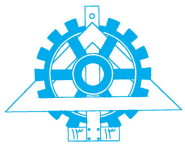
مبانی سیستم های توزیع شده

نام و نام خانوادگی: سیدمهدی حاجی سیدحسین

۸۱۰۱۰۰۱۱۸



پروژه صفر

فهرست مطالب

۱ بخش اول پروژه : پیاده سازی سرور

۱.۱ معرفی client struct

در ابتدای کار یک struct ساختیم که وظیفه ی آن نگهداری Channel برای هر client جهت برقراری ارتباط به صورت جداگانه با سرور است .

۲.۱ معرفی keyValueServer struct

keyValueServer یک ساختار داده‌ای است که مسئول مدیریت ارتباطات کلاینت‌ها و پردازش دستورات در پایگاه داده kvstore.KVStore می‌باشد. این ساختار شامل اجزای زیر است:

- **kvstore.KVStore : store**

  این متغیر وظیفه‌ی نگهداری پایگاه داده‌ی keyValue را بر عهده دارد.

- **map[net.Conn]*client : clients**

  این نقشه (map) هر اتصال (conn) را به شیء client متناظر نگاشت می‌کند.

- **struct chan : closeChan**

  این کانال جهت اطلاع‌رسانی به سرور برای خاتمه یافتن اجرا استفاده می‌شود.

- **net.Conn chan : joinChan**

  این کانال هنگام اتصال یک کلاینت جدید، سرور را مطلع می‌سازد. هنگامی که در تابع Start() یک اتصال جدید پذیرفته می‌شود، آن اتصال در joinChan قرار گرفته و گوروتین run() آن را پردازش می‌کند. این پردازش شامل اضافه کردن کلاینت به نقشه clients و راه‌اندازی روال‌های خواندن و نوشتن برای آن کلاینت است.

- **net.Conn chan : leaveChan**

  این کانال سرور را هنگام قطع ارتباط یک کلاینت مطلع می‌سازد. زمانی که یک کلاینت به هر دلیلی (مانند خطا یا بسته شدن اتصال) از سرور جدا شود، اتصال آن در leaveChan قرار می‌گیرد و گوروتین run() این رویداد را پردازش می‌کند. این پردازش شامل حذف کلاینت از clients و به‌روزرسانی شمارنده‌ی کلاینت‌های قطع‌شده است.

- **int32 : droppedClientsCounter**

  این متغیر تعداد کلاینت‌هایی که ارتباطشان قطع شده است را نگهداری می‌کند.

- **command chan : commandChannel**

  این کانال برای دسترسی ایمن به داده‌ها در kvstore.KVStore طراحی شده است. با توجه به اینکه چندین کلاینت می‌توانند هم‌زمان درخواست ارسال کنند، این کانال تضمین می‌کند که درخواست‌ها به صورت ترتیبی توسط گوروتین storeManager پردازش شوند. این روش باعث می‌شود که برنامه thread-safe باشد.

۳.۱ ساختار command در پروژه

ساختار command به‌عنوان یک واسط بین کلاینت‌ها و پردازشگر اصلی سرور عمل می‌کند. این ساختار، درخواست‌های مربوط به عملیات روی داده‌های ذخیره‌شده را مدل‌سازی کرده و شامل فیلدهای مورد نیاز برای پردازش یک دستور در سیستم key-value است.

هر گوروتین یا ماژولی که نیاز به اجرای یک دستور روی پایگاه داده دارد، یک نمونه از command ایجاد کرده و آن را در کانال پردازش فرمان‌ها ارسال می‌کند. سرور، دستورات دریافتی را پردازش کرده و نتیجه را از طریق respChan به درخواست‌دهنده بازمی‌گرداند.

۴.۱ تابع New و روتین storeManager

۱.۴.۱ تابع New

تابع New مسئول ایجاد و مقداردهی اولیه یک نمونه از keyValueServer است. این تابع معمولاً وظایف زیر را انجام می‌دهد:

- مقداردهی اولیه به ساختار keyValueServer، شامل ایجاد یک نمونه از KVStore برای ذخیره داده‌ها.
- مقداردهی map مربوط به کلاینت‌ها.
- ایجاد کانال‌های مورد نیاز برای مدیریت ارتباطات کلاینت‌ها (joinChan، leaveChan و commandChannel).
- مقداردهی متغیر شمارنده‌ی کلاینت‌های قطع‌شده (droppedClientsCounter).
- راه‌اندازی storeManager که وظیفه‌ی پردازش درخواست‌های ارسالی به commandChannel را دارد.

۲.۴.۱ روتین storeManager [۱] [۲]

این goroutine نقش اصلی در پردازش درخواست‌های کلاینت‌ها را بر عهده دارد. وظایف آن شامل موارد زیر است:

- دریافت دستورات (command) از commandChannel.
- پردازش عملیات روی KVStore (مانند GET، SET، DELETE).
- ارسال پاسخ مناسب از طریق respChan به کلاینت درخواست‌دهنده.
- اطمینان از اجرای دستورات به صورت ترتیبی، که باعث می‌شود سرور thread-safe باشد.

این معماری باعث هماهنگی بهتر بین درخواست‌های هم‌زمان کلاینت‌ها و بهبود عملکرد سرور می‌شود.

۵.۱ تابع Start

تابع Start مسئول راه‌اندازی سرور و مدیریت اتصال کلاینت‌ها است. این تابع یک goroutine برای ()run ایجاد می‌کند که به صورت مداوم کانال‌های joinChan و leaveChan را پردازش می‌کند. سپس، سرور در یک حلقه منتظر دریافت اتصالات جدید از طریق net.Listener می‌ماند. هنگام برقراری اتصال جدید، آن را به joinChan ارسال می‌کند تا در ()run پردازش شده و به لیست کلاینت‌ها اضافه شود. این طراحی باعث می‌شود که مدیریت کلاینت‌ها به صورت کارآمد و هم‌زمان انجام شود.

۶.۱ تابع Close

تابع Close مسئول خاموش کردن سرور و بستن تمام اتصالات کلاینت‌ها است. با بسته شدن closeChan، تمام گوروتین‌های وابسته از اجرای خود خارج می‌شوند. سپس، تمامی اتصالات موجود در clients بسته شده و این مپ پاک‌سازی می‌شود. این تابع اطمینان می‌دهد که سرور به‌درستی خاموش شده و هیچ اتصال بازی باقی نمی‌ماند.

۷.۱ تابع‌های CountActive و CountDropped

تابع CountActive تعداد کلاینت‌های متصل را با شمارش اعضای clients بازمی‌گرداند. تابع CountDropped تعداد کلاینت‌هایی که اتصالشان قطع شده را از طریق droppedClientsCounter گزارش می‌دهد. این توابع برای نظارت بر وضعیت سرور استفاده می‌شوند.

۸.۱ روال readRoutine [۳]

روال readRoutine مسئول پردازش داده‌های دریافتی از کلاینت است. این تابع با استفاده از یک bufio.Scanner ورودی را به‌صورت خط‌به‌خط می‌خواند و هر خط را تجزیه می‌کند. بر اساس نوع درخواست (مانند Put، Get، Delete و Update)، این داده‌ها به commandChannel ارسال می‌شوند تا به‌صورت ترتیبی پردازش شوند. در صورت قطع ارتباط کلاینت، اتصال آن به leaveChan ارسال می‌شود تا به درستی مدیریت شود.
در جریان پردازش درخواست‌ها، در هنگام ارسال پاسخ‌ها به کلاینت، ابتدا چک می‌شود که آیا کانال نوشتن writeChan برای آن کلاینت پر است یا نه. اگر کانال پر باشد، پیام از دست خواهد رفت و پیامی در کنسول نمایش داده می‌شود که نشان‌دهنده کند بودن خواندن از طرف کلاینت است. این مکانیزم برای جلوگیری از بلوکه شدن و حفظ کارایی سرور در هنگام کار با چندین کلاینت طراحی شده است.[۵]

۹.۱ روال writeRoutine

روال writeRoutine مسئول ارسال پاسخ‌ها به کلاینت‌ها از طریق کانال writeChan است. این تابع به‌طور مداوم در حال خواندن پیام‌ها از کانال است و هر پیام را به کلاینت متصل ارسال می‌کند. استفاده از writeChan با ظرفیت مشخص، از مسدود شدن سرور در صورت کند بودن کلاینت جلوگیری می‌کند و از ارسال پیام‌ها به‌صورت ایمن و کارآمد اطمینان می‌یابد.

۲ بخش دوم پروژه : تست نویسی

۱.۲ گزارش تست‌ها

۱.۱.۲ تست `TestBasicCorrectness` [۴]

در این تست، صحت عملکرد کلاس `SquarerImpl` برای محاسبه مربع یک عدد ساده بررسی می‌شود. عدد 2 به عنوان ورودی به کانال `input` ارسال می‌شود و سپس مربع آن (که باید برابر با ۴ باشد) از کانال `squares` دریافت می‌شود. اگر نتیجه‌ی محاسبه‌شده برابر با ۴ نباشد، خطا گزارش می‌شود. همچنین، اگر زمان تست از ۵۰۰۰ میلی‌ثانیه تجاوز کند، خطای `timeout` ایجاد می‌شود.

۲.۱.۲ تست `TestSquarerImplWithNegativeValue`

در این تست، عملکرد `SquarerImpl` برای ورودی منفی بررسی می‌شود. با استفاده از یک مقدار منفی -10، مربع آن محاسبه می‌شود که نتیجه‌ی صحیح باید برابر با 100 باشد. این تست از تابع کمکی `helperFuncForTest` استفاده می‌کند تا ورودی و خروجی را مقایسه کرده و در صورت بروز خطا پیام مناسب ارسال کند.

۳.۱.۲ تست `TestSquarerImplWithZeroValue`

در این تست، عملکرد `SquarerImpl` برای ورودی صفر بررسی می‌شود. عدد 0 به عنوان ورودی به کانال `input` ارسال شده و انتظار می‌رود که نتیجه‌ی محاسبه‌شده برابر با 0 باشد. همانند تست قبلی، از تابع کمکی `helperFuncForTest` برای انجام مقایسه و گزارش خطا استفاده می‌شود.

۴.۱.۲ تست `TestSquarerImplWithEmptyChannel`

این تست بررسی می‌کند که سرور `SquarerImpl` هنگام بسته شدن کانال ورودی چه عملکردی دارد. در این تست، ابتدا کانال `input` بسته می‌شود و هیچ مقداری به آن ارسال نمی‌شود. در این حالت، انتظار می‌رود که سرور مقدار صفر را به عنوان نتیجه‌ی خروجی ارسال کند. اگر نتیجه‌ی محاسبه‌شده غیر از صفر باشد یا زمان تست تمام شود، خطای مناسب گزارش می‌شود.

مراجع

[1] “Thead-safty using channels in golang,” https://stackoverflow.com/questions/13690734/thread-safe-way-to-funnel-data-from-multiple-go-routines.

[2] “Thead-safty in golang,” https://pavledjuric.medium.com/thread-safety-in-golang-47fa856fb8bb.

[3] “Buffered channel,” https://go.dev/tour/concurrency/3.

[4] “Test running in golang,” https://stackoverflow.com/questions/16935965/how-to-run-test-cases-in-a-specified-file.

[5] “Check if buffered channel is full,” https://stackoverflow.com/questions/25657207/how-to-know-a-buffered-channel-is-full.