej topológie a spojiť tie efemérnejšie. Keď sú naše zásuvky nastavené, odošleme požiadavku, prečítame ju a vytlačíme. Potom pošleme odpoveď, prečítame ju a vytlačíme. Pretože sokety request a response sú synchrónne, ich odosielanie a prijímanie sa musí uskutočňovať v špecifickom poradí. Soket request musi odoslať a potom prijať a naopak pre soket response. Ak tieto operácie vyskúšate v nesprávnom poradí alebo zdvojnásobíte jednu z nich, vyvolá sa výnimka.

Publish – Subscribe

Čo keby ste chceli zverejniť tok údajov a umožniť ľubovoľnému počtu klientov, aby ho spotrebovali. ZeroMQ má na to vzor pomocou soketov na publikovanie a odber. V tomto príklade rozdelíme klienta a server na samostatné procesy. To nám umožní predviesť schopnosť ZeroMQ komunikovať medzi jazykmi, takže server napíšeme v jazyku Java a klienta v jazyku C#.



Tu začneme vytvorením nášho kontextu a soketu *publisher* a naviazaním na náš koncový bod. Potom robíme slučku a zakaždým vložíme náhodnú hodnotu do jedného z 100 000 identifikátorov.

```
package org.PubSub;

import java.util.Random;

import org.zeromq.ZMQ;

public class Server {

 public static void main(String[] args) {
  try(ZMQ.Context ctx = ZMQ.context(1);
    ZMQ.Socket publisher = ctx.socket(ZMQ.PUB)) {
    publisher.bind("tcp://*:5556");

    Random random = new Random();
    while (true) {
      int id = random.nextInt(100000);
      int data = random.nextInt(500);
      publisher.send(String.format("%05d %d", id, data));
    }
  }
```

```
  }
}
```

Opäť teda vytvoríme náš kontext, *subscriber* zásuvku a pripojíme sa k nášmu koncovému bodu. Potom vyberieme náhodné ID na odber. Všimneme si, že aj keď sa chceme prihlásiť na odber všetkých udalostí publikovaných serverom, MUSÍME nastaviť odber (na získanie všetkých správ potrebujeme prázdny reťazec), inak klient nedostane nič. Subsciber vykona zhodu reťazca so začiatkom správy. Ak sa nájde zhoda, klient dostane správu. Od *publishera* dostávame sto správ, ktoré spolu predstavujú celkové množstvo údajov, ktoré sme dostali. Po spracovaní správ vytlačíme ID a priemer údajov.

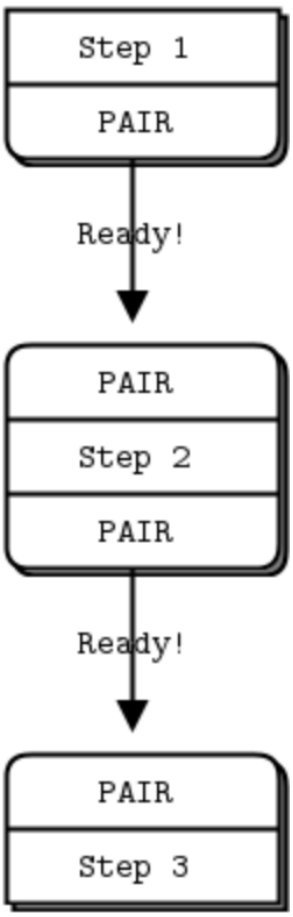
```
using System;

using NetMQ;

namespace PubSub {
 sealed class Program {
  public static void Main(string[] args) {
   using (NetMQContext ctx = NetMQContext.Create()) {
    using (var subscriber = ctx.CreateSubscriberSocket()) {
     subscriber.Connect("tcp://localhost:5556");
     int id = new Random().Next(100000);
     subscriber.Subscribe(id.ToString("D5"));

     long totalData = 0;
     int updateNumber = 0;
     for (; updateNumber < 100; updateNumber++)
     {
       string message = subscriber.ReceiveString();
       totalData += int.Parse(message.Split(' ')[1]);
     }

     Console.WriteLine(string.Format("Average data for id {0} was {1}",
       id, (totalData / updateNumber)));
    }
   }
  }
 }
}
```

Exkluzívny pár

Exkluzívne páry sa používajú na koordináciu viacvláknových aplikácií.

Exkluzívne páry pevne spájajú vlákna vašej aplikácie (podobne ako nemenné kolekcie) a nemôžu sa škálovať na procesy, takže jedinou skutočnou výhodou, ktorú vidím, by bolo, že sa nemusím

učiť o nemenných dátových štruktúrach. Ak však používate jeden z mnohých jazykov, pre ktoré takéto zbierky neexistujú, exkluzívne páry sú nekonečne lepšie ako zdieľanie premenlivého stavu a používanie zámkov, semaforov a/alebo mutexov.



V tomto príklade je vytvorený trojkrokový proces spojený exkluzívnymi párovými zásuvkami cez inproc transport. Kroky sa vytvárajú v opačnom poradí, v akom sa použijú, a čakajú na signál READY z predchádzajúceho kroku.

```
var zmq = require('zmq')

var step3Receiver = zmq.socket('pair');
step3Receiver.on('message', function() {
  console.log('Test successful');
});
step3Receiver.bindSync('inproc://step3');

var step2Receiver = zmq.socket('pair');
step2Receiver.on('message', function() {
  var xmitter = zmq.socket('pair');
  xmitter.connect('inproc://step3');
  console.log('Step 2 ready, signalling step 3');
  xmitter.send('READY');
});
step2Receiver.bindSync('inproc://step2');

var xmitter = zmq.socket('pair');
xmitter.connect('inproc://step2');
console.log('Step 1 ready, signalling step 2');
xmitter.send('READY');
```

4. Výhody

Viaceré jazykové a platformové integračné body, ktoré sú všetky integrované a podporované.V praxi sa ZeroMQ používa na správu soketovej vrstvy.Obsah správy je flexibilný a ľahko sa prispôsobuje modelu fázového merania: identifikácia, časová pečiatka, hodnota a metrika.

Dobrá škálovateľnosť architektúry. Dá sa ľahko rozšíriť z aplikačnej komunikácie na medziaplikačnú komunikáciu a rozšírenú komunikáciu.

Hlavné výhody používania ZeroMQ

Plne distribuované, takže nie je potrebný server, čo znižuje jediný proces.Multithreading možno použiť na čítanie z rôznych uzlov pomocou všetkých jadier CPU. Výkonnejší, pretože žiaden uzol nedokáže vypnúť celý systém.Je ľahký a rýchly.Podporované sú všetky moderné jazyky a platformy.Prenáša správy cez IPC, TCP, TPIC a multicast.

Poháňané veľkou a aktívnou open source komunitou, čiže ZeroMQ je vyvinutý veľkou komunitou prispievateľov. Existujú väzby tretích strán pre mnoho populárnych programovacích jazykov a natívne porty pre C# a Java.

5. Nevýhody

Pre bežných vývojárov a inžinierov je ťažké pochopiť a používať abstrakcie.

Difúzny a neefektívny model.

6. Vzorový kód

Prvým krokom je inštalácia knižnice: `pip install pyzmq`

Potom vytvoríme dva python súbory: klient a server.

Server spája soket REP s tcp://*:5555. Od klienta očakáva b"Ahoj" a odpovedá b"World"

```
import time
import zmq

context = zmq.Context()
socket = context.socket(zmq.REP)
socket.bind("tcp://*:5555")

while True:
    #  Wait for next request from client
    message = socket.recv()
    print("Received request: %s" % message)

    #  Do some 'work'
    time.sleep(1)

    #  Send reply back to client
    socket.send(b"World")
```

Klient pripája zásuvku REQ k tcp://localhost:5555. Pošle "Ahoj" na server, očakáva "World" späť.

```
import zmq

context = zmq.Context()

#  Socket to talk to server
print("Connecting to hello world server…")
socket = context.socket(zmq.REQ)
socket.connect("tcp://localhost:5555")
```

```
#  Do 10 requests, waiting each time for a response

for request in range(10):
    print("Sending request %s …" % request)
    socket.send(b"Hello")

    #  Get the reply.
    message = socket.recv()
    print("Received reply %s [ %s ]" % (request, message))
```

Odkazy:

https://zeromq.org/

https://cloudinfrastructureservices.co.uk/rabbitmq-vs-zeromq-whats-the-difference/

https://intelligentproduct.solutions/technical-software/introduction-to-zeromq/

https://madhusairavada.medium.com/zeromq-457fc762aa9f

https://blog.scottlogic.com/2015/03/20/ZeroMQ-Quick-Intro.html