МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ИНТЕРАКТИВНОЙ ONLINE-СРЕДЫ ПРОГРАММИРОВАНИЯ ДЛЯ ЯЗЫКА ПРОГРАММИРОВАНИЯ ELIXIR

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Романовой Виктории Васильевны

Научный руководитель
зав. каф., к. ф.-м. н., доцент _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность темы. Современное информационное общество стремительно эволюционирует, обозначая важность высокой квалификации и умений специалистов в области информационных технологий. Требования к ним включают владение современными языками программирования, а динамичность и гибкость в разработке становятся ключевыми аспектами успешного программного обеспечения. С увеличением популярности новых языков программирования возникает потребность в эффективных инструментах для их изучения.

Интерактивная среда программирования REPL является популярным и простым в использовании программным обеспечением, которое предоставляет возможности для проведения экспериментов, тестирования идей и быстрого освоения нового языка. Способность моментального просмотра результатов выполнения каждой строки кода способствует лучшему пониманию технологий. Кроме того, REPL создает удобное окружение, облегчая процесс обучения без необходимости настройки сложных сред разработки.

Согласно ежегодному опросу, проводимому порталом Stack Overflow для выявления наиболее предпочтительных новых языков программирования в сфере информационных технологий в 2022 году, 75,46% опрошенных работников IT сферы выразили предпочтение в пользу функционального языка программирования Elixir, который подходит для разработки систем, нацеленных на высокую нагрузку и стабильность, позволяет обрабатывать большие объемы данных в реальном времени и поддерживать параллелизм на высоком уровне [**?**].

Однако, в силу относительной непопулярности Elixir до недавнего времени, существует проблема в наличии широко доступных образовательных сред для знакомства с основными конструкциями и принципами языка.

Цель бакалаврской работы — создание online REPL-среды для языка программирования Elixir. Решение должно предоставлять инструмент для возможности интерактивного программирования без необходимости локальной установки дополнительного программного обеспечения, а также включать в себя функциональность обучения.

Поставленная цель определила **следующие задачи**:

1. изучить принципы построения языка программирования Elixir для осуществления синтаксического анализа пользовательского кода;
2. изучить теоретические аспекты построения и состава интерактивного про-

граммирования;

3. разработать алгоритм проведения синтаксического анализа пользовательского кода для выявления уязвимых, небезопасных и запрещенных к использованию конструкций;
4. разработать алгоритм обработки пользовательского кода в изолированной среде в серверной части веб-приложения с возможностью получения результатов оценки, предупреждений или ошибок выполнения;
5. разработать модуль для осуществления клиент-серверного взаимодействия веб-приложения;
6. разработать удобный и интуитивно понятный интерфейс для взаимодействия со средой.

Структура и объём работы. Бакалаврская работа состоит из введения, двух разделов, заключения, списка использованных источников и пяти приложений. Общий объем работы — 64 страницы, из них 52 страницы — основное содержание, включая 14 рисунков, цифровой носитель в качестве приложения, список использованных источников информации — 21 наименование.

1 REPL для языка программирования Elixir

1.1 Описание работы REPL

REPL представляет собой интерактивную среду, которая запускает активную сессию нужного языка программирования и ожидает дополнительные входные данные, что обеспечивает мгновенное взаимодействие с кодом.

Аббревиатура REPL сложилась из его базовой реализации с помощью функций языка Lisp:

```
(loop (print (eval (read))))
```

1. **Чтение (Read)**

   Функция чтения принимает выражение от пользователя из стандартного канала ввода `stdin` и разбирает его в структуру данных в памяти. Если канал пуст, система ожидает ввода с клавиатуры.
2. **Вычисление (Eval)**

   Функция `eval` принимает внутреннюю структуру данных, оценивает или исполняет ее.
3. **Печать (Print)**

   Функция печати принимает результат, полученный на предыдущем шаге, и выводит его пользователю. Это может быть значение выражения, сообщение, результат функции.
4. **Повтор (Loop)**

   После завершения шагов чтения, оценки и печати, REPL возвращается в режим ожидания нового ввода пользователя, создавая цикл, который продолжается до тех пор, пока пользователь не завершит сеанс. В этом процессе переменные и другие определения, созданные в текущем контексте, сохраняются до окончания работы с REPL или до явного удаления их из памяти.

REPL обеспечивает удобный способ тестирования и экспериментирования с выбранным языком программирования, а также обеспечивает немедленную обратную связь, что делает его популярным инструментом для обучения, прототипирования и отладки кода.

1.2 Обзор REPL

Инструмент REPL может быть реализован в разнообразных формах, включая локальные, веб-браузерные, мобильные и облачные платформы. Большин-

ство популярных программных комплексов для разработки включают в себя интегрированные средства REPL. Для Elixir существует два основных REPL-решения: IEx и Livebook.

IEx (Interactive Elixir) является локальной интерактивной оболочкой для языка программирования Elixir, требующей предварительной установки данного языка на текущую систему. IEx разработан с помощью метапрограммирования и макросов Elixir, в основе которого лежит виртуальная машина Erlang BEAM, позволяющая динамически компилировать фрагменты кода в байт-код путём создания абстрактного синтаксического дерева (АСД). Elixir, используя возможности Erlang, запускает каждую сессию IEx в своем процессе, гарантируя, что ошибки или сбои в коде пользователя не влияют на её работу или другие части системы. Поддерживается доступ к функциям языка, его стандартной библиотеке, а также историю команд.

Livebook является облачной средой программирования для Elixir, использующая компонент редактора CodeMirror 6 с поддержкой автодополнения, встроенной документации и форматирования кода. Платформа разработана в большей части для обработки больших объемов данных и выполнения сложных вычислений, однако ее также можно использовать в качестве REPL. Livebook представляет собой блокноты с поддержкой ячеек кода, в которых код Elixir оценивается по требованию и хранится в формате `.livemd`.

1.3 Реализация синтаксического анализатора

Одним из возможных вариантов синтаксической структуры, получаемой в ходе проведения синтаксического анализа является абстрактное синтаксическое дерево, которое позволяет проверять правильность программных конструкций и корректность использования их элементов в ходе проведения семантического анализа.

В реализации REPL такая структура данных является основополагающей для интерпретации кода пользователя, в ходе которой выполнение команд производится путём рекурсивного обхода АСД. Однако, большинство других программ на Elixir используют процесс компиляции, при которой АСД переводится в формат инструкций байт-кода BEAM, которые затем сохраняются в виде `.beam`-файлов.

Листовыми (конечными) элементами являются выражения, при приведение которых к АСД получается само выражение. Любой другой вид выражения

(нелистовой узел) состоит из кортежа из трех элементов, который может представлять переменные или вызовы. Первый элемент определяет собой выражение, описывающее тип узла, второй — список с методанными об узле, такими как номер строки, файл, номер столбца и т.д., третий — список дочерних узлов.

```
quote do
    1 + 2 * 4
end
=> {:+, [], [1, {:*, [], [2, 4]}]}
```

Вызовы — это основной строительный блок, с помощью которого другие узлы объединяются для построения правильного АСД. Вызовы бывают двух видов: неквалифицированные — не предварены вызовом модуля (в этом случае интерпретатор ищет функцию в текущем контексте выполнения: в текущем или импортированных модулях) и квалифицированные — вызовы функций, которые предварены именем модуля.

В случае неквалифицированных вызовов первым элементом кортежа выстукает атом, вторым — метаданные, третьим — список аргументов. Структуры данных, такие как словари, кортежи и битовые строки, представляются вызовом, где типом является имя атома соответствующего конструктора специальной формы, а аргументами являются их элементы.

Квалифицированные вызовы используют оператор `.` в качестве первого элемента кортежа АСД. Если метанные содержат `no_parens: true`, это означает, что каждое использование такого оператора будет представлено как вызов, в котором круглые скобки могут быть опущены. Оператор `.` может быть произвольно вложен:

Любой тип узла состоит из комбинации узлов. Для реализации анализа и оценки кода пользователя в REPL происходит сопоставление со следующими основными образцами конструкций:

1. `{:defmodule, meta, [aliases, do_block]}` — для определения модуля;
2. `{:def, _, [{local, _, _} | _]}` — для определения публичной функции модуля;
3. `{:def, _, [{:when, _, [{local, _, _} | _]} | _]}` — для определения публичной функции модуля с добавочным условием;

4. `{:defp, _, [{local, _, _} | _]}` — для определения приватной функции модуля;
5. `{:defp, _, [{:when, _, [{local, _, _} | _]} | _]}` — для определения приватной функции модуля с добавочным условием;
6. `{fun, _, args}` — для вызова локальных и импортированных функций;
7. `{{:., meta, [module, fun]}, info, args}` — для вызова именованной функции;
8. `{{:., _, lambda}, _, args}` — для вызова анонимной функции;
9. `{node, _, children}` — базовое определение узла;
10. `{leaf, _, nil}` — определение листовых узлов, например, переменных.

1.4 Технологический стек

Для реализации интерпретаторов обычно используется тот же язык программирования, для которого он и создается, гарантируя, что синтаксис и семантика будут полностью соответствовать языку. В связи с этим основным языком проекта является Elixir.

Для создания веб-модуля приложения используется спецификация Plug Elixir, которая является легковесной библиотекой, предоставляющей базовые компоненты для обработки HTTP-запросов.

Для создания индивидуальной сессии пользователя, которая будет хранить сессионную информацию, а также для ее управления используется модуль GenServer, который используется в паре с библиотекой Plug и реализует поведение общего сервера в рамках OTP — абстракции для построения параллельных и распределенных систем.

Для реализации пользовательского интерфейса использовался EEx — модуль языка программирования Elixir, который предоставляет возможности рендеринга динамического контента для веб-приложения черех HTML-шаблоны.

Создание интерактивной консоли на веб-странице реализовано с помощью библиотеки JavaScript jQuery Console, которая предоставляет инструменты для отображения результатов выполнения команд, их истории, предупреждений и ошибок выполнения. Для форматирования элементов веб-страницы используется язык таблиц стилей CSS. Отправка запросов к серверной части приложения для выполнения пользовательского кода осуществляется с помощью асинхронной технологии AJAX, которая позволяет обновлять только определенные части веб-страницы, чтобы сделать пользовательский опыт более динамичным.

2 Разработка интерактивной online-среды программирования

Для проектирования веб-приложения используется клиент-серверная модель, в которой клиенты не несут нагрузки по выполнению кода, что снижает требования к их вычислительной мощности.

Серверная логика состоит из API сервера, который отвечает за обработку запросов, содержащих код от клиента, их интерпретацию и отправление на выполнение, а также за формирование ответов, которые могут включать результаты выполнения, сообщения об ошибках и другие метаданные, и изолированной среды выполнения пользовательского кода. Одним из критических аспектов REPL является безопасность выполнения кода. Клиент-серверная модель позволяет контролировать выполнение кода в изолированной среде на сервере, предотвращая потенциальные атаки.

Клиентская часть представляет собой пользовательский интерфейс и механизм обработки ответов от сервера.

2.1 Синтаксический анализ пользовательского кода

Модуль `ElixirRepl.Sandbox.Analysis` предназначен для проверки кода, который пользователь хочет выполнить в изолированной среде. Основная его задача — анализ абстрактного синтаксического дерева на предмет запрещённых или потенциально опасных операций перед тем, как код будет выполнен.

Атрибут модуля `@allowed_locals` определяется набор из 118 разрешённых локальных функций, которые можно безопасно вызывать в контексте изолированной среды выполнения, исключая функции, которые могут привести к неправомерному доступу к системным ресурсам или изменению состояния системы вне контекста среды.

Следующий атрибут модуля `@allowed_named_functions` определяет словарь, указывающий на модули и их функции, разрешенные для вызова. Также для избежания исчерпания ресурсов системы предусмотрены следующие атрибуты модуля:

1. `@collection_length = 100` — для предотвращения создания чрезмерно больших списков;
2. `@binary_size_value = 128` — ограничение размера бинарных объектов;
3. `@binary_unit_value = 16` — ограничение единицы измерения бинарных объектов.

Проверка АСД начинается с функции `is_safe?/2`, определенная с использованием спецификации типов, указывающей, что на вход принимается переменная, представляющая АСД, и переменная, предоставляющая среду времени выполнения.

Прежде чем проверять, безопасен ли модуль, нужно проверить, не пытается ли пользователь переопределить зарезервированный или уже существующий в рамках данного окружения модуль. Это осуществляется с помощью приватной функции `check_module/2`.

Далее необходимо собрать все публичные и приватные функции, определенные модулем в параметр `local_functions`, что позволит пользователю вызывать функции, определенные модулем, после их проверки. Данных механизм осуществляется с помощью функции `module_locals/1`.

Проверка именованных функций осуществляется по следующему алгоритму:

1. преобразуется имя текущего модуля путем добавления к нему нового пространства имен с последующим расширением в контексте среды выполнения;
2. определяется количество аргументов функции;
3. осуществляется проверка, является ли рассматриваемый модуль частью списка модулей, разрешенных в данной среде;
4. если проверка прошла успешно, разрешается вызов всех функций в данном модуле, иначе вызывается ошибка;
5. разрешенные функции нормализуются в кортеж, в котором первое значение представляет имя функции, второе — ее арность;
6. проверяется, соответствует ли арность вызываемой функции одному из разрешённых вариантов. Если нет, вызывается исключение;
7. проверяется безопасность всех аргументов функции путем рекурсивного вызова для них функции `is_safe?/3`;

Для проверки неименованной функций осуществляется анализ тела функции на предмет использования только безопасных операций и элементов, а также рекурсивная проверка аргументов функции.

Для проверки диапазона коллекции, которую хочет создать пользователь, в теле модифицированной функции `is_safe?/3` вычисляется абсолютная разниц между начальным и конечным значениями диапазона, а также осуществля-

ется проверка значений на соответствие целочисленным значениям.

Чтобы типовые аннотации и операции с битовыми строками в коде были выполнены также корректно и безопасно, используется другая модификация функции `is_safe?/3`, в которой происходит проверка каждого элемента битовой строки на размер и единиц с помощью функции `check_bitstring/1`.

Далее рассматриваются типовые случае, возникающие в ходе прохода анализатора по АСД:

1. обработка списка узлов: происходит рекурсивная проверка каждого узла;
2. обработка узлов с детьми: происходит рекурсивная проверка родительского и дочернего узел;
3. обработка листовых узел, которые представляются атомами: функция возвращает исходное АСД, так как такие элементы не требуют дополнительной обработки безопасности;
4. другие случаи: по умолчанию, если узел не подпадает под специальные категории, он возвращается без изменений. Это предполагает, что узел не требует специальной обработки безопасности.

2.2 Обработка предупреждений

Модуль `ElixirRepl.Warning` представляет абстракцию, которая позволяет процессу контролируемой среды выполнения пользовательского кода извлекать и хранить предупреждающие сообщения от компилятора Elixir. Для записи сообщений о ходе выполнения программы используется функционал встроенного модуля `Logger`.

Для стандартизации формата предупреждающих сообщений с целью упрощения их анализа и обработки в рамках системы был определен новый тип `warning()`, который представляет собой кортеж из трех переменных:

1. `non_neg_integer()` — неотрицательное целое число для определения номера строки, на которой возникло предупреждение;
2. `charlist()` — список символов для хранения имени файла, в котором возникло предупреждение;
3. `charlist()` — список символов для сообщения предупреждения.

Функция `start_link` используется для создания нового процесса, который выполняется параллельно и обрабатывает предупреждения. Гарантируется, что если процесс-агент завершится с ошибкой, он не повлияет на другие работающие процессы.

Приватная функция `warnings/1` отвечает за приём и обработку различных типов сообщений, таких как обработка предупреждений, сброс предупреждений, ответ на запрос о доступности модуля, а также любых других неизвестных сообщений, которые логируются для отладки на уровне `debug`.

Для извлечения сохраненных предупреждений из процесса-агента используется функция `flush`, которая в качества аргумента принимает идентификатор процесса PID. В процессе работы функции `make_ref()` создаёт уникальную ссылку, которая используется для корреляции запросов и ответов между процессами.

2.3 Изолированная среда выполнения

Изолированная среда выполнения пользовательского кода реализована с помощью библиотеки GenServer для создания синхронных вызовов к серверу веб-приложения. Для создания GenServer определен модуль `ElixirRepl.Sandbox` с реализацией функций обратного вызова.

Автоматически при запуске процесса вызывается функция `init/1`, которая отвечает за инициализацию начального состояния сервера, настройку необходимых ресурсов и установление начальных параметров.

Функция `handle_call/3` обрабатывает синхронные вызовы, поступающие в сервер, работая по следующему алгоритму:

1. новый входной код пользователя объединяется с уже кешированным кодом, что позволяет обрабатывать многострочные входные данные;
2. происходит очистка всех предупреждений, оставшихся от предыдущих выполнений;
3. пользовательский код преобразуется в АСД и оценивается в контексте текущего состояния;
4. если при выполнении кода возникает исключение или ошибка, форматируется сообщение об, обнуляется кэш и увеличивается счётчик операций;
5. получаются все предупреждения, возникшие во время попытки выполнения кода;
6. собирается итоговый ответ, который включает результат выполнения, предупреждения и обновленный счетчик;
7. в качестве выходных данных получается кортеж состоящий из данных ответа и обновленного состояния сервера.

Непосредственная обработка кода пользователя происходит с помощью

функции `eval/3`, представленной в трех сценариях: обработка корректного кода, обработка неполного ввода и обработка некорректного ввода.

В случае успешного преобразования кода в АСД происходит его оценка на безопасность используемых конструкций, выполнение кода с применением текущих привязок и окружения, захват результата выполнения для обеспечения гарантии, что вывод не потерялся и создание нового состояния с обновлением привязок, сбрасыванием кеша, увеличением счетчика операции и сохранением текущих предупреждений.

В случае неполного ввода происходит обновление кешируемого кода новой строкой и возвращение обновленного состояния с меткой `:incomplete`.

В случае некорректного ввода пользователя, когда ввод не может быть правильно интерпретирован из-за синтаксических ошибок или других проблем с анализом, генерируется сообщение об ошибке, указывающее на место (строку) и тип ошибки.

В случае, если выполнение кода превышает допустимое время выполнения, происходит принудительное завершение процесса GenServer. Это гарантирует, что зависший или нереагирующий процесс будет остановлен, предотвращая возможные утечки ресурсов или блокировки.

Функция `handle_info/2` предназначена для обработки входящих сообщений, которые не связаны с вызовами или ответами, управляемыми через `handle_call/3`, обрабатывая тайм-аут бездействия для корректного освобождения ресурсов, связанных с неактивными или неиспользуемыми модулями.

2.4 Пользовательский интерфейс приложения

Для реализации клиентской части REPL используется принцип построения веб-приложений SPA (Single Page Application), в котором все необходимое для работы пользователя загружается в браузер один раз, и при последующих действиях пользователя страница не перезагружается полностью. Это достигается за счёт асинхронной загрузки данных в фоне, осуществляемой с помощью технологии JavaScript AJAX, и динамической перерисовки только тех участков интерфейса, которые должны измениться, что является удобной функциональностью в процессе взаимодействия пользователя с консолью.

Основными элементами экранной формы веб-приложения являются:

1. форма приветствия для ориентации новых пользователь на странице;

2. интерактивная консоль, позволяющая пользователям написать код и сразу увидеть результат его выполнения;
3. кнопка перехода к сайту Elixir School, который представляет собой открытый и бесплатный ресурс для изучения языка программирования Elixir.

Для реализации интерактивной консоль пользователя использована библиотека jQuery Console, которая представлена в файле `jquery.console.min.js`, содержащем ее минифицированную версию (файл представлен в открытом доступе). Основными элементами библиотеки являются обработка нажатия клавиш, ведение истории команд, функция автодополнения.

2.5 Клиент-серверное взаимодействие

Для реализации маршрутизатора веб-сервера приложения был создан модуль `ElixirRepl.Router`. Для управлениями сессиями пользователя с помощью механизма cookies используется модуль `Plug.Session`, который позволяет сохранять и извлекать данные сессии через зашифрованные и подписанные cookies.

`Plug.Parsers` используется для анализа тел запросов. Он настроен на обработку данных, закодированных в urlencoded-формате, а также пропуск запросов с типом содержимого `application/json`.

Для обеспечения доступа к статическому контенту без необходимости обработки каждого запроса через полноценные контроллеры с целью повышения производительности при обработке запросов используется модуль `Plug.Static`.

Функция `handle_errors` обрабатывает различные типы ошибок, которые могут возникнуть при работе сервера с HTTPS-запросами.

Обработчик `get` используется для обслуживания главной страницы вебсайта. Для оценка кода пользователя на стороне сервера в изолированной среде выполнения создан обработчик `post`.

Код, который должен быть выполнен, извлекается из параметров запроса. Если параметр отсутствует, функция вызывает исключение. Принятый код передается в изолированную среду выполнения, связанную с PID, для оценки.

Информирование клиента о том, что запрошенный ресурс не найден происходит с помощью ответа сервера `404 (Not Found)`.

Обработка команд пользователя, отправка их на сервер для выполнения и вывод результатов в консоль осуществляется с помощью скрипта на языке программирования JavaScript.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы были изучены основные принципы построения языка программирования Elixir и проведения синтаксического анализа пользовательского кода для обнаружения уязвимых, небезопасных и запрещенных к использованию конструкций. Было реализовано веб-приложение для интерактивной среды программирования Elixir REPL, которое включает три основных компонента:

- пользовательский интерфейс, содержащий консоль для работы с Elixir с возможностью просмотра результатов оценки кода и истории команд, многострочного ввода и автодополнения;
- сервер, отвечающий за обработку и оценку кода в изолированной среде выполнения для каждого пользователя, обнаружение и формирование предупреждений и ошибок выполнения;
- клиент-северное взаимодействие для отправки запросов на оценку кода со стороны клиента и обработки ответов от сервера.

Приложение соответствует всем требованиями интерактивной среды программирования и предоставляет пользователям возможность проведения экспериментов, тестирования и быстрого освоения языка Elixir без необходимости локальной установки дополнительного программного обеспечения.