1. Node.js是一个能够在服务器端运行JavaScript的开放源代码、跨平台JavaScript运行环境。
2. Node在处理请求时是单线程的，但是在后台拥有一个I/O线程池
3. 在Node中，每个js文件中的js代码都是独立运行在一个函数中，而非全局作用域，其他模块无法访问变量和函数，该函数会自动创建：
   1. function（exports，require，module，\_\_filename，\_\_dirname）｛代码｝
      1. module：代表当前模块本身 exports变量：exports = module.exports
4. 基本语法：
   1. delete 对象.属性
5. 模块化：
   1. 向外部暴露：exports.变量 = xx 或 export default 类名
      1. module.exports = ｛变量：值。。。｝ 赋值，可实现批量暴露
   2. 引入模块：require（“模块标识”） 若使用相对路径，必须使用 . 或 .. 开头
      1. 函数返回一个对象，该对象代表引入的模块。
      2. 使用模块名字引入模块时，首先从当前目录的node\_modules中寻找，没有则去上一级目录的node\_modules中寻找。。。，找不到则报错
6. 全局对象：global 在全局中创建的变量/函数都会作为global的属性/方法保存
7. 包规范：
   1. 包结构：包是一个压缩文件，解压后为目录，可包含以下文件及目录：（仅描述文件为必须）
      1. package.json（描述文件）、bin（可执行二进制文件）、lib（js代码）、doc（文档）、test（单元测试）
   2. 包描述文件：是一个json文件，位于包的根目录下（不能写注释）
      1. 字段：“name”、“description”、“version”、“keywords”、“maintainers”、“dependencies”。。。
8. NPM（Node Package Manager）：帮助完成第三方模块的发布、安装、依赖
   1. 在一个目录下安装包时，先用init创建package.json
   2. 命令：
      1. npm -v、npm search 包名、npm init
      2. npm install / i 包名 [操作符] 在当前目录安装包，若不写包名，表示安装当前项目所有依赖项
         1. 操作符：包名后面加 @版本号 表示安装指定版本的包
            1. --save（-S）添加到生产依赖；
            2. -g全局安装；
            3. cnpm --registry=url镜像安装，以后镜像直接用cnpm i 包名，将从之前网站下载
            4. --save-dev（-D）添加到开发依赖
      3. npm remove / r 包名 [操作符] 移除包
      4. npm config set registry url 设置镜像源，淘宝源：http://registry.npm.taobao.org
9. Buffer：缓冲区，Node自带包
   1. Buffer结构与数组类似，Buffer中可以存储二进制文件，性能强于传统数组。可以通过索引操作元素
   2. 在Buffer中以二进制存储，在显示时都是以十六进制的形式呈现，每个元素占8位，范围从00-ff。超出范围的截取二进制最后8位
   3. Buffer的大小一旦确定，则不能再修改
   4. 属性：let buf = Buffer.from（str）
      1. buf.length 占用内存的大小（Bytes）
   5. 方法：
      1. Buffer.from（二进制数据） 将数据保存在buffer中，返回Buffer对象
      2. Buffer.alloc（字节数） 返回指定大小的Buffer对象
      3. Buffer.allocUnsafe（字节数） 返回指定大小的Buffer对象，创建时不清空数据，有残留数据
      4. Buffer.toString（） 返回转换出来的字符串
10. fs模块：文件系统
    1. fs模块的方法可以选择同步或异步执行，同步方法名一般是异步方法名加上Sync
    2. 同步文件写入：通过返回值获取文件
       1. fs.openSync（路径，’操作的类型’ [，mode]） 同步打开文件，返回文件的描述符，可用于操作文件
          1. 操作类型：w 写（不存在文件则创建）；r 读（~异常）；a 追加（~创建）
          2. mode：操作权限，一般不传
       2. fs.writeSync（文件描述符，’内容’ [，写入起始位置[，‘编码’]]） 同步写入文件
          1. 编码默认为utf-8
       3. fs.closeSync（文件描述符） 保存并关闭文件（同步）
    3. 异步文件写入：通过回调函数的参数获取文件
       1. fs.open（路径，‘操作的类型’[，mode]，回调函数） 异步打开文件，无返回值
          1. 回调函数有两个参数：err（错误对象，若没有错误则为null）、fd（文件的描述符）
       2. fs.write（文件描述符，’内容’ [，写入起始位置[，‘编码’]]，回调函数） 异步写入文件
          1. 一般把该方法写在open的回调函数内，若err为null执行write
       3. fs.close（文件描述符，回调函数） 关闭文件，一般写在write的回调内，回调函数只有一个err参数
    4. 简单文件写入：
       1. fs.writeFile（文件路径，数据[，option]，回调函数） 写入数据并保存关闭，回调函数的参数：err
          1. option：一个对象，默认｛encoding：‘utf-8’，flag：‘w’，mode：0o666｝
    5. 简单文件读取：
       1. fs.readFile（‘文件路径’[，option]，回调函数） 读取文件，回调函数的参数：err，data（Buffer对象）
    6. 流式文件写入：同步、异步、简单写入不适合大文件的写入，性能差，容易导致内存溢出
       1. let ws = fs.createWriteStream（文件路径[，option]） 创建一个可写流
       2. ws.on / once（‘事件’，fn） 监听流，执行回调函数。once绑定的事件触发一次后失效
          1. 事件：open、close
       3. ws.write（内容） / ws.end（内容） 写入内容 / 关闭流
    7. 流式文件读取：适用于大文件
       1. let rs = fs.createReadStream（‘文件路径’）
       2. rs.on（‘data’，fn（data）｛｝） 绑定data事件后会自动读取，读取完自动关闭
       3. rs.pipe（ws） 将可读流中的内容直接输出到可写流，完成后自动关闭两个流
    8. 其他方法：
       1. fs.existsSync（‘文件路径‘） 检查文件是否存在（异步方法已废弃）
       2. fs.stat（‘文件路径’，fn（err，stat）｛｝） / fs.statSync（‘文件路径’） stat / 返回一个对象，对象中保存了文件状态的相关信息
          1. stat.isFile（） / .isDirectory（）
       3. fs.unlink（‘文件路径’，fn） / fs.unlinkSync（‘文件路径’） 删除文件，建议同步
       4. fs.readdir（‘路径’ [，options]，fn（err，files）） / fs.readdirSync（‘路径’[，options]） 读取一个目录的目录结构，返回一个字符串数组
       5. fs.truncate（‘路径’，字节数，fn）/ fs.truncateSync（‘路径’，字节数） 截断文件（将文件修改为指定大小）
       6. fs.mkdir（‘路径’[，mode]，fn）/ fs.mkdirSync（‘路径’[，options]） 创建一个目录
       7. fs.rmdir（‘路径’[，mode]，fn）/ fs.rmdirSync（‘路径’[，options]） 删除一个目录
       8. fs.rename（‘老路径’，‘新路径’，fn（err））/ fs.renameSync（‘老路径’，‘新路径’） 重命名文件
       9. fs.watchFile（‘文件路径’[，options]，fn（curr，prev）｛｝） 当文件发生变化时，执行回调函数
          1. curr：当前文件的状态；prev：修改前文件的状态。两个参数都是stats对象
11. path模块：
    1. 拼接绝对路径：const｛resolve｝=require（‘path’）； resolve（\_ \_dirname，‘路径’） //dirname变量代表当前文件所在目录的绝对路径