day11 网络编程

UDP 和 TCP 协议特点

UDP: 用户数据报协议 1. 无连接的不可靠的协议 2. 数据以包的形式发送, 一个包 64K 3. 速度快效率高, 容易丢包
TCP: 传输控制协议 1. 有连接的可靠协议 2. 直接传输数据没有大小限制 3. 速度慢, 安全性高

总结:

UDP: User Datagram Protocol, 用户数据报协议
特点:
1. 无连接的不可靠协议
2. 数据按包发送, 64K 一个包
3. 速度快效率高, 容易丢包
用于视频直播, 网络电话
TCP: Transmission Control Protocol, 传输控制协议
特点:
1. 需要建立连接的可靠协议 电话
2. 数据传输无大小限制
3. 速度慢效率低 重发机制
用于文件下载, 浏览网页
TCP 通信的三次握手: TCP 协议中, 在发送数据的准备阶段, 客户端与服务器之间的三次交互, 以保证连接的可
靠
1. 客户端向服务端发送验证信息, 等待服务器确认
2. 服务端收到验证信息后, 回复客户端验证信息, 同时发送自己的一条验证信息
3. 客户端收到服务端回复的信息, 确认自己之前发的信息无误, 并再次向服务器发回服务端的验证信息

ServerSocket

java.net.ServerSocket 类: TCP 服务端
// 构造方法
ServerSocket(int port): 创建一个 TCP 服务端, 并监听指定端口
// 成员方法
Socket accept(): 监听数据, 会阻塞. 收到数据后返回 Socket 对象
void close(): 关闭服务端 ServerSocket

Socket

java.net.Socket 类: TCP 客户端
// 构造方法
Socket(String ip, int port): 创建 TCP 客户端对象
// 成员方法
OutputStream getOutputStream(): 获取输出流对象, 用于发送数据
InputStream getInputStream(): 获取输入流, 用于接收数据
void shutdownOutput(): 关闭输出流, 告知服务端数据发送完毕
void close(): 关闭客户端 Socket

网络编程三要素:

1. 通信协议: 怎么传输
2. IP 地址: 传给哪个主机
3. 端口号: 传给主机的哪个进程

端口号

端口号: 计算机中进程的唯一标识
端口号的取值范围: 0~65535 的整数, 其中 0~1024 不建议使用
注意:
通信的两端是 2 个计算机中的 2 个程序在相互通信, 所以 2 个程序都要有端口号. 端口号可以相同, 也可以不同,
相互之间能识别就行

IP 地址

IP 地址: 互联网协议地址(Internet Protocol Address). 是网络中计算机的唯一标识
版本:
IPv4: 192.168.1.2
IPv6: ABCD:EF01:2345:6789:ABCD:EF01:2345:6789
特殊的 IP 地址: "127.0.0.1", "localhost", 都代表自己的电脑
常用 DOS 命令:
// 查看自己电脑的 IP 地址
ipconfig
// 查看是否能连接到指定 IP 地址
ping IP 地址
ping 192.168.31.2

TCP 协议下文件上传案例

客户端
读取本地图片文件
发送到服务端
读取服务端的响应结果
服务端
接收客户端的文件数据
写入到服务端磁盘
发送响应给客户端

软件结构

C/S: Client Server 客户端 服务端
B/S: Browser Server 浏览器 服务端

day12 函数式接口

@FunctionalInterface 注解

@FunctionalInterface 作用: 检测接口是否符合函数式接口的要求, 不符合就编译报错 用在哪: 在接口声明的上面
@FunctionalInterface public interface 接口名{}

几个常用的函数式接口

Supplier

java.util.function.Supplier<T>函数式接口: 生产型函数式接口
// 抽象方法
T get(): 用于获取一个对象或值. 至于获取什么值, 怎么获取, 需要我们根据应用场景编写 Lambda 来实现

Consumer

java.util.function.Consumer<T>函数式接口: 消费型函数式接口
// 抽象方法
void accept(T t): 用于消费(使用)一个对象或值. 至于怎么消费, 需要我们根据应用场景编写 Lambda 来实

现
// 默认方法
default Consumer<T> andThen(Consumer<? super T> after): 拼接两个 Consumer 接口的 Lambda 对象实
现连续操作. 谁写前面, 谁先消费

Predicate

java.util.function.Predicate<T>函数式接口: 条件接口, 用于判断
// 抽象方法
boolean test(T t): 判断参数传递的对象. 至于怎么判断, 要判断什么, 需要我们编写 Lambda 表达式来实现
// 默认方法 (用于连接多个判断条件)
default Predicate<T> and(Predicate<? super T> other): 与
default Predicate<T> or(Predicate<? super T> other): 或
default Predicate<T> negate(): 非, 取相反结果

Function

java.util.function.Function<T,R>: 根据一个 T 类型的数据得到另一个 R 类型的数据
T 称为前置条件, 也就是输入(input)的类型
R 称为后置条件, 也就是返回结果(result)的类型
有进有出, 所以称为"函数 Function"
// 抽象方法
R apply(T t): 将 T 转换为 R
// 默认方法
default <V> Function<T, V> andThen(Function<? super R, ? extends V> after): 拼接多个 Function 转换

day13 Stream 流 方法引用

常用的流操作

获取流对象 集合->流: 集合对象.stream() 数组->流: Stream.of(数组) Stream 方法:

延迟方法:

Stream filter(Predicate p): 过滤
Stream map(Function f): 转换映射
Stream limit(long n): 只要前 n 个
Stream skip(long n): 不要前 n 个

终结方法

long count(): 返回流中元素的个数
void forEach(Consumer c): 遍历消费每个元素

静态方法

Stream concat(Stream s1, Stream s2): 合并流

方法引用（简化 Lambda 表达式）

对象的成员方法: 对象名::成员方法名 静态方法: 类名::静态方法名 父类的方法: super::成员方法名 本类的方法:
this::成员方法名
类和数组的构造器引用
类: 类名::new
数组: 数据类型[]::new

day14 JUnit 反射 注解

获取类的字节码对象的三种方式

1. Class.forName("全类名");
2. 类名.class
3. 对象名.getClass()

通过反射获取成员方法对象，并且调用方法

1. 获取类的字节码对象 Class cls
2. Method m = cls.getDeclaredMethod("方法名", 参数类型.class, ...)
3. 返回值 = m.invoke(对象, 参数值,...);

通过反射创建类的实例对象

1. 获取类的字节码对象 Class cls
2. 无参: cls.newInstance()
   有参: Constructor con = cls.getConstructor(参数类型.class, ...)
   对象 = con.newInstance(参数值, ...)

注解的作用

1. 生成文档 @param @return
2. 代码分析 @ProAnno @Check
3. 编译检查 @Override @FunctionalInterface

注释: 给程序员看
注解: 给程序看

自定义注解

元注解
public @interface 注解名 {
 属性(抽象方法)
 返回值类型 方法名() default 默认值;
}
只有一个属性名 value, 使用时省略 value= @Pro(11)
数组元素只有一个, 省略{}
返回值类型:
基本类型
String
枚举
注解
以上的数组

常用的元注解及其作用

@Target: 规定注解所使用的位置
ElementType.TYPE 类 METHOD 方法 FIELD 成员变量
@Retention: 规定注解所保存的时期
RetentionPolicy.RUNTIME
@Documented: 保存到 API 文档中
@Inherited: 子类可以继承该注解