Урок Спринт 2.5

Кеширование данных

Аннотация

В этом уроке мы поговорим подробнее о синхронизации доступа к общему ресурсу с помощью мьютекса. Название мьютекс происходит от английского mutual exclusion. Мьютекс используется для создания критической секции (подробнее можно почитать здесь), которая гарантирует, что только одна горутина в данный момент может выполнять этот раздел кода.

Кеширование данных

В программировании часто возникает задача реализовать кеширование. Некоторые операции, такие как дорогостоящие вычисления, операции ввода-вывода или работа с базами данных, выполняются медленно, поэтому имеет смысл кешировать результаты после их получения.

Давайте попробуем написать кеш, который загружает объекты из базы данных и помещает их в мапу. Для этого создадим интерфейс, который описывает метод получения данных из базы :

```
type DataRetriever interface {
    Retrieve(ID string) (*Data, error)
}
```

Любая база данных, либо другие источники, реализующие интерфейс `DataRetriever` могут быть использованы в нашем кеше.

Далее создадим структуру для описания данных:

```
type Data struct {
    ID string // для упрощения содержит только ID
}
```

Теперь создадим сам кеш:

```
// Кеш для хранения данных
type Cache struct {
    // данные будем хранить здесь
    m  map[string]*Data
    dr DataRetriever
}
// Создание нового объекта
func NewCache(dr DataRetriever) *Cache {
    return &Cache{
        m:  make(map[string]*Data),
        dr: dr,
    }
}
```

Нужно реализовать метод `Get` для получения данных из кеша:

```
func (c *Cache) Get(ID string) (Data, bool) {
    // проверим, есть ли данные в кеше
    data, exists := c.m[ID]
    // нашли в мапе - вернём значение
    if exists {
        return *data, true
    }
    // данные не нашли - нужно запросить
    data, err := c.dr.Retrieve(ID)
    if err != nil {
        // ошибка получения данных - запишем в лог
        log.Printf("c.dr.Retrieve(ID): %s", err)
        // вернём пустое значение
        return Data{}, false
    }
    // получили значение - запомним
    c.m[data.ID] = data
    // вернём полученное значение
    return *data, true
}
```

Это вполне рабочий вариант, но при условии использования в одной горутине. Иначе, при чтении и записи в кеш несколькими горутинами мы получим состояние гонки (`race condition`). Но кеш обычно используется многими горутинами, поэтому он должен быть потокобезопасным. Для этого добавим мьютекс:

```
type Cache struct {
   m  map[string]*Data
   dr DataRetriever
   mu sync.Mutex
}
```

Вызов `mu.Lock()` и `mu.Unlock()` будет определять, что находится в критической секции. Только после вызова `Unlock()` другая горутина сможет заблокировать мьютекс `mu` с помощью `mu.Lock()`. Внесём

изменения в функцию `Get`:

```
func (c *Cache) Get(ID string) (Data, bool) {
   c.mu.Lock()
   data, exists := c.m[ID] // теперь доступ к мапе внутри критической секции
   c.mu.Unlock()
   // нашли в мапе - вернём значение
   if exists {
    return *data, true
   }
   // запрос данных из базы - не  в критической секции
   data, err := c.dr.Retrieve(ID)
   if err != nil {
    // ошибка получения данных - запишем в лог
    log.Printf("c.dr.Retrieve(ID): %s", err)
    // вернём пустое значение
    return Data{}, false
   }
   // перед обращением к мапе снова заблокируем мьютекс
   c.mu.Lock()
   // разблокируем при выходе из функциии
   defer c.mu.Unlock()
   // внутри критической секции нужно снова проверить на наличие значения в мапе
   data, exists = c.m[data.ID]
   if exists {
    return *data, true
   }
   // получили значение - запомним
   c.m[data.ID] = data
   // вернём полученное значение
   return *data, true
}
```

Обратите внимание, что код получения объекта из базы данных находится вне критической секции. Таким образом другие горутины могут продолжать использовать кеш.

После получения объекта из базы кеш снова блокируется, поскольку полученный объект необходимо поместить в мапу. После этого необходимо снова проверить, был ли объект уже помещён в кеш другой горутиной. Это возможно, поскольку несколько горутин могут одновременно запрашивать объект, используя один и тот же идентификатор. Повторная проверка после получения блокировки позволит убедиться, что если другая горутина уже поместила объект в кеш, он не будет перезаписан новой копией.

Справка