Блокировки выполнения

- Блокировки как средство управления параллелизмом
- Блокировки Что это? и Как работает?

Блокировки как средство управления параллелизмом

Управление блокировками является центральной концепцией в контексте многопользовательских систем при совместном, параллельном использовании ресурсов несколькими пользователями, точнее - параллельными процессами. Для построения правильной архитектуры параллельных процессов очень важно понимать основные принципы работы блокировок, чтобы не допускать распространенных ошибок.

Мы не будем приводить подробное описание принципа работы блокировок, тк такие статьи можно с легкостью найти в интернете, но для тех кто в этом совсем новичок, приведем список статей, которые имеет смысл изучить, хотя-бы поверхностно:

Состояние гонки (Race Condition)
Взаимная блокировка (Deadlock)
Атомарная операция (Atomic Operation)
Параллельные вычисления
Распределённый менеджер блокировок

Почему блокировки важны для вашего бизнеса

Не редко случаются ситуации, когда нескольким пользователям в одно и то же время требуется взаимодействие с одними и теми же данными. Например, два пользователя могут одновременно попытаться зарегистрироваться на один и тот же временной слот. Без применения блокировок, в базу данных поступят две заявки на одно и то же время, что приведет к проблемам, особенно если другие временные слоты уже заняты.

При использовании блокировок вы сможете обеспечить последовательное выполнение операций, запрошенных одновременно, гарантируя таким образом более надежную работу вашего приложения.

Приложение Metabot предоставляет различные функции для работы с блокировками в контексте вашего бизнеса или конкретного бота. Например, функции получения блокировки включают в себя проверку блокировки и захват блокировки. Эти две операции является атомарными, т.е. выполняются одномоментно, если блокировка доступна для захвата (т.е. если никакой другой процесс не захватил блокировку на момент проверки). Такая атомарная операция автоматически исключает возникновение состояния гонки (Race Condition).

Чтобы не возникло взаимной блокировки, предусмотрено указание времени жизни блокировки (time to leave) в параметрах функций захвата блокировки (см параметр ttlSec).

Также, для упрощения работы с блокировками, в функции захвата встроены алгоритмы, позволяющие автоматически выполнять повторную попытку захвата блокировки на протяжении указанного промежутка времени (см параметр maxWaitSec).

Все это может быть особенно полезно во многих сценариях, в которых важно обеспечить согласованность и предотвратить конкурирующие операции.

> **ВНИМАНИЕ!!!** Будьте очень аккуратны при использовании функций блокировок, неправильное использование может нарушить работоспособность всей системы! Ваш бот может быть внесен в черный список а все вебхуки и задания бота будут исключаться из очередей!

> Подробное описание назначения блокировок с описанием примеров смотрите в статье про блокировки в нашем блоге. **ДОБАВИТЬ ССЫЛКУ!**

Методы блокировок в Metabot

Приложение Metabot предлагает ряд методов, которые помогают управлять блокировками.

> Все перечисленные ниже методы вам не нужны, если вы используете стандартные функции связанные с контакт-центром: lead.**assignDialogToNextOperator**, lead.**assignDialogToOperator** или lead.**getNextOperators** и т.д., так как данные функции уже включают в себя алгоритмы блокировок.

Методы генерации имени блокировок

Использование на данный момент смысла не имеет, функции блокировок сами добавляют дополнительные префиксы к именам блокировок указанным в аргументах функций.

Метод	Описание

bot.**getLockNameForBot**(string $lockName, string $lockPrefix = ''): **string**	Генерирует имя блокировки по боту
bot.**getLockNameForBusiness**(string $lockName, string $lockPrefix = ''): **string**	Генерирует имя блокировки по бизнесу

Методы перехвата блокировок

Данные методы позволяют дождаться, когда блокировка будет освобождена, и затем захватить ее. Это может быть полезно, чтобы обеспечить выполнение операции только одним процессом в определенный момент времени.

Метод	Описание
bot.**waitForBusinessLock**(string $lockName, string $lockPrefix = '', ?int $ttlSec = null, $maxWaitSec = 300): **bool**	Захватывает блокировку по бизнесу
bot.**waitForBotLock**(string $lockName, string $lockPrefix = '', ?int $ttlSec = null, $maxWaitSec = 300): **bool**	Захватывает блокировку по боту

> Если нет понимания что указать в параметре ttlSec, укажите значение равное 30 (это значит что блокировка будет автоматически удалена спустя 30 секунд, если вы забыли это сделать в коде), это необходимо, чтобы не возникло взаимной блокировки, если же вы указываете время жизни равное 0, то блокировка не ограничена по времени, но в таком случае нужно подходить к алгоритмам еще тщательнее, чтобы не забывать освобождать блокировку.

Методы удаления блокировок

Данные методы позволяют освобождать различные виды блокировок. Какие именно из этих методов использовать зависит от того, была ли блокировка создана в текущем скрипте или нет, и нужно ли вам принудительно освободить блокировку.

Метод	Описание
bot.**releaseAllCurrentLocks**(): **bool**	Освобождает все блокировки захваченные в текущем скрипте
bot.**releaseCurrentLockForBusiness**(string $lockName, string $lockPrefix = ''): **bool**	Освобождает блокировку захваченную текущем скрипте по бизнесу

bot.**releaseCurrentLockForBot**(string $lockName, string $lockPrefix = ''): **bool**	Освобождает блокировку захваченную в текущем скрипте по боту
bot.**releaseLockForBusiness**(string $lockName, string $lockPrefix = ''): **bool**	Освобождает блокировку по бизнесу
bot.**releaseLockForBot**(string $lockName, string $lockPrefix = ''): **bool**	Освобождает блокировку по боту

Методы проверки наличия блокировок

Данные методы позволяют проверить наличие блокировок. Они помогут определить, должен ли ваш скрипт ожидать освобождения блокировки или может продолжить работу.

Метод	Описание
bot.**hasLockForBusiness**(string $lockName, string $lockPrefix = '', $checkValue = null): **bool**	Проверяет существование блокировки по бизнесу
bot.**hasLockForBot**(string $lockName, string $lockPrefix = '', $checkValue = null): **bool**	Проверяет существование блокировки по боту

Все эти методы обеспечивают гибкость при работе с блокировками в вашем приложении Metabot. С их помощью вы можете создавать более надежные и эффективные бизнес-процессы.

> Подробнее методы перечисленные выше и примеры к ним вы можете изучить в **Справочнике по функциям JS**

> **ВНИМАНИЕ!!!** Если у вас нет понимания как пользоваться функциями блокировок, то лучше не экспериментировать и обратиться к нашим специалистам!

Блокировки Что это? и Как работает?

Блокировки и состояние гонки являются центральными концепциями в контексте многопользовательских систем и совместного использования ресурсов. Понимание этих понятий особенно важно при работе с общими данными, с которыми могут взаимодействовать два или более пользователя.

Зачем это нужно

Возьмем для примера платформу Metabot. Её пользователи обычно действуют автономно и никак не влияют на действия друг друга. Тем не менее, случаются ситуации, когда одновременное взаимодействие нескольких пользователей с одними и теми же данными становится неизбежным.

Чтобы рассмотреть этот процесс подробнее возьмем простой пример - сервис записи на прием к специалисту. В этом сервисе, два пользователя могут одновременно попытаться зарегистрироваться на один и тот же временной слот. В базу данных поступят две заявки на одно и то же время, что приведет к проблемам, особенно если другие временные слоты уже заняты.

Здесь на помощь приходят блокировки.

Как это работает

Важно понимать, что блокировка сама по себе не блокирует доступ к ресурсу. Вместо этого, она служит своего рода условием или соглашением, общей переменной, которую могут проверять различные пользователи или процессы. Таким образом, когда пользователь (или, в нашем контексте, бот) хочет получить доступ к ресурсу, он проверяет состояние блокировки (равно ли значение переменной **true**).

Если переменная указывает, что ресурс свободен, пользователь "захватывает" блокировку, изменяя ее значение на **false** для всех остальных пользователей. Это действие символически сообщает системе, что ресурс теперь занят, и другие пользователи или процессы не могут получить к нему доступ до тех пор, пока текущий пользователь не "освободит" блокировку, вернув ей значение **true**.

Ниже можно посмотреть как данный процесс выглядит в форме кода:

```
let lockName = 'doctorsRecord' // Имя для вашей блокировки | string
let lockPrefix = '' // Добавляется к lockName, помогает в категоризации блокировок | string
let ttlSec = 30 // Через какое количество секунд блокировка снимется самостоятельно | int
let maxWaitSec = 5 // Сколько секунд бот будет пытаться получить блокировку (Максимум 300) | int
let isOpenRecord = bot.waitForBotLock(lockName, lockPrefix, ttlSec, maxWaitSec) // Метод захвата блокировки

if (isOpenRecord) {
  // Выполянется, если получилось захватить блокировку и isOpenRecord = true
} else{
  // Выполянется, если НЕ получилось захватить блокировку и isOpenRecord = false
}
```

В скрипте задаются значения переменным, необходимым для захвата блокировки. Затем вызывается метод, возвращающий значение блокировки. И в завершении происходит проверка: если блокировка свободна (её значение равно **true**), то выполняются одни действия, если блокировка занята (её значение равно **false**), то выполняются другие действия.

> Важно понимать, что операция по захвату и проверке осуществляется одновременно. Если функция вернула **true**, значит у текущего пользователя получилось захватить блокировку, а у других пользователей - нет.

Время жизни блокировки

Бывают ситуации, когда пользователь забывает или просто уходит, оставляя блокировку в "захваченном" состоянии, что закрывает доступ других пользователей к ресурсу. Для решения этой проблемы используется "время жизни" блокировки. Это означает, что блокировка автоматически "освобождается" после определенного периода времени (например, 30 секунд), даже если пользователь, изначально захвативший блокировку, не освободил её самостоятельно. Это обеспечивает более справедливый и эффективный доступ к ограниченным ресурсам.

Что это дает

Важно понимать, что само по себе использование блокировок не является гарантией исключения состояний гонки. Такие ситуации могут по-прежнему возникать, если не соблюдается порядок и последовательность операций. Блокировки просто предоставляют новый уровень контроля над использованием ресурсов, и являются семафором для сигнализации другим пользователям или процессам о статусе ресурса.

Блокировки, при правильном использовании, позволяют системам эффективно обрабатывать совместную работу с ресурсами и поддерживать согласованность данных, предотвращая конфликты и ошибки. Они являются весомым инструментом в руках разработчиков программного обеспечения для управления совместным доступом к ресурсам в многопользовательских и многопоточных системах.