TEST RUNNER PLUGIN HAND-LEIDING

VOOR DE PERFORMANCE TESTING TOOL

Lex Felix

28 November 2019

INHOUDSOPGAVE

INTRODUCTIE

In deze handleiding wordt beschreven hoe een testrunner plugin geschreven wordt voor de performance testing tool (PTT). Er is een git repository beschikbaar voor het performance testing tool project. Hierin zit ook documentatie voor de werking van de performance testing tool, een handleiding voor het schrijven van een blockchain provider plugin en een module (empty-testrunner) in het performance testing tool maven project die gebruikt wordt als startpunt voor deze handleiding.

DEPENDENCIES

Een blockchain provider plugin **moet** gebruik maken van een aantal libraries:

- PF4J 3.1.0 – Plugin framework
- Akka 2.5.26 (classic) – Concurrency framework
- API module PTT – De klassen die overeen komen
- SLF4J – Logging framework
- Akka Testkit 2.5.26 – Voor het testen van de providers
- Junit 5.5.2 – Voor het testen van de providers

Hiernaast kunnen er naar eigen dependencies toegevoegd worden zolang dat deze dependencies werken met de performance testing tool die gecompileerd wordt voor Java 11. Een aantal geadviseerde dependencies staan in al in de pom van de start module.

Akka is de belangrijkste library om te begrijpen. Interactie me je plugin gebeurt via Akka. Akka is een concurrency framework gebaseerd op het Actor Model. Hierin worden berichten naar de mailbox van Actoren gestuurd. Elke Actor heeft zijn eigen mailbox en handelt de berichten synchroon af. Doordat Actoren parallel kunnen draaien op aparte Threads wordt zo concurrency verkleint doordat alleen nagedacht hoeft te worden aan het uitwisselen van berichten. Meer

over Akka is te lezen in hun documentatie. **Er wordt gebruik gemaakt van Akka classic binnen de performance testing tool.**

BOUWEN

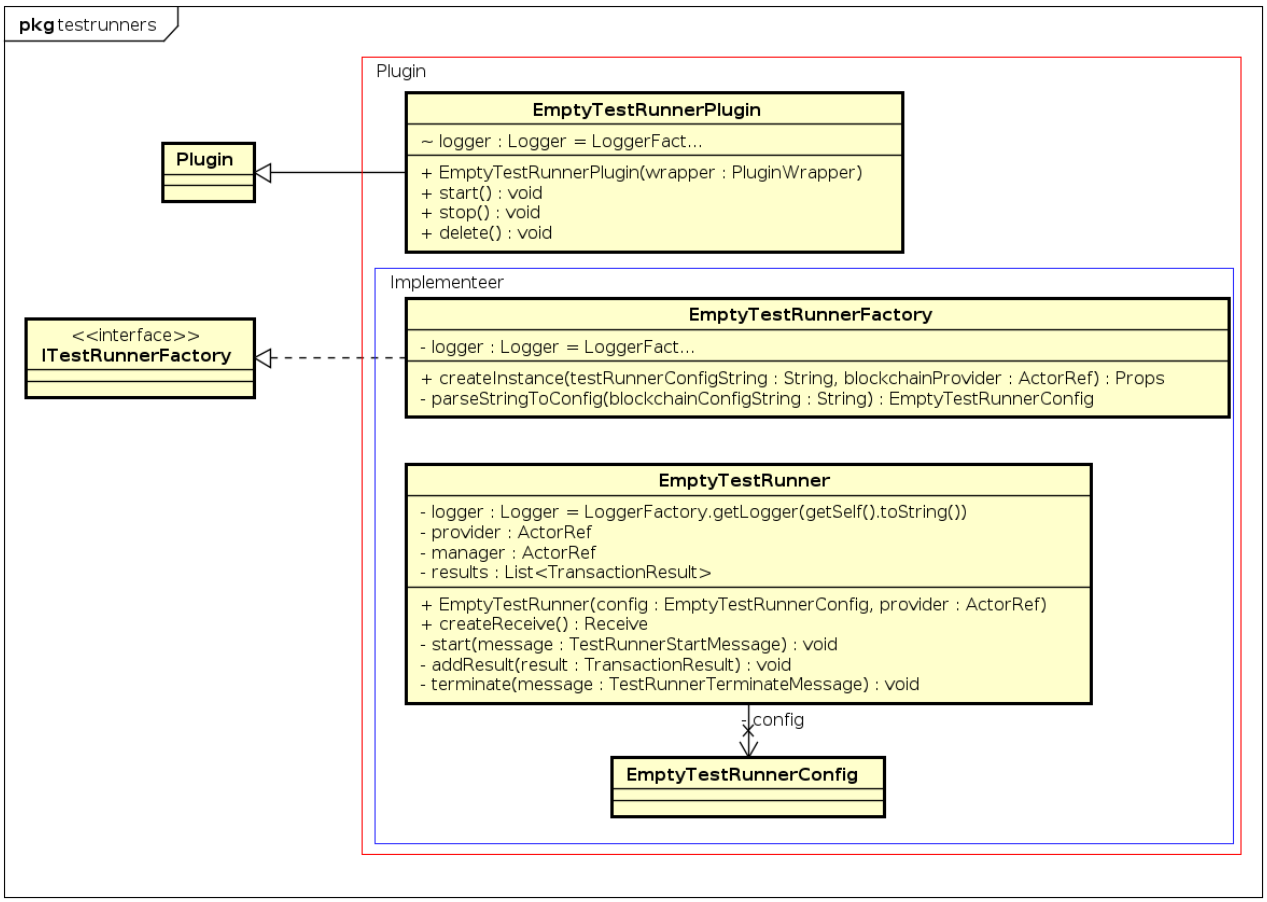
De plugin wordt via maven als een fat jar gebouwd. In de start module hoeft niets aan de build in de pom.xml veranderd te worden. Wel moeten wat properties veranderd worden zodat de plugin gebruikt kan worden in de PTT.

artifactId	ArtifactId van het project
name	Naam van het project
jar.name	Uiteindelijke naam van de plugin jar
plugin.class	Volledige klasse naam van de EmptyTestrunnerPlugin
plugin.id	De unique identifier van de plugin, wordt gebruikt als type in het testscenario bestand
plugin.version	Versie van de plugin (Semantic Versioning Specification)
plugin.description	Beschrijving van de plugin (optioneel)
plugin.provider	Auteur van de plugin (optioneel)
plugin.license	Licentie van de plugin (optioneel)

Nadat de plugin gebouwd is, is het te vinden in de target map zoals normaal. De plugin kan gebruikt worden in de PTT door het in een folder genaamd “plugins” te stoppen die naast de ptt.jar staat en testscenario bestand te schrijven die gebruik maakt van je plugin.

START MODULE

De start module in de git repository ziet er als volgt uit.



Figuur 1: Klassen diagram start module

Alles in de blauwe box moet geïmplementeerd worden en alles in de rode box is minimaal nodig in het project voor een werkende plugin. De EmptyTestRunnerPlugin hoeft dus niets aan veranderd te worden. Dit is namelijk een klasse die het plugin framework PF4J gebruikt voor het inladen van alle extensies. Het implementeren van de EmptyTestRunnerFactory is te vinden in het hoofdstuk Factory en voor de EmptyTestRunner is het hoofdstuk Testrunner. De klasse EmptyTestRunnerConfig is een POJO voor je configuratie variabelen, meer informatie in het hoofdstuk Configuratie. Het is mogelijk om zelf nieuwe klassen toe te voegen die gebruikt worden in deze klassen.

FACTORY

De EmptyTestRunnerFactory klasse is een implementatie van de ITestRunnerFactory. Deze factory wordt in een abstract factory pattern gebruikt binnen de PTT voor het instantiëren van een testrunner. De PTT kan een instantie aanvra-

gen door de createInstance methode aan te roepen van je factory. Deze methode heeft de taak om een instantie van je EmptyTestRunner terug te geven die gebruikt kan worden zoals beschreven in het Hoofdstuk Testrunner. Om dit voor elkaar te krijgen moet gezorgd worden dat er een Props object gemaakt wordt met daarin een valide EmptyTestRunnerConfig object. Een Props object is nodig omdat, binnen de PTT gebruikt wordt gemaakt van Akka, een concurrency framework. Hierdoor worden testrunners actoren in een Akka systeem en verandert het instantiëren. Waar normaal gesproken een nieuw object aangemaakt moet worden moet voor een actor een Props object aangemaakt worden.

Het aanmaken verschilt niet veel van het gebruiken van een normale constructor. Binnen de parameters moet de eerste de klasse zijn en de rest moeten overeen komen met parameters van een constructor in EmptyTestRunner. Hierdoor kan Akka zelf de actoren aanmaken op andere threads wanneer dat nodig is.

In het code voorbeeld is config een valide EmptyTestRunnerConfig object. Dit object is een POJO met alle configuratie variabelen die in de configuratie blok van je testrunner staan (Hoofdstuk Configuratie voor meer informatie). Dit object kan verkregen worden door de blockchainConfigString, die als parameter wordt meegegeven, te parsen naar een EmptyTestRunnerConfig object en deze vervolgens te valideren. Vergelijkbare functionaliteit wordt binnen de PTT gedaan met een aantal jackson libraries deze staan in de start module onder geadviseerde dependencies. Wanneer de configuratie niet valide blijkt te zijn moet er een TestRunnerConfigException worden gethrowed. Deze exception wordt gebruikt om aan de gebruiker door te geven wat er precies fout is gegaan, wees zo expliciet mogelijk in de message.

Ook wordt er een referentie naar een blockchain provider meegegeven. Met deze referentie kan de testrunner transacties laten verzenden door de blockchain provider (meer informatie in het hoofdstuk Testrunner). De relatie tussen testrunner en blockchain provider is één op één.

TESTRUNNER

De EmptyTestRunner wordt in de PTT gebruikt als actor in een AkkaSystem. Hierdoor extend de EmptyTestRunner van de AbstractActor klasse of een klasse die AbstractActor extend zoals AbstractActorWithTimers.

Aangezien Akka gebruikt wordt kan de PTT geen interface gebruiken om specifieke methoden aan te roepen van de EmptyTestRunner. Om deze reden is er ook geen interface voor de EmptyTestRunner beschikbaar gesteld. In de plaats van een interface zijn een set aan bericht klassen opgesteld waar een functionaliteit aan gekoppeld is. Het afhandelen van deze berichten en de bijbehorende functionaliteit uit te voeren is de taak van de EmptyTestRunner.

Bericht klassen moeten gekoppeld worden aan een methode in de createReceive method.

```
@Override
public Receive createReceive() {
    return receiveBuilder()
            .match(TestRunnerStartMessage.class, this::start)
            .match(TransactionResult.class, this::addResult)
            .match(TestRunnerTerminateMessage.class, this::terminate)
            .build();
}
```

Figuur 2: createReceive methode

Een testrunner kan een drietal berichten ontvangen:

- TestRunnerStartMessage
- TransactionResult
- TestRunnerTerminateMessage

TESTRUNNERSTARTMESSAGE

Een TestRunnerStartMessage wordt door de TestRunnerManager gestuurd. Wanneer dit bericht wordt ontvangen moet de testrunner starten. Dit betekend dat er transactie verstuurd mogen worden naar eigen wens. Dit kan gedaan worden door een SendTransactionMessage object te sturen via de referentie die is meegegeven aan de EmptyTestRunnerFactory bij het instantiëren van de testrunner. Door de functionId in het SendTransactionMessage object te veranderen kunnen specifieke transacties worden verstuurd (kijk naar de blockchain provider plugin om te achterhalen welke ondersteund worden).

Tijdens het afhandelen van een TestRunnerStartMessage moet rekening gehouden worden dat andere berichten ook verwerkt moeten worden. Het afhandelen van dit bericht moet de testrunner dus niet blokkeren. Dit kan gedaan worden door een combinatie van berichten naar zichzelf te laten sturen of gebruikt te maken van timers of scheduled messages. Ook moet de referentie naar de TestRunnerManager opgeslagen worden zodat alle resultaten terug gestuurd kunnen worden.

TRANSACTIONRESULT

Wanneer een blockchain provider een SendTransactionMessage message ontvangt wordt een transactie verstuurd. Wanneer een transactie geconfirmeerd of te wel gefaald is heeft de transactie een geldige eind staat bereikt en wordt de TransactionResult terug gestuurd. Dit is een object waarin data staat omtrent de transactie die de PTT gebruikt om metrieken mee te genereren.

Een testrunner moet, wanneer deze ontvangen is, zijn unique identifier in het TransactionResult object opslaan in de runner variabelen. Hierdoor kan de PTT zien welke transactie verstuurd is door welke testrunner. Hierna moet het resultaat opgeslagen worden bij alle andere resultaten. Wanneer alle resultaten van alle transacties ontvangen zijn moet de gehele lijst in een TestRunnerResultMessage worden verwerkt en naar de manager terug gestuurd worden die opgeslagen is tijdens het starten.

TESTRUNNERTERMINATEMESSAGE

De gebruiker kan een timeout instellen bij het uitvoeren van een test scenario. Wanneer dit het geval is dan wordt er een TestRunnerTerminateMessage ingepland om verstuurd te worden naar de TestRunnerManager. Deze zal wanneer deze ontvangen worden ook een TestRunnerTerminateMessage versturen naar alle testrunners. De testrunner moet dan stoppen met het verzenden van transacties en de blockchain provider stoppen. De blockchain provider kan gebruik maken van timers, scheduled messages of zichzelf berichten sturen voor het checken van staat veranderingen van verzonden transacties. Dit is niet meer nodig dus is het stoppen van de blockchain provider nodig om onnodige handelingen te voorkomen.

Na het stoppen moeten alle TransactionResults die tot dan toe zijn verzameld naar de manager worden gestuurd zodat er nog een test rapport kan worden gegenereerd.

CONFIGURATIE

Een testrunner kan worden voorzien van configuratie variabelen. Deze variabelen kan zelf bepaald worden en is onafhankelijk van andere testrunners. Het is dus belangrijk dat een gebruiker kan lezen welke configuratie variabelen er gebruikt worden in je testrunner.

Deze configuratie variabelen worden vervolgens geplaatst in het test scenario bestand. De enige eis aan je configuratie variabelen is dus dat het als geldige yaml kan worden geschreven in het testscenario bestand.

```yaml
# this is just an example of a testscenario file your blockchain provider will only receive the config value block
# for testing put the config values inside testBlockchainProviderConfigBlock.yml file
runners:
  testrunner1:
    &testrunner
    type: testrunner
    providerId: blockchainprovider1
    #  testrunner config block BEGIN

    #  testrunner config block END
  testrunner2:
    <<: *testrunner
    providerId: blockchainprovider2
useProvidersForSetup:
  blockchainprovider1:
    passthroughSetupDataTo:
      - blockchainprovider2
providers:
  blockchainprovider1: &blockchainprovider
    type: empty-blockchainprovider
    functions:
      - function1
      - function2
    #  blockchain provider config block BEGIN

    #  blockchain provider config block END
  blockchainprovider2:
    <<: *blockchainprovider
```

Figuur 3: Test scenario bestand voorbeeld

Dit bestand wordt door de PTT geparsed en daar wordt je testrunner config blok uit gehaald en doorgestuurd naar je EmptyTestRunnerFactory tijdens de createInstance methode in de vorm van een yaml string. Dit parsen ondersteund merge keys waardoor het dupliceren van configuratie blokken mogelijk is.

TESTING

Om de EmptyTestRunner goed te testen is het TestKit van Akka nodig. Dit is een klasse om een actor binnen een Akka systeem te mocken en te gebruiken. In de start module staan al twee testen waar een TestKit gebruikt wordt om een TestRunnerManager te simuleren. Deze verstuurd dan de mogelijke berichten om te kijken of de juiste berichten terug worden gestuurd. Voor de blockchain provider is een MockProvider klasse gemaakt. Dit is een blockchain provider die bij het ontvangen van een SendTransactionMessage zal hij het verzenden van een transactie nabootsen. Dit doet hij door middel van scheduled messages te gebruiken om na een bepaalde tijd de transactie te ontvangen en daarna te confirmeren. Bij het ontvangen of confirmeren van een transactie is er een kans dat de transactie faalt. Deze kans is in te stellen in de MockProvider. Hierdoor kunnen alle mogelijke geldige TransactionResult objecten worden verstuurd naar je testrunner.