1 Тема 1. Введение в теорию распределенных вычислительных систем

1.1 Общая классификация систем обработки данных

Система обработки данных (СОД) — система, выполняющая автоматизированную обработку данных.

СОД (исходя из схемы Ларионова) **подразделяется на** класс сосредоточенных и распределенных систем.

Сосредоточенные системы включают в себя: отдельные ЭВМ, вычислительные системы и вычислительные комплексы.

Распределенные системы включают в себя: системы телеобработки и вычислительные сети — процессы обработки данных рассредоточены по многим компонентам.

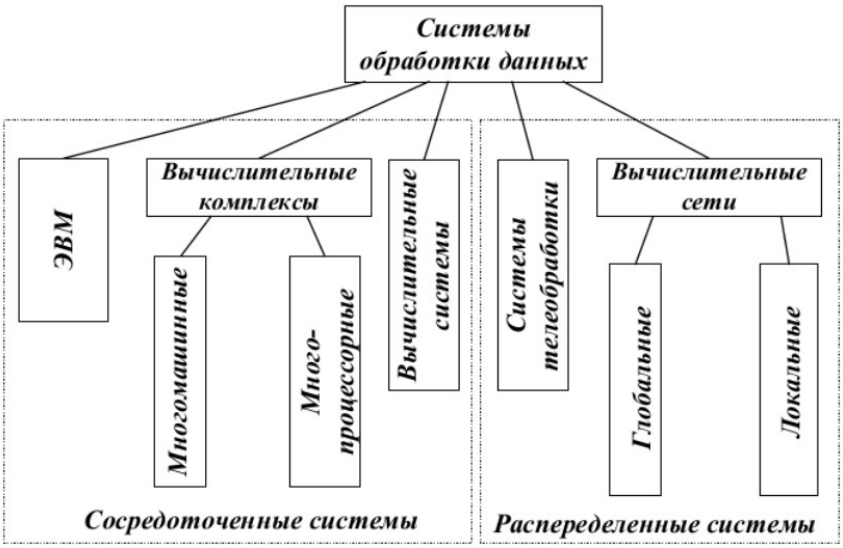

Рисунок 1.1 — Классификация СОД

1.1.1 Сосредоточенные системы

Сосредоточенные системы — часть СОД. **Включают в себя:** отдельные ЭВМ, вычислительные системы и вычислительные комплексы (многомашинные и многопроцессорные ВК).

Важнейшая характеристика сосредоточенных систем — быстродействие вычислений.

ЭВМ — одномашинная СОД.

Вычислительные комплексы — СОД построенные по одному из двух принципов: «Многомашинные вычислительные комплексы» или «Многопроцессорные вычислительные комплексы».

Многомашинные вычислительные комплексы — разновидность систем, которые включают несколько ЭВМ и предназначены для повышения надёжности и производительности СОД.

Многопроцессорные вычислительные комплексы — разновидность систем, которые включают более одного процессора, но используют общую память СОД.

Вычислительные системы (комплексы дополненные прикладным ПО) — СОД настроенные на решение задач конкретной области применения.

1.1.2 Распределенные системы

Распределенные системы — часть СОД. **Включают в себя:** системы телеобработки и вычислительные сети — процессы обработки данных рассредоточены по многим компонентам.

Системы телеобработки — системы, предназначенные для обработки данных, передаваемых по каналам связи. Потеряли свою актуальность и используются только в специализированных областях таких, как системы дальней космической связи, SCADA.

Вычислительные сети — совокупность ЭВМ, объединённых сетью (каналы и узлы связи) передачи данных. Сети подразделяются на локальные (LAN) и глобальные (WAN). Примером локальной сети является LAN — сеть компьютеров, сосредоточенных на небольшой территории.

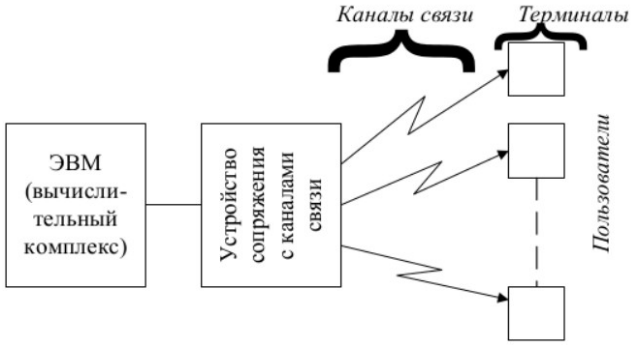

Рисунок 1.2 — Состав системы телеобработки данных

1.1.3 Распределенные вычислительные сети

Распределенные вычислительные сети (РВ-сети, вычислительные сети) **—** совокупность ЭВМ, объединённых сетью (каналы и узлы связи) передачи данных. Сети подразделяются на локальные (LAN) и глобальные (WAN). Примером локальной сети является LAN — сеть компьютеров, сосредоточенных на небольшой территории.

Для РВ-сетей характерно распределение функций, ресурсов между множеством элементов (узлов) и отсутствие единого управляющего центра, поэтому выход из строя одного из узлов не приводит к полной остановке всей системы.

Пример РВ-сетей является Интернет, компьютерная сеть, распределенные базы данных.

1.2 Сетевые объектные системы

Сетевые объектные системы — системы взаимосвязанных объектов в сети, в которые входит: узел, сеть (соединяющаяся логическим/физическим кабелем), зона (логическая группа из нескольких сетей) — все есть объекты! К примеру, через модель OSI описывается любое взаимодействие распределенных приложений в сети.

Подходы реализации распределенных приложений смещались в сторону **“Распределенных объектных систем РОС”**, **использующих**:

- классические модели, основанные на модели OSI;
- инструменты распределенной вычислительной среды (DCE);
- специальные сетевые системы: брокеры сообщений (CORBA);
- вызов методов удаленных объектов (RMI).

1.2.1 Классические приложения модели OSI

Любое взаимодействие распределенных приложений в сети описывается моделью OSI. **Модель определяет 7 уровней** взаимодействия систем (физический, канальный, сетевой, транспортный, сеансовый, представлений, прикладной уровень — 7). Каждый уровень выполняет определенные функции при таком взаимодействии.

Классические приложения модели OSI:

- telnet/ssh/PuTTY (сеансовый) — обеспечивает защищенную работу в сети;
- FTP (прикладной) — протокол, обеспечивающий передачу файлов по сети;
- HTTP (прикладной) — протокол передачи гипертекста. Приложения реализованы по общей архитектуре клиент-сервер.

1.2.2 Распределенная вычислительная среда (DCE)

Распределительная вычислительная среда DCE — технология, которая формирует программное окружение ОС. Разработана OSF. Является промежуточным ПО (middlewere). Использует модель клиент-сервер.

DCE предоставляет сервисы: служба каталогов, служба связанная со временем, служба проверки подлинности, файловая система — **и средства разработки** клиент-серверных приложений.

Удаленный вызов процедур RPC (запрос-ответ в клиент-серверной архитектуре) — важнейший технологический инструмент DCE. Класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве (на удаленных компьютерах). Особенностью реализации технологии RPC является необходимость создания программных заглушек как на стороне клиента, так и на стороне сервера.

Для упрощения разработки интерфейсов RPC часто используются **язык определения интерфейсов IDL**.

1.2.3 Технология CORBA

CORBA — общая архитектура брокера (посредника) объектных запросов — технологический стандарт написания распределенных приложений. Технология CORBA создана для поддержки разработки и развертывания сложных объектно-ориентированных прикладных систем.

CORBA — механизм в программном обеспечении, который дает возможность программам, написанным на разных языках программирования, работающим в разных узлах сети, просто взаимодействовать друг с другом.

GIOP — абстрактный протокол в стандарте CORBA, обеспечивающий способность к взаимодействию брокеров. Включает несколько протоколов: IIOP, SSLIOP, HTIOP.

1.2.4 Удаленный вызов методов RMI

RMI — программный интерфейс вызова удаленных методов в языке Java. Является простейшим инструментом реализации не очень сложных распределенных систем. Технологии RMI и CORBA заложили основы “Сервис-ориентированных технологий”.

1.3 Сервис-ориентированные системы

В конце 90-х годов появляется термин «сервис» и формируется парадигма: «все есть сервис».

Сервис-ориентированные архитектура SOA — подход к разработке ПО, в основе которого лежат сервисы со стандартизированными и простыми интерфейсами.

Разновидности сервисов:

- Platform-as-a-Service (PaaS) – платформа как услуга.
- Software-as-a-Service (SaaS) — программное обеспечение как услуга.
- Hardware-as-a-Service (HaaS) — аппаратное обеспечение как услуга.
- Data-as-a-Service (DaaS) – данные как услуга.

Общая тенденция этого движения привела к аутсорсингу.

Аутсорсинг — использование внешнего источника/ресурса. Передача организацией на основании договора определённых бизнес-процессов на обслуживание другой компании-специалисту.

1.3.1 Функции и сервисы

Понятие сервиса в ИТ интерпретируется с помощью модели SOA.

Сервис-ориентированная архитектура (SOA) — подход к разработке ПО, в основе которого лежат сервисы со стандартизированными интерфейсами.

С помощью SOA реализуется три аспекта ИТ-сервисов:

• **Сервисы бизнес-функций** — автоматизация компонентов конкретных бизнес-функций, необходимых потребителю.

• **Сервисы инфраструктуры.** Данные сервисы выполняют проводящую функцию, посредством платформы, через которую поставляются сервисы бизнес-функций.

• **Сервисы жизненного цикла —** сервисы жизненного цикла отвечают за дизайн, внедрение, управление, изменение сервисов инфраструктуры и бизнес-функций.

Сервис-ориентированный подход вводит такие понятия, как:

• Управление бизнес-процессами (BPM) — концепция процессного управления организацией;

• EAI — общее название сервиса интеграции прикладных систем;

• AOP — парадигма программирования, основанная на идее разделения функциональности программы на модули.

1.3.2 Системы middleware

В процессе создания и эксплуатации систем, созданных на основе подхода SOA, стало ясно, что необходима промежуточная (связующая) сетевая система.

Связующее программное обеспечение (middleware) — слой или комплекс технологического программного обеспечения для обеспечения взаимодействия между различными приложениями, системами. (к примеру, вызов удаленных процедур RPC).

SOAP — простой протокол доступа к объектам, протокол обмена структурированными сообщениями в распределенной вычислительной среде. На основе этого протокола реализуется SOA. Сейчас протокол используется для обмена произвольными сообщениями в формате XML.

1.3.3 Сервисные шины предприятий

Сервисная шина предприятия ESB — связующее программное обеспечение, обеспечивающее централизованный и унифицированный обмен сообщениями между различными информационными системами на принципах сервис-ориентированной архитектуры.

Основной принцип сервисной шины — концентрация обмена сообщениями между различными системами через единую точку, в которой обеспечивается транзакционный контроль, преобразование данных, сохранность сообщений.

По своей сути, позволяет создавать такие инфраструктуры, которые **скрывают детали реализации сетей**, определённых моделью OSI, **обеспечивая доступность**

только протоколов прикладного уровня, таким как: SMTP, FTP, HTTP, HTTPS и другим.

1.4 Виртуальные системы

Виртуальные системы — системы, предоставляющие набор вычислительных ресурсов и обеспечивающие при этом логическую изоляцию друг от друга вычислительных процессов, выполняемых на одном физическом ресурсе.

К ним относятся: виртуальные машины, виртуализация вычислительных комплексов на уровне ОС, виртуализация ПО на уровне языка, виртуальная машина языка Java.

1.4.1 Виртуальные машины

Виртуальная машина — программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой платформы и исполняющая программы или виртуализирующая некоторую платформу и создающая на ней среды, изолирующие друг от друга программы и даже операционные системы.

Таким образом, согласно модели СОД мы приходим к модели ЭВМ как совокупности аппаратных средств и программного обеспечения ОС, реализующего конкретную виртуальную машину — ЭВМ.

1.4.2 Виртуализация вычислительных комплексов на уровне ОС

Вычислительный комплекс — базовая основа вычислительной системы и содержит ОС, которая, реализуя виртуальную машину, позволяет рассматривать комплекс как отдельную ЭВМ.

Повышение эффективности работы ВС реализуются посредством:

• виртуализации многоядерных процессоров в виде отдельных вычислителей, объединённых общей памятью;

• виртуализации многомашинных систем средствами ПО MPI.

1.4.3 Виртуализация ПО на уровне языка

На уровне языка виртуализация ПО означает разные способы обращения к памяти.

ЭВМ, построенные на архитектуре фон-Неймана, породило большое количество систем адресации, с которыми возникали некоторые проблемы. Инструментом решения проблем является **именование** и **типизация** имён языковых конструкций, например:

• языки ассемблеров, посредством мнемоник команд, скрывают многие особенности методов адресации данных и размеров самих команд, перекладывая эти проблемы на компиляторы и линковщики программ;

• язык Fortran, ориентированный исключительно на вычисления, полностью освобождается от прямой адресации, но вынужден поддерживать типизацию и работу с массивами данных;

• в языке C — наоборот, вводится тип указателя, необходимый для решения системных задач и написания программ максимально заменяющих языки ассемблеров;

• языки, поддерживающие динамическую типизацию, например, Python, PHP, Perl, JavaScript и другие максимально скрывают саму типизацию, перекладывая решение вопросов адресации на интерпретаторы этих языков.

1.4.4 Виртуальная машина языка Java

Java Virtual Machine JVM (виртуальная машина Java) — основная часть исполняющей системы Java Runtime Environment JRE. Виртуальная машина Java исполняет байт-код Java, предварительно созданный из исходного текста Java-программы компилятором Java (javac).

Первая особенность: **полная независимость байт-кода от ОС и оборудования**, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью является **гибкая система безопасности**, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы, вызывают немедленное прерывание.

Часто **к недостаткам** концепции виртуальной машины относят **снижение производительности**. Для семи разных задач время выполнения на Java составляет в среднем в полтора - два раза больше, чем для C/C++, в некоторых случаях Java быстрее, а в отдельных случаях в 7 раз медленнее. С другой стороны, для большинства из них потребление памяти Java-машиной было в 10—30 раз больше, чем программой на C/C++.

2 Тема 2. Инструментальные средства языка Java

2.1 Общее описание инструментальных средств языка Java

Java Development Kit JDK — дистрибутив инструментальных средств для разработки приложений на языке Java, обязательно включающий в себя базовый дистрибутив JRE (среда исполнения Java-приложений). В пособии используется дистрибутив OpenJDK

2.1.1 Инструментальные средства командной строки

Две наиболее важные утилиты инструментальных средств языка Java являются:

• java — главная программа, запускающая виртуальную машину Java;

• javac — компилятор исходных текстов языка, входящий в дистрибутив JDK.

Дополнительные настройки инструментальной среды Java определяются с помощью четырёх переменных среды ОС:

• PATH — список директорий, где ОС будет искать исполняемые файлы для запуска;

• CLASSPATH — список директорий, где Java будет искать нужные библиотеки (пакеты библиотек);

• JRE_HOME — указывает на директорию инсталляции JRE;

• JAVA_HOME — указывает на директорию инсталляции JDK.

2.1.2 Пакетная организация языка Java

Важной особенностью языка Java является хорошая структуризация его системных программных средств на базе множества пакетов.

• java.lang — базовый пакет, обеспечивающий основные возможности языка объекты классы, исключения, математические функции, интерфейс с JVM и другие;

• java.util — инструментальный пакет классов: коллекции, дата, время;

• java.io — операции с файлами, потоковый ввод\вывод;

• java.math — набор функций: sin, cos и другие;

• java.net — операции с сетью, сокеты, URL;

• java.security — генерация ключей, шифрование и дешифрование;

• java.sql Java Database Connectivity (JDBC) — доступ к базам данных;

• java.awt — базовый пакет для работы с графикой;

• javax.swing — графические компоненты для разработки приложений: кнопки, текстовые поля.

2.1.3 Инструментальные средства Eclipse

Технологии IDE, которые связаны с реализацией проектов на языке Java:

• создание проекта выполняется командами: File→New→Java Project;

• добавление класса: Package Explorer→proj1/src→New→Class;

• оператор package задается правила хранения классов;

• архивация JAR производится в окне “Export”.

Завершив создание, мы получаем в среде IDE вкладку Example1.java с исходным текстом шаблона класса Example1.

В проекте находятся следующие директории:

• директория bin соответствует дереву каталогов для классов проекта;

• директория src соответствует дереву каталогов для исходных текстов проекта;

2.2 Классы и простые типы данных

Java является объектно-ориентированным языком программирования. Он поддерживает следующие основные понятия: полиморфизм; наследование; инкапсуляция;

Базовая конструкция языка Java:

public class Object,

— здесь Object адресуется как java.lang.Object и является суперклассом для всех других классов.

Имеет восемь простейших типов данных: boolean (?), byte (1), char (2), short (2), int (4), long (8), float (4), double (8).

2.2.1 Операторы и простые типы данных.

Имеет восемь простейших типов данных: boolean (?), byte (1), char (2), short (2), int (4), long (8), float (4), double (8). float и double имеют специальные значения бесконечности и NaN. Имеется набор зарезервированных слов (case, if, else, new, null, …). Имеется перечень операторов (+, *, -, <=, …). Есть неявное преобразование типов: double→float→long→int.

2.2.2 Синтаксис определения классов

[Экз. ответ]:

Кроме восьми простых типов данных, все остальные типы языка Java относятся к классам, а чтобы иметь возможность преобразования между ними, используются классы обертки (Boolean, Byte, …).

Определение класса: **[модификаторы] class имя [extends] [implements]**

Модификаторы: abstract, public, final.

extends: наследование.

implements: перечень интерфейсов.

[Полный ответ]:

[модификаторы] **class** имя-класса

[**extends** суперкласс]

[**implements** список_интерфейсов]

{

…

// переменные и методы класса

…

}

• class – слово, являющееся обязательным при объявлении класса;

• extends – слово, которое используется при объявлении нового класса, на основе уже созданного класса (суперкласса); объявляемый класс называется классом-потомком уже существующего класса;

• implements – слово, которое указывает, что класс поддерживает методы, определённые в последующем списке интерфейсов; отдельные элементы списка интерфейсов разделяются запятыми.

Объявляемый класс может не иметь модификатора, тогда «областью его видимости» является файл, в котором он определён. Если модификатор присутствует, то он может быть одним из трёх видов:

• abstract — класс, имеющий хотя бы один абстрактный метод (метод объявлен, но не имеет реализации); может иметь классы-потомки;

• public — открытые (публичные) классы, которые можно использовать: внутри программы, внутри пакета программ или за пределами пакета программ;

• final — класс, который не может иметь потомков; примерами таких классов являются: String и StringBuffer.

2.2.3 Синтаксис и семантика методов

[Экз. ответ]:

Определение метода: [спецификатор_доступа] [модификатор] [тип_данных] имя метода (аргументы) [throws исключения]

Спецификаторы: public, private, protected — область видимости методов/свойств.

Модификаторы: static, abstract, final, native, synchronized — особенность использования методов.

[Полный ответ]:

[спецификатор_доступа]

[static] [abstract] [final] [native] [synchronized] // модификаторы

тип_данных имя_метода

([параметр1], [параметр2], ...)

[**throws** список_исключений]

Спецификатор доступа — может отсутствовать, но в любом случае он определяет область видимости не только методов, но и переменных:

• public — имеется видимость переменной константы или метода из любого класса;

• private — доступ разрешается только внутри класса;

• protected — доступ разрешается как внутри класса, в котором даётся определение, так и внутри классов-потомков;

• <пробел> - доступ разрешается для любых классов пакета, в котором класс определён.

Модификаторы — определяют особенности использования методов, если к ним имеется доступ:

• static — метод, принадлежащий классу и используемый всеми объектами класса; обращение к методу производится указанием имени класса и, после точки, указывается метод (оператор new указывать не нужно);

• abstract — объявляет метод, но не даёт его реализации;

• final — метод нельзя будет переопределять (перегружать) в классах-потомках;

• native — определяет метод, реализованный на других языках, отличных от языка Java, например, - на языке C;

• synchronized — при обращении к методу, доступ к остальным методам класса прекращается до завершения работы данного метода.

Тип_данных — возвращаемый методом тип результата, к которым относятся: типы объявленных классов, простейшие типы данных или void, если метод ничего не возвращает.

throws список_исключений — если в методе используются классы или ситуации, приводящие к исключениям, то указывает список необрабатываемых в методе исключений; имена исключений в списке разделяются запятыми.

2.2.4 Синтаксис определения интерфейсов

[Экз. ответ]:

Интерфейсы — специальные типы классов, альтернатива абстрактным типам классов, чтобы расширить их возможности (допускают объявление объединений других интерфейсов).

Общий синтаксис объявления интерфейсов: [public] interface имя [extends] { }

public — необязательный модификатор доступа.

extends — перечень интерфейсов.

[Полный ответ]:

Интерфейсы — специальные типы классов. были предложены как альтернатива абстрактным типам классов, чтобы расширить возможности последних. Интерфейсы можно интерпретировать как незавершённые классы, подобные абстрактным классам,

но в отличие от последних они допускают объявление объединений других интерфейсов. Как показано выше, список интерфейсов может присутствовать при объявлении класса, подключая к классу методы, которые должны быть в нем реализованы. Общий синтаксис объявления интерфейсов имеет вид:

[**public**] **interface** имя_интерфейса

[**extends** интерфейс1, интерфейс2, ...]

{

[тип_переменной1 имя_переменной1 = значение1;]

[тип_переменной2 имя_переменной1 = значение2;]

...

[метод_доступа] тип_метода название_метода1([тип_аргумента1 аргумент1], [тип_аргумента2 аргумент2], ...);

[метод_доступа] тип_метода название_метода2([тип_аргумента1 аргумент1], [тип_аргумента2 аргумент2], ...);

...

}

В определении интерфейса может присутствовать необязательный модификатор public и два ключевых слова:

• interface – слово, являющееся обязательным при объявлении интерфейса

• extends – слово, которое используется при объявлении нового интерфейса, включающего один или более уже объявленных интерфейсов.

Тело интерфейса может содержать:

• объявленные и инициализированные типы данных;

• описание не реализованных методов.

2.2.5 Объекты и переменные

[Экз. ответ]:

Переменные — это область памяти, в которую мы можем сохранить данные, осуществлять доступ к этим данным или изменять их. Все переменные являются объектными ссылками.

Объекты — это область памяти, которая содержит поля и методы класса. Создаются только динамически.

[Полный ответ]:

В языке Java имеются только динамически создаваемые объекты и все переменные языка являются объектными ссылками. Но в Java отсутствует понятие ссылки и операции адресации.

```
package ru.tusur.asu;
public class Example2 {
        // Объявление переменной типв String
        String text1;
```

```
        // Объявление статического массива из двух целых чисел
        static int[] im = new int[2];
        // Конструктор класса
        Example2(String text2, int n) {
                // Присваиваем значение части переменных
                text1 = text2;
                im[0] = n;
        }
        // Первый (обычный) метод
        public void print1() {
                System.out.println(text1 + ":"
                + " im[0]=" + im[0]
                + " im[1]=" + im[1]);
        }
}
```

2.3 Управляющие операторы языка

Оператор if, switch, цикла while, do … while, цикла for, перехода (break, continue, return).

2.4 Потоки ввода-вывода

В языке Java для организации ввода-вывода используются специальные классы, которые должны агрегироваться во вновь определяемые классы и создаваемые объекты.

Технология программирования на языке Java **использует два общих подхода**:

• методы стандартного ввода-вывода;

• обобщенные методы, основанные на двух классах InputStream и OutputStream.

2.4.1 Стандартный ввод/вывод

[Экз. ответ]:

Стандартный ввод/вывод — осуществляет ввод/вывод информации.

Методы стандартного ввода-вывода определяются объектами в пакете java.lang: InputStream, PrintStream.

Объекты ввода/вывода являются статическими и финальными (не имеют возможности иметь дочерние классы).

[Полный ответ]:

Методы стандартного ввода-вывода определяются объектами в пакете java.lang:

• public static final InputStream in — объект ввода in порожден из класса InputStream и имеет три основных метода: читает по одному байту поток и возвращает число, читает содержимое потока и помещает в массив, читает len байт и помещает в массив b со смещением off;

• public static final PrintStream out — порожден классом OutputStream.PrintStream, имеет методы: print, println, write (выводит массив байт), write (выводит len байт из массива b);

• public static final PrintStream err — также, как и out;

Таким образом, объекты стандартного ввода-вывода являются статическими, что уже отмечалось в рассмотренных примерах, и финальными, т.е. — не имеющими возможность иметь дочерние классы.

Программируя ввод-вывод, необходимо всегда помнить, что многие методы порождают исключения IOException, поэтому, определяя собственные методы, необходимо проектировать их обработку или игнорирование.

2.4.2 Классы потоков ввода

[Экз. ответ]:

Классы потоково ввода: InputSream, отраженный в специальном пакете java.io.

Объектам класса InputStream доступны все методы, которые также доступны стандартному вводу.

На практике используются методы: available, skip, close.

[Полный ответ]:

Для решения общих задач ввода-вывода язык Java использует специальный пакет java.io, обладающий набором классов и интерфейсов, позволяющих управлять процессами внутри различных потоков. Базовым классом для методов ввода является InputStream.

InputStream имя_объекта = new Конструктор_одного_из_классов_ввода

Объектам класса InputStream доступны все методы, которые также доступны стандартному вводу. На практике используют еще методы: available (число байт, доступных для чтения в потоке ввода), skip (попускает во входном потоке N байт), close (закрывает входной поток ввода).

Поскольку входной буфер всегда ограничен в размере, то реальное чтение происходит блоками переменной длины.

2.4.3 Классы потоков вывода

Для реализации общих задач вывода информации в языке Java используется базовый класс OutputStream, также принадлежащий пакету java.io.

OutputStream имя_объекта = new Конструктор_одного_из_классов_вывода

Наиболее значимые методы OutputStream :

write(int b),
write(byte[] b),
write(byte[], int off, int len),
flush(принудительный сброс данных из промежуточного буфера в поток вывода) — перед закрытием всегда нужно сбрасывать, close().

2.5 Управление сетевыми соединениями

Управление сетевыми соединениями — осуществляется с помощью специального пакета java.net, содержащий базовые классы и методы для работы со стеком протоколов TCP/IP.

Содержанию ПО пакета java.net:

• адресация в Internet, основанная на классе InetAddress;
• адресация в Internet на основе классов URL и URLConnection;
• сокеты протокола TCP;
• сокеты протокола UDP;

2.5.1 Сетевая адресация языка Java

[Экз. ответ]:

Адресация в языке Java производится на базе класса InetAddress или URL и URLConnection.

Объекты класса InetAddress — используются для указания адреса. Имеют два собственных метода: getHostAddress(), getHostName().

Объекты URL — класс для непосредственной работы с URL-адресами. Реализованы в пакете java.net.

Объекты URLConnection — класс, методы которого обеспечивают процессы загрузки ресурсов Internet.

[Полные ответ]:

Адресация на базе класса InetAddress

В языке Java в качестве адресов используются объекты класса InetAddress, а привычные нам адреса Internet — в качестве строковых аргументов в трех статических методах этого класса: getLocalHost (создает объект адреса для локального компьютера), getByName(String host), getAllByName(String host) — создается массив объектов адресов для hosts.

Созданные объекты класса InetAddress используются в методах других классов, например в классах сокетов, но также имеют два собственных метода: getHostAddress(), getHostName().

Адресация на базе URL и URLConnection

[Экз. ответ]:

Для работы с web-протоколами такими как, http, ftp и другими, в пакете java.net имеются классы:

• URL — класс для непосредственной работы с URL-адресами; объекты класса URL имеют ряд важных методов: getProtocol(), getHost(), getPort(), getFile(), openConnection(), openStream.

• URLConnection — класс, методы которого обеспечивают процессы загрузки ресурсов Internet и их информационную поддержку с использованием объектов класса URL. Имеет следующие методы: getContentLength(), getContentType(), getDate().

[Полные ответ]:

Поддерживается следующая схема URL-адресов protocol://host:port/file, которая обеспечивается тремя конструкторами класса URL:

• URL(String saddr);

• URL(String protocol, String host, String file);

• URL(String protocol, String host, int port, String file).

Объекты класса URL имеют ряд важных методов:

• getProtocol(), getHost(), getPort(), getFile() - возвращают значения компонент соответствующего URL-адреса;

• openConnection() - возвращает объект класса URLConnection;

• openStream() - возвращает объект входного потока класса InputStream.

Для манипулирования адресуемыми ресурсами используется объект класса URLConnection, который можно получить командой:

URLConnection ucon = url.openConnection();

где url — объект класса URL, имеющий множество методов. Например:

• connect() - устанавливает соединение с ресурсом, если оно ещё не было установлено, или вызывает исключение, если соединение — невозможно;

• getContentLength() - возвращает размер ресурса;

• getContentType() - возвращает тип содержимого адресуемого ресурса;

• getDate() - возвращает дату загрузки;

• getExpiration() - возвращает срок хранения;

• getPermission() - определяет параметры доступа к ресурсу;

• getInputStream() - возвращает объект потока ввода класса InputStream.

2.5.3 Сокеты стека протоколов TCP/IP.

[Экз. ответ]:

Стек протоколов TCP/IP — сетевая модель данных.

Для организации взаимодействия между двумя программами по технологии клиент-сервер по протоколу **TCP** пакет java.net предоставляет два класса:

• Socket — класс для создания объектов клиентского сокета;

• ServerSocket — класс создающий объекты, используемые серверными программами для организации соединения с программами клиентов.

Имеются важные методы:

accept — соглашение на соединение;

getInputStream/Output — создание потоков для возможности обмена данными;

read/write — методы объектов потоков для обмена сообщениями.

Для асинхронного взаимодействия по протоколу **UDP** используется класс DatagramSocket.

Его основные методы: get, send, receive.

[Полные ответ]:

Сокеты протокола TCP

Для организации взаимодействия между двумя программами по протоколу TCP пакет java.net предоставляет два класса:

• Socket — класс для создания объектов клиентского сокета;

• ServerSocket — класс создающий объекты, используемые серверными программами для организации соединения с программами клиентов.

Чтобы программа-клиент могла соединиться с сервером, она должна знать адрес и порт сервера, а затем создать объект клиентского сокета командой:

Socket client = new Socket(InetAddress address, int port);

Чтобы программа-сервер могла принимать соединения от клиентов, она должна создать объект типа ServerSocket командой:

ServerSocket server = new ServerSocket(int port);

Получив запрос на соединение, программа-сервер должна согласиться на него и создать объект клиентского сокета командой:

Socket client = server.accept();

Для возможности непосредственного обмена данными обе программы, и клиент и сервер, должны создать входные и выходные потоки, используя классы пакета java.io:

InputStream in = client.getInputStream();

И методы объектов класса Socket:

OutputStream out = client.getOutputStream();

Далее, обмен данными между программами выполняется через объекты потоков ввода-вывода с помощью соответствующих методов read(...) и write(...).

Описанная выше технология обмена данными между двумя программами называется — технология клиент-сервер, а метод взаимодействия — синхронным. Алгоритм, по которому осуществляется обмен данными — протоколом.

Сокеты протокола UDP

Для организации асинхронного взаимодействия между программами пакет java.net предоставляет классы DatagramSocket и DatagramPacket, реалиованные на транспортном уровне протокола UDP.

Сокет протокола UDP создаётся объектом класса DatagramSocket, который имеет три конструктора:

DatagramSocket();

DatagramSocket(int port);

DatagramSocket(int port, InetAddress addr);

Каждый объект класса DatagramSocket имеет следующие основные методы:

• getInetAddress() - возвращает адресс, к которому осуществляется подключение;

• getPort() - возвращает порт, к которому осуществляется подключение;

• getLocalAddress() - возвращает локальный адрес компьютера, с которого осуществляется подключение;

• getLocalPort() - возвращает локальный порт, через который осуществляется подключение;

• send(DatagramPacket pack) — передача пакета;

• receive(DatagramPacket pack) - приём пакета.

Каждый объект класса DatagramPacket имеет следующие основные методы:

• getAddress() и setAddress() - возвращает или устанавливает адрес подключения;

• getPort() и setPort() - возвращает или устанавливает порт подключения;

• getLength() и setLength() - возвращает или устанавливает размер дейтаграммы;

• getData() - возвращает массив байт содержимого дейтаграммы;

2.6 Организация доступа к базам данных

Доступ к базам данных осуществляется посредством СУБД, которая обслуживает запросы клиента.

Общий доступ приложений к СУБД осуществляется за счет классов и методов пакета java.sql. Пример СУБД: Apache Derby.

2.6.1 Инструментальные средства СУБД Apache Derby

Apache Derby — реляционная СУБД, написанная на языке Java, предназначенная для встраивания в Java-приложения. Если используется в серверном варианте, необходимо определиться с адресом и портом (1527), по которым СУБД будет принимать соединения.

Работа СУБД обеспечивается интерактивной **утилитой ij.** Она позволяет создавать и удалять базы данных, делать к ним различные SQL-запросы и выводить результаты на стандартный ввод-вывод.

SQL-запросы и драйверы баз данных

JDBC (соединение с базами данных) — инструментальное средство, предназначенное для взаимодействия СУБД в языке Java. JDBC — платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета java.sql.

JDBC Driver — группа интерфейсов, для взаимодействия с базой данных.

JDBC DriverManager — Для манипулирования драйверами и подключения к СУБД, пакет java.sql содержит класс DriverManager со статическими методами: registerDriver, deregisterDriver, getDrivers, getConnection.

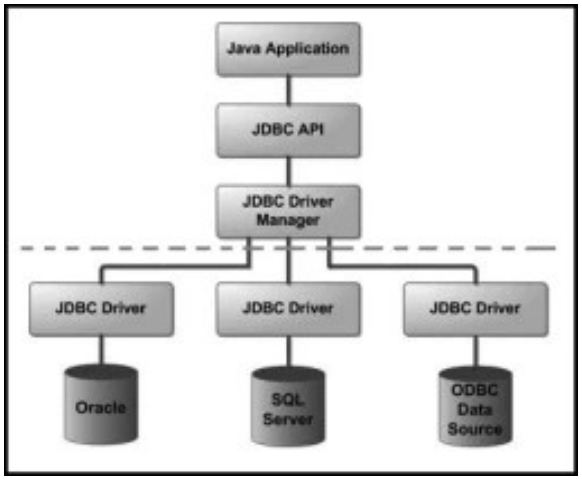

Рисунок 2.1 — Инструментальное средство JDBC

----------дальше для экза

3 Тема 3. Объектные распределенные системы

3.1 Брокерные архитектуры

Брокер — ПО-посредник (служба промежуточного уровня middleware).

Классическая архитектура: модель “клиент-сервер”, на базе TCP протокола пакета java.net. Брокер принимает запросы от программы-клиента, передает их программе-серверу, получает ответ от сервера и передает его клиенту. Для взаимодействия клиента с базой данных используется брокер в лице СУБД.

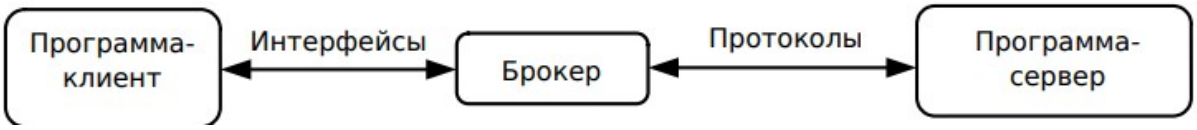

Рисунок 3.1 — Брокерная архитектурная модель взаимодействия “Клиент-сервер”

Брокерная модель нацелена на упрощение технологии создания программ-клиентов. Она предоставляет клиентским программам наборы интерфейсов. Таким образом, логика реализации программ-клиентов сводится к технологиям вызова удалённых процедур (RPC).

Пример брокерной технологии: прокси-сервер (сервер-посредник) — промежуточный сервер (комплекс программ) в компьютерных сетях, выполняющий роль посредника между пользователем и целевым сервером (при этом о посредничестве могут как знать, так и не знать обе стороны), позволяющий клиентам как выполнять косвенные запросы (принимая и передавая их через прокси-сервер) к другим сетевым службам, так и получать ответы. Дополняется технологией языков описания интерфейсов IDL.

3.1.1 Вызов удалённых процедур (архитектура)

RPC (remote procedure call) — технология удаленного вызова процедур. Основу составляет синхронная схема взаимодействия программ клиента и сервера. ПО брокера распределялось между программами клиента и сервера в виде программ-заглушек (рисунок 3.2), которые реализовывали все протоколы взаимодействия.

Основной недостаток устаревшего RPC — ограниченное время жизни вызываемой процедуры на стороне сервера. В условиях принципиальной ненадёжности взаимодействия в сети, возникают проблемы подтверждения результата или ошибок исполнения вызываемых процедур.

RPC и RMI — это механизмы, которые позволяют клиенту вызывать процедуру или метод с сервера посредством установления связи между клиентом и сервером.

Общее различие между RPC и RMI состоит в том, что RPC поддерживает только процедурное программирование, тогда как RMI поддерживает объектно-ориентированное программирование.

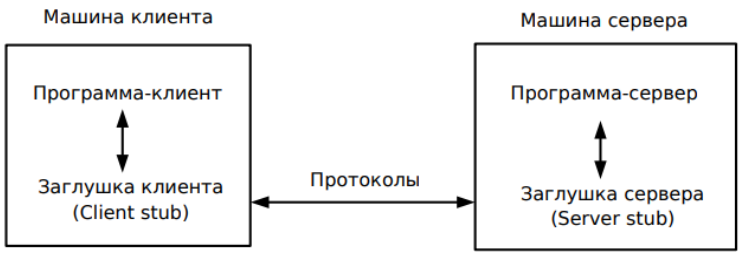

Рисунок 3.2 — Брокерная модель RPC

3.1.2 Использование удалённых объектов

RMI — механизм вызова метода удаленного объекта (который находится на другом сервере). Серверно-ориентированная модель. Специфичная для Java. Для реализации используется интерфейс Java. Удаленный объект представляет собой некоторые данные и с ними можно работать посредством некоторого интерфейса программ-заглушек.

В общем случае считается, что удалённый объект (Remote object), расположенный на сервере, имеет:

• Состояние (State) — данные инкапсулируемые (включаемые) объектом.

• Методы (Methods) — операции над данными состояния объекта.

• Интерфейс (Interface) — программный код, обеспечивающий доступ к методам объекта.

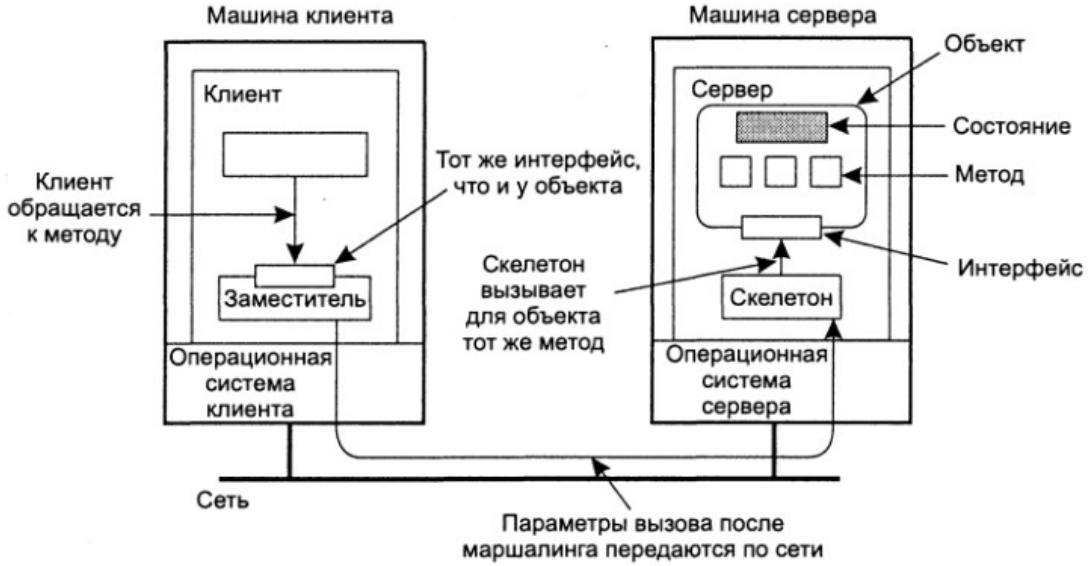

Рисунок 3.3 — Обобщенная организация удаленных объектов

Описание работы заглушек RMI:

• на стороне программы-клиента создаётся объект-заглушка (stub, proxy), реализующий методы маршалинга: преобразования имени объекта, вызываемого метода и его аргументов в поток данных, передаваемых по протоколам серверу;

• на стороне программы-сервера создаётся объект-заглушка (skeleton), реализующий методы демаршалинга: преобразования входного потока данных в запрос к объекту сервера.

3.2 Технология CORBA

CORBA — объектно-ориентированная технология создания распределенных приложений. Технология основана на использовании брокера объектных запросов (ORB). Технология позволяет строить приложения из распределенных объектов, реализованных на различных языках программирования. Одноранговая система

Имеет собственный абстрактный протокол GIOP, на основе которого разработаны три интернет протокола: IIOP, SSLIOP, HTIOP.

Шесть этапов проектирования распределенных приложений:

1. Общее описание брокерной архитектуры CORBA.
2. Серверная часть будущего распределенного объекта.
3. Клиентская часть будущего распределенного объекта.
4. Генерация базового описания распределенного класса OrbPad, который демонстрирует использование инструментального средства языка IDL.
5. Пример реализации удаленного объекта.
6. Пример реализации клиентской части.

3.2.1 Брокерная архитектура CORBA

CORBA — объектно-ориентированная технология создания распределенных приложений.

Базовые принципы CORBA:

• независим от физического размещения объекта;

• независим от платформы;

• независим от языка программирования.

Главная особенность CORBA — использование компонента ORB (брокер объектных запросов), формирующего “мост” между программой-клиентом и программой-сервером.

Имеет собственный абстрактный протокол GIOP, на основе которого разработаны три интернет протокола: IIOP, SSLIOP, HTIOP.

Интерфейс объекта CORBA создается на языке IDL.

Для разработчиков приложений наиболее важными являются **шестнадцать общих объектных служб, являющиеся ядром CORBA-систем** или сервисами: служба имён, управления событиями, жизненных циклов, устойчивых состояний, транзакций, времени, безопасности, уведомлений…

CORBA обеспечивает разработчиков сложных приложений инструментами, облегчающими создание приложений клиентов на разных языках программирования. Для этого предоставляется универсальный язык описания интерфейсов (IDL) и

специализированные компиляторы, которые освобождают программистов от деталей реализации сетевого взаимодействия посредством генерации необходимого набора компонент как для для клиентской, так и серверной частей распределенных приложений.

3.2.4 Генерация распределенных объектов OrbPad

OrbPad — интерфейс распределенного объекта. С помощью него, к примеру, можно взаимодействовать с базами данных (insert, delete, …).

Работа по созданию (генерации) распределенных объектов **начинается с** описания интерфейсов, которые удалённый объект (созданный на сервере) будет предоставлять программам-клиентам. Технология CORBA требует, чтобы используемые интерфейсы были описаны на языке IDL, независящим от языка реализации. Затем, это описание компилируется в конкретный язык реализации.

Язык Java имеет собственный компилятор **idlj, обеспечивающий преобразование** формального описания IDL в набор файлов с шаблонами исходных текстов на языке Java.

3.2.5 Реализация серверной части ORB-приложения

Реализация серверной часть ORB-приложений — обеспечивает интерфейс удалённого объекта OrbPad.

Простейший вариант **архитектуры** такого приложения состоит из двух классов:

• класса сервера — организующего приём запросов от клиентов и отправку им результатов выполненных методов;

• класса серванта (слуга) — реализующего только методы объявленного интерфейса. Его задача: создать объекты NotePad и ORB, передать их серванту OrbPadServer (сервант), создать и зарегистрировать удаленный объект, запустить цикл приема запросов от программ клиентов.

Сервер должен ожидать приёма запросов от программ клиентов.

3.2.6 Реализация клиентской части ORB-приложения

Приложение клиента реализуем в виде класса OrbPadClient:

Создаем и инициализируем ORB. Получаем доступ к серверу по его имени. Получаем доступ к объекту. Клиент объекта имеет доступ к объектной ссылке и вызывает операции объекта. Клиент знает только логическую структуру объекта в соответствии с его интерфейсом и может наблюдать за поведением объекта через вызовы методов.

3.3 Технология RMI

Технология RMI — упрощенный вариант CORBA (специализированный для языка Java). Современная реализация технологии RMI, основанная на собственном протоколе JRMP (Java Remote Method Protocol), не требует генерации «стабов» (client stub) или «скелетонов» (server stub, sceleton), хотя использует своего брокера rmiregistry как службу «Сервиса имён». В такой интерпретации, общая организация технологии RMI может быть представлена рисунком 3.4.

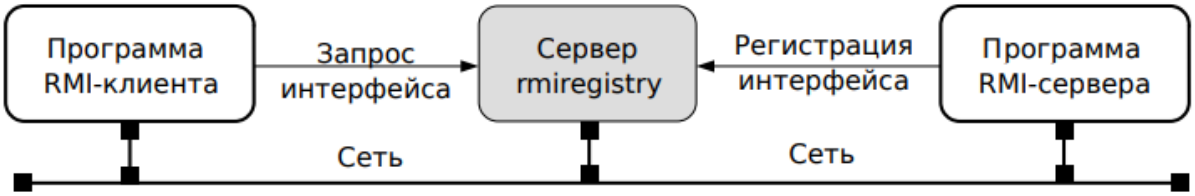

Рисунок 3.4 — Общая организация технологии RMI

Сервер службы имет rmiregistry — брокер, является центральным местом организации технологии RMI.

Соответственно, технология проектирования распределенного приложения сводится к трём основным этапам:

1. Написание интерфейса распределенного приложения.

2. Написание программы сервера, реализующей функциональность удалённого объекта и обеспечивающей способность регистрации интерфейса объекта на сервере rmiregistry.

3. Написание программы клиента, реализующей функциональность локального приложения и обеспечивающей способность получения интерфейса удалённого объекта с сервера rmiregistry.

3.3.1 Интерфейсы удалённых объектов технологии RMI

Интерфейсы RMI — описываются классическими интерфейсами языка Java, которые расширяют интерфейс java.rmi.Remote.

Описание интерфейсов с помощью IDL и java.rmi.Remote схожи, за исключением:

• удалён метод void setClose(), поскольку его отсутствие не уменьшает функциональности приложения, а назначение является достаточно одиозным для удалённых объектов, разрешающих клиентским приложениям останавливать сервера;

• аргументы методов, передающих строковые значения, заменены на «родной» для языка Java тип — String;

• метод getList() теперь возвращает массив строк (тип String[]), что не требует при его реализации и использовании дополнительных преобразований, связанных с упаковкой/распаковкой содержимого.

3.3.2 Реализация RMI-сервера

Реализация отдельного RMI-сервера обычно состоит из двух компонент на языке Java, содержащих:

• описание интерфейса;

• реализацию интерфейса и исходного текста самого сервера, обеспечивающего регистрацию интерфейса на сервере rmiregistry и принимающего запросы программ клиентов;

Сервер во время своего запуска выполняет три базовых действия:

Создаем объект взаимодействия с базой данных → передаем его удаленному объекту → регистрируем удаленный объект на сервере rmiregister.

Запущенный RmiPadServer будет ожидать запросы от клиентских программ по некоторому произвольному порту и внешнему адресу компьютера, которые хранятся на сервере rmiregistry. Клиентские RMI-программы будут обращаться к серверу имён и получать от него координаты доступа к RmiPadServer. Все это скрыто внутри технологии RMI.

3.3.3 Реализация RMI-клиента

Создадим класс с именем RmiPadClient: создаем и инициализируем объект локального клиентского класса RMI → получаем объектную ссылку на удаленных объект методом lookup → обрабатываем запросы.

Основное отличие RMI-клиента от текста ORB-клиента заключается в том, что RMI-клиент подключает интерфейс удалённого объекта всего лишь одним методом lookup() класса Naming. Все остальные изменения связаны только с различными типами разных технологий и приложений.

4 Тема 4. Web-технологии распределенных систем

4.1 Общее описание технологии web

WEB (всемирная паутина) — распределенная система, предоставляющая доступ к связанным между собой документам, расположенным на различных компьютерах, подключенных к Интернету. Распределенная система рассматривается как некий “механизм” по перемещению “ресурсов” (чего угодно) с одной машины на другую.

Для реализации механизма **нужно три технологии:**

1. Адресация ресурсов — гибкий и расширяемый способ именования произвольных ресурсов (URI).

2. Представление ресурсов — ресурсы представлены в виде потока бит и передаются по сети (HTML).

3. Передача ресурсов — протоколы, поддерживающие минимально необходимый набор операций передачи данных (HTTP).

4.1.1 Унифицированный идентификатор ресурсов (URI)

URI — символьная строка, позволяющая идентифицировать какой-либо ресурс (документ, изображение, файл), с помощью которого осуществляется адресация ресурсов web.

URI имеет две формы представления:

1. URL — система унифицированных адресов электронных ресурсов. Определяет, где и как найти ресурс.

2. URN — постоянная последовательность символов, идентифицирующая абстрактный или физический ресурс. Определяет, как ресурс идентифицировать.

Недостатки адресации URL — хорошо известны:

• использование ограниченного набора ASCII-символов, делающего нечитаемыми слова национальных языков;

• сильная привязка к стеку протоколов TCP/IP, требующая указания адреса сети и порта транспортного соединения;

• наличие привязок к тексту HTML и методам протокола HTTP.

4.1.2 Общее распределение ресурсов (HTML)

Средством представления ресурсов распределенных систем сети Интернет является HTML.

HTML — стандартизированный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML.

Он первоначально **обеспечивал**:

• предоставление ссылок (адресацию) на различные ресурсы Интернет;

• форматирование текста, включая представление списков и таблиц;

• отображение рисунков в формате .gif;

• интерактивные средства взаимодействия клиента с удаленным приложением сервера с помощью, например, GET и POST протокола HTTP.

Существенным событием web-технологий — стала возможность включения в текст языка HTML объектов языка Java, известных как апплеты.

Java апплет — прикладная программа в формате байт-кода. Апплеты используются для предоставления интерактивных возможностей веб-приложений, которые не могут быть представлены HTML.

4.1.3 Протокол передачи гипертекста (HTTP)

Протокол передачи гипертекста HTTP — протокол прикладного уровня передачи данных. В настоящий момент используется для передачи произвольных данных. Основой HTTP является технология “клиент-сервер”.

HTTP включает в себя достаточно большой набор стандартизированных методов:

• GET — основной метод запроса ресурсов в браузерах;

• POST — дополнительный метод запроса ресурса, используемый в браузерах с помощью специальных конструкций <FORM>;

• HEAD — метод, по форме запроса аналогичный методу GET, но ответ сервера не содержит тела ресурса;

• PUT — аналогичен методу POST, но содержит более детальный диалог;

• DELETE — предназначен для удаления ресурсов.

4.2 Модель клиент-сервер

Модель “клиент-сервер” — архитектура РВ-сети, которая делится на клиента и сервер. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов.

1. Клиент — конечный потребитель результатов вычислений, инициирует сетевое взаимодействие.

2. Сервер — пассивная часть, обслуживающая одного или множество клиентов.

4.2.1 Распределение приложений по уровням

Трехзвённая архитектура вертикального типа построения РВ-сетей. Входят:

• верхний уровень представления (пользовательского интерфейса) — ПО поддержки пользовательского интерфейса, расположенное на машине клиента;

• средний уровень бизнес-логики (функциональная поддержка приложений) – функциональная часть приложения, реализованная на сервере, но не включающая хранилище используемых данных, которое реализовано как отдельная система.

• нижний уровень данных (накопление, хранение и извлечение данных) — это сервер или набор серверов, содержащих программы, которые хранят и предоставляют данные программам уровня бизнес-логики. Фактически — это сервера, содержащие базы данных управляемые некоторыми СУБД.

4.2.2 Типы клиент-серверной архитектуры

Однозвенная архитектура — модель, где все прикладные программы находятся на рабочих станциях, которые обращаются к общему серверу баз данных или к общему файловому серверу. Никаких прикладных программ сервер при этом не исполняет, только предоставляет данные. [клиент: прикладное ПО; сервер: предоставляет данные]

Двухзвенная архитектура — прикладные программы находятся на сервере приложений, а в рабочих станциях находятся программы-клиенты, которые предоставляют для пользователей интерфейс для работы с приложениями на общем сервере. [клиент: программы-клиент; сервер: прикладное ПО]

Трёхзвенная архитектура — сервер баз данных, файловый сервер и другие представляют собой отдельный уровень, результаты работы которого использует сервер приложений. Логика данных и бизнес-логика находятся в сервере приложений. Все обращения клиентов к базе данных происходят через промежуточное программное обеспечение (middleware), которое находится на сервере приложений. Вследствие этого, повышается гибкость работы и производительность. [клиент: программа-клиент; сервер: сервер приложений + middleware; сервер баз данных]

Распределение предполагает разбиение компонентов на мелкие части и последующее разнесение этих частей по системе.

Вертикального типа распределения приложений — размещение на разных машинах логически разных компонент приложения.

Горизонтальное распределение — когда логически одинаковые компоненты клиентов и серверов размещаются на разных машинах. Центральный элемент — «Сервер распределения» — распределяет нагрузку запросов от множества клиентов по множеству серверов.

4.3 Технология Java-сервлетов

Сервлет — программа, которая работает на сервере и взаимодействует с клиентом по принципу запрос-ответ.

Или иначе говоря — интерфейс Java, который расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ. Технология Java Servlet определяет HTTP-специфичные сервлет классы.

Пакеты javax.servlet и javax.servlet.http обеспечивают интерфейсы и классы для создания сервлетов.

Благодаря сервлетам **появилась возможность** проектировать полноценные графические приложения, используя только функциональные возможности браузеров.

Apache Tomcat — web-сервер, контейнер сервлетов. Tomcat позволяет запускать веб-приложения. Tomcat используется в качестве самостоятельного веб-сервера, в качестве сервера контента в сочетании с веб-сервером Apache HTTP Server.

— таким образом, **был создан полный набор инструментов,** позволяющих развивать web-технологии Java, даже без ориентации на технологию апплетов.

Жизненный цикл сервлета:

1. Запуск: tomcat загружает класс сервлета → создает объект класса сервлета → вызывает метод init() сервлета.

2. Обслуживание запросов: tomcat получает запрос → определяет какому сервлету предназначен → передает запрос в метод service() → для каждого запроса создается свой поток.

3. Прекращение работы сервлета: tomcat вызывает метод destroy().

4.3.1 Классы Servlet и HttpServlet

Класс Servlet — публичный интерфейс, который реализован в пакете javax.servlet. Содержит описание пяти методов:

• destroy — вызывается контейнером сервлета, чтобы указать сервлету, что он выводится из эксплуатации;

• getServletConfig — возвращает объект ServletConfig, который содержит параметры инициализации и запуска для этого сервлета.

• getServletInfo — возвращает информацию о сервлете, такую как автор, версия и авторские права.

• init — вызывается контейнером сервлета, чтобы указать сервлету, что он вводится в эксплуатацию.

• service — вызывается контейнером сервлета, чтобы сервлет мог ответить на запрос.

Класс HttpServlet — предоставляет абстрактный класс, разделяющийся на подклассы для создания HTTP-сервлета, подходящего для веб-сайта. Предназначен для расширения возможностей сервлета. Подкласс HttpServlet должен переопределить хотя бы один метод, обычно один из следующих:

• doGet(...) — если сервлет поддерживает запросы HTTP GET;

• doPost(...) — для запросов HTTP POST;

• doPut(...) — для запросов HTTP PUT;

• doDelete(...) — для запросов HTTP DELETE;

• init(...) и destroy(...) — чтобы управлять ресурсами, которые управляют жизненным циклом сервлета;

• getServletInfo(...) — когда необходимо предоставить информацию о себе

Любой сервлет, который создаёт проектировщик является обычным публичным Java-классом, который расширяет абстрактный класс HttpServlet.

Жизненный цикл сервлета состоит из трёх периодов:

1. Когда сервер стартует, то загружает доступные ему сервлеты. При этом, каждый сервлет выполняет метод init(...). При необходимости, проектировщик может переопределить этот метод.

2. В процессе работы сервера, сервлет выполняет методы, которые запрашивает клиент, кроме методов init(...) и destroy(...). При необходимости, проектировщик переопределяет нужные методы, обычно — методы doGet(…) и doPost(...).

3. Когда сервер завершает работу, он для каждого сервлета вызывает его метод destroy(...). При необходимости, проектировщик может переопределить этот метод.

4.3.2 Контейнер сервлетов Apache Tomcat

Apache Tomcat — web-сервер, классическое представление контейнера сервлетов. Обеспечивает поддержку жизненного цикла сервлета. Tomcat позволяет запускать веб-приложения. Tomcat используется в качестве самостоятельного веб-сервера, в качестве сервера контента в сочетании с веб-сервером Apache HTTP Server.

Минимальная настройка дистрибутива Apache Tomcat требует задания системной переменной **CATALINA_HOME**. Минимальная проверка работоспособности дистрибутива Apache Tomcat осуществляется с помощью сценария **startup.sh**. После старта, сервер Apache Tomcat начинает прослушивать порт **8080**. Это отличает его от обычных web-серверов, которые по умолчанию прослушивают порт 80. Соответственно, для остановки запущенного сервера используется сценарий **shutdown.sh**.

4.3.3 Диспетчер запросов — RequestDispatcher

Диспетчер запросов RequestDispatcher — интерфейс, используется для внутренней коммуникации между сервлетами в одном контексте.

RequestDispatcher реализует два метода :

• forward — передача запроса/управления другому ресурсу на сервере;

• include — включение контента дополнительного ресурса в ответ.

Сервлет получает запрос от браузера в виде объекта request определённого интерфейсом HttpServletRequest и проводит его анализ. После анализа запроса, разработчик должен привязать к объекту request некоторый ресурс сервера, который должен быть передан клиенту. Это делается с помощью реализации объекта интерфейса RequestDispatcher.

4.3.4 Технология JSP-страниц

JSP (JavaServer Pages) — технология, которая используется для разработки веб-страниц путем вставки Java-кода в HTML-страницы с помощью специальных тегов JSP. В сравнении с Servlet: обычно используется, когда нет большой обработки данных. Медленнее. Может принимать только HTTP запросы.

— технология, позволяющая веб-разработчикам создавать содержимое, которое имеет как статические, так и динамические компоненты. Позволяют отделить динамическую часть страниц от статического HTML. Страница JSP содержит текст двух типов: статические исходные данные, которые могут быть оформлены в одном из текстовых форматов HTML, SVG, WML, или XML, и JSP-элементы, которые конструируют динамическое содержимое.

Элементы JSP: выражение, скриплет, объявление, директива, комментарий, действие.

Директивы JSP — распространяются на всю структуру класса, в который компилируется страница.

JSP-действие — является удобным, когда у нас имеются уже готовые HTML-страницы, которые можно включить в JSP-страницу. Например, файлы Title.html и post1.html.

Объявления, выражения, скриплеты — вставляют код языка Java в JSP-страницу.

4.3.5 Модель MVC

MVC — шаблон проектирования, основанный на принципе разделения представления данных и функционала, где эти данные формируются и обрабатываются.

Главная цель MVC: отделить модель (бизнес-логику) от представления (того, что видет пользователь). Разделяются посредством контроллера.

MVC состоит из: модель [Model]; вид (представление) [View]; контроллер [Controller].

Модель — поставщик доступа к данным, главные задачи модели заключаются в представлении доступа к данным для их просмотра или актуализации.

Представление — выполняется вывод данных на экран. При классическом подходе к web-разработкам в представлении будет формироваться HTML код.

Контроллер — компонент MVC, предназначенный для связи между моделями и представлениями, а также для обработки данных, которые пришли от пользователя к серверу через формы запроса и другие источники.

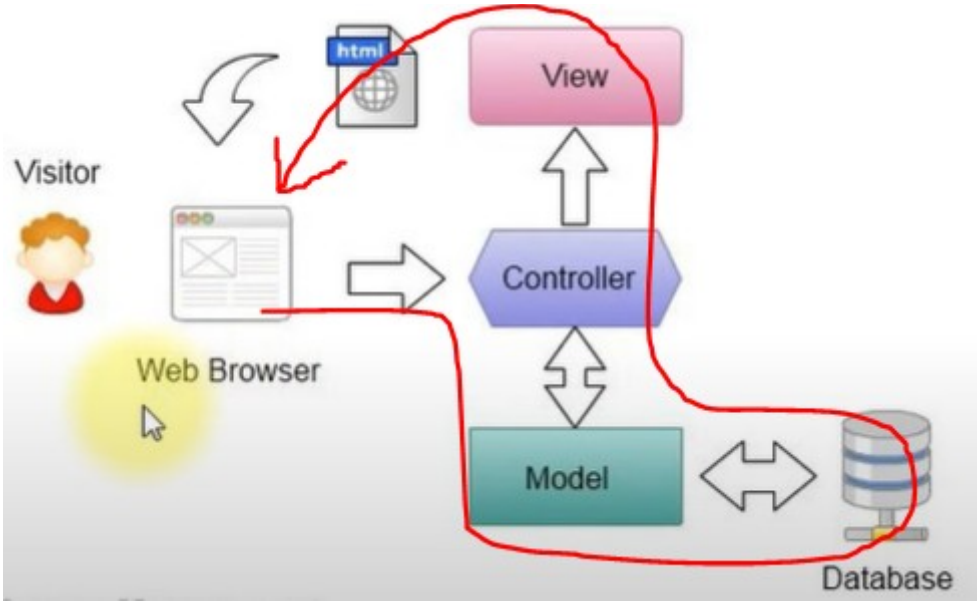

Рисунок 4.3 — MVC Web model

5 Тема 5. Сервис-ориентированные архитектуры

5.1 Концепция SOA

Сервис-ориентированная архитектура SOA — модель взаимодействия компонент, которая связывает различные сервисы между собой с помощью четко определяемых интерфейсов.

В самом общем виде SOA предполагает наличие трех основных участников: поставщика сервиса, потребителя сервиса и реестра сервисов. Взаимодействие участников выглядит достаточно просто: поставщик сервиса регистрирует свои сервисы в реестре, а потребитель обращается к реестру с запросом.

Базовыми составляющими SOA: сервисные компоненты, интерфейс сервиса, соединитель сервисов, механизмы обнаружения сервисов.

• Сервисные компоненты (или сервисы) — описываются программными компонентами, которые обеспечивают прозрачную сетевую адресацию.

• Интерфейс сервиса — обеспечивает описание возможностей и качества предоставляемых сервисом услуг. В таком описании определяется формат сообщений, используемых для обмена информацией, а также входные и выходные параметры методов, поддерживаемых сервисным компонентом.

• Соединитель сервисов — это транспорт, обеспечивающий обмен информацией между отдельными сервисными компонентами.

• Механизмы обнаружения сервисов — предназначены для поиска сервисных компонентов, обеспечивающих требуемую функциональность сервиса. Среди всего множества вариантов, обеспечивающих обнаружения сервисов, выделяются две основные категории: системы динамического и статического обнаружения.

5.1.1 Связывание распределенных программных систем

Связанность сервисов — одна их характеристик РВ-сетей.

Подразделяются на два типа:

• Сильносвязанные системы — зависимый класс содержит ссылку на класс предоставляющий сервис.

• Слабосвязанные системы — зависимый класс содержит ссылку на интерфейс, который реализован различными классами.

5.1.2 Web-сервисы первого и второго поколений

Первое поколение веб-сервисов (до 2007 года) — опиралось на парадигму XML веб-служб, позволяющих создавать независимые масштабируемые слабосвязанные приложения. Для этого использовалось **три основных стандарта**: WSDL, UDDI и SOAP, образующие так называемый **«Треугольник SOA».**

UDDI — инструмент для расположения описаний веб-сервисов (WSDL) для последующего их поиска другими организациями и интеграции в свои системы. Использует специальное хранилище (репозитория), где предприятия и организации могут размещать данные о предоставляемых ими сервисах.

WSDL — язык описания веб-сервисов и доступа к ним, основанный на языке XML.

Качественные характеристики веб-сервисов первого поколения:

• контент сервисов формируется поставщиками сервисов;

• отсутствуют единые стандарты протоколов авторизации и аутентификации пользователей, что требовало от поставщиков сервиса проводить самостоятельные разработки;

• отсутствует понятие состояния сервиса и после обращения клиента к серверу, его состояние на сервере не сохраняется

Второе поколение веб-сервисов (Начиная с 2004 года) — начинают публиковаться и внедряться различные стандарты, повышение качество работы веб-сервисов. К ним относятся:

• WS-Security – обеспечение безопасности веб-сервисов;

• WS-Addressing – маршрутизация и адресация SOAP-сообщений;

• WSRF, WS-Notification – работа с состоянием веб-сервисов.

Качественные характеристики:

• контент сервисов формируется пользователями сервисов;

• используются единые стандарты протоколов авторизации и аутентификации пользователей;

5.1.3 Брокерные архитектуры Web-сервисов

Прямой вызов через UDDI — каждый раз, когда потребитель хочет вызвать службу, он должен запросить URI оконечных точек в службе UDDI. Такой подход также вынуждает потребителя самостоятельно оценивать качество провайдера (поставщика сервиса) каким-либо способом.

Синхронный вызов через посредника — предполагает использование «интеллектуального брокера», который способен самостоятельно проводить оценку провайдеров сервиса на предмет их присутствия в сети, качества обслуживания пользователей и загруженности запросами. Потребитель вызывает прокси-службу такого брокера, который, в свою очередь, оценивает загруженность провайдеров, выбирает одного из них и передаёт UDDI. Он знает только о том, что может использовать этот URI для вызова Web-службы.

Асинхронный вызов через посредника — потребитель использует два асинхронных канала связи с брокером. По одному каналу передаёт запрос брокеру, который ставит его в свою очередь запросов. По другому каналу, потребитель асинхронно забирает ответ из очереди ответов брокера.

5.2 Частные подходы к реализации сервисных технологий

Три технологических направлений, которые напрямую не ассоциируют себя с веб-сервисами, но в практическом плане являются реализациями сервисных технологий. **К ним относятся:** технологии одноранговых сетей, технологии Grid и облачные вычисления

5.2.1 Технологии одноранговых сетей

Иначе говоря, одноранговая сеть — компьютерная сеть, основанная на равноправии участников. Часто в такой сети отсутствуют выделенные серверы, а каждый узел (peer) является как клиентом, так и выполняет функции сервера. В отличие от архитектуры клиент-сервера, такая организация позволяет сохранять работоспособность сети при любом количестве и любом сочетании доступных узлов.

Технология одноранговых или P2P сетей — обеспечивает взаимодействие приложений РВ-сетей на базе принципа децентрализации, когда разделение вычислительных ресурсов и сервисов производится напрямую посредством прямого взаимодействия между участниками сети друг с другом.

5.2.2 Технологии GRID

Грид-вычисления (англ. grid — решётка, сеть) — это форма распределенных вычислений, в которой «виртуальный суперкомпьютер» представлен в виде кластеров, соединённых с помощью сети, слабосвязанных гетерогенных компьютеров, работающих вместе для выполнения огромного количества заданий (операций, работ).

Уровни архитектуры GRID:

1. Базовый уровень (Fabric) – содержит различные ресурсы, такие как компьютеры, устройства хранения, сети, сенсоры и другие.
2. Связывающий уровень (Connectivity) – определяет коммуникационные протоколы и протоколы аутентификации.
3. Ресурсный уровень (Resource) – реализует протоколы взаимодействия с ресурсами РВС и их управления.
4. Коллективный уровень (Collective) – управление каталогами ресурсов, диагностика, мониторинг.
5. Прикладной уровень (Applications) – инструментарий для работы с Grid и пользовательские приложения.

5.2.3 Облачные вычисления

Облачные вычисления (cloud computing) — технология распределенной обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как Интернет-сервис.

Облачная обработка данных — это парадигма, в рамках которой информация постоянно хранится на серверах в Интернет и временно кэшируется на клиентской стороне, например, на персональных компьютерах, игровых приставках, ноутбуках, смартфонах и тому подобных устройствах.

Конкретные модели облачных моделей: частное, публичное, гибридное, общественное облако.