Základní nástroje pro paralelní a distribuované systémy v jazyce Go

Tomáš Urbanec

Verze: 31.10.2024

Tento dokument slouží jako ukázka typu a rozsahu informací, které by měly být předány kolegům při představování možností daného jazyka pro paralelní a distribuované výpočty. K dokumentu je dostupný malý repozitář s ukázkami použití zmiňovaných nástrojů. Zároveň slouží jako ukázka možné formy doporovdného handoutu. Představení možností daného jazyka a doprovodný handout jsou součástí zápočtových úkolů v předmětu KMI/PDS.

Paměťový model jazyka Go

„Nekomunikuj sdílením paměti, sdílej paměť komunikováním.“

V jazyce Go preferujeme komunikaci pomoci tzv. kanálů a přeposílání zpráv/dat před klasickým přístupem ke sdílený proměnným.

Model paměti v Go je silně ovlivněn CSP (Communicating sequential processes), které synchronizuje procesy pomocí předávání zpráv. Go pro konkurenční vykonávaní používá tzv. gorutiny, což jsou velmi lehká vlákna spravována přímo jazykem. CSP se v Go manifestuje prostřednictvím kanálů, což umožňuje gorutinám komunikovat, aniž by sdílely paměť přímo. Go nicméně nabízí i „klasické“ synchronizační nástroje typu zámků, podmíněných proměnných, atd.

Klíčovým konceptem v rámci modelu paměti Go je vztah „happens-before“, tj. princip, kde pokud jedna operace nastane před jinou, bude její výsledek viditelný pro tu druhou. Právě vynucováním happens-before programátor zajišťuje komunikaci mezi gorutinami. Například pokud je hodnota odeslána na kanál (což z definice nastane před odpovídající operací přijetí), je zaručeno, že jiná gorutina po jejím přijetí uvidí důsledky předchozích operací odesílatele. Takovýmto místům s vynuceným vztahem happens-before se říká body synchronizace. Při vhodném použití jsme schopni zamezit chybám souběhu a zajistit integritu dat. Kde programátor happens-before nezaručí, musí počítat s problémy (např. kvůli optimalizacím, kterém mimo body synchronizace mohou pracovat dle libosti). Více se dočtete v oficiálním popisu modelu [1].

Následuje zevrubný popis nástrojů pro vývoj paralelních či distribuovaných systémů dostupných přímo v jazyce a jeho standardní knihovně. Nedílnou součástí dokumentu jsou ukázky použití jednotlivých nástrojů dostupných spolu s tímto dokumentem.

Synchronizační nástroje

Kanály

Základní komunikační nástroj v Go. Zpráva je z kanálu přijata vždy až po jejím odeslání (happens-before).

Operace kanálu:

1. vytvoření
2. přijetí z kanálu, blokující `i, ok := <-chan`
3. odeslání do kanálu, blokující `chan <- i`
4. uzavření kanálu `chan.close()`
5. iterace přes kanál `range chan`
6. výběr kanálu s daty (select)

Kanály jsou bezpečné pro přístup z více gorutin (přijetí/odeslání/uzavření). Kanály mohou být bufferované (mají buffer s danou kapacitou, pak blokují) nebo nebufferované (udrží nejvýše jednu zprávu, pak blokují). Kanály mohou být jednosměrné – můžeme někomu předat kanál jen pro zápis nebo jen pro čtení.

Balíček *sync*

Mimo kanály Go nabízí i klasická synchronizační primitiva. Vše nalezneme v balíčku sync (součást standardní knihovny). Primitiva zajišťují happens-before, tedy body v běhu, kde jsou gorutiny synchronizovány.

Mutex: Klasický zámek. Maximálně jedna gorutina může držet zamčený mutex v daném okamžiku. Operace:

- Lock(): zamkne zámek,
- Unlock(): odemkne zámek.

RWMutex: Zámek pro čtenáře a písaře. Nechá číst více čtenářů, nebo zapisovat jednoho písaře v jednom okamžiku. Operace:

- RLock() a RUnlock() pro čtenáře (může více),
- Lock() a Unlock() pro písaře (jediný).

Cond: Podmíněná proměnná. Potřebuje dodat zámek (rozhraní Locker, např. Mutex) a poskytuje operace:

- Wait(): atomicky zablokuje gorutinu, odemkne zámek a čeká na Signal() nebo Broadcast().
- Signal(): probudí jednu čekající gorutinu.
- Broadcast(): probudí všechny čekající gorutiny.

Waitgroup: Nástroj umožňující blokovat gorutinu než přijde signál od n procesů. Často používána pro čekání na dokončení spuštěných gorutin. Operace:

- Add(n): přičte n do vnitřního čítače.
- Done(): sníží vnitřní čítač o 1.
- Wait(): blokuje volající gorutinu/y dokud vnitřní čítač není 0.

Balíček *sync/atomic*

Podbalíček pro práci s nízkoúrovňovými atomickými operacemi. Umožňuje načítat/ukládat složené hodnoty bez zámků, používat klasické složené atomické operace atd.

Pomocné nástroje

Once Zajistí, že zadaná funkce poběží nejvýše jednou nezávisle na počtu zavolání. Často používáno pro inicializaci.

Pool Pool je zásobárna pro dočasné objekty. Často používaný pro efektivní práci se znovupoužitelnými objekty (např. buffery), kde zajistí méně alokací. Operace:

- Get() získá objekt z poolu nebo vytvoří nový objekt, je-li pool prázdný,
- Put() vrátí objekt zpět do poolu.

Map Mapa/Slovník s bezpečným přístupem z více gorutin.
Operace:

- Store(key, value): přidá (nebo upraví) záznam do mapy
- Load(key): najde záznam v mapě, vrací hodnotu a flag indikující platnost
- Delete(key): Smaže záznam v mapě.
- LoadOrStore(key, value): Vrátí záznam klíče v mapě, pokud existuje. Jinak vytvoří nový s dodanou hodnotou a vrátí jej.
- Range(f func(key, value interface) bool): Iteruje dodanou funkci přes mapu, dokud nevrátí false nebo neprojde celou mapu.

Nástroje pro distribuované systémy

Základní nástroje pro práci se sítí nalezneme ve standardním balíčku *net*. Ten je dost rozsáhlý, zde si zmíníme jen pár zajímavých podbalíčků.

Balíček *net/http*

Standardní balíček [4] pro práci s HTTP klienty/servery/spojeními. Nabízí snadnou práci s klientem, serverem, handlery, requesty, ..., přímočarou podporu pro middleware, atd. Normálně prakticky použitelný.

Balíček *net/rpc*

Implementace RPC. Pro serializaci používá gob formát [2], lze upravit. Podporuje pouze TCP a HTTP pro přenos. Podporuje synchronní i asynchronní volání. Nabízené procedury mají omezený formát (vstupní a výstupní argument, návratová hodnota).

Pomocné nástroje

1. Balíček context [5]: kontext umožňuje přehlednou správu života gorutiny – rušení, timeouty, metadata, deadliny, ... Užitečný i jinde než při správě gorutin.
2. Race detector: nástroj umožňující detekovat možné chyby souběhu v programu. Spouští se flagem -race u run/test/build (výrazně zatěžuje zdroje – pro vývoj a testování ne pro produkci).

Další zajímavé knihovny

Balíček *gRPC*

Go implementace gRPC.
https://github.com/grpc/grpc-go

Balíček *gorilla/websocket*

Často používaná Go implementace websocketů. Za zmínku stojí celý balíček gorilla.
https://github.com/gorilla/websocket

Exsitují i další implementace websocketů:

- https://pkg.go.dev/golang.org/x/net/websocket
- https://pkg.go.dev/github.com/coder/websocket

Balíček *go-micro*

Go framework pro vývoj mikroslužeb (a distribuovaných systému obecně).
https://github.com/micro/go-micro

Zajímavé projekty napsané v Go

- https://github.com/docker – kontejnerizace
- https://github.com/kubernetes/kubernetes – orchestrační nástroj
- https://github.com/grafana/grafana – nástroj pro monitoring/observabilitu
- https://github.com/etcd-io/etcd – distribuované key-value úložiště
- https://github.com/hashicorp/terraform – nástroj pro správu infrastruktury jako kódu
- https://github.com/jaegertracing/jaeger – distribuovaný tracing
- https://github.com/prometheus/prometheus – monitorovací nástroj a DB pro časové řady
- https://github.com/grpc/grpc-go – RPC protokol

Reference

[1] *Paměťový model Go*, https://go.dev/ref/mem, [Citováno 31.10.2024]

[2] *Gob formát*, https://go.dev/blog/gob, [Citováno 31.10.2024]

[3] *Balíček sync*, https://pkg.go.dev/sync, [Citováno 31.10.2024]

[4] *Balíček net/http*, https://pkg.go.dev/net/http, [Citováno 31.10.2024]

[5] *Balíček context*, https://pkg.go.dev/context, [Citováno 31.10.2024]