МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

КУРСОВОЙ ПРОЕКТ

по дисциплине: Компьютерные сети

тема: «Разработка FTP клиент-серверного приложения»

Автор работы _____ Пахомов Владислав Андреевич ПВ-223

(подпись)

Руководитель проекта _____ Панченко Максим Владимирович

(подпись)

Оценка ____

Белгород 2025 г.

Оглавление

1 Введение 3

2 Анализ протоколов передачи данных 6

2.1 FTP 6

2.2 SFTP 9

2.3 SCP 10

2.4 HTTP/HTTPS 11

3 Разработка клиент-серверного приложения 14

3.1 Разработка сервера 14

3.1.1 Выбор стека 14

3.1.2 Слой Core 15

3.1.3 Слой Dependencies 21

4 Работа приложения 22

5 Вывод о проделанной работе 22

6 Список источников и литературы 23

7 Список источников и литературы 23

1 Введение

Передача файлов на сегодняшний день является одним из самых важных сценариев использования компьютерной сети. Перечислим лишь несколько из сценариев:

- **Интернет**. Ежедневно пользователи интернета посещают миллиарды сайтов и загружают копии сайтов на портативные и стационарные устройства. В свою очередь сайты могут выполнять большое количество функций, как образовательных так и развлекательных.

- **Домашняя локальная сеть**. Ещё одним сценарием передачи файлов между компьютерами является формирование локальной сети. Например, в доме можно установить NAS - специализированный компьютер, который выполняет функции хранения файлов, разграничения ролей, резервного копирования. Пользователи локальной сети должны иметь доступ как загружать данные на домашний сервер, так и получать данные с домашнего сервера. Файлы могут иметь различный характер, например это может быть домашний фотоархив или видеозаписи, исполняемые файлы, текстовые документы.

- **Нейронные сети**. Ещё одним сценарием использования компьютерных сетей может являться организация работы в высоконагруженных системах. Например, в сегодняшние дни всё больше набирает популярность использование нейронных сетей. Для обучения нейронных сетей требуются большие вычислительные мощности. Но также для таких систем требуются и большое хранилище данных, дело в том что нейронные сети обучаются на большом количестве датасетов - наборов

данных, которые нужно где-то хранить. Для этого и есть компьютерные сети - они позволяют распределить нагрузку и создать сеткомьютеров, которая позволяет эффективно хранить данные и использовать их. В случае нейронных сетей также может появляться дополнительная нагрузка в виде интерпретации данных, так как нейронные сети не могут работать с сырыми данными напрямую, их необходимо закодировать в бинарную последовательность а также представить в памяти в соответствующем виде. И так как сегодня нейронные сети используются в очень большом спектре задач, хранилища должны уметь подстраиваться под выбранный формат данных.

Задача передачи файлов была актуальной и сохраняет свою актуальность по сегодняшний день. В связи с этим было разработано множество протоколов, которые позволяют эффективно организовать передачу файлов.

Цель работы заключается в исследовании популярных протоколов передачи данных, а также создания клиент-серверного приложения для управления файлами на удалённом компьютере на основе протокола FTP.

Задачи

- Исследовать известные протоколы передачи данных, их сильные и слабые стороны
- Разработать сервер
  - Выбрать подходящий стек технологий
  - Выполнить декомпозицию задач
  - Разработать архитектуру приложения
  - Разработать серверное приложение
- Разработать клиент

  - Выбрать подходящий стек технологий
  - Выполнить декомпозицию задач
  - Разработать архитектуру приложения
  - Разработать клиентское приложение
- Выполнить анализ функционала клиент-серверного взаимодействия

2 Анализ протоколов передачи данных

Как уже было сказано ранее, в компьютерных сетях передача файлов - очень популярная задача. В связи с этим существует множество протоколов, которые подходят для решения конкретных специфических задач. Перечислим некоторые из них.

2.1 FTP

FTP - один из самых старых и самых распространённых протоколов. За полвека протокол постоянно изменялся и расширялся под главенством Internet Engineering Task Force.

Впервые протокол FTP появился в 1971 году как один из механизмов передачи файлов в RFC-114 и применялся для обмена файлами в MIT.

В дальнейшем протокол дополнялся и дорабатывался, RFC-294 ввёл возможность изменения типа передаваемых данных. RFC-354 закрепил задачу протокола как протокол для эффективной и надёжной передачи данных между устройствами сети в сети ARPANET, в 1973 с появлением RFC-454 протокол закрепился официально.

Следующим большим скачком для протокола стал переход от основного используемого протокола для обмена данными между устройствами сети от NCP к TCP вместе с RFC-765.

На сегодняшний день актуальным описанием для протокола является RFC-959, это описание в основном вводит некоторые дополнительные команды для более удобной работы с каталогами, команду уникального сохранения файла и получение информации о системе.

В дальнейшем выходили только дополнения к текущему описанию. Так, RFC-2640 вводит кодировку UTF-8, которая используется многими современными компьютерами сегодня. Стандарт RFC-2228 вводит более широкие возможности для обеспечения безопасности при передаче файлов.

Положительные стороны

- Протокол FTP имеет очень широкую поддержку, имеет множество рабочих проверенных временем решений, большинство компьютеров имеют поддержку FTP, в том числе и на системном уровне, то есть нет необходимости устанавливать стороннее программное обеспечение для использования FTP.

- Очень хорошая совместимость со старыми компьютерами. Реализации FTP есть как для старых так и для современных компьютеров. Кроме того, протокол FTP поддерживает очень большое количество способов передачи файлов, что позволит запустить сервер на множестве старых систем, вроде компьтютеров IBM. Такие случаи редки сегодня, однако они позволят администрировать устаревшие системы, которые по сегодняшний день всё ещё являются частью множества бизнес процессов.

- Поддержка многих способов передачи файлов. Протокол может передавать и получать данные как постранично, так и в скомпрессированном виде.

- Относительная простота протокола для пользователя. Изначально FTP разрабатывался как консольное приложение, поэтому множество ответов имеют понятное описание, так же как и команды выражаются относительно просты в освоении и понимании.

Отрицательные стороны

- Большинство команд протокола при условии использования современных систем просто мусорные, вряд ли сегодня найдётся устройство, которое будет использовать в качестве системной кодировки EBCDIC или обычный ASCII.

- Протокол

  не подходит для автоматизации. В сегодняшнем мире пользователю приятней всего использовать понятный и интуитивный графический интерфейс, однако как уже было сказано ранее, протокол заточен для использования в консоли, из-за чего возникает множество проблем. Например, команда вывода списка файлов в текущей директории платформоспецифична, что вызывает трудности в интерпретации для оформления графического интерфейса в зависимости от того, на какой машине запущен протокол.

- Протокол

  не предназначен для нагруженных систем. Протокол использует два соединения - одно для передачи данных, а второе для передачи команд (порт 21 и 20 соответственно). В нагруженных системах с большим количеством соединение это непозволительная роскошь.

- Безопасность. Протокол в текущем описании никак не шифрует данные, любой может подключиться к сети и прослушать пакеты, из-за чего существует угроза несанкционированного доступа к данным.

- Некое моральное устаревание протокола нивелируется новыми стандартами, дополняющими протокол. Однако заставить пересесть множество уже отлаженных серверов на старых стандартах на новые и неисследованные команды может быть экономически неоправданно и технологически сложно, в том числе и при оптимизации процессов.

Протокол FTP хорошо отлажен и проверен временем, он подходит для консольной работы на старых системах. Однако для построения новых систем рекомендуется использовать более современные протоколы.

2.2 SFTP

На смену протоколу FTP пришёл протокол SFTP, который решает большинство проблем протокола FTP.

Этот протокол был разработан The OpenSSH Project, его описание можно найти в расширении к описанию протокола SSH в RFC-4253 от 2006 года. В этом и первое основное отличие протокола FTP, SFTP работает поверх SSH, в то время как FTP работает на устаревшем протоколе Telnet.

SFTP не является дополнением FTP, хотя и берёт очень много из его функционала. Протокол SFTP нужно рассматривать как совершенно новый переработанный протокол для современных систем.

Протокол SFTP берёт очень много из соединения по SSH, его основной особенностью стала перекочевавшая из SFTP повышенная безопасность, большое количество доступных методов шифрования.

Положительные стороны

- Безопасность. Как уже было сказано ранее, протокол SFTP обеспечивает гораздо большую безопасность, чем FTP, позволяя передавать данные, например, с использованием ассиметричного шифрования.

- Протокол SFTP использует одно соединение, а не два, как в FTP. Это делает работу в высоконагруженных системах более удобной, нежели в FTP, так как для передачи используется только один порт.

- Расширенный функционал. SFTP дополняет возможности FTP, позволяя например получать атрибуты файлов.

- Протокол SFTP не создавался для работы в консоли, ответы сервера не заточены для под конкретную операционную систему, а также предоставляется гораздо большеее количество данных, что позволяет

легко распарсить полученные данные и представить их в графическом виде.

Отрицательные стороны

- Протокол SFTP гораздо сложнее запрограммировать. Расширенный функционал и требования к безопасности ведут к повышенной нагрузке при написании функционала, что может занять большее количество времени у программистов и следовательно большие расходы при реализации и отладке собственного SFTP сервера.

- Из-за
  повышенной сложности, большего количество шагов для подготовки к передаче и кодировании данных возникают дополнительная нагрузка как на клиент, так и на сервер, вследствие чего передача файлов может работать медленнее.

Несмотря на повышенную вычислительную нагрузку а также стоимость разработки, протокол SFTP - это настоящее и будущее для манипуляции файловой системой сервера. В первую очередь, этот протокол обеспечивает высокую безопасность, что позволяет работать с сервером в публичной сети. Протокол FTP пусть и проще в исполнении и быстрее, однако безопасно может использоваться в хорошо изолированных локальных сетях.

2.3 SCP

Протокол SCP был также разработан The OpenSSH Project.

В 2019 коду протокол был признан OpenSSH как устаревший, в последней версии SCP используется SFTP для передачи файлов.

Отличительной способностью SCP является его относительная простота. Он также работает на SSH, однако предлагает гораздо более узкий функционах

Положительные стороны

- Безопасность. Протокол SCP работает на основе SSH, а значит может использовать надёжное ассиметричное шифрование.
- Протокол SCP гораздо менее громоздок по сравнению с SFTP. Он содержит одну единственную команду - скопировать указанный файл или получить его. Из-за этого его гораздо легче запрограммировать, он гораздо меньше нагружает как сервер так и клиент. Узкий спектр задач протокола позволяет решать конкретную задачу более эффективно.

Отрицательные стороны

- Уменьшенный функционал протокола привёл к некоторым проблемам с безопасностью. Так, согласно CVE-2019-6111 SCP-клиент не проверяет пути полученных файлов, и поэтому вредоносный SCP-сервер может перезаписать файлы на клиенте. SFTP протокол может принимать только целые пути, в то время как SCP клиент может выбрать файл по определённому паттерну.

Протокол SCP подходит, если нужно быстро передать файл от одного компьютера на другой. На данный момент этот протокол признан небезопасным и при использовании SCP используется SFTP.

2.4 HTTP/HTTPS

Протокол HTTP, так же как и FTP, прожил очень долгую историю развития, однако в отличие от своего современника, сегодняшний интернет без HTTP/HTTPS незаменим.

Именно при помощи HTTP сегодня происходит передача большинства файлов в интернете: загружаются миллионы HTML, CSS, JS, JSON, XML, медиафайлов.

HTTP/1.0 впервые был описан в 1997 году в RFC-2068. Он описывал формат передачи данных, заголовки, коды ответов. Также в HTTP/1.0 появилась важная особенность, позволяющая обмениваться данными - появилась возможность задавать MIME-тип ответа и размер ответа, что позволило передавать файлы как от клиента так и получать их от сервера. Также появился в дополнение к методу GET метод POST, который позволял редактировать состояние сервера.

HTTP/1.1 - самый известный на данный момент протокол. Он обеспечил необходимую безопасность при передаче данных, введя опциональное SSL/TLS ассиметричное кодирование данных (протокол HTTPS). Также новый стандарт добавил новые методы взаимодействия.

Положительные стороны

- Безопасность. Протокол поддерживает безопасное соединение с использованием шифрования, что позволяет безопасно передавать данные.
- Простота. В отличие от громоздкой настройки SFTP сервера, с HTTP сервером можно начинать работу без предварительной настройки.
- Большая гибкость. Обмен информации с использованием HTTP/HTTPS очень гибок и зависит исключительно от реализации веб-сервера.
- Широкая поддержка. На современном рынке существует большое количество браузеров, которые умеют эффективно и отлаженно выполнять HTTP запросы, обрабатывать их и использовать мощности современных устройств максимально эффективно. Также существуют браузеры и для старых систем, которые также могут использовать как безопасный так и небезопасный вариант HTTP. Существует большое количество фрейморков и библиотек, позволяющих создать свой сервер (FastAPI, Flask, Django, Spring, Poem, Express.js)

- Простота в реализации. Веб-сервер гораздо проще написать, чем FTP или SFTP сервер, что позволяет бизнесу создавать бюджетные быстроразвивающиеся решения.

Отрицательные стороны

- Отсутствие сессии.
  Преимущество всех вышеперечисленных протоколов являлось то, что пользователь всегда находился в контексте сессии, например у сессии была рабочая директория и пользователь без дополнительных средств проверки. HTTP лишён такой возможности, из-за чего программистам приходится обеспечивать эту сессию вручную, например при помощи Session-Cookie, JWT-токена и др.

- Необходимость разрыва соединения.
  После успешного получения ответа от сервера соединение немедленно разрывается в HTTP. Открытие нового сокета для общения с сервером требует дополнительных затрат.

HTTP/HTTPS - очень гибкий, простой и, следовательно, дешёвый протокол. Так же как в компьютерных сетях победил не совсем оптимальный но дешёвый Ethernet, в качестве прикладного протокола для обмена файлов сегодня лидирует HTTP.

3 Разработка клиент-серверного приложения

3.1 Разработка сервера

Одним из преимуществ протокола HTTP является его гибкость, возможность создавать любое количество команд и модифицировать их поведение.

Текущим решениям FTP этого не хватает, пусть они и являются надёжными и отлаженными, однако не предоставляют возможности кастомизации, переопределения поведения.

Именно на эти концепции и был сделан акцент при разработке FTP сервера, были позаимствованы некоторые паттерны современных веб-серверов.

Для разделения слоёв приложения было выделено три слоя: CtrlFTP-Core, CtrlFTP-Dependencies и сам сервер. Рассмотрим все три слоя ниже

3.1.1 Выбор стека

В качестве языка программирования был выбран язык программирования Java. Java очень давно находится на рынке, кроме того фреймворк Spring для Java сегодня используется многими компаниями для создания высоконагруженных веб-серверов, то есть Java множество лет использовалась для создания серверов.

В качестве ORM для проекта был выбран Hibernate, так как он является основным инструментом на выбранном языке программирования для манипуляирования базы данных.

В качестве системы сборки был выбран Gradle, так как он хорошо себя зарекомендовал на рынке как система для сборки проектов на Java.

3.1.2 Слой Core

Этот слой позволяет выстроить базовую работу сервера, которая позволит серверу определять команды, которые он должен выполнять, а также зависимости в командах.

Команды

CtrlFTP-Core содержит классы, которые позволяют создать свой сервер. Каждая команда создаётся при помощи аннотации `@Command`, которая находится в пакете `rchat.info.ctrlftp.core.annotations.Command`. Например:

```
public class ServiceService {
    @Command(name = "NOOP")
    public static Response noop() {
        return new Response(ResponseTypes.COMMAND_OK, "Glad to work with ya, human 'fella!");
    }
}
```

Таким образом можно определить FTP-команду NOOP, которая возвращает код OK (200) вместе с сообщением `Glad to work with ya, human 'fella!`. Каждая такая команда должна быть статична, а также должна возвращать класс Response из пакета `rchat.info.ctrlftp.core.responses`

Работа с аннотациями в Java происходит при помощи Reflections API. FTP сервер открывает сокет и ждёт сообщения. Когда это сообщение приходит, формируется новая виртуальная нить, которая обрабатывает запросы пользователя. Когда пользователь отправил сообщение, сопровождающееся переносом строки, сервер просканирует все доступные ему команды и будет искать команду с соответствующим именем. В этом примере, если пользователь пришлёт команду NOOP, сервер проанализирует все классы-контролеры и их методы, найдёт статический метод с именем NOOP (нечувствительно к регистру) и вызовет этот метод.

Метод должен вернуть Response, который сервер должен

перенаправить клиенту, выполнившему запрос.

Зависимости Сейчас метод noop достаточно скучен, он не может получать параметр а также оригинальную команду от пользователя.

Для таких случаев CtrlFTP предоставляет систему Dependency Injection, рассмотрим её на примере.

Каждый метод может принимать следующие зависимости:

- `String` - команда в ”сыром”виде передаётся в метод
- `Session` - класс текущей сессии из пакета `rchat.info.ctrlftp.core`
- `Server` - класс текущего сервера из пакета `rchat.info.ctrlftp.core`
- `AbstractDependency` - зависимость, разработанная самим разработчиком, мы рассмотрим этот механизм позже

Зависимость внедряется в метод когда мы передаём параметр в метод в случае команд, например так:

```
public class ServiceService {
    @Command(name = "NOOP")
    public static Response noop(String command) {
        return new Response(ResponseTypes.COMMAND_OK, "Glad to work with ya, human 'fella!");
    }
}
```

Теперь мы можем получить доступ к ”сырой”команде, которую отправил пользователь на сервер.

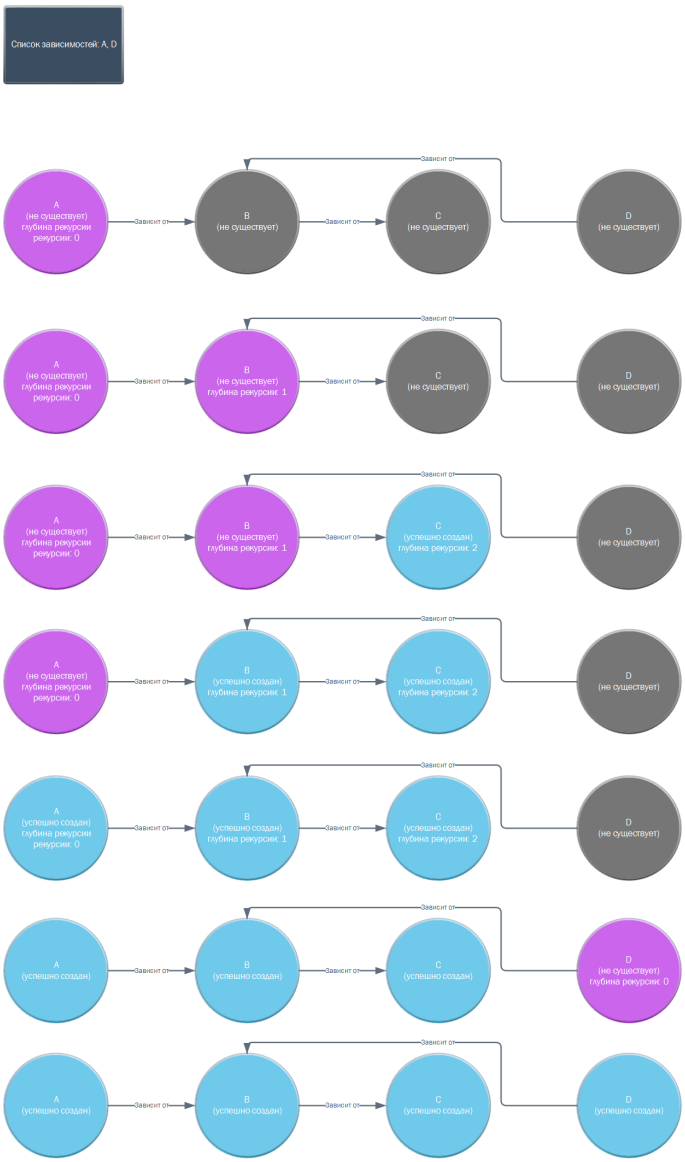
Библиотека CtrlFTP также позволяет написать собственные зависимости, которые также могут включать в себя другие зависимости, взаимодействовать с ними. Создать свою зависимость можно при помощи `@AbstractDependency` и аннотации `@Dependency` из пакета `rchat.info.ctrlftp.core.dependencies`. Аннотация принимает в качестве аргумента уровень зависимости, на уровне которого она будет существовать как объект. На данный момент доступно три уровня:

- `Global` - зависимость используется всё время существования сервера, уникальна для каждого сервера
- `Session` - зависимость используется всё время существования сессии, уникальна для каждой сессии
- `Command` - зависимость используется всё время существования команды, уникальна для каждой команды

Для создания зависимостей используется публичный конструктор, в котором, как и в команде, зависимости передаются в качестве параметров.

Есть и определённые ограничения, связанные с зависимостями разных уровней. Так, например, зависимость глобального уровня не может получить доступ к зависимости сессионного или командного уровня. В то время как зависимость командного уровня может обратиться к зависимости глобального уровня. Также нельзя ссылаться на саму себя, рекурсивные зависимости также недопустимы.

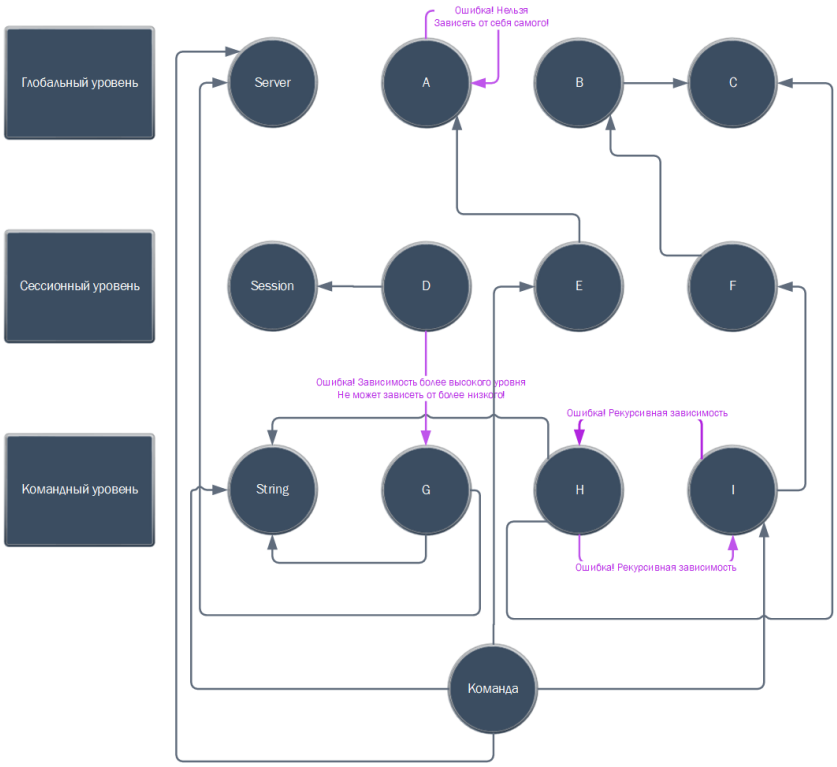
Текущие ограничения связаны также с работой Reflections API. Специальный класс `DependencyManager`, который существует на трёх уровнях - сессии, сервера и команды, выполняет функции внедрения необходимых зависимостей в команду и в другие зависимости. При поступлении списка аргументов менеджер зависимостей менеджер сначала проверяет, какой уровень у переданной зависимости. Если это зависимость его уровня, он проходит по списку всех своих зависимостей и ”запоминает”найденную зависимость. Иначе он пытается рекуррентно создать зависимость. То есть, берёт параметры из конструктора зависимости, снова вызывает метод, который ищет эти зависимости, конструирует объект-зависимость и сохраняет у себя, чтобы в дальнейшем использовать её.



Выбранный подход может привести к бесконечной рекурсии, поэтому метод отслеживает глубину своей рекурсии и выбрсывает исключение, если она слишком большая. Также при попытке сконструировать самого себя можно попасть в бесконечную рекурсию.

Если же зависимость находится уровнем выше, чем текущая зависимость, то менеджер зависимостей просит у родительского менеджера зависимостей найти эту зависимость. Если же запрашиваемая зависимость находится на нижнем уровне, метод выбросит исключение.

Менеджер зависимостей воспринимает класс строки как зависимость командного уровня, сессию как зависимость сессионного уровня и сервер как зависимость глобального уровня.



	String	Session	Server	Уровень команды	Уровень сессии	Уровень глобальный
Уровень команды	+	+	+	+	+	+
Уровень сессии	-	+	+	-	+	+
Уровень глобальный	-	-	+	-	-	+
Команда	+	+	+	+	+	+

Можно заметить, что сейчас наследование от AbstractDependency бесполезно - это пустой класс, ничего не делающий. Однако в будущем возможно он будет использован для пост- и предпроцессинга ввода/вывода.

Пример зависимости:

```
package rchat.info.ctrlftp.dependencies.deserializer;

/**
 * A dependency to parse single string after a command name
 */
```

```
public class SingleStringDeserializer extends BaseDeserializer<SingleStringDeserialized> {
    public SingleStringDeserializer(String command) {
        super(command);
    }

    @Override
    public SingleStringDeserialized deserialize(String command) {
        var firstSpaceIndex = command.indexOf(' ');

        return new SingleStringDeserialized(firstSpaceIndex == -1 ? "" :
        ↪  command.substring(firstSpaceIndex + 1));
    }
}
```

Эта зависимость принимает команду и оставляет только её аргумент, иногда очень полезная зависимость, если нужно извлечь только аргумент.

Запуск сервера Сервер можно запустить при помощи конструктора, который принимает конфигурационные файлы

```
package rchat.info.ctrlftp.examplebasic;

import rchat.info.ctrlftp.core.Server;

import java.io.IOException;
import java.util.List;

public class Main {
    public static void main(String[] args) throws ClassNotFoundException, IOException {
        var u = new Server(List.of("dependencies.xml"));
        u.mainLoop();
    }
}
```

Путь к конфигурационному файлу - это имя ресурса Java. Он представляет из себя XML-файл, который декларирует контролеры (сервисы) и зависимости для сервера. Именно в указанных классах сервер будет искать методы-команды и зависимости. Он может выглядеть так

```
<?xml version="1.0" encoding="utf-8"?>
<config>
    <services>
        <service>path.to.class.with.commands.ServiceClass</service>
```

```
        ...
    </services>
    <dependencies>
        <dependency>path.to.dependency.class.DependencyClass</dependency>
        ...
    </dependencies>
</config>
```

Файл конфигурации содержит корень `<config>`, который содержит в себе список сервисов `<services>` и список зависимостей `<dependencies>`.

3.1.3 Слой Dependencies

Пакет `rchat.info.ctrlftp.dependencies` предоставляет зависимости, которые могут быть полезны при разработке FTP сервера.

Аутентификация Класс `BaseAuthenticationDependency` позволяет проверить, аутентифицирован ли вошедший пользователь и позволяет ему выйти. Абстракция не включает в себя ввод логина и пароля, так как разработчик может захотеть позволить анонимный доступ к файловой системе. Метод `authenticate` возвращает `AuthenticationResult`, который содержит информацию о том, аутентифицирован ли пользователь, информацию об аутентификации и ошибку, если она произошла при проверке пользователя.

Десериализатор

Класс-десериализатор предназначен для расшифровки пользовательского ввода, парсинга команд. Класс `BaseDeserializer` расшифровывает полученную команду. Реализация расшифровки предоставлена разработчику. Есть также подготовленный заранее `SingleStringDeserializer`, который предназначен для частых случаев расшифровки единственного аргумента после названия команды.

Передача файлов Передача файлов является одним из ключевых функционалов работы FTP сервера. Класс

4 Работа приложения

Протестировать портал можно на адресе http://82.97.246.215/

Ссылка на репозиторий: https://github.com/IAmProgrammist/BookSTU

5 Вывод о проделанной работе

6 Список источников и литературы

1. Django Documentation [Электронный ресурс]
   Режим доступа: https://docs.djangoproject.com/en/5.1/

2. Django REST framework [Электронный ресурс]
   Режим доступа: https://www.django-rest-framework.org/

3. React [Электронный ресурс]
   Режим доступа: https://react.dev/

4. Redux Toolkit [Электронный ресурс]
   Режим доступа: https://redux-toolkit.js.org/rtk-query/overview

5. React Hook Form [Электронный ресурс]
   Режим доступа: https://react-hook-form.com/docs/useform

7 Список источников и литературы

1. Django Documentation [Электронный ресурс]
   Режим доступа: https://docs.djangoproject.com/en/5.1/

2. Django REST framework [Электронный ресурс]
   Режим доступа: https://www.django-rest-framework.org/

3. React [Электронный ресурс]
   Режим доступа: https://react.dev/

4. Redux Toolkit [Электронный ресурс]
   Режим доступа: https://redux-toolkit.js.org/rtk-query/overview

5. React Hook Form [Электронный ресурс]
   Режим доступа: https://react-hook-form.com/docs/useform