Сервис CLI. Описание

CLI сервис — это backend часть системы CLI.

Функциональность:

* Позволяет одновременно работать нескольким пользователям.
* Поддерживает возможность по аутентификации и авторизации пользователей.
* Связывает с каждым пользователем, подключившимся к cli сервису, его клиентскую сессию: обработчик клиентских запросов и клиентское состояние. Клиентские сессии двух разных пользователей не зависят друг от друга.
* Поддерживает автоматический выход пользователя по таймауту (и очистку его клиентской сессии).
* Поддерживает динамическую конфигурацию при запуске: информация о наборе команд, конфигурации машины состояний CLI, уровнях доступа загружается из соответствующих конфигурационных файлов при запуске.
* Содержит команды, режимы работы CLI и переходы между ними согласно спецификации CLI.
* Каждое клиентское состояние содержит свою машину состояний (конечный автомат) CLI, используемую для управления режимами работы CLI.
* Информация о наборе команд содержит следующую информацию: команды для работы с реальным железом, команды, реализующие некоторую логику на стороне CLI сервиса и команды, реализующие некоторую логику на стороне терминального клиента. Информация о командах терминального клиента нужна исключительно для формирования помощи и данных автодополнения.
* Принимает ввод от пользователя, разбирает его и выполняет соответствующую цепочку команд в случае успешности разбора. Ввод пользователя может быть разобран как цепочка из одной команды, так и из нескольких команд.
* Перед выполнением команды происходит динамическая проверка возможности выполнить команду в данный момент времени. Результат динамической проверки зависит от следующих факторов: прошел ли пользователь процесс аутентификации, наличие у пользователя прав для выполнения команды, доступность команды в данном режиме работы CLI.
* Во время выполнения, команда может изменить состояние клиента, взаимодействовать с железом, а также передать данные в потоки вывода и выводы ошибок. Результатом выполнения каждой команды является ее код возврата: код возврата 0 означает, что команды выполнилась успешно, в противном случае значение кода возврата будет отлично от 0. Обработчик клиентских запросов может преобразовать этот код возврата в сообщение и отправить это сообщение клиенту.
* Реализует протокол взаимодействия с терминальным клиентом и создает точку подключения для терминального клиента через сокеты.
* Каждая команда cli сервиса содержит поток ввода, поток вывода и поток вывода ошибок. Команды в цепочке команд соединяются через потоки ввода, вывода и вывода ошибок. Обработчик клиентских запросов является итоговым приемником вывода цепочки команд.
* Каждая команда cli сервиса содержит справочную информацию о самой себе. Также каждая команда содержит свойства, упрощающие разбор пользовательского ввода и построения цепочки команд. Эти свойства содержат следующую информацию: принимает ли команда ввод через поток ввода, взаимодействует ли команда с реальным железом, является ли команда заглушкой для представления команды терминального клиента, необходимо ли пользователю пройти аутентификацию для использования этой команды и т. д.

Поведение

* При запуске cli сервиса для взаимодействия с клиентами создается только слушающий сокет (listen сокет). При подключении клиента (при запуске терминального клиента) к этому сокету, создается клиентский сокет, через который и происходит взаимодействие клиента с cli сервисом; также создается обработчик запросов для взаимодействия только с этим клиентом и клиентское состояние.
* После того как пользователь успешно вошел в систему (в случае успешного вызова команды **login**) в его клиентское состояние помещаются его реквизиты (по крайней мере имя пользователя и уровень его доступа).
* При получении запроса от клиента на выполнение некоторой cli команды (при получении ввода пользователя) происходит ее разбор. В результате разбора создается цепочка из одной или нескольких команд.
* После разбора запроса от клиента в цепочку команд, все команды из этой цепочки выполняются последовательно. Перед выполнением команды происходит проверка, может ли данная команда быть выполнена в данный момент времени. Доступность команды для выполнения зависит от следующих факторов: прошел ли пользователь процесс аутентификации, достаточно ли у текущего пользователя прав для выполнения данной команды, доступна ли команда в данном режиме работы CLI. Если команда для выполнения в данный момент времени не доступна, она не будет выполнена и выполнение все цепочки команд прервется. При этом результат работы предыдущих команд из цепочки останется (транзакционность выполнения цепочки команд не предусмотрена).
* Если выполнение некоторой команды из цепочки команд завершилось успешно, то эта команда (имя команды) используется для возможного изменения текущего состояния машины состояний CLI (не всякая команда приводит к изменению состояния) и, соответственно, режима работы CLI. Машина состояний CLI для данного клиента хранится в его клиентском сосоянии.
* Если выполнение некоторой команды из цепочки команд закончилось с ошибкой, то выполнение цепочки команд прерывается; код завершения выполнения этой команды будет кодом завершения выполнения всей цепочки.
* Если выполнение всех команд из цепочки команд завершилось успешно, то мы считаем, что выполнение всей цепочки команд завершилось успешно; кодом завершения выполнения всей цепочки команд будет 0.
* Если команда не является последней командой в цепочке команд, то вывод из потока вывода этой команды направляется в поток ввода следующей за ней команды. Если команда является последней командой в цепочке команд, то вывод из потока вывода этой команды направляется обработчику клиентских запросов. Если перенаправления потока вывода ошибок (в поток вывода) нет, то данные из потока вывода ошибок направляются обработчику клиентских запросов. Если перенаправления потока вывода ошибок (в поток вывода) есть, то данные из потока вывода ошибок направляются в поток вывода этой же команды.
* В случае выхода пользователя из системы (в случае успешного вызова команды **logout**) из клиентского состояния удаляются реквизиты этого пользователя; при этом как обработчик клиентских запросов, так и клиентский сокет продолжают работать.
* При отключении клиента от cli сервиса обработчик клиентских запросов завершает свою работу и закрывает клиентский сокет на стороне cli сервиса; если для клиента было создано клиентское состояние, то это состояние будет очищено.

Детали реализации

* Все конфигурационные файлы будут хранить данные в виде выражений языка Erlang. С одной стороны это сильно облегчит и ускорит разбор таких файлов, а с другой стороны особо не усложнит их синтаксис.
* Каждой команде cli сервиса соответствует некоторый модуль, реализующий поведение команды.
* В качестве потока вывода для команды мы будем использовать специальный процесс, накапливающий вывод пользователя. В этот же специальный процесс мы будем накапливать вывод ошибок в случае перенаправления вывода ошибок. Когда команда закончит свое выполнение весь вывод, накопленный в этом специальном процессе, становится потоком ввода для следующей команды. Весь вывод последней команды в цепочке команд, а также весь вывод ошибок, которые не были перенаправлены в поток вывода, отправляются на обработчик клиентских запросов. Поэтому каждой команде необходимо передавать три идентификатора процесса в качестве потоков ввода, вывода и вывода ошибок.
* Если запрос от клиента корректен, то будет создана цепочка команд, соответствующая запросу от клиента. Помимо этого, будет создан контекст выполнения этой цепочки, связывающий команды в цепочке команд друг с другом. Контекст выполнения команд передает каждой команде соответствующие идентификаторы процессов (в качестве потоков ввода, вывода и вывода ошибок). Также контекст выполнения после выполнения каждой команды посылает специальному процессу накопителю сообщение о том, что весь накопленный вывод можно считать вводом.
* После разбора запроса от клиента и создания цепочки команд, эта цепочка команд передается контексту выполнения; контекст выполнения — это отдельный процесс, выполняющий каждую команду из цепочки в своем процессе.
* Если выполнение цепочки команд завершилось с ошибкой (если какая-либо команда из цепочки команд вернула код возврата отличный от 0), то обработчик клиентских запросов вернет клиенту сообщение об этом через стандартный поток вывода ошибок.

Машина состояний (конечный автомат) CLI

Клиентское состояние, помимо всего прочего содержит машину состояний для управления режимами работы CLI. На данный момент конечный автомат имеет следующий вид:

*1) Состояния (режимы работы CLI)*

* Режим работы до аутентификации пользователя
* Фундаментальный режим работы
* Режим глобальной настройки
* Режим настройки интерфейсов
* Режим настройки группы интерфейсов
* Режим настройки VLAN

2) События, приводящие к смене состояния (команды)

* Команда **login**: режим работы до аутентификации пользователя → фундаментальный режим работы
* Команда ***configure terminal***: фундаментальный режим работы → режим глобальной настройки.
* Команда ***interface***: режим глобальной настройки → режим настройки интерфейсов.
* Команда ***interface range***: режим глобальной настройки → режим настройки группы интерфейсов.
* Команда ***vlan***: режим глобальной настройки → режим настройки VLAN.
* Команда ***end***: режим глобальной настройки, режим настройки интерфейсов, режим настройки группы интерфейсов, режим настройки VLAN → фундаментальный режим работы.
* Команда ***exit***: режим настройки интерфейсов, режим настройки группы интерфейсов, режим настройки VLAN → режим глобальной настройки; режим глобальной настройки → фундаментальный режим работы.
* Команда **logout**: фундаментальный режим работы, режим глобальной настройки, режим настройки интерфейсов, режим настройки группы интерфейсов, режим настройки VLAN → режим работы до аутентификации пользователя.

Все остальные команды состояние машины состояний CLI не меняют. В машину состояний CLI уведомление о выполнении команды, попадает только после ее успешного выполнения. Если команда выполняется не успешно, то машина состояний CLI уведомления не получает.

Структура (состояния и переходы) машины состояний будет меняться. Так, например, настройка некоторых протоколов (BGP, OSPF, MSTP) требуют добавления одного или более режима CLI. Поэтому мы будем загружать конфигурации конечного автомата из внешнего источника, вместо жесткой реализации в теле конечного автомата.

Разбор ввода пользователя и построения цепочки команд

При разборе ввода пользователя и построение цепочки команд мы будем делать следующие шаги:

1. Разбираем ввод пользователя на лексемы при помощи лексического анализатора.
2. Разбираем полученный набор лексем и строим AST при помощи синтаксического анализатора (LL(1)).
3. По построенной AST генерируем вызовы соответствующих команд (с соответствующими аргументами) и строим связи между этими командами (на основе перенаправления вывода и вывода ошибок).

Правила лексического анализа:

*Letter* = **A** | … | **Z** | **a** | … | **z**

*Word* = *Letter* {*Letter*}

*Character* = …

*Digit* = **0** | … | **9**

*String* = (*Digit* | *Letter*) {*Digit* | *Letter*} | **'"'** *Character* {*Character*} **'"'**

Правила синтаксического анализа:

*CompositeCommand* = *Command* | *Command* **'|'** *CompositeCommand* | *Command* **'|&'** *CompositeCommand*

*Command* = *Word* *Word* | *Word* *Word* *Args* | *Word* | *Word* *Args*

Args = *String* | *String* *Args*

Замечание по *Command*:

Во время запуска cli сервиса мы будем динамически формировать таблицу для синтаксического анализа исходя из известных на момент загрузки команд (для разбора нетерминала Command). У нас есть команды, состоящие из двух слов (например, configure terminal); более того, у нас есть ряд команд разной длины, имеющие общий префикс (например, interface и interface range), что создает неоднозначности во время синтаксического анализа. Для решения этой проблему мы будем формировать таблицу для синтаксического анализа так, чтобы команды с большим количеством слов располагались раньше, команд с меньшим количеством слов. При синтаксическом анализе при выборе варианта для Command мы будем делать пред просмотр не на один, а на несколько символов и выбирать вариант, расположенный раньше в таблице для синтаксического анализа. Другими словами, для Comand синтаксический анализатор у нас будет не LL(1), а LL(N) (в нашем случае, LL(2)).

Упрощенная реализация:

Для данных правил лексического и синтаксического анализа можно предложить следующую упрощенную реализацию разбора ввода пользователя и создания цепочки команд:

1. Разбираем ввод пользователя на лексемы при помощи лексического анализатора.
2. В списке лексем, полученном выше, находим лексемы со значениями **'|'** и **'|&'**. Эти лексемы делят список лексем на подгруппы: первая подгруппа расположена от начала списка до первой лексемы, вторая подгруппа — между первой и второй лексемами и т.д.
3. Для каждой полученной подгруппы проводим синтаксический анализ, причем начальным нетерминальным символом будет *Command*.
4. Для каждой полученной подгруппы по результатам синтаксического анализа создаем команду и передаем ей аргументы. Полученные команды объединяем в цепочку, причем тип объединения зависит от терминального символа (лексемы) разделителя из п.2. Если разделителем был терминальный символ **'|'**, то только вывод предыдущей команды подается на вход последующей команды, вывод ошибок в этом случае отправляется клиенту. Если разделителем был терминальный символ **'|&'**, то вывод и вывод ошибок предыдущей команды подаются на вход последующей команды в порядке их возникновения.

Команды cli сервиса