**1. Как организовать клиент-серверное взаимодействие?**

Клиент – программа, которая запрашивает сервер выполнить то или иное

действие (задачу) и вернуть полученные данные клиенту.

Клиент подключается к порту, выделенному на сервере.

Алгоритм работы клиента выглядит следующим образом:

1) Клиент пытается соединиться с сервером по известному адресу и

порту. Cокеты реализуются на основе класса Socket и предназначены для отправки данных на определённый сервер

2) Клиент создает поток чтения сообщений от сервера.

3) Клиент передает сообщение серверу.

4) Клиент, при получении команды от сервера, выполняет ее.

Сервер – это специальная программа, обычно запущенная на отдельном

компьютере, которая ожидает от клиентских программ запросы и

предоставляет им свои ресурсы в виде данных или сервисных функций. На

сервере для работы нужно выделить порт.

Алгоритм работы сервера выглядит следующим образом:

1) Сервер открывает порт для подключения.

2) Сервер создает поток чтения сообщений от вновь подключенных

клиентов.

3) Сервер, при получении команды, выполняет ее и пересылает ответ

клиенту

**2. Как и в каком виде передавать файлы?**

Создайте соединение между клиентом и сервером. После подключения найдите файл, который вам нужно отправить клиенту. Затем отправьте размер файла клиенту. Затем отправьте файл, разбитый на части.

на клиентском После создания соединения запросите файл. Получите размер файла, затем примите данные до тех пор, пока u не достигнет размера файла.

1. Создается экземпляр потока

2. Поток открывается (для чтения или записи)

3. Производится чтение из потока/запись в поток

4. Поток закрывается

Для работы с файлами существуют два специализированных класса FileInputStream и FileOutputStream. У обоих есть методы read и write, которые считывают и пишут байт.

При небольших размерах файлов все работает вполне прилично. Но если вы попробуете скопировать файл хотя бы в несколько мегабайт это может происходить крайне долго т.к. чтение происходит по 1 байту. Это крайне неэффективно.

**3. Как пересылать большие файлы?**

На помощь придет буферизованные версии чтения/записи. Т.е. Вы считываете сразу несколько килобайт за один раз. Это радикально повышает производительность. Для этого надо использовать другой вариант вызова методов read и write — он принимает в качестве входного параметра массив байтов. Т.е. при чтении поток сразу заполняет весь массив (или его часть) и делает операцию сразу для большого количества — читает или пишет.

**4. Как пересылать служебные команды?**

Не понял вопроса

**5. Что хранить в базе данных?**

-Учетные данные пользователей

- Права на доступ к папкам

- Путь к файлам

**6. Как передавать структуру каталогов/файлов?**

Класс File, определенный в пакете java.io, не работает напрямую с потоками. Его задачей является управление информацией о файлах и каталогах. Хотя на уровне операционной системы файлы и каталоги отличаются, но в Java они описываются одним классом File.

В зависимости от того, что должен представлять объект File - файл или каталог, мы можем использовать один из конструкторов для создания объекта:

|  |  |
| --- | --- |
|  | File(String путь\_к\_каталогу)  File(String путь\_к\_каталогу, String имя\_файла)  File(File каталог, String имя\_файла) |

Например:

|  |  |
| --- | --- |
|  | // создаем объект File для каталога  File dir1 = new File("[C://SomeDir](file:///C:\SomeDir)");  // создаем объекты для файлов, которые находятся в каталоге  File file1 = new File("[C://SomeDir](file:///C:\SomeDir)", "Hello.txt");  File file2 = new File(dir1, "Hello2.txt"); |

Класс File имеет ряд методов, которые позволяют управлять файлами и каталогами. Рассмотрим некоторые из них:

* **boolean createNewFile()**: создает новый файл по пути, который передан в конструктор. В случае удачного создания возвращает true, иначе false
* **boolean delete()**: удаляет каталог или файл по пути, который передан в конструктор. При удачном удалении возвращает true.
* **boolean exists()**: проверяет, существует ли по указанному в конструкторе пути файл или каталог. И если файл или каталог существует, то возвращает true, иначе возвращает false
* **String getAbsolutePath()**: возвращает абсолютный путь для пути, переданного в конструктор объекта
* **String getName()**: возвращает краткое имя файла или каталога
* **String getParent()**: возвращает имя родительского каталога
* **boolean isDirectory()**: возвращает значение true, если по указанному пути располагается каталог
* **boolean isFile()**: возвращает значение true, если по указанному пути находится файл
* **boolean isHidden()**: возвращает значение true, если каталог или файл являются скрытыми
* **long length()**: возвращает размер файла в байтах
* **long lastModified()**: возвращает время последнего изменения файла или каталога. Значение представляет количество миллисекунд, прошедших с начала эпохи Unix
* **String[] list()**: возвращает массив файлов и подкаталогов, которые находятся в определенном каталоге
* **File[] listFiles()**: возвращает массив файлов и подкаталогов, которые находятся в определенном каталоге
* **boolean mkdir()**: создает новый каталог и при удачном создании возвращает значение true
* **boolean renameTo(File dest)**: переименовывает файл или каталог

**Работа с каталогами**

Если объект File представляет каталог, то его метод isDirectory() возвращает true. И поэтому мы можем получить его содержимое - вложенные подкаталоги и файлы с помощью методов list() и listFiles(). Получим все подкаталоги и файлы в определенном каталоге:

|  |  |
| --- | --- |
|  | import java.io.File;  public class Program {  public static void main(String[] args) {  // определяем объект для каталога  File dir = new File("[C://SomeDir](file:///C:\SomeDir)");  // если объект представляет каталог  if(dir.isDirectory())  {  // получаем все вложенные объекты в каталоге  for(File item : dir.listFiles()){  if(item.isDirectory()){  System.out.println(item.getName() + " \t folder");  }  else{  System.out.println(item.getName() + "\t file");  }  }  }  }  } |

Теперь выполним еще ряд операций с каталогами, как удаление, переименование и создание:

|  |  |
| --- | --- |
|  | import java.io.File;    public class Program {  public static void main(String[] args) {  // определяем объект для каталога  File dir = new File("[C://SomeDir//NewDir](file:///C:\SomeDir\NewDir)");  boolean created = dir.mkdir();  if(created)  System.out.println("Folder has been created");  // переименуем каталог  File newDir = new File("[C://SomeDir//NewDirRenamed](file:///C:\SomeDir\NewDirRenamed)");  dir.renameTo(newDir);  // удалим каталог  boolean deleted = newDir.delete();  if(deleted)  System.out.println("Folder has been deleted");  }  } |

**Работа с файлами**

Работа с файлами аналогична работе с каталога. Например, получим данные по одному из файлов и создадим еще один файл:

|  |  |
| --- | --- |
|  | import java.io.File;  import java.io.IOException;  public class Program {  public static void main(String[] args) {  // определяем объект для каталога  File myFile = new File("[C://SomeDir//notes.txt](file:///C:\SomeDir\notes.txt)");  System.out.println("File name: " + myFile.getName());  System.out.println("Parent folder: " + myFile.getParent());  if(myFile.exists())  System.out.println("File exists");  else  System.out.println("File not found");    System.out.println("File size: " + myFile.length());  if(myFile.canRead())  System.out.println("File can be read");  else  System.out.println("File can not be read");    if(myFile.canWrite())  System.out.println("File can be written");  else  System.out.println("File can not be written");  // создадим новый файл  File newFile = new File("[C://SomeDir//MyFile](file:///C:\SomeDir\MyFile)");  try  {  boolean created = newFile.createNewFile();  if(created)  System.out.println("File has been created");  }  catch(IOException ex){    System.out.println(ex.getMessage());  }  }  } |

При создании нового файла метод createNewFile() в случае неудачи выбрасывает исключение IOException, поэтому нам надо его отлавливать, например, в блоке try...catch, как делается в примере выше.