МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«ПЕРМСКИЙ ГОСУДАРСТВЕННЫЙ ГУМАНИТАРНО-ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

Выпускная квалификационная работа

УЧЕБНАЯ СИСТЕМА РАСПРЕДЕЛЕНИЯ ВЫЧИСЛЕНИЙ МЕЖДУ НЕСКОЛЬКИМИ КЛИЕНТАМИ (ПРОГРАММА-КЛИЕНТ)

направление 09.03.02 Информационные системы и технологии,
профиль «Информационные технологии в образовании»

Работу выполнил:
студент 843 группы
Смирнов Роман Сергеевич
_____
(подпись)

Научный руководитель:
доцент кафедры МД и ИТО
Еремин Евгений Александрович
_____
(подпись)

«Допущен к защите в ГЭК»
Зав. кафедрой МД и ИТО:
_____ / Е.В. Оспенникова
(подпись)
« » 2016 г.

Пермь
2016

Оглавление

Введение

Языки высокого уровня стали основным средством разработки программ. С одной стороны, компьютеры умеют понимать только коды машинных команд. С другой стороны, разработчики не имеют возможности создавать прикладные и системные программы на уровне машинных кодов – слишком велик процент ошибок, а также трудоёмкость такой работы. Поэтому давно возникла потребность в появлении «переводчиков» с различных языков программирования на язык машинных кодов. Такими «переводчиками» стали трансляторы.

Большую часть задач, решаемых с помощью программирования, составляют задачи, в которых широко применяются методы вычислительной математики, а в них входят арифметические и логические выражения. Поэтому трансляцией выражений занимались очень многие исследователи и разработчики трансляторов.

Цель транслятора заключается в том, чтобы преобразовать выражение, записанное в привычной человеку форме, в последовательность арифметических (или логических) команд, которые способно выполнить вычислительное устройство.

Важной характеристикой транслятора является создание эффективной (т.е. быстро работающей) программы вычислений. В последнее время большое внимание уделяется при этом распределению вычислительных команд между несколькими арифметическими устройствами (или несколькими процессорами).

Изучением проблемы трансляции арифметических выражений первым занялся швейцарский математик Хайнц Рутисхаузер. В 1952 году была опубликована его работа, в которой он раскрыл свой метод. В эти же годы этой проблемой заинтересовался российский математик

Л. Л. Ляпунов. Свои идеи Ляпунов положил в основу курса по программированию, читавшегося им в Московском Университете.

Целью работы является создание программы-вычислителя арифметических выражений, с помощью которой можно оценить эффективность работы разных алгоритмов, использующих параллельные вычисления отдельных частей этих выражений.

Предполагается, что программа будет работать совместно с программой разбора арифметических выражений, которая разработана в рамках выпускной квалификационной работы студента Шалагинова Петра Николаевича. Вместе обе программы образуют клиент-серверную систему распределения арифметических выражений между несколькими клиентами.

Объект: применение языка программирования Java для разработки программ

Предмет: разработка программ на языке Java

Цель работы: разработать клиентскую часть учебной системы распределения вычислений на языке Java

Задачи:

1. Изучить основы трансляции арифметических выражений
2. Рассмотреть существующие алгоритмы трансляции арифметических выражений
3. Выбрать оптимальный алгоритм трансляции для разработки программы
4. Изучить технологию клиент-сервер и методы ее реализации на языке Java
5. Разработать программу-клиент на языке Java

ГЛАВА 1. ОСНОВЫ ТРАНСЛЯЦИИ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

1.1. Что такое трансляция

Все идеи трансляции базируются на простом, но важном соображении: так как программа хранима, как и данные, то она может быть аргументом или результатом работы некоторых алгоритмов, которые реализуют те или иные аспекты конструирования программ [10].

Начальные идеи по трансляции были выдвинуты в 1952 году, на самом раннем этапе развития программирования. В этом году Рутисхаузер опубликовал работу [8], в которой изложил идею о том, что по естественной записи арифметического выражения можно построить эквивалентный ей фрагмент машинного кода. Соответствующий алгоритм был сформулирован. Эта работа дала толчок к возникновению трансляторов, в своем входном языке допускающих естественную запись выражений.

Трансляция — преобразование программы, представленной на одном из языков программирования, в программу на другом языке. При трансляции выполняется перевод программы, понятной человеку, на язык, понятный компьютеру. Выполняется специальными программными средствами – трансляторами.

Транслятор - это программа, переводящая текст на языке программирования в форму, пригодную для исполнения (на язык исполнительного устройства). Такой формой обычно являются машинные команды, которые могут непосредственно исполняться компьютером. Совокупность машинных команд данного компьютера образует его систему команд или машинный язык. Программу, которую обрабатывает транслятор, называют исходной программой, а язык, на котором

записывается исходная программа — входным языком этого транслятора [4].

Транслятор связан в общем случае с тремя языками. Во-первых, это входной язык, с которого выполняется перевод. Во-вторых, целевой (объектный) язык, на который выполняется перевод. И, наконец, третий язык — это язык, на котором написан сам транслятор, — инструментальный язык.

Первые трансляторы, появившиеся в 1950-е годы, программировались на машинном языке или на языке низкого уровня — языке ассемблера. Сейчас для разработки трансляторов используются языки высокого уровня [9].

1.2. Задача разбора арифметического выражения

В работе любого транслятора присутствует фаза разбора входной программы, представленной, как привило, в виде текста. Задача транслятора – преобразовать одно представление программы в эквивалентное другое представление.

Процесс разбора арифметического выражения, часто неформально называемый парсингом (parse), как правило, состоит из двух этапов: лексического анализа и синтаксического анализа.

На этапе лексического анализа входная строка из последовательности символов переводится в последовательность лексем. *Лексемами* называются отдельные части арифметического выражения, такие как имена переменных, числа, знаки операций и скобки. Это своего рода атомы, которые считаются неделимыми и рассматриваются как нечто целое.

Например, в арифметическом выражении

15*(A+30)

можно выделить следующие лексемы в порядке их появления: "15", "*", "(", "A", "+", "30" и ")". Каждая лексема относится к тому или иному классу: "A" относится к классу переменных, "15" и "30" - к классу чисел, "*" и "+" - к классу операций, а "(" и ")" - к классу разделителей.

Суть лексического анализа заключается в выделении из последовательности «букв» входной программы последовательности «слов», неделимых с точки зрения последующего анализа.

На этапе синтаксического анализа проверяется, принадлежит ли входная строка языку, с которым ее пытаются сопоставить. Однако на практике одного такого ответа (принадлежит или нет) недостаточно и, как правило, параллельно с синтаксическим разбором происходят какие-либо действия, направленные на дальнейшее преобразование входной строки.

1.3. Существующие алгоритмы трансляции арифметических выражений

Первые процедурно-ориентированные языки программирования высокого уровня предназначались для решения инженерных и научно-технических задач, в которых широко применяются методы вычислительной математики. Значительную часть программ решения таких задач составляют арифметические и логические выражения. Поэтому трансляцией выражений занимались очень многие исследователи и разработчики трансляторов.

Исторически первым решением задачи трансляции арифметических выражений был метод Рутисхаузера.

Метод Рутисхаузера

В 1952 году швейцарский математик Хайнц Рутисхаузер опубликовал работу, в которой предложил алгоритм трансляции арифметических выражений. Особенностью метода является

предположение о полной скобочной структуре выражения. Под полной скобочной записью выражения понимается запись, в которой порядок действий задается расстановкой скобок. Неявное старшинство операций при этом не учитывается.

Так, выражение D = A+B*C должно быть записано в виде D = (A+(B*C)). При обработке данного выражения алгоритм присваивает каждому символу из строки номер уровня по следующему правилу:

- если это открывающаяся скобка или переменная, то значение увеличивается на 1;
- если знак операции или закрывающаяся скобка, то уменьшается на 1

Для выражения (A+(B*C)) присваивание значений уровня будет происходить следующим образом:

Таблица 1. Присваивание значений уровня

Номер символа	1	2	3	4	5	6	7	8	9
Символы строки	(	A	+	B	(	*	C	)	)
Номера уровней	1	2	1	2	3	2	3	2	1

Алгоритм складывается из следующих шагов.

1. Выполнить расстановку уровней.
2. Выполнить поиск элементов строки с максимальным значением уровня.
3. Выделить тройку — два операнда с максимальным значением уровня и операцию, которая заключена между ними.
4. Результат вычисления тройки обозначить вспомогательной переменной.
5. Из исходной строки удалить выделенную тройку вместе с её скобками, а на её место поместить вспомогательную переменную, обозначающую результат, со значением уровня на единицу меньше, чем у выделенной тройки.

6. Выполнять шаги 2 — 5 до тех пор, пока во входной строке не останется одна переменная, обозначающая общий результат выражения [9].

Недостаток метода – требование полной скобочной структуры.

Классическим считается метод трансляции выражений, основанный на использовании промежуточной обратной польской записи, названной так в честь польского математика Яна Лукашевича, который впервые использовал эту форму представления выражений в математической логике. Обратную польскую запись также называют польской инверсной записью (ПОЛИЗ) [3].

Рассмотрим сущность обратной польской записи на примере. Простое арифметическое выражение с вещественными переменными:

A + B * C - D / (A + B)

можно представить в виде обратной польской записи:

A B C * + D A B + / -.

Эту бесскобочную запись называют также постфиксной записью, потому что знак каждой операции записан после соответствующих операндов. В обратной польской записи операнды располагаются в том же порядке, что в исходном выражении, а знаки операций при просмотре записи слева направо встречаются в том порядке, в котором нужно выполнять соответствующие действия. Отсюда вытекает основное преимущество обратной польской записи перед обычной записью выражений со скобками:

1. Действия, описываемые в ПОЛИЗ, можно выполнять в процессе ее однократного безвозвратного просмотра слева направо, т.к. операнды в ПОЛИЗ всегда предшествуют знаку операции и на момент извлечения операции оба операнда уже готовы.

2. Операнды ПОЛИЗ расположены в том же порядке, что и в исходном (инфиксном) выражении. Перевод выражения в ПОЛИЗ сводится только к изменению порядка следования знаков операций.

Перевод арифметического выражения в ПОЛИЗ

Существует несколько алгоритмов для превращения инфиксных формул в обратную польскую запись. Рассмотрим переработанный алгоритм на основе идеи Э. Дейстра, который Эндрю Таненбаум предложил в своей книге [12].

Предположим, что формула состоит из переменных, двухоперандных операторов «+», «-», «*», «/», а также левой и правой скобок.

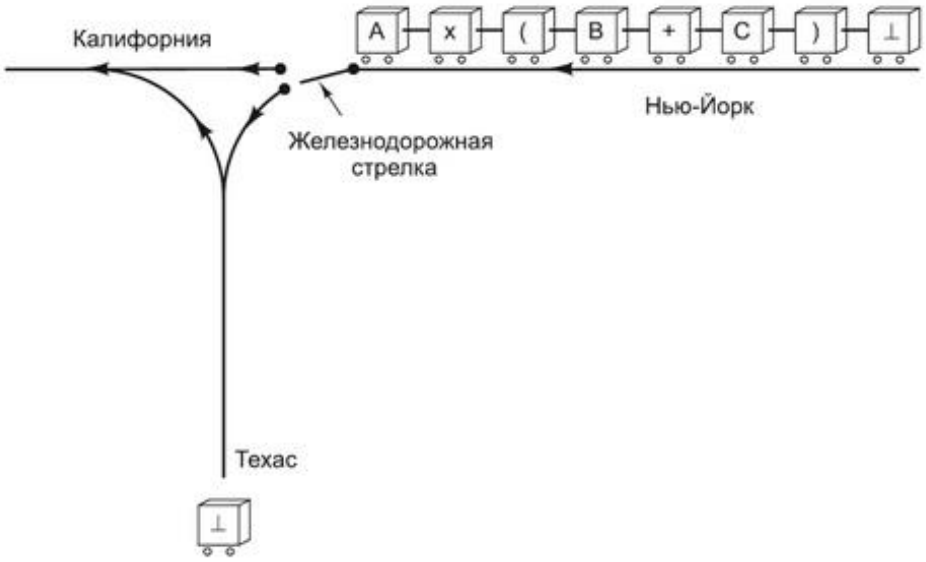

Рис. 1. Схема перевода выражения в ПОЛИЗ

На рисунке 1 схематично показана железная дорога из Нью-Йорка в Калифорнию с ответвлением, ведущим в Техас. Каждый символ формулы представлен одним вагоном. Поезд движется справа налево. Перед стрелкой каждый вагон должен останавливаться и узнавать, должен ли он двигаться прямо в Калифорнию, или ему нужно будет по пути заехать в

Техас. Вагоны, содержащие переменные, всегда направляются в Калифорнию и никогда не едут в Техас. Вагоны, содержащие все прочие символы, должны перед прохождением стрелки узнавать о содержимом ближайшего вагона, отправившегося в Техас.

В таблице 2 показана зависимость ситуации от того, какой вагон отправился в Техас последним и какой вагон находится у стрелки. Первый вагон (помеченный символом ⊥) всегда отправляется в Техас.

Таблица 2.

	⊥	**+**	**P**	*****	**/**	**(**	**)**
⊥	4	1	1	1	1	1	5
+	2	2	2	1	1	1	2
P	2	2	2	1	1	1	2
*****	2	2	2	2	2	1	2
/	2	2	2	2	2	1	2
(	5	1	1	1	1	1	3

Числа соответствуют следующим ситуациям:

1. Вагон на стрелке отправляется в Техас

2. Последний вагон, направившийся в Техас, разворачивается и направляется в Калифорнию

3. Вагон, находящийся на стрелке, и последний вагон, отправившийся в Техас, угоняются и исчезают

4. Остановка. Символы, находящиеся на Калифорнийской ветке, представляют собой формулу в обратной польской записи, если читать слева направо

5. Остановка. Произошла ошибка. Изначальная формула была некорректно сбалансирована

После каждого действия производится новое сравнение вагона, находящегося у стрелки, и вагона, который на данный момент последним ушёл на Техас. Этот процесс продолжается до тех пор, пока не будет достигнут шаг 4. Отметим, что линия на Техас используется как стек, где отправка вагона в Техас – это помещение элемента в стек, а разворот отправленного в Техас вагона в сторону Калифорнии – это выталкивание элемента из стека.

Порядок следования переменных в инфиксной и постфиксной записи одинаков. Однако порядок следования операторов не всегда один и тот же. В обратной польской записи операторы появляются в том порядке, в котором они будут выполняться [12].

Метод ПОЛИЗ

Выражения в обратной польской записи вычисляются с помощью стека. *Стеком* называется упорядоченный набор элементов, в котором размещение новых и удаление существующих происходит с одного конца, называемого вершиной. Принцип работы стека сравнивают со стопкой листов бумаги: чтобы взять второй сверху, нужно сначала снять верхний. Разберем процесс вычисления выражения в обратной польской записи [1].

Выражение состоит из N символов, каждый из которых является либо операндом, либо оператором. При чтении ПОЛИЗ слева направо операнды помещаются в стек, а операции применяются к верхним элементам стека. Результат выполнения операции возвращается в стек, заменяя собою операнды. По исчерпании всех символов выражения результат его вычисления находится на вершине стека [10].

В качестве примера рассмотрим вычисление следующего выражения:

(8+2*5)/(1+3*2-4).

Соответствующая формула в обратной польской записи выглядит так:

8 2 5 * + 1 3 2 * + 4 - /.

Порядок вычисления данного выражения покажем в виде следующей таблицы:

Таблица 3. Порядок вычисления выражения в ПОЛИЗ

Шаг	Символ	Оставшаяся цепочка	Стек
1	8	2 5 * + 1 3 2 * + 4 - /	8
2	2	5 * + 1 3 2 * + 4 - /	8, 2
3	5	* + 1 3 2 * + 4 - /	8, 2, 5
4	*	+ 1 3 2 * + 4 - /	8, 10
5	+	1 3 2 * + 4 - /	18
6	1	3 2 * + 4 - /	18, 1
7	3	2 * + 4 - /	18, 1, 3
8	2	* + 4 - /	18, 1, 3, 2
9	*	+ 4 - /	18, 1, 6
10	+	4 - /	18, 7
11	4	- /	18, 7, 4
12	-	/	18, 3
13	/		6

Число на вершине стека – это правый операнд. Это очень важно для операций деления, вычитания и возведения в степень, поскольку порядок следования операндов в данном случае имеет значение (в отличие от операций сложения и умножения). Другими словами, операция деления действует следующим образом: сначала в стек помещается числитель, потом знаменатель, и тогда операция даёт правильный результат.

Метод ПОЛИЗ часто применяется при вычислении арифметических выражений. Поэтому именно он и был выбран для разработки программы-вычислителя арифметических выражений.

ГЛАВА 2. ТЕХНОЛОГИЯ КЛИЕНТ-СЕРВЕР

2.1. Основные идеи технологии «клиент-сервер»

Самой простой и распространенной моделью распределенной обработки данных является модель типа «клиент-сервер». В этой модели программа разделяется на две части: одна часть называется сервером, а другая – клиентом. Сервер имеет прямой доступ к некоторым аппаратным и программным ресурсам, которые желает использовать клиент [13].

Появление клиент-серверных систем тесным образом связано с парадигмой открытых систем, активно развивающихся с начала 1980-х годов. Программисты пришли к выводу о необходимости распределения задач между элементами системы, в данном случае между двумя типами процессов, которые могут выполняться на различных компьютерах, объединенных в вычислительную сеть. Процесс сервера отвечает за предоставление каких-либо услуг клиентскому процессу. Клиентский процесс запрашивает у сервера определенные услуги и ресурсы.

Клиент - объект, запрашивающий информацию по сети. Как правило, это персональный компьютер или рабочая станция, запрашивающая информацию у сервера.

Клиентские машины, как правило, представляют собой однопользовательские персональные компьютеры или рабочие станции, предоставляющие конечным пользователям дружественный интерфейс. Клиентская станция обычно имеет наиболее удобный графический интерфейс пользователя, предполагающий наличие окон и мыши.

Сервер - объект, обслуживающий запросы клиентов на получение ресурсов определенного вида. Как правило, это высокопроизводительная рабочая станция.

Каждый сервер в окружении клиент-сервер предоставляет клиентам набор услуг. Наиболее распространенным типом сервера в архитектуре клиент-сервер является сервер баз данных. Высокопроизводительный сервер обеспечивает коллективный доступ нескольких клиентов к одной и той же базе данных.

Клиент и сервер могут находиться как на одном компьютере, так и на разных компьютерах в сети. В большинстве случаев сервер и клиент располагаются на разных компьютерах. Также может возникать такая ситуация, когда некоторый программный блок будет одновременно выполнять функции сервера по отношению к одному блоку и клиента по отношению к другому.

Существует несколько подходов к распределению обязанностей между сервером и клиентом. В простейшем случае работает модель «тонкий» клиент (Рис. 2). Это вариант, когда клиентское ПО максимально упрощено и представляет собой только интерфейсную часть, состоящую из форм ввода и просмотра данных. «Тонкий» клиент фактически отвечает только за презентацию данных, поэтому можно сказать, что он не умеет думать, и вся программная логика сосредоточена на стороне сервера.

Рис. 2. «Тонкий» клиент

Есть и обратное решение – «толстый» клиент (Рис. 3). В противовес своему «тонкому» собрату, «толстый» клиент старается максимально облегчить работу сервера, например, реализует дополнительные правила, сортирует полученные данные на клиентской стороне, выполняет вспомогательные расчеты и т.п. Проектирование полнофункционального «толстого» клиентского приложения значительно сложнее, чем разработка его упрощенного собрата. Но, в конечном итоге, мы получаем прибавку в производительности, а это весьма важно в больших многопользовательских системах [5].

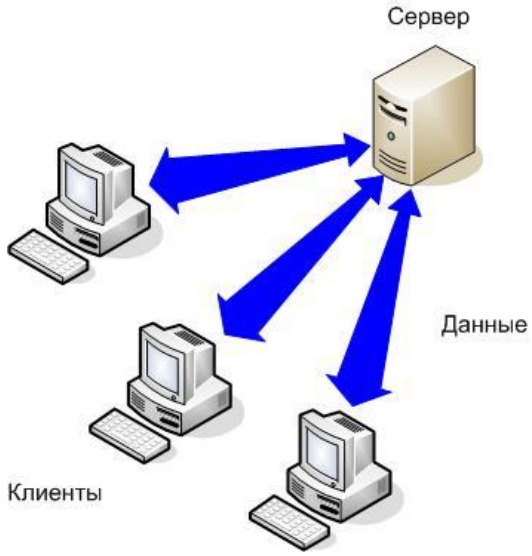

Рис.3. «Толстый» клиент

Несмотря на то, что клиент и сервер представляют собой отдельные программы, выполняющиеся на разных компьютерах, вместе они составляют единое приложение. Разделение ПО на части клиента и сервера и есть основной метод распределенного программирования.

Отличия в функционировании приложений в рамках технологии «клиент-сервер» отвечают на четыре вопроса:

- какие виды программного обеспечения используются в логических компонентах;

- какие механизмы программного обеспечения используются для реализации функций логических компонентов;
- как логические компоненты распределяются компьютерами в сети;
- какие способы используются для связи компонентов друг с другом.

Существуют три подхода, каждый из которых реализован в определённой модели технологии «Клиент-сервер»:

- модель доступа к удаленным данным (Remote Date Access - RDA);
- модель сервера базы данных (DateBase Server - DBS);
- модель сервера приложений (Application Server - AS).

Рассмотрим функции и характеристики этих моделей технологии «клиент-сервер» [4].

Модель доступа к удаленным данным (RDA) – сетевая архитектура технологии «клиент – сервер», при которой коды компонента представления и прикладного компонента совмещены и выполняются на компьютере – клиенте (Рис. 4). Доступ к информационным ресурсам осуществляется при помощи непроцедурного языка или вызовами функций специальной библиотеки, если есть определённый интерфейс прикладного программирования – API.

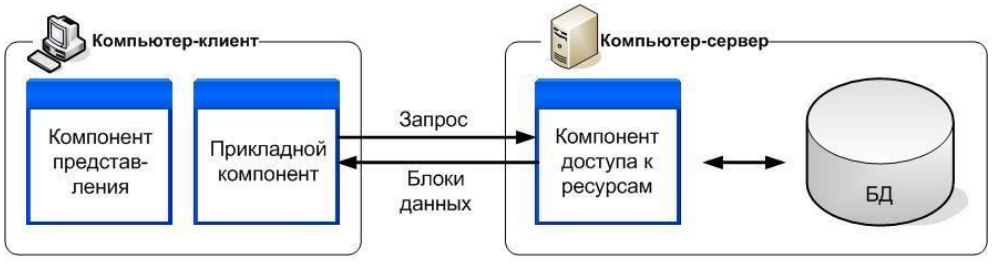

Рис.4. Модель доступа к удаленным данным

Запросы к информационным ресурсам направляются по сети к удаленному компьютеру, который анализирует их, возвращая клиенту блоки данных.

Главным преимуществом RDA-модели является широкий выбор инструментальных средств разработки приложений, позволяющих быстро создать приложения, работающие с SQL-ориентированными СУБД. Обычно инструментальные средства поддерживают графический интерфейс пользователя с операционной системой, а также средства автоматической генерации кода, в которых смешаны прикладные функции и функции представления.

Несмотря на широкое распространение, RDA-модель вытесняется более технологичной DBS-моделью.

Модель сервера баз данных (DBS) – сетевая архитектура технологии «клиент-сервер», основу которой составляет механизм хранимых процедур, реализующий прикладные функции (Рис. 5). В DBS-модели понятие информационного ресурса сведено до базы данных из-за того же механизма хранимых процедур, который осуществлён в СУБД.

В DBS-модели приложение является распределенным. Компонент представления выполняется на компьютере-клиенте, в то время как прикладной компонент представлен как набор хранимых процедур и функционирует на компьютере-сервере.

Рис.5. Модель сервера баз данных

Преимущества DBS-модели перед RDA-моделью очевидны: это и возможность централизованного администрирования различных функций, и снижение трафика сети из-за того, что вместо SQL-запросов по сети передаются вызовы хранимых процедур, и возможность распределения процедуры между несколькими приложениями.

Модель сервера приложений (AS) - сетевая архитектура технологии «клиент-сервер», представляющая собой процесс, выполняемый на компьютере-клиенте и отвечающий за интерфейс с пользователем (ввод и отображение данных) (Рис. 6). Главным элементом этой модели является прикладной компонент, называющийся сервером приложения, работающий на удаленном компьютере. Сервер приложений реализован как группа прикладных функций, представленных в виде сервисов. Каждый сервис предоставляет некоторые услуги всем программам, которые могут ими воспользоваться.

Серверов приложений может быть несколько, и каждый из них предоставляет определенные услуги. Любая программа, которая пользуется ими, рассматривается как клиент приложения.

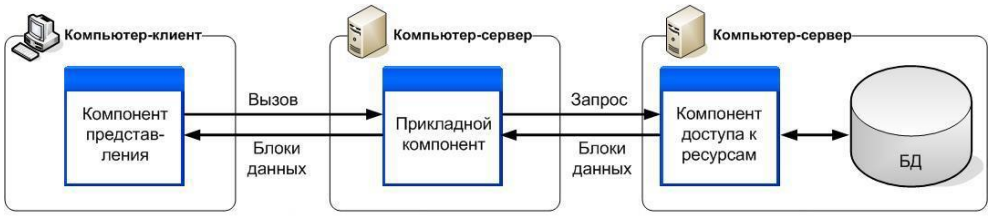

Рис.6. Модель сервера приложений

Подробности реализации прикладных функций в сервере приложений полностью недоступны для клиента приложения. Запросы поступают от клиента по очереди к AS-процессу, который извлекает и передает их для обработки службе в соответствии с приоритетами.

Так как данные «спрятаны» за сервером приложений, в котором обычно встроена проверка полномочий клиента, в СУБД обеспечивается защита данных.

Доступ к информационным ресурсам обеспечивается, как и в RDA-модели, менеджером ресурсов. Из прикладных компонентов доступны такие ресурсы как, базы данных, очереди, почтовые службы и другие. Серверы приложений выполняются, как правило, на том же компьютере, где функционирует менеджер ресурсов, что избавляет от необходимости направления SQL-запросов по сети и повышает производительность системы, Также серверы приложений могут выполняться и на других компьютерах.

AS-модель является универсальной системой, в которой может быть сколько угодно уровней, сообщающихся между собой. Четкое разделение логических компонентов, возможность баланса загрузки между несколькими серверами, и оптимальный выбор программных средств для их осуществления обеспечивают модели такой уровень гибкости, защиты данных и открытости, который пока недостижим в RDA- и DBS-моделях. В AS-модели возможна работа по медленным линиям связи, что определённо снижает трафик между клиентом и сервером приложений.

Следовательно, если необходима система с графическим интерфейсом, то следует использовать RDA-модель. DBS-модель является предпочтительным вариантом для СУБД. AS-модель является лучшим вариантом для создания больших информационных систем.

2.2. Реализация сетевого взаимодействия в языке Java

Язык Java является одним из лучших языков программирования для реализации сетевого взаимодействия. Это возможно благодаря классам, определенным в пакете java.net. Они обеспечивают легкие в использовании

средства, с помощью которых имеется возможность обращаться к сетевым ресурсам.

В основе сетевой поддержки Java лежит концепция сокета (socket). Сокет идентифицирует конечную точку сети. Сокеты — основа современных сетей, поскольку сокет позволяет отдельному компьютеру обслуживать одновременно как множество разных клиентов, так и множество различных типов информации. Это достигается за счет использования порта (port) — нумерованного сокета на определенной машине. Говорят, что серверный процесс «слушает» порт до тех пор, пока клиент не соединится с ним. Сервер в состоянии принять множество клиентов, подключенных к одному и тому же номеру порта, хотя каждый сеанс является уникальным. Чтобы обработать множество клиентских соединений, серверный процесс должен быть многопоточным либо обладать какими-то другими средствами обработки одновременного ввода-вывода [11].

Сокетные коммуникации происходят по определенному протоколу. Internet-протокол (IP) — это низкоуровневый маршрутизирующий протокол, который разбивает данные на небольшие пакеты и посылает их через сеть по определенному адресу, что не гарантирует доставки всех этих пакетов по этому адресу.

Протокол управления передачей (Transmission Control Protocol — TCP) является протоколом более высокого уровня, обеспечивающий надежную сборку этих пакетов, сортировку и повторную передачу, необходимую для надежной доставки данных. Третий протокол — протокол пользовательских дейтаграмм (User Datagram Protocol — UDP), стоящий непосредственно за TCP, может быть использован непосредственно для поддержки быстрой, не требующей постоянного соединения и ненадежной транспортировки пакетов.

Java поддерживает семейства протоколов как TCP , так и UDP. TCP применяется для надежного потокового ввода-вывода по сети. UDP поддерживает более простую, а потому быструю модель передачи дейтаграмм от точки к точке.

Дейтаграммы — это порции информации, передаваемые между машинами.

Язык Java также поддерживает работу с высокоуровневыми WEB-протоколами. Для нахождения ресурсов в Internet используется унифицированный локатор ресурсов (URL). Внутри пакета Java.net класс URL представляет простой согласованный программный интерфейс для доступа к информации по всей сети Internet посредством использования URL.

Java-класс URL имеет несколько конструкторов.

Одна из часто используемых форм специфицирует URL в виде строки, идентичной тому, что вы видите в браузере:

URL(String urlSpecifier)

Следующая форма конструктора позволяет разбить URL на части-компоненты:

URL(String protocolName, String hostName, int port, String path)

Более функциональным классом является URLConnection. У этого класса есть два метода – getInputStream(), для получения входящих данных, и getOutputStream(), который можно использовать для передачи данных на сервер, если он поддерживает такую операцию.

Как упоминалось ранее, пакет java.net также предоставляет доступ к протоколам TCP и UDP. Для реализации взаимодействия по низкоуровневым протоколам TCP и UDP необходим класс InetAdress. Он используется для инкапсуляции как числового IP-адреса, так и доменного имени для этого адреса.

Экземпляры этого класса создаются не с помощью конструкторов, а с помощью статических методов:

- InetAddress getLocalHost()
- InetAddress getByName(String name)
- InetAddress[] getAllByName(String name)

Первый метод возвращает IP-адрес машины, на которой исполняется Java-программа. Второй метод возвращает адрес сервера, чье имя передается в качестве параметра. Это может быть как DNS-имя, так и числовой IP, записанный в виде текста, например, "67.11.12.101". Наконец, третий метод определяет все IP-адреса указанного сервера.

Для работы с TCP-протоколом используются классы Socket и ServerSocket. Первым создается ServerSocket – сокет на стороне сервера. Его простейший конструктор имеет только один параметр – номер порта, на котором будут приниматься входящие запросы. После создания вызывается метод accept(), который приостанавливает выполнение программы и ожидает, пока какой-нибудь клиент не инициирует соединение. В этом случае работа сервера возобновляется, а метод возвращает экземпляр класса Socket для взаимодействия с клиентом.

Клиент для подключения к серверу также использует класс Socket. Его простейший конструктор принимает два параметра - адрес сервера (в виде строки, или экземпляра InetAddress) и номер порта. Если сервер принял запрос, то сокет конструируется успешно и далее можно воспользоваться методами getInputStream() или getOutputStream().

На стороне сервера класс Socket используется точно таким же образом – через методы getInputStream() и getOutputStream().

Рассмотрим классы Socket и ServerSocket более подробно. Во-первых, класс ServerSocket имеет конструктор, в который передается, кроме номера порта, еще и адрес машины. Это может показаться

странным. Однако, если компьютер имеет несколько сетевых интерфейсов, то он имеет и несколько сетевых адресов. С помощью такого детализированного конструктора можно указать, по какому именно адресу ожидать подключения. Это должен быть именно локальный адрес машины, иначе возникнет ошибка.

Аналогично, класс Socket имеет расширенный конструктор для указания как локального адреса, с которого будет устанавливаться соединение, так и локального порта.

Во-вторых, можно воспользоваться методом setSoTimeout(int timeout) класса ServerSocket, чтобы указать время в миллисекундах, на протяжении которого нужно ожидать подключение клиента. Это позволяет серверу не «зависать», если никто не пытается начать с ним работать. Тайм-аут задается в миллисекундах, нулевое значение означает бесконечное время ожидания.

Важно подчеркнуть, что после установления соединения с клиентом сервер выходит из метода accept(), то есть перестает быть готовым принимать новые запросы. Однако, как правило, желательно, чтобы сервер мог работать с несколькими клиентами одновременно. Для этого необходимо при подключении очередного пользователя создавать новый поток исполнения, который будет обслуживать его, а основной поток снова войдет в метод accept().

ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ

3.1. Общее описание программной системы

В ходе работы была разработана программная система для выполнения параллельного вычисления арифметических выражений. Для реализации системы была использована технология «клиент-сервер». Соответственно, система состоит из двух частей: программы-сервера и программы-клиента.

Программа-сервер является основным звеном системы. На стороне сервера вводится арифметическое выражение в обычной инфиксной форме в виде строки. Операндами являются вещественные положительные числа. Допустимыми операциями являются «+», «-», «*», «/», также можно использовать скобки. Затем программа-сервер выполняет перевод выражения в обратную польскую запись и строит сбалансированное дерево, необходимое для оптимального разбиения выражения на части - ветви. Эти части выражения в дальнейшем будут отправлены клиентам для параллельного вычисления.

Программа-клиент является рабочей частью системы. На стороне клиента производится основной процесс вычисления выражения. Программа-клиент запускается на нескольких компьютерах, вводится IP-адрес сервера и устанавливается соединение. Как только к серверу подключится хотя бы один клиент, можно начать вычисление. Используя сбалансированное дерево, сервер рассылает каждому клиенту часть исходного выражения, записанного в обратной польской записи. Клиент принимает выражение и тут же производит его вычисление. После завершения вычисления клиент отправляет результат и время вычисления обратно на сервер. Если выражение вычислено полностью, то сервер выводит на экран результат вычисления. Если же в сбалансированном

дереве остались не рассчитанные ветви, то сервер отправляет их клиентам, пока расчет не будет полностью завершен.

Благодаря подсчету времени вычисления можно провести анализ эффективности разработанной системы. Например, можно отослать все выражение одному клиенту и засечь время. Затем подключить четырех клиентов и рассчитать то же самое выражение. А после этого сравнить время вычисления.

Разработка программной системы проводилась совместно с Петром Шалагиновым. Его работа посвящена разработке программы-сервера. Данная работа посвящена разработке клиентской части системы.

3.2. Описание программы-клиента

Для взаимодействия программы-клиента с пользователем был разработан простой интерфейс (Рис.7).

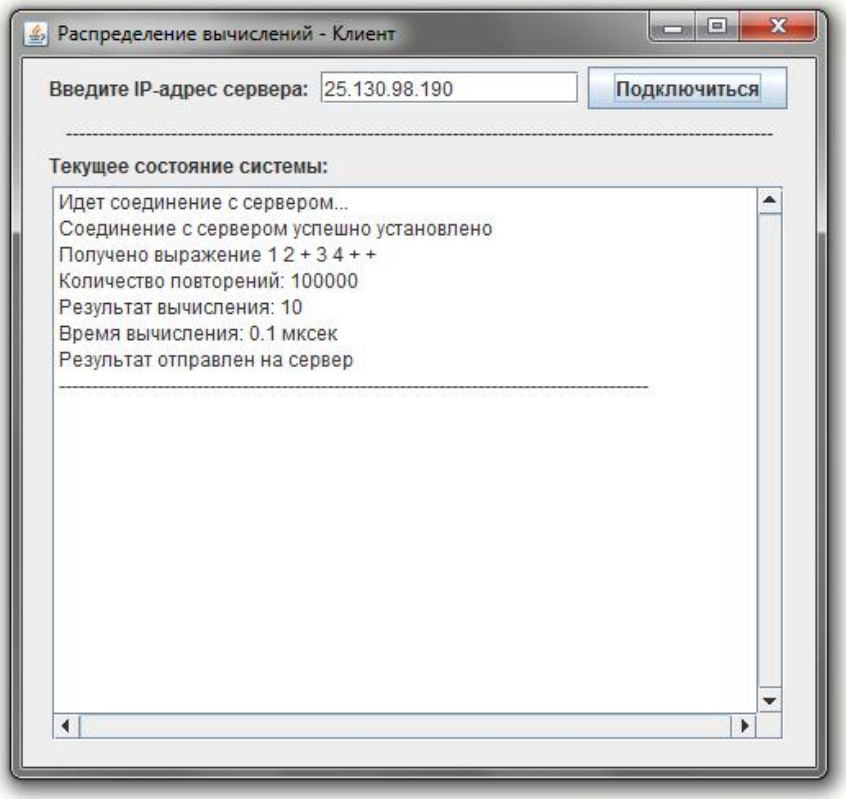

Рис.7. Интерфейс программы-клиента

В главном окне программы расположены следующие элементы:

- поле для ввода IP-адреса сервера
- кнопка «Подключение», для активации соединения с сервером
- информационное окно, в котором выводится текущее состояние системы: статус соединения, полученное выражение, результат и время вычисления, сообщение об отправке или ошибке.

После запуска программы-клиента необходимо установить соединение с сервером. Для этого потребуется ввести IP-адрес сервера в соответствующее поле. Адрес вводится в обычном формате вида xxx.xxx.xxx.xxx. Например, «192.168.1.1». После ввода адреса клиент должен нажать на кнопку «Подключиться», чтобы инициализировать подключение. Если сервер в данный момент активен, то пользователь получит сообщение об успешном подключении (Рис.8). Если сервер выключен, то в информационном окне появится сообщение об ошибке.

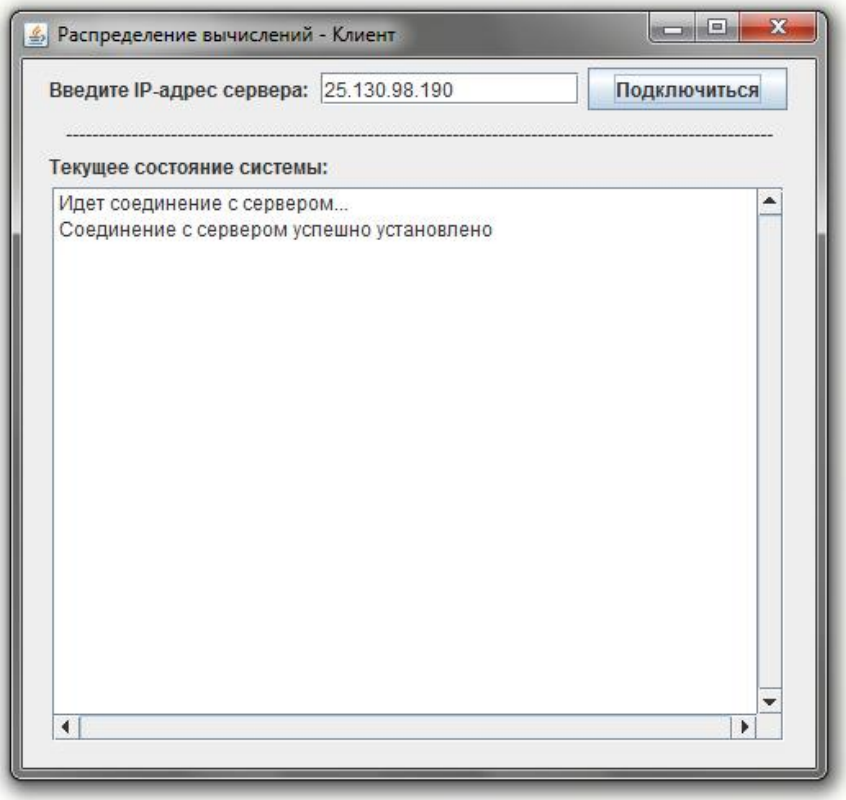

Рис.8. Успешное подключение к серверу

Если соединение с сервером прошло успешно, то клиент готов к началу вычислений. Сервер получит уведомление о подключении нового клиента. Далее клиент ожидает команды от сервера. Как только сервер дождется подключения нескольких клиентов, исходное выражение будет разбито на несколько частей и каждый клиент получит свою часть выражения.

После получения выражения клиент сразу же начнет вычисление. Как только результат будет получен, он отобразиться в информационном окне, и будет отправлен обратно на сервер. Также в информационном окне появится сообщение об успешной отправке (Рис.9), а клиент будет ожидать команды от сервера.

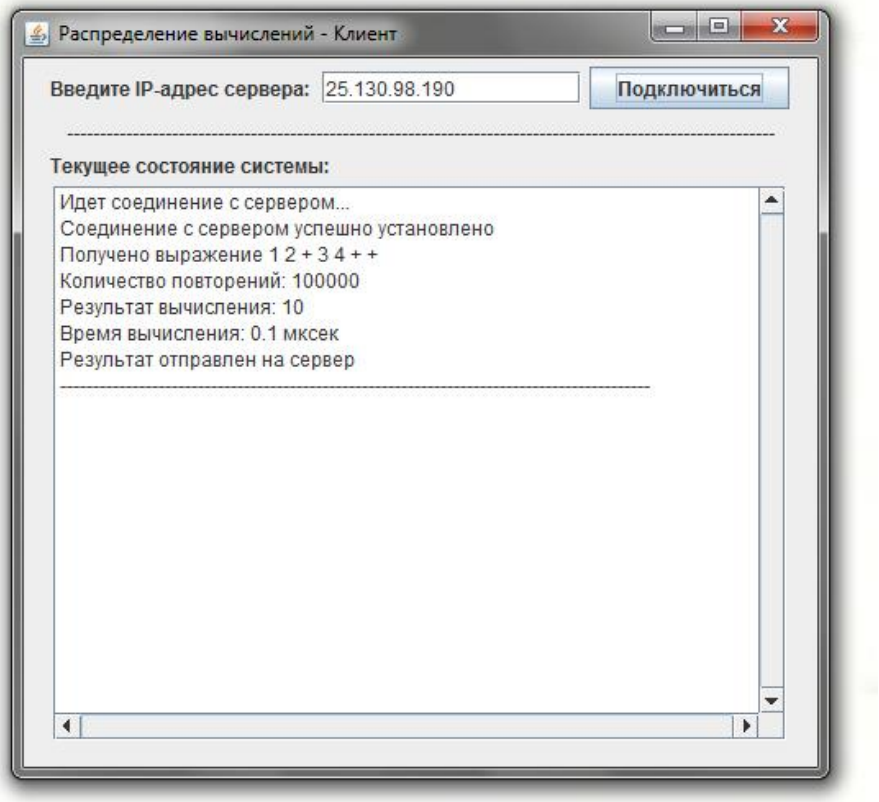

Рис. 8. Результат вычисления выражения и сообщение об отправке

3.3. Программа на языке Java и её структура

При разработке программы-клиента были использованы некоторые классы из встроенных библиотек языка Java.

1. *Библиотека net*. Классы данной библиотеки позволяют реализовать сетевое взаимодействие между программой-сервером и программами-клиентами. Для создания подключения потребовались следующие классы данной библиотеки:
    - InetAddress. Этот класс используется для работы с IP адресами. С его помощью можно указать программе адрес сервера, необходимый при создании сокета. Создание экземпляра происходит не с помощью оператора new, а с применением статических методов. В нашем случае был использован метод getByName, который возвращает адрес узла.
    - Socket. Данный класс реализует клиентские сокеты. Сокет является конечной точкой для связи узлов. Для создания сокета необходимо указать IP-адрес сервера, используя метод класса InetAddress, и порт, по которому будет идти передача.
2. *Библиотека IO*. Классы этой библиотеки работают с потоками ввода и вывода. Они позволяют получать поток данных, обрабатывать его и отправлять дальше. В программе-клиенте были использованы 2 класса из этой библиотеки:
    - ObjectInputStream. Данный класс необходим для приема входящего потока. Для этого используется метод getInputStream(), который принимает входящие данные и записывает их в переменную input. Для чтения полученных данных используется метод ReadObject().

- ObjectOutputStream. Данный класс необходим для обработки исходящего потока. Для записи данных в поток используется метод writeObject().

3. *Библиотека Until.* Из данной библиотеки был использован класс LinkedList. Класс используется для создания связанного двунаправленного списка, в котором каждый элемент содержит указатели на предыдущий и следующий элементы. LinkedList может менять свой размер во время исполнения программы, при этом не обязательно указывать размерность при создании объекта. LinkedList реализует методы получения, удаления и вставки в начало, середину и конец списка, а также позволяет добавлять любые элементы.
    - Метод add(). Данный метод позволяет добавить в список новый элемент. Новый элемент оказывается на вершине.
    - Метод removeLast(). С помощью данного метода можно получить последний элемент списка и сразу удалить его.
    - Метод size(). Используется для получения размера списка, т.е. количества элементов, находящихся в нем.
4. *Библиотека Swing*. Данная библиотека содержит необходимый набор компонентов для создания графического интерфейса пользователя. Основана на более ранней библиотеке AWT. Для создания интерфейса программы потребовались следующие классы:
    - JFrame – главное окно приложения;
    - JPanel – простые панели для группировки элементов;
    - JButton – кнопка;
    - JCheckBox – флажок проверки;
    - JTextField – однострочное текстовое поле;
    - JTextArea – многострочное текстовое поле.

5. *Библиотека AWT*. Пакет *awt.event*. Событие (event) в библиотеке AWT возникает при воздействии на компонент при манипуляциях мышью, вводе с клавиатуры, перемещении окна. При возникновении события исполняющая система Java автоматически создает объект соответствующего событию класса. Этот объект не производит никаких действий, он только хранит все сведения о событии.

   - ActionEvent – событие классов JButton и JTextField;
   - ItemEvent – событие, возникающее в классе JCheckbox.

   Методы обработки событий описаны в интерфейсах - слушателях. Для каждого события существует свой интерфейс. Методы интерфейса "слушают", что происходит в потенциальном источнике события. При возникновении события эти методы автоматически выполняются, получая в качестве аргумента объект-событие и используя при обработке сведения о событии.

   - ActionListner – слушатель события ActionEvent;
   - ItemListner – слушатель события ItemEvent.

   На схеме (Рис. 9) изображена структура разработанной программы.

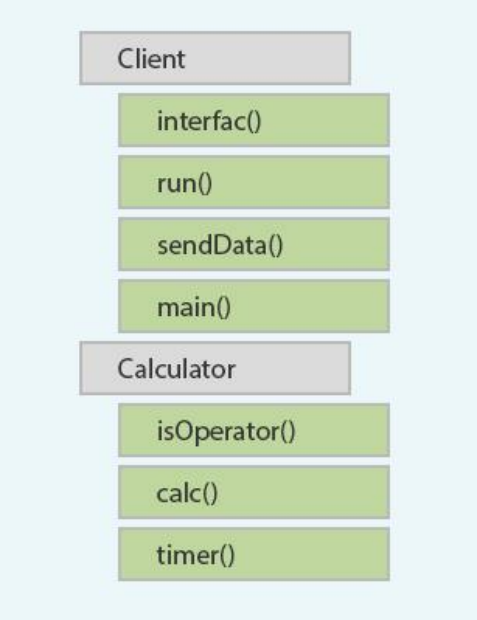

Рис. 9. Схема классов программы-вычислителя

Программа состоит из двух основных классов: calculator и client.

Класс client включает в себя 4 метода:

1. *Метод interface()*. Данный метод предназначен для создания пользовательского интерфейса программы-вычислителя.
2. *Метод run()*. Класс client реализует интерфейс runnable. Данный метод используется для создания потока подключения к серверу.
3. *Метод sendData()*. Данный метод используется для отправки результата и времени вычисления на сервер.
4. *Метод main()*. Главный метод программы, в котором инициализируется интерфейс.

Класс calculator включает в себя 3 метода:

1. *Метод isOperator()*. Данный метод предназначен для определения того, является ли очередной элемент операцией. Если очередной элемент является оператором, то метод возвращает значение true, иначе возвращается значение false.
2. *Метод calc()*. Основной метод программы-вычислителя, в котором производится вычисление заданного выражения.
3. *Метод timer()*. Данный метод предназначен для определения времени вычисления. Вычисление времени производится на основе системного времени. Засекается время в начале и в конце вычисления выражения. Разность этих значений является временем, затраченным на вычисление. Данный метод отличается быстродействием и неплохой точностью вычисления.

Наиболее интересным является метод run() класса client. Данный метод реализует подключение к серверу. Рассмотрим его более подробно.

Для создания соединения необходимо создать клиентский сокет. Для этого создадим новый сокет connection, указав в качестве параметров IP-адрес сервера и порт, по которому будет происходить соединение.

```
connection = new Socket(InetAddress.getByName(ipserv), 3333);
```

После создания сокета возьмем входящий и исходящий потоки. Входящий поток позволяет считывать данные с сокета, а исходящий поток позволяет отправлять объекты в сокет.

```
output = new ObjectOutputStream(connection.getOutputStream());
input = new ObjectInputStream(connection.getInputStream());
```

Далее необходимо прослушивать входящий поток. Для этого запускаем цикл while с параметром true и считываем входящий поток с помощью метода readObject().

```
while(true) {
try { inp = (String)input.readObject(); }}
```

Сервер отправляет клиенту строку вида «выражение, количество повторений». Как только клиент считает строку из входного потока, ее необходимо разделить на части: выражение отдельно, количество повторений отдельно. После этого можно запускать метод calc() класса calculator, в качестве параметров указав только что полученные данные.

```
String in[] = inp.split(",");
String result = c.calc(in[0],true,Integer.valueOf(in[1]));
```

Рассмотрим более подробно метод calc() класса calculator, так как он используется для вычисления полученного выражения. Ниже представлена блок-схема (Рис. 10), отражающая алгоритм работы метода. Блок-схема нарисована с помощью интернет-сервиса draw.io.

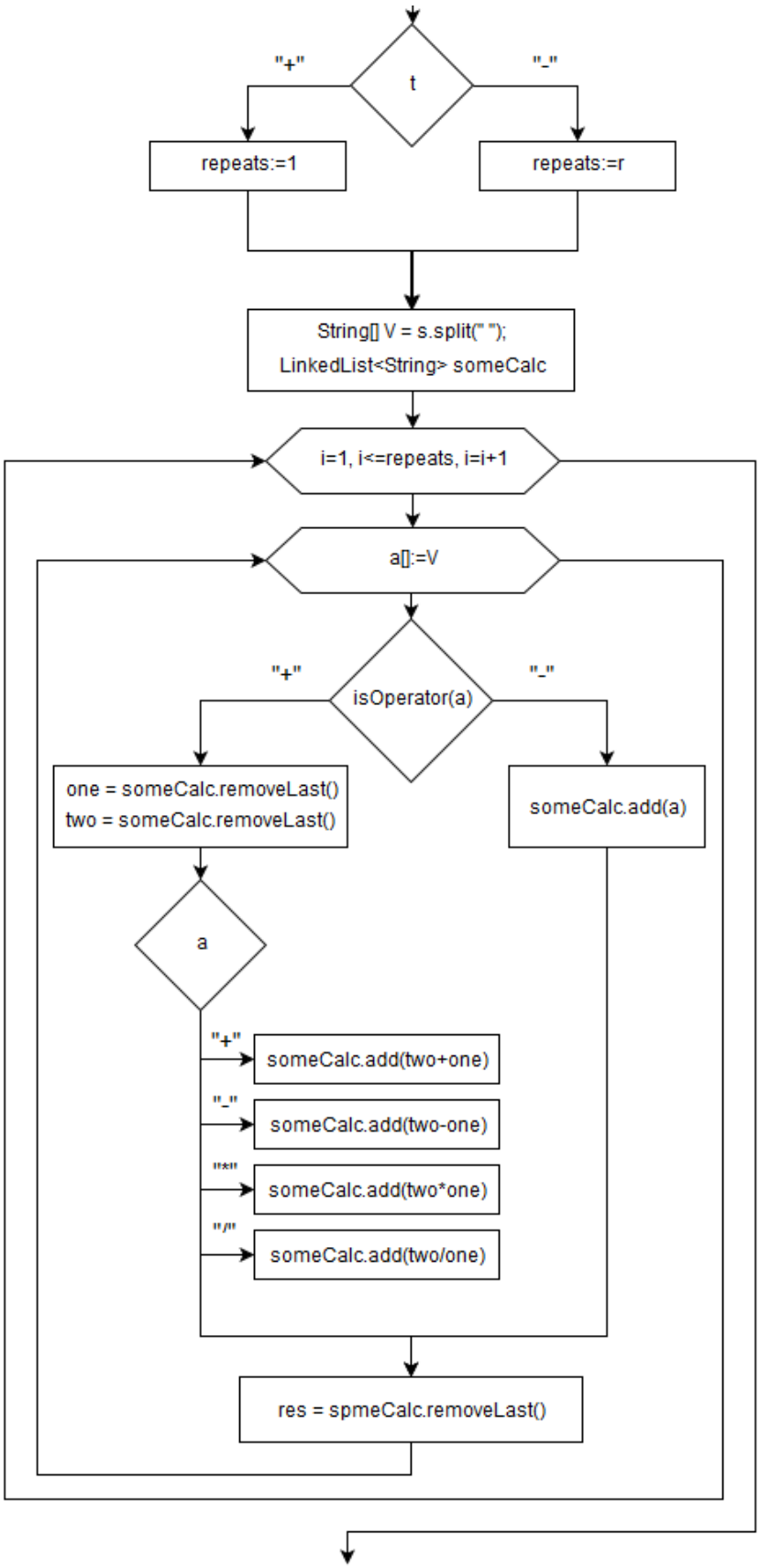

Рис. 10. Блок-схема метода calc() класса calculator

На вход метода подаются три параметра:

1. Входная строка «S», которая содержит выражение, полученное от сервера.
2. Логическая переменная «t», необходимая для включения или отключения функции подсчета времени вычисления.
3. Целочисленная переменная «r», которая задает количество повторений вычисления выражения, указанное сервером.

Вначале проверяем значение входного параметра «t». Если он равен false, значит произвести вычисление выражения нужно один раз. Переменной «repeats», присваиваем значение 1. Если параметр «t» имеет значение true, то произвести вычисление нужно некоторое количество раз. Переменной «repeats» в данном случае присваиваем значение входного параметра «r», который указывает количество необходимых повторений. При этом необходимо записать время начала измерения в переменную Start, используя функцию получения системного времени.

long Start =System.currentTimeMillis();

Далее необходимо разбить входную строку «S» на элементы и поместить их в массив «V». Для этого применяем метод split(String regExp) к строке «S», в качестве параметра «regExp» указав символ пробела, т.к. он используется для разделения операндов.

Теперь, когда входное выражение разделено на элементы, можно начать вычисление. Как упоминалось ранее, для реализации вычисления выражений в ПОЛИЗ необходима структура данных, известная как стек. Создадим объект someCalc типа LinkedList. Он и будет являться стеком.

Запустим первый цикл for, который будет повторять вычисление выражения некоторое количество раз, заданное переменной «repeats». Внутри запустим второй цикл for, который будет поочередно

рассматривать каждый элемент в массиве «V», в котором хранится выражение.

Проверяем, является ли очередной элемент массива «V» операцией. Для этого используется метод логического типа isOperator(). Он сравнивает полученный элемент со списком допустимых операций и на выходе выдает значение true или false.

```
boolean isOperator(String c) {
if (c == “+” || c == “-“ || c == “*” || c == “/”) {
return true;
} else {
return false;
} }
```

Если очередной элемент является операцией, то проверяем, имеется ли в стеке два операнда, необходимых для выполнения операции. Если в стеке нет двух операндов, то выводим сообщение об ошибке. Если в стеке есть два операнда, то заносим каждый из них в переменные «one» и «two» соответственно, используя метод removeLast(). После этого производим операцию над двумя полученными операндами и записываем результат обратно в стек, используя метод add(). Если же очередной элемент не является операцией, значит это операнд. В таком случае заносим его в стек, используя метод add().

Когда элементы массива «V» закончится, то результатом вычисления выражения будет являться элемент, расположенный на вершине стека. Запишем его в переменную «res», используя знакомый метод removeLast().

После этого следует засечь время окончания расчета и записать ее в переменную Stop.

```
long Stop = System.currentTimeMillis();
```

Используя метод timer() подсчитаем время, затраченное на вычисление выражения, найдя разницу между данными о времени, записанными в переменных Start и Stop. Данный метод вычисления является достаточно точным, так как использует системное время компьютера. Вычисление времени производится для дальнейшей оценки эффективности всей клиент-серверной системы.

Как только результат и время вычисления будут получены, необходимо отправить данные серверу. Для этого будем использовать метод sendData(). В качестве параметра укажем переменную, в которой хранится результат вычисления выражения и время, за которое оно было вычислено.

Метод sendData работает следующим образом. Для начала нужно очистить исходящий поток, используя метод flush(). Это необходимо для исключения смешивания пакетов. Далее, используя метод writeObject(), записываем в исходящий поток объект, указанный в качестве параметра, т.е. результат и время вычисления.

Результатом применения метода sendData() станет отправка результата и времени вычисления на сервер. После этого программа-клиент продолжит прослушивать входящий поток, ожидая дальнейших команд от сервера.

Полный исходный код программы-клиента представлен в приложении 1.

3.4. Возможности применения разработанного ПО в курсе информатики

Разработанную клиент-серверную систему можно применять как в средней школе, так и в высших учебных заведениях.

Например, в некоторые школьные программы входит формирование у учащихся понятий технологии «клиент-сервер». Полученную учебную

систему можно демонстрировать при рассмотрении темы «клиент-сервер» при обучении по учебнику «Информатика-базовый курс», 8-9 классы, Семакина И., Залоговой Л. Запустив систему, можно демонстрировать учащимся как обмениваются информацией клиент и сервер, а также как происходит подключение.

Изучение технологии «клиент-сервер» также входит в программы курсов высших учебных заведений, связанных с информатикой и информационными системами. При изучении этой темы можно демонстрировать студентам разработанную учебную систему, более детально углубляясь в технологию.

Кроме того, данную систему можно показывать, как пример реализации сетевого взаимодействия, при изучении языка программирования Java.

Учебная система распределения вычислений между несколькими клиентами хорошо подойдет для изучения курса про параллельные вычисления, который в последнее время активно включается в учебные программы для студентов высших учебных заведений. Используя разработанную систему можно показать студентам, как можно параллельно вычислить арифметическое выражение с помощью клиент-серверного приложения. Студенты смогут увидеть, каким образом сервер разделяет выражение на части и отправляет их клиентам, которые в последствии параллельно выполняют вычисление. После завершения расчетов можно оценить эффективность параллельного метода вычисления арифметического выражения, сравнив время вычисления исходного выражения одним клиентом и несколькими клиентами.

Таким образом, разработанную учебную систему можно активно применять в образовательной среде.

Заключение

В дипломной работе ставилась задача исследования алгоритмов разбора арифметических выражений и технологии клиент-сервер, а также разработка клиентской части учебной системы распределения вычислений на языке Java.

В результате работы получены следующие результаты:

1. Изучены основы трансляции арифметических выражений.
2. Изучены два основных алгоритма трансляции арифметических выражений:
    - метод Рутисхаузера, который является исторически первым решением задачи трансляции арифметических выражений;
    - классический метод ПОЛИЗ, основанный на промежуточной обратной польской записи.
3. Для разработки программы-вычислителя был выбран метод ПОЛИЗ, благодаря которому выражение можно вычислить в процессе однократного просмотра слева направо.
4. Изучены основные идеи технологии «клиент-сервер» и методы реализации сетевого взаимодействия на языке Java.
5. Разработана клиентская часть учебной системы распределения вычислений. Полученная программа работает совместно с программой-сервером, разработанной Петром Шалагиновым. Программа-сервер отвечает за разбиение арифметического выражения на части и отправку этих частей клиентам. Программа-клиент используется для расчета этих частей на стороне клиентов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Зеленяк О. П. Практикум программирования. Задачи, алгоритмы и решения / О. П. Зеленяк. – М.: ДМК Пресс, 2013. – 313 с.
2. Коржов В. А. Многоуровневые системы клиент-сервер [Текст] / В. А. Коржов // Сети/Network World. – 1997. – № 6. – С. 72-75.
3. Лебедев В. Н. Введение в системы программирования / В. Н. Лебедев. – М.: Статистика, 1975. – 309 с.
4. Молчанов А. Ю. Системное программное обеспечение: учеб. для студентов вузов, обучающихся по специальностям "Вычисл. машины, комплексы, системы и сети" / А. Ю. Молчанов. – СПб.: Питер, 2010. - 395 с.
5. Осипов Д. Л. Базы данных и Delphi. Теория и практика / Д. Л. Осипов. – СПб: БХВ-Петербург, 2011. – 752 с.
6. Поляков К.Ю. Информатика. Углубленный уровень: Учебник для 10 класса: в 2 ч. Ч. 2 / К.Ю. Поляков, Е.А.Еремин. – М.: БИНОМ, 2013. – 304 с.
7. Поттосин И. В. От программирующих программ к системам программирования / Э. З. Любимский, И. В. Поттосин, М. Р. Шура-Бура // Становление Новосибирской школы программирования. Мозаика воспоминаний. – Новосибирск: Институт систем информатики им. А.П. Ершова СО РАН, 2001. – С. 18-27.
8. Рутисхаузер Г. Алгоритм частных и разностей. – М.: Изд-во инофстранной литературы, 1960 – 93 с.
9. Свердлов С. З. Конструирование компиляторов: учебное пособие / С. З. Свердлов. – Германия: AP LAMBERT Academic Publishing, 2015. – 571 с.

10. Собчак А. П. Модель аппаратной реализации алгоритма Рутисхаузера / А. П. Собчак, К. В. Ходарев // Радіоелектронні і комп’ютерні системи. – 2005. – № 4. – С. 96–101.
11. Сухов С. А. Программирования сетевых взаимодействий в Java: методические указания / С. А. Сухов. – Ульяновск: УлГТУ, 2010. – 52 с.
12. Таненбаум Э. Архитектура компьютера / Э. Таненбаум. – СПб: Питер, 2013. – 843 с.
13. Хьюз К. Параллельное и распределенное программирование с использованием C++ / К. Хьюз. – М.: Вильямс, 2004. – 672 с.
14. Шилдт Г. Java 8. Полное руководство. – М.: Вильямс, 2015. – 1376 с.

Приложение 1. Исходный код программы-клиента.

```
import java.util.LinkedList;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import javax.swing.*;

public class calculator {
boolean isOperator(String a) {
if (a.equals("+") || a.equals("-") || a.equals("*") || a.equals("/")) {
return true;
} else {
return false;
}
}

public String calc(String s, boolean t, int r) {
int repeats = 1;
float res = 0;
float one = 0;
float two = 0;
```

```
boolean error = false;
if ((t==false)) { repeats = 1; } else { repeats = r; }
long Start =System.currentTimeMillis();
String[] V = s.split(" ");
LinkedList<String> someCalc = new LinkedList<>();
for (int i = 1; i <= repeats; i = i + 1) {
        if(error) { break; }
for (String a:V) {
if (isOperator(a)){
if(!someCalc.isEmpty()) {
        one = Float.parseFloat(someCalc.removeLast());
} else {
        error = true;
        break;
}
if(!someCalc.isEmpty()) {
        two = Float.parseFloat(someCalc.removeLast());
} else {
        error = true;
        break;
}
switch(a) {
case "+":
someCalc.add(String.valueOf(two + one));
break;
case "-":
someCalc.add(String.valueOf(two - one));
break;
```

```
case "*":
someCalc.add(String.valueOf(two * one));
break;
case "/":
someCalc.add(String.valueOf(two / one));
break;
default:

}
} else {
someCalc.add(a);
}
/* System.out.println(someCalc.getLast()); */
}
if (!error) { res = Float.parseFloat(someCalc.removeLast()); }
if (someCalc.size() != 0) {
        error=true;
}
}
long Stop = System.currentTimeMillis();

if (!error) {
        if(t) {
        return ""+res+","+timer(Start,Stop,repeats);
        } else {
                return ""+res;
        }
} else {
```

```
        return "Ошибка в выражении!";
}
}
public String timer(long ts,long te,int r) {
        double tres=te-ts;
        return ""+tres*1000/r;
}
}

public class client extends JFrame implements Runnable {
        private static final long serialVersionUID = 1L;
        static private Socket connection;
        static private ObjectOutputStream output;
        static private ObjectInputStream input;
        static calculator c = new calculator();
        public static boolean timer = false;
        public static JTextArea area;
        public static JButton connect;
        public static String ipserv;
        public static boolean connected = false;

        public static void interfac() {
       JFrame window = new JFrame("Распределение вычислений - Клиент");
       window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       final JPanel panel = new JPanel();

       JLabel label3 = new JLabel("Введите IP-адрес сервера: ");
       final JTextField ip = new JTextField("25.130.98.190");
```

```
        ip.setColumns(14);
        JLabel space2 = new JLabel("
");
        JLabel label2 = new JLabel("Текущее состояние системы:");
        JLabel label1 = new JLabel("--------------");

        area = new JTextArea("",20,39);
        area.setEditable(false);
        //area.setLineWrap(true);
        //area.setWrapStyleWord(true);
        JScrollPane scroll = new JScrollPane (area);
        scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_
SCROLLBAR_ALWAYS);
         scroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL
_SCROLLBAR_ALWAYS);

        connect = new JButton("Подключиться");
        connect.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                ipserv = ip.getText();
                                new Thread(new client()).start();
                                area.setText("");
                                area.append(" Идет соединение с сервером... \n");
            }
        });

        window.addWindowListener(new WindowListener() {
         @Override
```

```
public void windowActivated(WindowEvent arg0) {}
@Override
public void windowClosed(WindowEvent arg0) {}
@Override
public void windowClosing(WindowEvent arg0) {
if(connection==null) {
System.exit(0);
} else {
        if(connection.isConnected()) {
        sendData("Exit");
        }
}
}
@Override
public void windowDeactivated(WindowEvent arg0) {}
@Override
public void windowDeiconified(WindowEvent arg0) {}
@Override
public void windowIconified(WindowEvent arg0) {}
@Override
public void windowOpened(WindowEvent arg0) {}
});
panel.add(label3);
panel.add(ip);
panel.add(connect);
panel.add(label1);
panel.add(label2);
panel.add(space2);
```

```
        panel.add(scroll);
        window.getContentPane().add(panel);
        window.setPreferredSize(new Dimension(500, 470));
        window.pack();
        window.setLocationRelativeTo(null);
        window.setVisible(true);
        window.setResizable(false);
    }

    public static void main(String[] args) {
         interfac();
    }
         @Override
         public void run() {
                try {
                connection = new Socket(InetAddress.getByName(ipserv), 3333);
                if (connection.isConnected()) { connect.setEnabled(false); }
                output = new ObjectOutputStream(connection.getOutputStream());
                input = new ObjectInputStream(connection.getInputStream());
                        String inp = null;
                        area.append(" Соединение с сервером установлено \n");
                        area.append(" ----------------------------\n");
                        while(true) {
                        try {
                        inp = (String)input.readObject();
                        } catch (ClassNotFoundException e) {}
                        String in[] = inp.split(",");
                        area.append(" Получено выражение: " +in[0]+"\n");
```

```
                    area.append(" Количество повторений: " +in[1]+"\n");
                    String result = c.calc(in[0],true,Integer.valueOf(in[1]));
                    sendData(result);
                    String res[] = result.split(",");
                    area.append(" Результат вычисления: " +res[0]+"\n");
                    area.append(" Время вычисления: " +res[1]+" мксек.\n");
                    area.append(" ----------\n");
                    }
            } catch (UnknownHostException e) {
            } catch (IOException e) {}
        }
        private static void sendData(Object obj) {
            try {
                    output.flush();
                    output.writeObject(obj);
            } catch (IOException e) {}
        }
}
```