

目录

说明	1.1
----	-----

I 学习笔记

HTML5	2.1
CSS3	2.2
JS	2.3
ES6	2.4
Python	2.5
Linux	2.6
Docker	2.7
C语言	2.8

II 其他记录

postgresql	3.1
git	3.2
adb	3.3
树莓派	3.4
Termux	3.5
未分类	3.6

- [说明](#)

说明

- I. 根据视频教程学习敲下来的笔记
- II. 其他个人使用记录

- HTML5
 - 新语义标签
 - 网页布局结构标签及兼容处理
 - 多媒体标签及属性介绍
 - 新表单元素及属性
 - 智能表单控件
 - 表单属性
 - API
 - 获取页面元素及类名操作和自定义属性
 - 自定义属性
 - 文件读取
 - 获取网络状态
 - 获取地理定位
 - 本地存储
 - Canvas
 - 绘图方法
 - 渐变方案
 - 填充效果
 - 非零环绕原则
 - 绘制虚线
 - 绘制动画效果
 - 绘制文本
 - 绘制图片
 - 绘制圆弧
 - 平移【坐标系圆点的平移】
 - 旋转【坐标系旋转】
 - 伸缩

HTML5

新语义标签

网页布局结构标签及兼容处理

HTML5 语义元素

```
<header></header>
<footer></footer>
<article></article>
<aside></aside>
<nav></nav>
<section></section>
```

多媒体标签及属性介绍

HTML 视频

- `<video></video>` 视频
 - 属性: controls 显示控制栏
 - 属性: autoplay 自动播放
 - 属性: loop 设置循环播放
- `<audio></audio>` 音频
 - 属性: controls 显示控制栏
 - 属性: autoplay 自动播放
 - 属性: loop 设置循环播放

```
<video>
<source src="code/多媒体标签/trailer.mp4">
<source src="trailer.ogg">
<source src="trailer.WebM">
</video>
```

新表单元素及属性

智能表单控件

```

<input type="email">
<!--
email: 输入合法的邮箱地址
url:   输入合法的网址
number: 只能输入数字
range: 滑块
color: 拾色器
date: 显示日期
month: 显示月份
week : 显示第几周
time: 显示时间
-->

```

表单属性

- form属性:
 - autocomplete=on | off 自动完成
 - novalidate=true | false 是否关闭校验
- input属性:
 - autofocus: 自动获取焦点
 - form:
 - multiple: 实现多选效果
 - placeholder: 占位符 (提示信息)
 - required: 必填项
 - list

```

<input type="text" list="abc"/>
<datalist id="abc">
  <option value="123">12312</option>
  <option value="123">12312</option>
  <option value="123">12312</option>
  <option value="123">12312</option>
</datalist>

```

API

获取页面元素及类名操作和自定义属性

```

//选择器： 可以是css中的任意一种选择器
//通过该选择器只能选中第一个元素。
document.querySelector("选择器");

//与document.querySelector区别： querySelectorAll 可以选中所有符合
document.querySelectorAll("选择器");

Dom.classList.add("类名"); //给当前dom元素添加类样式

Dom.classList.remove("类名"); //给当前dom元素移除类样式

classList.contains("类名"); //检测是否包含类样式

classList.toggle("active"); //切换类样式（有就删除，没有就添加）

```

自定义属性

- data-自定义属性名在标签中，以data-自定义名称
 - 获取自定义属性 Dom.dataset 返回的是一个对象
 - Dom.dataset.属性名 或者 Dom.dataset[属性名]¹
 - Dom.dataset.自定义属性名=值 或者 Dom.dataset[自定义属性名]=值;

文件读取

FileReader

FileReader 接口有3个用来读取文件方法返回结果在result中

readAsBinaryString ---将文件读取为二进制编码

readAsText ---将文件读取为文本

readAsDataURL ---将文件读取为DataURL

FileReader 提供的事件模型

onabort 中断时触发

onerror 出错时触发

onload 文件读取成功完成时触发

onloadend 读取完成触发，无论成功或失败

onloadstart 读取开始时触发

onprogress 读取中

获取网络状态

- 获取当前网络状态 window.navigator.onLine 返回一个布尔值
- 网络状态事件
 1. window.ononline
 - i. window.onoffline

获取地理定位

1. 获取一次当前位置

```
window.navigator.geolocation.getCurrentPosition(success,error)
1. coords.latitude    纬度
2. coords.longitude   经度
```

2. 实时获取当前位置

```
window.navigator.geolocation.watchPosition(success,error);
```

本地存储

- localStorage:
 - 永久生效
 - 多窗口共享
 - 容量大约为20M

```
window.localStorage.setItem(key,value) //设置存储内容
window.localStorage.getItem(key)       //获取内容
window.localStorage.removeItem(key)    //删除内容
window.localStorage.clear()            //清空内容
```

- sessionStorage:
 - 生命周期为关闭当前浏览器窗口
 - 可以在同一个窗口下访问
 - 数据大小为5M左右

```
window.sessionStorage.setItem(key,value)
window.sessionStorage.getItem(key)
window.sessionStorage.removeItem(key)
window.sessionStorage.clear()
```

Canvas

绘图方法

```
<canvas id="myCanvas" width="200" height="100"></canvas>
<script type="text/javascript">
    var ctx=document.getElementById("myCanvas");
    ctx.moveTo(x,y)    //落笔
    ctx.lineTo(x,y)    //连线
    ctx.stroke();      //描边
    ctx.beginPath();   //开启新的图层
</script>
```

- 样式: strokeStyle="值"
- 线宽: lineWidth="值" 备注: 不需要带单位
- 线连接方式: lineJoin: round | bevel | miter (默认)
- 线帽 (线两端的结束方式): lineCap: butt(默认值) | round | square

```
ctx.closePath(); //闭合路径
```

渐变方案

线性渐变

```
var grd=ctx.createLinearGradient(x0,y0,x1,y1);
```

- x0-->渐变开始的x坐标
- y0-->渐变开始的y坐标
- x1-->渐变结束的x坐标
- y1-->渐变结束的y坐标

```
grd.addColorStop(0,"black");    //设置渐变的开始颜色
grd.addColorStop(0.1,"yellow"); //设置渐变的中间颜色
grd.addColorStop(1,"red");      //设置渐变的结束颜色
ctx.strokeStyle=grd;
ctx.stroke();
```

- 备注: 渐变的开始位置和结束位置介于0-1之间, 0代表开始, 1代表结束。中间可以设置任何小数

径向渐变


```
ctx.createRadialGradient(x0,y0,r0,x1,y1,r1);
```

- (x0,y0): 渐变的开始圆的 x,y 坐标
- r0: 开始圆的半径
- (x1,y1): 渐变的结束圆的 x,y 坐标
- r1: 结束圆的半径

填充效果

```
ctx.fill();           //设置填充效果  
ctx.fillStyle="值";  //设置填充颜色
```

非零环绕原则

绘制一个如下图形

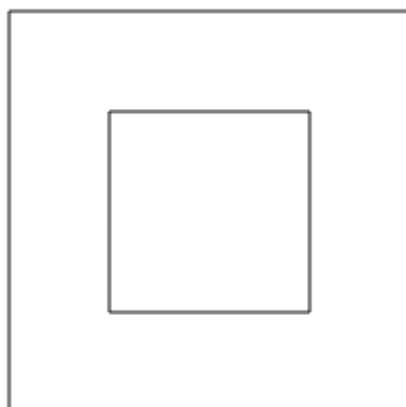


图 - 矩形

- 非零环绕原则:
 - 任意找一点，越简单越好
 - 以点为圆心，绘制一条射线，越简单越好（相交的边越少越好）
 - 以射线为半径顺时针旋转，相交的边同向记为+1，反方向记为-1，如果相加的区域等于0，则不填充。
 - 非零区域填充

绘制虚线

1. 原理:

设置虚线其实就是设置实线与空白部分直接的距离,利用数组描述其中的关系

例如: [10,10] 实线部分10px 空白部分10px

例如: [10,5] 实线部分10px 空白部分5px

例如: [10,5,20] 实线部分10px 空白5px 实线20px 空白部分10px 实线5px 空白20px....

2. 绘制:

```
ctx.setLineDash(数组);
ctx.stroke();
ctx.moveTo(100, 100);
ctx.lineTo(300, 100);
ctx.setLineDash([2,4]);
ctx.stroke();
```

注意: 如果要将虚线改为实线, 只要将数组改为空数组即可。

绘制动画效果

绘制一个描边矩形: `content.strokeRect(x,y,width,height)`

绘制一个填充矩形: `content.fillRect(x,y,width,height)`

清除: `content.clearRect(x,y,width,height)`

实现动画效果:

1. 先清屏
2. 绘制图形
3. 处理变量

绘制文本

绘制填充文本

```
content.fillText(文本的内容,x,y)
```

绘制镂空文本

```
content.strokeText();
```

设置文字大小:

```
content.font="20px 微软雅黑"
```

文字水平对齐方式【文字在圆心点位置的对齐方式】

```
content.textAlign="left | right | center"
```

文字垂直对齐方式

```
content.textBaseline="top | middle | bottom | alphabetic(默认)"
```

- 文字阴影效果
 - ctx.shadowColor="red"; 设置文字阴影的颜色
 - ctx.shadowOffsetX=值; 设置文字阴影的水平偏移量
 - ctx.shadowOffsetY=值; 设置文字阴影的垂直偏移量
 - ctx.shadowBlur=值; 设置文字阴影的模糊度

绘制图片

```
//将图片绘制到画布的指定位置  
content.drawImage(图片对象,x,y);  
//将图片绘制到指定区域大小的位置 x,y指的是矩形区域的位置, width和height指的是矩形区域的大小  
content.drawImage(图片对象,x,y,width,height);  
content.drawImage(图片对象,sx,sy,swidth,sheight,dx,dy,dwidth,dheight);
```

- sx,sy 指的是要从图片哪块区域开始绘制,
- swidth, sheight 是值 截取图片区域的大小
- dx,dy 是指矩形区域的位置, dwidth,dheight是值矩形区域的大小

解决图片绘制到某一个区域的按原比例缩放绘制: 绘制宽: 绘制高==原始宽: 原始高

绘制圆弧

```
content.arc(x,y,radius,starttradian,endradian[,direct]);
```

1. x,y 圆心的坐标
2. radius 半径
3. starttradian 开始弧度
4. endradian 结束弧度
5. direct 方向（默认顺时针 false） true 代表逆时针
6. **角度 和 弧度的关系： 角度:弧度= 180:pi**
7. 0度角: 以圆心为中心向右为0角 顺时针为正，逆时针为负
8. 特殊值

0度 = 0弧度

30度 = $\pi/6$ （180度的六分之一）

45度 = $\pi/4$

60度 = $\pi/3$

90度 = $\pi/2$

180度 = π

360度 = 2π

绘制圆上任意点：

9. 公式：
 - $x=ox+r\cos(\text{弧度})$
 - $y=oy+r*\sin(\text{弧度})$
 - ox: 圆心的横坐标
 - oy: 圆心的纵坐标
 - r: 圆的半径

平移【坐标系圆点的平移】

```
ctx.translate(x,y);
```

- 通过该方法可以将原点的位置进行重新设置。
- translate(x,y) 中不能设置一个值
- 与moveTo(x,y) 的区别：
 - moveTo(x,y) 指的是将画笔的落笔点的位置改变，而坐标系中的原点位置并没有发生改变
 - translate(x,y) 是将坐标系中的原点位置发生改变

旋转【坐标系旋转】

```
ctx.rotate(弧度)
```

伸缩

```
ctx.scale(x,y)
```

备注：沿着x轴和y轴缩放 x,y 为倍数 例如： 0.5 1

¹. 属性名是不包含data-** ↩

- CSS3
 - 样式
 - 背景
 - 边框
 - 文本
 - 选择器
 - 颜色渐变
 - 2D转换
 - 3D转换
 - 动画
 - 过渡
 - 自定义动画
 - 伸缩布局或者弹性布局

CSS3

样式

背景

规定背景图片的定位区域

<code>background-origin:</code>	
<code>padding-box</code>	背景图像相对内边距定位（默认值）
<code>border-box</code>	背景图像相对边框定位【以边框左上角为参照进行位置设置】
<code>content-box</code>	背景图像相对内容区域定位【以内容区域左上角为参照进行位置设置】

备注：1. 默认盒子的背景图片是在盒子的内边距左上角对齐设置。

规定背景的绘制区域

<code>background-clip:</code>	
<code>border-box</code>	背景被裁切到边框盒子位置【将背景图片在整个容器中显示】
<code>padding-box</code>	背景被裁切到内边距区域【将背景图片在内边距区域（包含内边距）显示】
<code>content-box</code>	背景被裁切到内容区域【将背景图片在内容区域显示】

规定背景图片的尺寸

```
background-size:
    cover
    contain
```

边框

- box-shadow: 盒子阴影
- border-radius: 边框圆角
- border-image: 边框图片

```
/* 设置边框图片 */
border-image-source: url("2.png");

/* 边框图片裁切 : 不需要带单位*/
border-image-slice: 20;

/* 设置边框图片的平铺方式 */
/* border-image-repeat: stretch; */
border-image-repeat: round;
/* border-image-repeat: repeat; */

border-image-width: 20px;
```

文本

```
text-shadow: 设置文本阴影
```

选择器

1. 属性选择器: [属性名=值] {} [属性名] {} 匹配对应的属性即可 [属性名^=值] {} 以值开头 [属性名*=值] {} 包含 [属性名\$=值] {} 以值结束
2. 结构伪类选择器: :first-child {} 选中父元素中第一个子元素
:last-child {} 选中父元素中最后一个子元素 :nth-child(n) {}
选中父元素中正数第n个子元素 :nth-last-child(n) {} 选中父元素中倒数第n个子元素
 - 备注:
 - n 的取值大于等于0
 - n 可以设置预定义的值
 - odd[选中奇数位置的元素]

- even 【选中偶数位置的元素】
 - n 可以是一个表达式：an+b的格式
3. 其他选择器： :target 被锚链接指向的时候会触发该选择器
- ::selection 当被鼠标选中的时候的样式 ::first-line 选中第一行 ::first-letter 选中第一个字符

颜色渐变

- 线性渐变：

1. 开始颜色和结束颜色
2. 渐变的方向
3. 渐变的范围

```
background-image: linear-gradient(
    to right,
    red,
    blue
);
```

备注：

4. to + right | top | bottom | left
5. 通过角度表示一个方向 0deg [从下向上渐变] 90deg 【从左向右】

- 径向渐变：

```
/* 径向渐变 */
background-image: radial-gradient(
    100px at center,
    red,
    blue
);
```

2D转换

- 位移 transform: translate(100px,100px);
 - 备注：位移是相对元素自身的位置发生位置改变
- 旋转 transform: rotate(60deg);
 - 备注：取值为角度
- 缩放 transform: scale(0.5,1);

- 备注：取值为倍数关系，缩小大于0小于1，放大设置大于1
- 倾斜 `transform: skew(30deg, 30deg);`
 - 备注：
 1. 第一个值代表沿着x轴方向倾斜
 2. 第二个值代表沿着y轴方向倾斜

3D转换

- 位移 `transform: translateX() translateY() translateZ()`
- 旋转 `transform: rotateX(60deg) rotateY(60deg) rotateZ(60deg);`
- 缩放 `transform: scaleX(0.5) scaleY(1) scaleZ(1);`
- 倾斜 `transform: skewX(30deg) skewY();`
- 将平面图形转换为立体图形 `transform-style: preserve-3d;`

动画

过渡

```
/* 设置哪些属性要参与到过渡动画效果中: all */
transition-property: all;

/* 设置过渡执行时间 */

transition-duration: 1s;

/* 设置过渡延时执行时间 */
transition-delay: 1s;

/* 设置过渡的速度类型 */

transition-timing-function: linear;
```

自定义动画

```

/* 1定义动画集 */
@keyframes rotate {

    /* 定义开始状态 0%*/
    from {
        transform: rotateZ(0deg);
    }

    /* 结束状态 100%*/
    to {
        transform: rotateZ(360deg);
    }
}

```

注意：如果设置动画集使用的是百分比，那么记住百分比是相对整个动画执行时间的。

伸缩布局或者弹性布局

- 设置父元素为伸缩盒子【直接父元素】

```
display: flex
```

为什么在伸缩盒子中，子元素会在一行上显示？

1. 子元素是按照伸缩盒子中主轴方向显示
 2. 只有伸缩盒子才有主轴和侧轴
 3. 主轴：默认水平从左向右显示
 4. 侧轴：始终要垂直于主轴
- 设置伸缩盒子主轴方向（flex-direction）

```

flex-direction: row;
flex-direction: row-reverse;
flex-direction: column;
flex-direction: column-reverse;

```

- 设置元素在主轴的对齐方式(justify-content)

```
/* 设置子元素在主轴方向的对齐方式 */  
justify-content: flex-start;  
justify-content: flex-end;  
justify-content: center;  
justify-content: space-between;  
justify-content: space-around;
```

- 设置元素在侧轴的对齐方式 (align-items)

```
align-items: flex-start;  
align-items: flex-end;  
align-items: center;  
/* 默认值 */  
align-items: stretch;
```

- 设置元素是否换行显示 (flex-wrap)
 1. 在伸缩盒子中所有的元素默认都会在一行上显示
 2. 如果希望换行: `flex-wrap: wrap | nowrap;`
- 设置元素换行后的对齐方式 (align-content)

```
align-content: flex-start;  
align-content: flex-end;  
align-content: center;  
align-content: space-around;  
align-content: space-between;  
/* 换行后的默认值 */  
align-content: stretch;
```

- JS学习
 - 数据类型
 - 对象的基本使用
 - 创建一个对象
 - 对象属性操作
 - 通过构造函数创建对象
 - 自定义一个构造函数来创建对象
 - 构造函数的概念
 - 构造函数的执行过程
 - 继承
 - 为什么要使用继承?
 - 原型链继承1
 - 原型链继承2
 - 拷贝继承(混入继承)
 - 原型式继承
 - 借用构造函数实现继承
 - 原型链（家族族谱）
 - 闭包
 - 变量作用域
 - 作用域链
 - 闭包的问题
 - 闭包问题的产生原因
 - 函数的4种调用方式

JS学习

数据类型

- 基本数据类型——值类型：(数字、字符串、布尔值、null、undefined)
- 复杂数据类型——引用类型：(对象)
 - 数组
 - 函数
 - 正则表达式
 - Date

对象的基本使用

创建一个对象

```

var student={
  name:"李白" , //student有一个name属性, 值为"李白"
  grade:"初一" ,
  //a、 student有一个say属性, 值为一个函数
  //b、 student有一个say方法
  say:function(){
    console.log("你好");
  },
  run:function(speed){
    console.log("正在以"+speed+"米/秒的速度奔跑");
  }
}

```

对象是键值对的集合：对象是由属性和方法构成的 (ps：也有说法为：对象里面皆属性，认为方法也是一个属性)

- name是属性 grade是属性
- say是方法 run是方法

对象属性操作

获取属性

第一种方式：.语法

- `student.name` 获取到 name 属性的值
- `student.say` 获取到一个函数

第二种方式：[]语法

- `student["name"]` 等价于 `student.name`
- `student["say"]` 等价于 `student.say`

两种方式的差异：

- .语法更方便，但是坑比较多(有局限性)
 - .后面不能使用js中的关键字、保留字(class、this、function。。。)
 - .后面不能使用数字

```

var obj={};
obj.this=5; //语法错误
obj.0=10;   //语法错误

```

- []使用更广泛
 - o1[变量name]
 - ["class"]、["this"]都可以随意使用 `obj["this"]=10`
 - `obj[3]=50 = obj["3"]=50`
 - `["[object Array)"]` jquery里面就有这样的实现
 - `["{abc}"]` 给对象添加了 {abc} 属性

设置属性

- `student["gender"]="男"` 等价于 `student.gender="男"`
 - 含义：如果student对象中没有gender属性，就添加一个gender属性，值为"男"
 - 如果student对象中有gender属性，就修改gender属性的值为"男"
- 案例1: `student.isFemale=true`
- 案例2: `student["children"]=[1,2,5]`
- 案例3:

```
student.toShanghai=function(){
    console.log("正在去往上海的路上")
}
```

删除属性

- `delete student["gender"]`
- `delete student.gender`

通过构造函数创建对象

构造函数创建对象的例子

- `var xiaoming = new Object()` --> `var xiaoming = {};`
- `var now = new Date()`
- `var rooms = new Array(1,3,5)` --> `var rooms = [1,3,5]`
- `var isMale=/123/;` ==> `var isMale=new RegExp("123")`
 - isMale是通过RegExp构造函数创建出来的对象
 - isMale是RegExp构造函数的实例
- 以上例子中，Object、Date、Array都是内置的构造函数

自定义一个构造函数来创建对象

- 构造函数

```
function Person(name,age){
    this.name=name;
    this.age=age;
}
var p1=new Person("赵云",18)
```

- 说明： p1就是根据【Person构造函数】创建出来的对象

构造函数的概念

- 任何函数都可以当成构造函数 `function CreateFunc(){ }`
- 只要把一个函数通过new的方式来进行调用，我们就把这一次函数的调用方式称之为：构造函数的调用
 - `new CreateFunc();` 此时CreateFunc就是一个构造函数
 - `CreateFunc();` 此时的CreateFunc并不是构造函数
- `new Object()`等同于对象字面量`{}`

构造函数的执行过程

1. 创建一个对象 `var p1=new Person();` (我们把这个对象称之为 Person构造函数的实例)- `_p1`
2. 创建一个内部对象， `this`，将this指向该实例(`_p1`)
3. 执行函数内部的代码，其中，操作this的部分就是操作了该实例(`_p1`)
4. 返回值：
 - a、如果函数没有返回值(没有return语句)，那么就会返回构造函数的实例(`p1`)
 - b、如果函数返回了一个基本数据类型的值，那么本次构造函数的返回值是该实例(`_p1`)

```
function fn(){
}
var f1=new fn();    //f1就是fn的实例

function fn2(){
    return "abc";
}
var f2=new fn2();    //f2是fn2构造函数的实例
```

- c、如果函数返回了一个复杂数据类型的值，那么本次函数的返回值就是该值

```
function fn3(){
    return [1,3,5];
    //数组是一个对象类型的值,
    //所以数组是一个复杂数据类型的值
    //本次构造函数的真正返回值就是该数组
    //不再是fn3构造函数的实例
}
var f3=new fn3();    //f3还是fn3的实例吗? 错
//f3值为[1,3,5]
```

问题:

1. 如何判断一个数据是否是复杂数据类型?

使用排除法:

- 看它的值是不是数字、字符串、布尔值、null、undefined
- 如果不是以上5种值, 那就是复杂数据类型

2. 为什么要理解构造函数的返回值?

String是一个内置函数: a、String() b、new String()

一个函数通过new调用, 或者不通过new调用, 很多时候会有截然不同的返回值

3. 我们如何分辨出一个对象到底是不是某个构造函数的实例?

var isTrue=xxx instanceof Person

4. 如何识别xxx对象是哪个构造函数的实例?

xxx.**proto**属性, 也是对象, 该对象中一般会有一个constructor属性, 这个值指向PPP, 那么xxx就是PPP的构造函数

5. typeof运算符不能用来判断对象的构造函数

继承

通过【某种方式】让一个对象可以访问到另一个对象中的属性和方法, 我们把这种方式称之为继承 并不是所谓的xxx extends yyy

为什么要使用继承?

有些对象会有方法(动作、行为), 而这些方法都是函数, 如果把这些方法和函数都放在构造函数中声明就会导致内存的浪费


```
function Person(){
    this.say=function(){
        console.log("你好")
    }
}
var p1=new Person();
var p2=new Person();
console.log(p1.say === p2.say); //false
```

原型链继承1

```
Person.prototype.say=function(){
    console.log("你好")
}
```

缺点：添加1、2个方法无所谓，但是如果方法很多会导致过多的代码冗余

原型链继承2

```
Person.prototype={
    constructor:Person,
    say:function(){
        console.log("你好");
    },
    run:function(){
        console.log("正在进行百米冲刺");
    }
}
```

注意点：

- a、一般情况下，应该先改变原型对象，再创建对象
- b、一般情况下，对于新原型，会添加一个constructor属性，从而不破坏原有的原型对象的结构

拷贝继承(混入继承)

场景：有时候想使用某个对象中的属性，但是又不能直接修改它，于是就可以创建一个该对象的拷贝

```
var o1 = {age:2};  
var o2 = o1;  
o2.age = 18;
```

1. 修改了o2对象的age
2. o2对象跟o1对象是同一个对象
3. o1对象的age属性也被修改了

如下代码中，如果使用拷贝继承对代码进行优化会非常和谐

```
var o3 = {gender:"男",grade="初三",group:"第五组",name:"张三"};  
var o4 = {gender:"男",grade="初三",group:"第五组",name:"李四"};
```

1. 已经有了o3对象
2. 创建o3对象的拷贝：for...in循环

```
//a. 取出o3对象中每个属性  
for(var key in o3){  
    //key是o3中每个属性  
    //b. 获取到对应属性值  
    var value = o3[key];  
    //c. 把属性值放入o4  
    o4[key] = value;  
}
```

3. 修改克隆对象，把该对象的name改为“李四”

```
o4.name="李四";
```

浅拷贝和深拷贝

- 浅拷贝只是拷贝一层属性，没有内部对象
- 深拷贝其实是利用了递归的原理，将对象的若干层属性拷贝出来

实现：

```
var source={name:"李白",age:15}  
var target={};  
target.name=source.name  
target.age=source.age;
```

上面的方式很明显无法重用，实际代码编写过程中，很多时候都会使用拷贝继承的方式，所以为了重用，可以编写一个函数把他们封装起来：

```
function extend(target,source){
    for(key in source){
        target[key]=source[key];
    }
    return target;
}
extend(target,source)
```

由于拷贝继承在实际开发中使用场景非常多，所以很多库都对此有了实现

- jquery: `$.extend`
- es6中有了对象扩展运算符仿佛就是专门为了拷贝继承而生：

```
var source={name:"李白",age:15}
var target={ ...source }
var target2={...source,age:18}
```

原型式继承

- 场景：
 - 创建一个纯洁的对象
 - 创建一个继承自某个父对象的子对象

```
var parent = {age:18,gender:"男"};
var student = Object.create(parent);
console.log(student)
```

- 使用方式：
 - 空对象: `Object.create(null)`

```
var o1={ say:function(){}}
var o2=Object.create(o1);
```

`var o3={}` o3并不纯洁

借用构造函数实现继承

- 场景：适用于2种构造函数之间逻辑有相似的情况

- 原理：函数的call、apply调用方式
- 局限性：Animal（父类构造函数）的代码必须完全适用于Person（子类构造函数）

```
function Animal(name,age,gender){
    this.name=name;
    this.age=age;
    this.gender=gender;
}
function Person(name,age,gender,address){
    // Animal.call(this,name,age,gender);
    //等价于
    Animal.apply(this,[name,age,gender]);
    //this.name=name;
    //this.age=age;
    // this.gender=gender;
    this.address=address;
}
```

- 寄生继承、寄生组合继承

原型链（家族族谱）

- 概念：JS里面的对象可能会有父对象，父对象还会有父对象，。。。。。。祖先
- 根本：继承
 - 属性：对象中几乎都会有一个**proto**属性，指向他的父对象

-意义：可以实现让该对象访问到父对象中相关属性

- 根对象： `Object.prototype`
 - `var arr=[1,3,5]`
 - `arr.proto`: `Array.prototype`
 - `arr.proto.proto`找到了根对象

```
function Animal(){}
var cat=new Animal();
//cat.__proto__: Animal.prototype
//cat.__proto__.__proto__:根对象
```

- 错误的理解：万物继承自Object?

闭包

变量作用域

- 变量作用域的概念：就是一个变量可以使用的范围
- JS中首先有一个最外层的作用域：称之为全局作用域
- JS中还可以通过函数创建出一个独立的作用域，其中函数可以嵌套，所以作用域也可以嵌套

```
var age=18;//age是全局作用域中
function f1(){
    var name="张三";//name是f1函数内部声明的变量，所以作用域在f1内部
    console.log(name);//可以访问到name变量
    console.log(age);//age是全局作用域，所以也可以访问
}
console.log(age);//也可以访问
```

```
//多级作用域
//1级作用域
var gender="男";
function fn(){
    console.log(age);//age:undefined undefined为初始值，既然有

    console.log(height);//height不是在该作用域内部声明，故不能访问
//2级作用域
    return function(){
        //3级作用域
        var height=180;
    }
    var age=5;
}
```

注意：

- 变量的声明和赋值是在两个不同的时期的。
- fn函数在执行的时候，首先找到函数内部所有变量、函数声明，把他们放在作用域中，给变量一个初始值undefined --变量可以访问
- 逐行执行代码过程中，如果有赋值语句，对变量进行赋值

作用域链

- 由于作用域是相对于变量而言的，而如果存在多级作用域，这个变量又来自于哪里？这个问题就需要好好地探究一下了，我们把这个变量的查找过程称之为变量的作用域链
- 意义：查找变量（确定变量来自于哪里，变量是否可以访问）
- 简单来说，作用域链可以用几句话来概括：（或者说：确定一个变量来自于哪个作用域）
 1. 查看当前作用域，如果当前作用域声明了这个变量，就确定结果
 2. 查找当前作用域的上级作用域，也就是当前函数的上级函数，看看上级函数中有没有声明
 3. 再查找上级函数的上级函数，直到全局作用域为止
 4. 如果全局作用域中也没有，我们就认为这个变量未声明(xxx is not defined)

```
function fn(callback){
  var age=18;
  callback();
}
fn(function(){
  console.log(age);
  //分析：age变量
  //1.查找当前作用域并没有
  // 2.查找上一级作用域，全局作用域
})
```

注意：看上一级作用域，不是看函数在哪里调用，而是看函数在哪里编写，因为这种特别，我们通常会把作用域说成是**词法作用域**

- 举例1:

```
var name="张三";
function f1(){
  var name="abc";
  console.log(name);
}
f1();
```

- 举例2:

```
var name="张三";
function f1(){
    console.log(name);
    var name="abc";
}
f1();
```

- 举例3:

```
var name="张三";
function f1(){
    return function(){
        console.log(name);
    }
    var name="abc";
}
var fn=f1();
fn();
```

- 举例4:

```
var name="张三";
function f1(){
    return {
        say:function(){
            console.log(name);
            var name="abc";
        }
    }
}
var fn=f1();
```

闭包的问题

```

function fn(){
    var a=5;
    return function(){
        a++;
        console.log(a);
    }
}
var f1=fn();
//执行到上一行，fn函数完毕，返回匿名函数
// 一般认为函数执行完毕，变量就会释放
// 但是由于此时由于js引擎发现匿名函数要使用a变量
//所以a变量并不能得到释放
// 而是把a变量放在匿名函数可以访问到的地方去了
f1();//6
f1();//7
f1();//8

```

```

function q2(){
    var a={};
    return function(){
        return a;
    }
}
var t3=q2();
var o5=t3();
var o6=t3();
console.log(o5==o6);//true
var w3=q2();
var o8=w3();
console.log(o5==o8);//false

```

闭包问题的产生原因

- 函数执行完毕后，作用域中保留了最新的a变量的值
- 闭包内存释放


```
function f1(){
    var a=5;
    return function(){
        a++;
        console.log(a);
    }
}
var q1=f1();
//要想释放q1里边保存的a,只能通过释放q1
q1=null;//q1=undefined;
```

闭包的应用场景

- 模块化
- 防止变量被破坏

函数的4种调用方式

1. 函数调用

```
var age=18;
var p={
    age:15,
    say:function(){
        console.log(this.age);
    }
}
var s1=p.say;
s1();//函数调用--> this: window 输出18
```

结论：函数内部的this指向window

2. 方法调用

```

var age=18;
var p={
  age:15,
  say:function(){
    console.log(this.age);
  }
}
p.say(); //打印结果15
//
function Person(){
  this.age = 20;
}
Person.prototype.run=function(){
  console.log(this.age);
}
var p1=new Person();
p1.run(); //打印结果20
//
var clear=function(){
  console.log(this.length);
}
var length=50;
var tom={c:clear,length:100};
tom.c();//打印100 this指向tom

```

结论：由于clear函数被当成tom.c()这种方法的形式调用，所以函数内部的this指向调用该方法的对象：tom

3. new调用（构造函数）

```

//1
function fn(name){
    this.name=name;
}
//通过new关键字来调用的，这种方式就是构造函数调用方式
var _n=new fn("小明");
//2
function jQuery(){
    var _init=jQuery.prototype.init;
    return new _init();
}
jQuery.prototype={
    constructor:jQuery,
    length:100,
    init:function(){
        //1.this指向init构造函数的实例
        //2.如果本身没有该属性，那么去他的原型对象中去找
        //3.如果原型对象中没有，那么就去原型对象的原型对象中查找
        //4.最终没有找到，则属性值为：undefined
        console.log(this.length);
    }
}
jQuery.prototype.init.prototype=jQuery.prototype;
jQuery();//结果为100

```

4. 上下文方式 (call,apply,bind)

```

var name=21;
function f1(){
    console.log(this.name);
}
f1.call([1,3,5]);
f1.apply(this);
f1.call(5);
//总结
//call函数的第一个参数:
//1.如果是一个对象类型,那么函数内部this指向对象
//2.如果是undefined、null,那么函数内部this指向window
//3.如果是数字-->this指向new Number(数字),字符串-->this指向new String(字符串)

//bind是ES5(ie9+)
var obj={
    age:18,
    run:function(){
        setTimeout((function(){
            console.log(this.age)
        }).bind(this),500)
        //通过执行了bind方法,匿名函数本身并没有执行,只是改变了this指向
    }
}
obj.run();
//bind基本用法
function speed(){
    console.log(this.seconds);
}
//执行bind方法之后产生了一个新函数,这个新函数里边的逻辑和原来还是
var speedBind=speed.bind({seconds:100});
speedBind();//打印100

```

- call和apply都可以改变函数内部this的值
- 传参的形式不同

```

function toString(a,b,c){
    console.log(a+" "+b+" "+c);
}
toString.call(null,1,3,5);
toString.apply(null,[1,3,5])

```

在ES6的箭头函数之前的时代，想要判断一个函数内部的this指向谁，就是根据上边的四种方式来决定

- bind方法实现

```
//target表示新函数的内部的this值
Function.prototype._bind=function(target){

    //利用闭包创建一个内部函数，返回那个所谓的新函数
    return ()=>{
        //执行fn里边的逻辑
        this.call(target)
    }
    //等价于
    // var _that=this;
    // return function(){
    //     _that.call(target);
    // }
}
function fn(){
    console.log(this);
}
var _fn=fn._bind({age:18});
```

- bind方法放在函数的原型中 `fn.__proto__ === fn的构造函数.prototype`
- 所有的函数对象的构造函数都是Function

- ES6(常用的、重点的)
 - 模板字符串
 - 解构赋值
 - 函数的扩展
 - rest参数
 - 箭头函数
 - 对象的扩展
 - Promise
 - 回调地狱
 - Promise函数基本用法
 - Promise函数实现多层回调
 - Promise函数错误处理
 - async
 - class
 - 定义一个类
 - 添加实例方法
 - 添加静态方法
 - 类的继承
 - module
 - 基本用法
 - 模块有多个导出
 - 模块导入导出取别名

ES6(常用的、重点的)

模板字符串

模板字符串的基本用法

```

//基本用法
var s1 = ` abc `
var s2 = " abc "
//模板字符串解决了一些痛点:
//字符串拼接
var s3=" a "+s1+" b "+s2;
var s4=` a ${s1} b ${s2}`
var s5=`<div>
    <p>
        <span>222</span>
        <span>${s1}</span>
        <span>${s2}</span>
    </p>
</div>
`

```

解构赋值

对象的解构赋值

```

var obj={name:"张三",age:18}

var {name,age}=obj;
//生成2个变量,
//  name值来自于obj.name、
//  age值来自于obj.age

var {name:title}=obj;
//生成一个变量: title, 值来自于obj.name

```

函数参数的解构赋值

```
function f1(obj){
    console.log(obj.age);
    console.log(obj.height)
}
//等价于
function f1({ age,height }){
    console.log(age);
    console.log(height)
}

f1({age:5,height:180})
```

补充：属性的简写

```
var a = 3 ;
var c = 10;
var b = { a,c } ;
//b对象有一个a属性，a属性的值，来自于a变量，
//还有一个c属性，c属性的值来自于c变量
console.log(b); //{a: 3, c: 10}
```

函数的扩展

rest参数

- 使用背景：es6的
- 优点：arguments是伪数组，而rest参数是真数组

```
function fn(...args){
    //验证args是不是数组?
    console.log(args instanceof Array); //true
    console.log(Object.prototype.toString.call(args));
    console.log(Array.isArray(args));
    console.log(args); //数组: [1,2,3,4,5]
}

fn(1,2,3,4,5)
```

- 判断数据类型
 - typeof只能判断：数字、字符串、布尔、undefined、函数

- `Object.prototype.toString.call()`
- `Array.isArray()`
- `isNaN()`
- `isInfinity()`

箭头函数

- 场景：用于替换匿名函数
- 基本用法：

```
//匿名函数
div.onclick=function(){
    console.log("你好")
}
//箭头函数
div.onclick=()=>{
    console.log("你好")
}
```

- 有一个参数的箭头函数

```
var fn=(a)=>{
    console.log("abc");
}
//等价于：
var fn=a=>{
    console.log("abc");
}
```

- 有2个及更多参数的箭头函数

```
var f=(a,b,c)=>{
    console.log("abc")
}
```

- 箭头函数和普通匿名函数有哪些不同？
 - 函数体内的`this`对象，就是定义时所在的对象，而不是使用时所在的对象。
 - 不可以当作构造函数，也就是说，不可以使用`new`命令，否则会抛出一个错误。

- 不可以使用arguments对象，该对象在函数体内不存在。如果要
用，可以用 rest 参数代替。
- （不常用）不可以使用yield命令，因此箭头函数不能用作
Generator 函数。
 - generator函数现在经常用async替代

```
var p={
  age:18,
  run:()=>{
    setTimeout(()=>{
      console.log("run",this);//this指向window
    },100)
  },
  say(){
    setTimeout(()=>{
      console.log("say",this);//this指向p
    },100)
  },
  //推荐使用的方式
  travel:function(){
    setTimeout(()=>{
      console.log("travel",this);//this指向p
    },100)
  }
}
```

对象的扩展

- Object.assign : 实现拷贝继承

```
var source={age:18,height:170,className:"三年级"};
var newObj=Object.assign({},source);
```

- 对象扩展运算符

```
var obj1={ age:5,gender:"男" }
var obj2={ ...obj1 }
var obj3={ ...obj1 , age:10 }
var s1=[1,3,5,7,9];
var s2=[...s1];
```

Promise

为什么要有promise：解决回调地狱的问题

回调地狱

```
$.get("/getUser",function(res){
    $.get("/getUserDetail",function(){
        $.get("/getCart",function(){
            //...
        })
    })
})

function f1(){
    return new Promise((resolve)=>{
        setTimeout(()=>{
            console.log("第一步");
            //异步执行完毕，必须告诫外界执行结束
            resolve();
        },1000)
    })
}

function f2(){
    return new Promise((resolve)=>{
        setTimeout(()=>{
            console.log("第二步");
            //到这里异步操作就已经结束了，如何让外界得知
            resolve();
        },1000)
    })
}

f1().then(res=>{
    //return promise对象
    return f2();
}).then(res=>{
    setTimeout(()=>{
        console.log("结束");
    },1000)
})
```

Promise函数基本用法

```
var promise=new Promise((resolve,reject)=>{
    $.get("/getUser",res=>{
        resolve(res)
    })
})
promise.then(res=>{
    console.log(res);
})
```

Promise函数实现多层回调

```
new Promise((resolve,reject)=>{
    $.get("/getUser",res=>{
        resolve(res)
    })
}).then(res=>{
    //用户基本信息
    return new Promise(resolve=>{
        $.get("/getUserDetail",res=>{
            resolve(res)
        })
    })
}).then(res=>{
    //用户详情
    return new Promise(resolve=>{
        $.get("/getCart",res=>{
            resolve(res)
        })
    })
}).then(res=>{
    //购物车信息
})
```

Promise函数错误处理

- 第一种方式

```

new Promise((resolve, reject) => {
  $.ajax({
    url: "/getUser",
    type: "GET",
    success: res => {
      resolve(res);
    },
    error: res => {
      reject(res)
    }
  })
}).then(resSuccess => {
  // 成功的返回值
}, resError => {
  // 失败的返回值
})

```

- 第二种方式（推荐使用这一种，reject的错误和代码发生的粗误都可以捕捉到）

```

new Promise((resolve, reject) => {
  $.ajax({
    url: "/getUser",
    type: "GET",
    success: res => {
      resolve(res);
    },
    error: res => {
      reject(res)
    }
  })
}).then(resSuccess => {
  // 成功的返回值
}).catch(resError => {
  // 失败的返回值
})

```

async

await可以执行异步操作，但是await必须在async函数内执行

```

function f1(){
  return new Promise((resolve)=>{
    setTimeout(()=>{
      console.log("第一步");
      //异步执行完毕，必须告诫外界执行结束
      resolve();
    },1000)
  })
}

(async function(){
  await f1();
  console.log("第二步");
})();

async function get(){
  console.log('开始执行');
  var res = await timer()
  console.log('执行结束: ',res);
}

function timer(){
  return new Promise((resolve,reject)=>{
    setTimeout(()=>{
      resolve("你好");
    },1000)
  })
}

get();

```

错误处理，只能使用try-catch

```

function q(){
  return new Promise((resolve,reject)=>{
    setTimeout(()=>{
      reject("你好");
    },1000)
  })
}
(async function(){
  try{
    let res=await q();
    console.log(res);
  }catch(e){
    console.log(e);
  }
})();

```

class

定义一个类

```

class Person {
  constructor(name,age) {
    this.name=name;
    this.age=age;
  }
}
//相当于:
function Person(name,age){
  this.name=name;
  this.age=age;
}

```

添加实例方法

```

//不使用class添加实例方法
function Person(){

}
Person.prototype.run={()=>{
    console.log("run");
}}
//使用class添加实例方法
class Person {
    constructor(name,age) {
        this.name=name;
        this.age=age;
    }
    //定义方法
    say() {
        console.log("大家好, 我叫: "+this.name+", 今年: "+this
    }
    travel(){
        console.log("坐着飞机去巴厘岛");
    }
}

```

添加静态方法

```

class Animal {
    constructor(){

    }
    static age=18;
    static born(){
        console.log("小呆萌出生了")
    }
}
//访问静态方法
Animal.born();
console.log(Animal.age);

```

类的继承


```

class Person {
    constructor(name){
        this.name=name;
    }
}
class Student extends Person {
    constructor(name,grade){
        super(name);    //调用父类构造函数
        this.grade=grade;
    }
}

```

module

基本用法

- 导出模块：

```

//common.js
export default { name:"abc" }

```

- 导入模块：

```

//b.js
import common from "common.js"

console.log( common.name ) //"abc"

```

模块有多个导出

```

//person.js
export const jim = { country : "France" }
export const tony = { color: "gray" }
//默认的导出
export default { name: "abc" }

```

```
//index.js
import person , { jim , tony } from "person.js"

//person: { name:"abc" }
//jim: { country : "France" }
//tony: { color:"gray" }
```

模块导入导出取别名

```
//person.js
export const tony = { color:"gray" }
export { tony as Tony }

//index.js
import { Tony } from "person.js"
import { Tony as man} from "person.js"

console.log(man)    //{ color:"gray" }
```

- python
 - Anaconda
 - 安裝配置
 - 使用

python

Anaconda

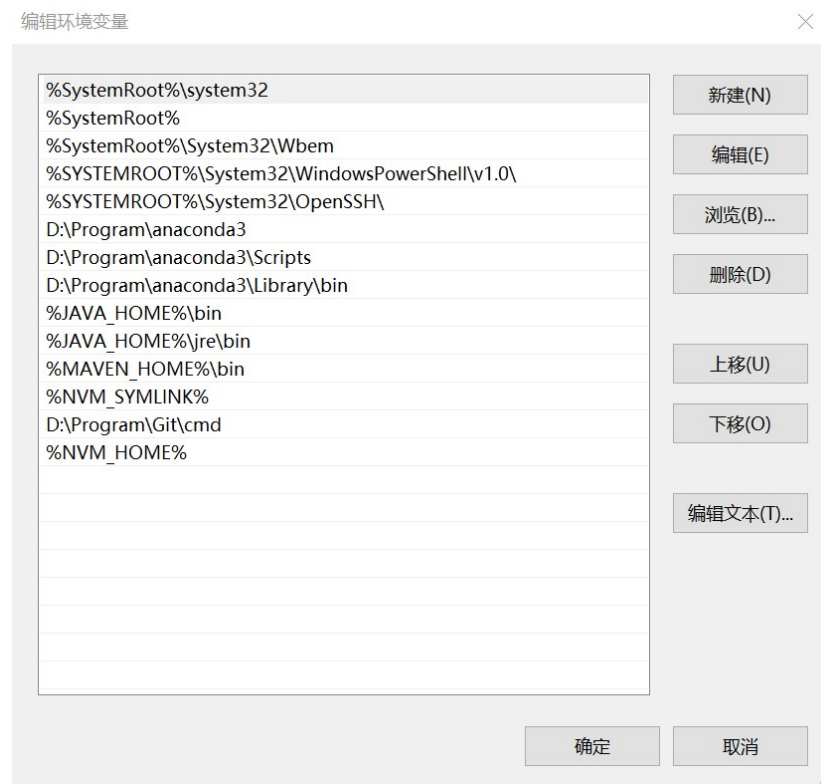
安装配置

1. 下载

- 注意选择与操作系统匹配的版本，可根据需要下载对应版本Python的Anaconda3
- 下载地址为：<https://www.anaconda.com/download/>
- 根据安装向导逐步进行，注意安装路径中的文件夹名称不要存在空格

2. 环境变量 系统变量PATH添

加 D:\Program\anaconda3;D:\Program\anaconda3\Scripts;D:\Program\anaconda3\Library\bin; (根据安装路径确定)



【注意】 此图PATH变量中也显示了后续安装过程中添加的信息

3. 测试

- 测试Python: 在cmd中输入 `Python` 命令, 查看Python版本信息
- 测试Conda: 在cmd中输入 `conda` 或 `pip` 命令, 查看具体信息
- `conda --version` 输出版本号说明设置成功

使用

1. 修改清华源

- Windows 用户无法直接创建名为 `.condarc` 的文件, 可先执行 `conda config --set show_channel_urls yes` 生成该文件之后再修改

```
channels:
- defaults
show_channel_urls: true
default_channels:
- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/
- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/
- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/
custom_channels:
conda-forge: https://mirrors.tuna.tsinghua.edu.cn/ana
msys2: https://mirrors.tuna.tsinghua.edu.cn/anaconda/
bioconda: https://mirrors.tuna.tsinghua.edu.cn/anacon
menpo: https://mirrors.tuna.tsinghua.edu.cn/anaconda/
pytorch: https://mirrors.tuna.tsinghua.edu.cn/anacond
simpleitk: https://mirrors.tuna.tsinghua.edu.cn/anaco
```

- 运行 `conda clean -i` 清除索引缓存, 保证用的是镜像站提供的索引。

2. 管理虚拟环境

- 创建虚拟环境 `code`

```
conda create -n code python=3
```

- 查看所有虚拟环境

```
conda env list
```

- 切换环境

```
activate code
```

- 删除虚拟环境

```
conda remove -n aiconfig --all
```

- 安装ipython

```
conda install -c anaconda ipython
```

- 退出虚拟环境 deactivate

3. 修改pip源

- Windows下和conda一样修改用户目录下的pip目录下的 pip.init C:\Users\xxx\pip\pip.ini

```
[global]
index-url=http://mirrors.aliyun.com/pypi/simple/
[install]
trusted-host=mirrors.aliyun.com
disable-pip-version-check = true
timeout = 6000
```

- [linux](#)
 - [目录结构](#)
 - [快捷键](#)
 - [文件权限](#)
 - [帮助](#)
 - [ls命令](#)
 - [切换目录](#)
 - [创建目录](#)
 - [链接](#)
 - [vi命令](#)
 - [文本搜索](#)
 - [文件搜索](#)
 - [解压缩](#)
 - [bzip2压缩](#)
 - [bzip2解压缩](#)
 - [zip压缩](#)
 - [unzip解压缩](#)
 - [查看所有用户](#)

linux

目录结构

- /: 根目录
- /bin: /usr/bin: 可执行二进制文件的目录，如常用的命令ls、tar、mv、cat等。
- /boot: 放置linux系统启动时用到的一些文件，如Linux的内核文件: /boot/vmlinuz, 系统引导管理器: /boot/grub。
- /dev: 存放linux系统下的设备文件，访问该目录下某个文件，相当于访问某个设备，常用的是挂载光驱 mount /dev/cdrom /mnt。
- /etc: 系统配置文件存放的目录，不建议在此目录下存放可执行文件，重要的配置文件有
/etc/inittab、/etc/fstab、/etc/init.d、/etc/X11、/etc/sysconfig、/etc/xinetd.d。
- /home: 系统默认的用户家目录，新增用户账号时，用户的家目录都存放在此目录下
- /lib: /usr/lib: /usr/local/lib: 系统使用的函数库的目录，程序在执行过程中，需要调用一些额外的参数时需要函数库的协助。
- /lost+fount: 系统异常产生错误时，会将一些遗失的片段放置于此目录下。
- /mnt: /media: 光盘默认挂载点，通常光盘挂载于 /mnt/cdrom 下，也不一定，可以选择任意位置进行挂载。

- /opt: 给主机额外安装软件所摆放的目录。
- /proc: 此目录的数据都在内存中, 如系统核心, 外部设备, 网络状态, 由于数据都存放于内存中, 所以不占用磁盘空间, 比较重要的目录有
/proc/cpuinfo、/proc/interrupts、/proc/dma、/proc/ioports、/proc/net/* 等。
- /root: 系统管理员root的家目录。
- /sbin: /usr/sbin: /usr/local/sbin: 放置系统管理员使用的可执行命令, 如fdisk、shutdown、mount 等。与 /bin 不同的是, 这几个目录是给系统管理员 root使用的命令, 一般用户只能"查看"而不能设置和使用。
- /tmp: 一般用户或正在执行的程序临时存放文件的目录, 任何人都可以访问, 重要数据不可放置在此目录下。
- /srv: 服务启动之后需要访问的数据目录, 如 www 服务需要访问的网页数据存放在 /srv/www 内。
- /usr: 应用程序存放目录, /usr/bin 存放应用程序, /usr/share 存放共享数据, /usr/lib 存放不能直接运行的, 却是许多程序运行所必需的一些函数库文件。/usr/local: 存放软件升级包。/usr/share/doc: 系统说明文件存放目录。/usr/share/man: 程序说明文件存放目录。
- /var: 放置系统执行过程中经常变化的文件, 如随时更改的日志文件 /var/log, /var/log/message: 所有的登录文件存放目录, /var/spool/mail: 邮件存放的目录, /var/run:程序或服务启动后, 其PID存放在该目录下。

快捷键

- 清屏: `ctrl+l`
- 在终端在退出锁定: `ctrl+c`

文件权限

- 读 r : read
- 写 w : write
- 执行 x execute
- d rwx rwx rwx 分为三组

drwxr-xr-x	12	wsj0051	wsj0051	4096	Nov 26 20:55	./
drwxr-xr-x	3	root	root	4096	Nov 19 17:18	../
-rw-----	1	wsj0051	wsj0051	7700	Nov 27 11:43	.bash_history
-rw-r--r--	1	wsj0051	wsj0051	220	Nov 19 17:18	.bash_logout
-rw-r--r--	1	wsj0051	wsj0051	4055	Nov 25 21:28	.bashrc
drwxr-xr-x	3	wsj0051	wsj0051	4096	Nov 25 21:17	.cache/
drwx-----	5	wsj0051	wsj0051	4096	Nov 25 21:17	.config/
1	2	3	4	5	6	

1. 文件类型
2. 所有者的权限

3. 用户组的权限
4. 其他用户的权限
5. 所有者
6. 用户组

帮助

- 命令 `--help`
- `man` 命令

ls命令

- `ls` 文件名 查看文件
 - `ls -a` 查看所有文件包含隐藏文件
 - `ls -l` 以列表形式查看文件，不包含隐藏文件
 - `ls -lh` 以列表形式查看文件，不包含隐藏文件，按照1024倍数显示{KB MB GB}
 - `ls -all` 以列表形式查看文件，包含隐藏文件
 - `ll` 同上
- 通配符：
 - `*` 匹配任意多个字符 0-256
 - `a*` 一个以上字符 256以下
 - `?` 匹配任意一个字符
 - `a?` 两个字符
 - `[a-z]` 区间法
 - 匹配a到z的所有字符，只能确定一个字符
 - `[abcde]` 穷举法
 - 匹配abcde的所有字符，只能确定一个字符

切换目录

- `cd` 切换到用户主目录
- `cd ~` 切换到当前用户的主目录
- `cd ..` 切换到上级目录
- `cd .` 切换到当前目录
- `cd -` 切换到上一次目录

创建目录

- `mkdir` 目录名 创建目录
- `mkdir` 目录名 `-p` 递归创建
- `rmdir` 目录名 目录一定是空的

- `rm 文件名` 删除文件
- `rm 文件名 -i` 删除时询问
- `rm -r 目录名` 递归删除

链接

1. 硬链接 `ln 源文件 链接文件`
 - 硬链接文件占磁盘空间，但是删除源文件不会影响硬链接文件，与copy类似
 - 无论你修改了哪一个链接之后的文件，两个文件都会改变保持一致，但是拷贝不会
2. 软连接 `ln -s 源文件 链接文件`
 - 软链接文件不占磁盘空间 但是删除源文件会影响软链接文件
 - 改变软链接文件就是相当于间接的改变了源文件
 - 查看文件时默认链接数为1 如果有链接依次递增
 - 如果创建的软链接文件和源文件在不同的目录下，需要使用绝对路径

vi命令

- `ctrl+z` 放到后台
- `jobs` 查看后台
- `fg 编号` 唤醒

按键	功能
ZZ	退出保存
:wq	退出保存
:q!	退出不保存
[n]x	删除光标后 n 个字符
[n]X	删除光标前 n 个字符
D	删除光标所在开始到此行尾的字符
[n]dd	删除从当前行开始的 n 行（准确来讲，是剪切，剪切不粘贴即为删除）
[n]yy	复制从当前行开始的 n 行
p	把粘贴板上的内容插入到当前行
dG	删除光标所在开始到文件尾的所有字符
J	将光标所在行和下一行进行合并，中间用空格分隔
.	执行上一次操作
u	撤销前一个命令
gg	定位到行首
G	定位到行尾
:set ic	搜寻时忽略大小写
:set noic	搜寻时不忽略大小写
:set nu	显示行号
:set nonu	不显示行号
a	光标位置右边插入文字
i	光标位置当前处插入文字
o	光标位置下方开启新行
O	光标位置上方开启新行
I	光标所在行首插入文字

按键	功能
A	光标所在行尾插入文字
:r 文件名	在光标下一行插入一个新的文件内容
:s/w1/w2/g	在当前行用w2替换w1
:g/p1/s//p2/g	在整个文本中用p2替换p1
:10,20s/p1/p2/g	第10到20行所有p1用p2代替
/内容	查找字符串，n向下查找，N向上查找

文本搜索

```
grep '搜索内容' 文件名
```

- 参数
 - -n 显示行号
 - -v 反选
 - -i 忽略大小写
- 通配符
 - ^a 以a为起始的字符搜索文件
 - a\$ 以a为结尾的字符搜索文件
 - . 匹配任意一个非换行的字符
 - * 匹配任意字符（大于0的整数）

文件搜索

```
find 目录 参数 文件名
```

- -name 文件名
- -size 大小
- -perm rwx

解压缩

tar归档

```
tar -cvf 归档文件名.tar 文件1 文件2 目录1 目录2
```

tar解归档

```
tar -xvf 归档文件名.tar -C 路径
```

gzip 压缩生成一个【归档文件名.tar.gz】，文件大小小于归档文件大小，
【归档文件名.tar】不存在了

```
gzip 归档文件名.tar
```

```
gzip -d 解压生成文件【归档文件名.tar】
```

```
gzip -d 归档文件名.tar.gz
```

一步归档压缩

```
tar -czvf 文件名.tar.gz 文件1 文件2 目录1 目录2
```

一步解归档压缩

```
tar -xzvf 文件名.tar.gz -C 路径
```

bzip2压缩

```
tar -cjvf 文件名.tar.bz2 文件1 文件2 目录1 目录2
```

bzip2解压缩

```
tar -xjvf 文件名.tar.bz2 -C 路径
```

zip压缩

```
zip 文件名 文件1 文件2 目录1 目录2 生成一个文件为：文件名.zip
```

unzip解压缩

```
unzip 文件名.zip -C 路径
```

查看所有用户

```
cat /etc/passwd
```

- [wsl2 ubuntu docker](#)
 - [安装](#)
 - [修改默认源](#)
 - [安装软件包以允许 apt 通过 HTTPS 使用存储库](#)
 - [添加 Docker 的官方 GPG 密钥](#)
 - [设置稳定的存储库](#)
 - [更新 apt 包索引](#)
 - [docker镜像源修改](#)
 - [启动、停止与重启](#)
 - [镜像相关](#)
 - [查看镜像](#)
 - [搜索镜像](#)
 - [下载](#)
 - [删除镜像](#)
 - [容器相关](#)
 - [查看容器](#)
 - [创建与启动容器](#)
 - [查看容器的ip地址](#)
 - [删除容器](#)
 - [备份与迁移](#)
 - [Dockerfile](#)
 - [Docker私有仓库](#)
 - [常用应用安装](#)
 - [docker安装nextcloud](#)
 - [tomcat](#)
 - [nginx](#)
 - [redis](#)
 - [设置代理](#)
 - [Docker 代理](#)
 - [Container 代理](#)
 - [Docker Build 代理](#)
 - [重启生效](#)

wsl2 ubuntu docker

安装

修改默认源

```
cp /etc/apt/sources.list /etc/apt/sources.list.bak
```

编辑文件

```
vim /etc/apt/sources.list
```

删除原有内容并替换为：

```
deb http://mirrors.aliyun.com/ubuntu/ bionic main restricted uni
deb http://mirrors.aliyun.com/ubuntu/ bionic-security main restr
deb http://mirrors.aliyun.com/ubuntu/ bionic-updates main restri
deb http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restr
deb http://mirrors.aliyun.com/ubuntu/ bionic-backports main rest
deb-src http://mirrors.aliyun.com/ubuntu/ bionic main restricted
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-security main r
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-updates main re
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-proposed main r
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-backports main
```

更新源

```
apt-get update
apt-get upgrade
```

安装软件包以允许 apt 通过 HTTPS 使用存储库

```
apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

添加 Docker 的官方 GPG 密钥

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo a
```

设置稳定的存储库

```
add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"
```

更新 apt 包索引

```
apt-get update
docker --version
sudo gpasswd -a 用户名 docker
```

docker 镜像源修改

vim /etc/docker/daemon.json 添加以下内容

```
{
  "registry-mirrors":["https://hub-mirror.c.163.com","https://regi
}
```

启动、停止与重启

```
sudo service docker start
sudo service docker stop
sudo service docker restart
```

镜像相关

查看镜像

```
docker images
```

搜索镜像

```
docker search 镜像名
```


下载

```
docker pull 镜像名
```

删除镜像

按镜像id删除

```
docker rmi 镜像id
```

删除所有镜像

```
docker rmi `docker images -q`
```

容器相关

查看容器

查看正在运行的容器

```
docker ps
```

查看所有容器

```
docker ps -a
```

查看停止的容器

```
docker ps -f status=exited
```

创建与启动容器

创建容器命令：`docker run` -i: 表示运行容器 -t: 表示启动后进入其命令行 --name: 为容器命名 -v: 表示目录的映射关系（宿主机目录:容器的目录） -d: 创建一个守护式容器在后台运行 -p: 表示端口映射（宿主机端口: 容器内的映射端口）

1. 交互式方式创建容器

```
docker run -it --name=容器名称 镜像名称:标签 /bin/bash
```

2. 守护式创建容器

```
docker run -di --name=容器名称 镜像名称:标签
```

3. 登录守护式容器方式

```
docker exec -it 容器名称或容器id /bin/bash
```

4. 启动和停止容器

```
docker start 容器名称或容器id  
docker stop 容器名称或容器id
```

5. 将文件拷贝到容器

```
docker cp 要拷贝的文件或目录 容器名称:容器目录
```

6. 将文件从容器拷贝出来

```
docker cp 要拷贝的容器名称:要拷贝的容器目录 本地目录
```

7. 目录挂载

```
docker run -di -v 宿主机目录:容器目录 --name=自定义容器名 镜像名称
```

查看容器的ip地址

```
docker inspect 容器名称 (容器id)  
docker inspect --format='{{.NetworkSettings.IPAddress}}' 容器名
```

删除容器

```
docker stop 容器名  
docker rm 容器名
```

备份与迁移

1. 保存为镜像

```
docker commit nginx mynginx
```

2. 镜像备份，保存为tar

```
docker save -o mynginx.tar mynginx
```

3. 恢复与迁移，加载tar文件

```
docker load -i mynginx.tar
```

Dockerfile

Dockerfile是一系列命令和参数构成的脚本，这些命令用于基础镜像并最终创建一个新的镜像。

命令	作用
FROM image_name:tag	定义了使用哪个基础镜像启动构建流程
MAINTAINER user_name	声明镜像的创建者
ENV key value	设置环境变量
RUN command	Dockerfile的核心部分
ADD source_dir/file dest_dir/file	将宿主机文件复制到容器内，压缩文件自动解压
WORKDIR path_dir	设置工作目录

以jdk8为例创建Dockerfile

```
mkdir -p /user/local/docker-jdk8  
vi Dockerfile
```

以下为Dockerfile内容：

```
FROM ubuntu
MAINTAINER wsj0051
WORKDIR /usr
RUN mkdir /usr/local/java
ADD jdk*.tar.gz /usr/local/java/
ENV JAVA_HOME /usr/local/java/jdk1.8
ENV JRE_HOME $JAVA_HOME/jre
ENV CLASSPATH $JAVA_HOME/bin/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE
ENV PATH $JAVA_HOME/bin:$PATH
```

执行命令构建镜像

```
docker build -t='jdk1.8' .
```

Docker私有仓库

1. 拉取私有仓库镜像

```
docker pull registry
```

2. 启动私有仓库

```
docker run -di --name=registry -p 5000:5000 registry
```

3. 浏览器输入地址http://ip:5000/v2/_catalog看到{"repositories":[]}表示私有仓库搭建成功并且内容为空

4. 修改 /etc/docker/daemon.json

```
{"insecure-registries":["ip:5000"]}
```

5. 将镜像上传到私有仓库

```
docker tag jdk1.8 ip:5000/jdk1.8
docker push ip:5000/jdk1.8
```

常用应用安装

docker安装nextcloud

```
# mysql
docker run -d --name mysql \
    -v /data/nextcloud/mysql:/var/lib/mysql \
    -e MYSQL_ROOT_PASSWORD=password \
    -e MYSQL_DATABASE=nextcloud \
    -e MYSQL_USER=nextcloud \
    -e MYSQL_PASSWORD=password \
    -p 3307:3306 \
    --restart=always \
    mysql
# 树莓派64位系统安装nextcloud
docker run -d --name nextcloud \
    -v /data/nextcloud/data:/var/www/html \
    --link mysql:mysql \
    --restart=always \
    --privileged=true \
    -p 8888:80 arm64v8/nextcloud
```

-e 代表添加环境变量 MYSQL_ROOT_PASSWORD 是root用户的登录密码

tomcat

```
docker run -di --name=mytomcat -p 8080:8080 -v /usr/local/webapp
```

nginx

```
docker pull nginx
docker run -di --name=nginx -p 80:80 nginx
docker exec -it nginx /bin/bash
docker cp html nginx:/usr/share/nginx/ --将本地的html文件夹复制到n
```

redis

```
docker run -di --name=myredis -p 5379:6379 redis
```

设置代理

Docker 代理

在执行docker pull时，是由守护进程dockerd来执行。因此，代理需要配在dockerd的环境中。而这个环境，则是受systemd所管控，因此实际是systemd的配置。

```
sudo mkdir -p /etc/systemd/system/docker.service.d
sudo touch /etc/systemd/system/docker.service.d/proxy.conf
```

在这个proxy.conf文件（可以是任意*.conf的形式）中，添加以下内容：

```
[Service]
Environment="HTTP_PROXY=http://proxy.example.com:8080/"
Environment="HTTPS_PROXY=http://proxy.example.com:8080/"
Environment="NO_PROXY=localhost,127.0.0.1,.example.com"
```

其中，proxy.example.com:8080 要换成可用的免密代理。通常使用 cntlm 在本机自建免密代理，去对接公司的代理。可参考《Linux下安装配置 Cntlm 代理》。

Container 代理

在容器运行阶段，如果需要代理上网，则需要配置 ~/.docker/config.json。以下配置，只在Docker 17.07及以上版本生效。

```
{
  "proxies":
  {
    "default":
    {
      "httpProxy": "http://proxy.example.com:8080",
      "httpsProxy": "http://proxy.example.com:8080",
      "noProxy": "localhost,127.0.0.1,.example.com"
    }
  }
}
```

这个为用户级的配置，除了 proxies，docker login 等相关信息也会在其中。而且还可以配置信息展示的格式、插件参数等。

此外，容器的网络代理，也可以直接在其运行时通过 `-e` 注入 `http_proxy` 等环境变量。这两种方法分别适合不同场景。`config.json` 非常方便，默认在所有配置修改后启动的容器生效，适合个人开发环境。在CI/CD的自动构建环境、或者实际上线运行的环境中，这种方法就不太合适，用 `-e` 注入这种显式配置会更好，减轻对构建、部署环境的依赖。当然，在这些环境中，最好用良好的设计避免配置代理上网。

Docker Build 代理

虽然 `docker build` 的本质，也是启动一个容器，但是环境会略有不同，用户级配置无效。在构建时，需要注入 `http_proxy` 等参数。

```
docker build . \
  --build-arg "HTTP_PROXY=http://proxy.example.com:8080/" \
  --build-arg "HTTPS_PROXY=http://proxy.example.com:8080/" \
  --build-arg "NO_PROXY=localhost,127.0.0.1,.example.com" \
  -t your/image:tag
```

注意：无论是 `docker run` 还是 `docker build`，默认是网络隔绝的。如果代理使用的是 `localhost:3128` 这类，则会无效。这类仅限本地的代理，必须加上 `--network host` 才能正常使用。而一般则需要配置代理的外部IP，而且代理本身要开启 Gateway 模式。

重启生效

代理配置完成后，`reboot` 重启当然可以生效，但不重启也行。

`docker build` 代理是在执行前设置的，所以修改后，下次执行立即生效。
Container 代理的修改也是立即生效的，但是只针对以后启动的 Container，对已经启动的 Container 无效。

`dockerd` 代理的修改比较特殊，它实际上是改 `systemd` 的配置，因此需要重载 `systemd` 并重启 `dockerd` 才能生效。

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

- C语言学习
 - [Hello World](#)
 - [System系统函数](#)
 - [C语言编译过程](#)
 - [汇编语言](#)
 - [处理C语言函数的警告](#)

C语言学习

Hello World

新建hello.c文件，输入以下内容

```
#include <stdio.h>
int main(void)
{
    printf("Hello World\n");
    return 0;
}
```

```
gcc -o hello.cgi hello.c
```

System系统函数

- 使用system函数可以调用其他程序
 - 需要使用系统库： `<stdlib.h>`
 - 可以用在Linux平台和windows平台，但是调用的命令行不同

C语言编译过程

1. 预处理 `gcc -E a.c -o a.i`
 - 宏定义展开
 - 头文件展开
 - 删除注释
 - 条件编译
2. 编译 `gcc -S a.i -o a.s`
 - 检查语法
 - 转化成汇编语言
3. 汇编 `gcc -c a.s -o a.o`
 - 将汇编语言转化为机器语言

4. 链接 `gcc a.o -o a.exe`
 - 将库文件链接变成可执行文件

汇编语言

1. 新建项目创建文件
2. 写c语言源代码添加断点，调试执行
3. 程序会停止在断点处，在调试菜单栏中选择窗口，在列表中选择反汇编，查看汇编源代码

```
//汇编代码
__asm
{
    mov a, 3
    mov b, 4
    mov eax, a
    add eax, b
    mov c, eax
}
```

处理C语言函数的警告

1. 放在程序第一行

```
#define _CRT_SECURE_NO_WARNINGS
```

2. 任意位置 `#pragma warning(disable:4996)`
3. 项目右击选择属性，在打开对话框中选择C/C++处理器，在预处理器定义中编辑 `_CRT_SECURE_NO_WARNINGS`
4. linux下没有该错误

- [postgresql 使用记录](#)
 - [查询获取字段名，字段类型，长度，主键，非空，自增，默认值，描述](#)
 - [查询函数的详细sql](#)
 - [锁表问题解决](#)

postgresql 使用记录

查询获取字段名，字段类型，长度，主键，非空，自增，默认值，描述

- 使用sql查询

```

select
  c.relname as 表名,
  a.attname as 列名,
  (case
    when a.attnotnull = true then true
    else false end) as 非空,
  (case
    when (
      select
        count(pg_constraint.*)
      from
        pg_constraint
      inner join pg_class on
        pg_constraint.conrelid = pg_class.oid
      inner join pg_attribute on
        pg_attribute.attrelid = pg_class.oid
        and pg_attribute.attnum = any(pg_constraint.conkey)
      inner join pg_type on
        pg_type.oid = pg_attribute.atttypid
    where
      pg_class.relname = c.relname
      and pg_constraint.contype = 'p'
      and pg_attribute.attname = a.attname) > 0 then true
    else false end) as 主键,
  concat_ws('', t.typname) as 字段类型,
  (case
    when a.attlen > 0 then a.attlen
    when t.typname='bit' then a.atttypmod
    else a.atttypmod - 4 end) as 长度,
  col.is_identity as 自增,
  col.column_default as 默认值,
  (select description from pg_description where objoid = a.a
    and objsubid = a.attnum) as 备注
from
  pg_class c,
  pg_attribute a ,
  pg_type t,
  information_schema.columns as col
where
  c.relname = '表名'
  and a.attnum>0
  and a.attrelid = c.oid

```

```

    and a.atttypid = t.oid
    and col.table_name=c.relname and col.column_name=a.attname
order by
    c.relname desc,
    a.attnum asc

```

- 命令行查询

```
\d tablename
```

查询函数的详细sql

- 命令行查询

```
\sf 函数名
```

- sql查询

```
select prosrc from pg_proc where proname='function_name'
```

锁表问题解决

[原文链接](#)

1. 查询锁表pid与锁表语句

```
select pid, state, username, query, query_start from pg_stat_
```

2. 查找所有活动的被锁的表

```

SELECT pid, state, username, query, query_start
from pg_stat_activity
where pid in (
    select pid from pg_locks l join pg_class t on l.relation
    and t.relkind = 'r'
);

```

3. 解锁二选一

```
SELECT pg_cancel_backend(94827); -- session还在, 事物回退;  
SELECT pg_terminate_backend(94827); --session消失, 事物回退
```

- [git](#)
 - [ssh生成密钥](#)
 - [SSH协议代理设置](#)
 - [git设置代理](#)
 - [ssh协议置代理](#)
 - [只对github.com使用代理，其他仓库不走代理](#)

git

配置全局用户名邮箱信息

```
git config --global user.name "your user name"
git config --global user.email "your email"
```

让Git显示颜色

```
git config --global color.ui true
```

git设置别名

```
git config --global alias.lg "log --color --graph --pretty=format:"
```

Git在clone仓库时，有两种URL可以选择，分别为HTTPS和SSH：

1. HTTPS的格式为： `https://github.com/用户名/仓库名.git`
2. SSH的格式为： `git@github.com:用户名/仓库名.git`

git图形化工具[sourcetree](#)

ssh生成密钥

```
ssh-keygen -t rsa -C "your email"
```

一个密钥同时在github和gitee使用，修改 `.ssh` 目录下config文件

```
# gitee
Host gitee.com
HostName gitee.com
HostkeyAlgorithms +ssh-rsa
PubkeyAcceptedAlgorithms +ssh-rsa
PreferredAuthentications publickey
IdentityFile ~/.ssh/id_rsa

# github
Host github.com
HostName github.com
PreferredAuthentications publickey
IdentityFile ~/.ssh/id_rsa
```

将公钥配置到github后,验证公钥是否绑定成功

```
ssh -T git@github.com
```

SSH协议代理设置

修改 ssh 配置文件 ~/.ssh/config Windows ssh 配置文件路径: C:\Users\你的用户名\.ssh\config Linux ssh 配置文件路径: /home/你的用户名/.ssh/config

```
# 全局 -S 代表走socks代理。
//对于使用git@协议的, 可以配置socks5代理
//在~/.ssh/config 文件后面添加几行, 没有可以新建一个
//socks5

# 只为特定域名设定
Host github.com
User git
ProxyCommand connect -S 127.0.0.1:10808 %h %p

//http || https
Host github.com
User git
ProxyCommand connect -H 127.0.0.1:1080 %h %p
```

git设置代理

ssh协议置代理

```
//http
git config --global https.proxy http://127.0.0.1:1080
//https
git config --global https.proxy https://127.0.0.1:1080
//使用socks5代理的 例如ss, ssr 1080是windows下ss的默认代理端口,mac下是10808
git config --global http.proxy socks5://127.0.0.1:1080
git config --global https.proxy socks5://127.0.0.1:1080
//取消全局代理
git config --global --unset http.proxy
git config --global --unset https.proxy
```

只对github.com使用代理，其他仓库不走代理

```
git config --global http.https://github.com.proxy socks5://127.0.0.1:1080
git config --global https.https://github.com.proxy socks5://127.0.0.1:1080
//取消github代理
git config --global --unset http.https://github.com.proxy
git config --global --unset https.https://github.com.proxy
```


- [adb](#)
 - [adb 连接安卓](#)
 - [安装apk](#)
 - [推送文件](#)
 - [使用adb fastboot 线刷](#)
 - [通过 fastboot 安装 Recovery](#)
 - [通过 Recovery 安装魔趣ROM](#)
 - [fastboot命令](#)
 - [win11 wsa安卓相关](#)

adb

adb 连接安卓

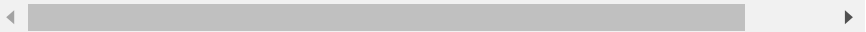
```
adb connect 127.0.0.1:58526
```

安装apk

```
adb install 目录/安装包.apk
```

推送文件

```
adb push ./Download/2023-02-28-231021 /storage/emulated/0/tvbox_
```



使用adb fastboot 线刷

1. 安装adb fastboot工具

```
sudo apt install adb fastboot
```

2. 通过USB将您的设备连接到电脑，并验证手机连接

```
adb devices
```

3. 重启到fastboot模式

```
adb reboot-bootloader
```

4. 以小米为例，下载官方线刷包后解压[链接](#)

```
tar zxvf clover_images_V10.3.2.0.ODJCNXM_20190515.0000.00_8.
```

5. 进入解压后的目录

```
cd clover_images_V10.3.2.0.ODJCNXM_20190515.0000.00_8.1_cn
```

6. 开始刷机

- 修改为可执行 `chmod +x flash_all.sh`
- fastboot模式下再次验证手机连接 `fastboot devices`
- 执行刷机脚本¹ `sudo sh flash_all.sh`

通过 fastboot 安装 Recovery

1. 下载 Recovery - 例如 [TWRP](#)。

- 您可以前往 [魔趣下载站](#) 的设备页面，点击 **Recovery 下载**。
- 如果没有找到适用于您设备的 Recovery，请借助互联网搜索设备维护者或资深玩家发布的教程。

2. 通过USB将您的设备连接到电脑。

3. 在电脑上打开命令提示符（Windows）或 终端（Linux 或 macOS）并输入：

```
adb reboot bootloader
```

您也可以通过组合键启动 fastboot 模式：

- 关闭设备后，按住音量调低 + 电源键，直到屏幕上方出现“FASTBOOT”字样，然后松开。
4. 一旦设备处于 fastboot 模式，请通过键入以下内容验证您的 PC 是否找到它：

```
fastboot devices
```

5. 将 Recovery 刷入到您的设备。

```
fastboot flash recovery twrp-x.x.x-x-x.img # 仅刷入
fastboot boot .\twrp-x.x.x-x-x.img # 刷入重启到recovery
```

6. 现在进入 Recovery 模式以验证安装：

- 关闭设备后，按住音量调高 + 电源键，直到进入 Recovery 模式，然后松开。

通过 Recovery 安装魔趣ROM

1. 下载你想要安装的魔趣ROM包。

- 可选项，下载第三方扩展包，例如 [OpenGapps](#)

2. 如果您尚未进入 Recovery 模式，请重启到 Recovery 模式。

- 关闭设备后，按住音量调高 + 电源键，直到进入 Recovery 模式，然后松开。

¹ 该脚本删除所有数据，可以选择其他sh结尾的脚本，根据名字可以猜到大概意思 ↩

fastboot命令

1. 列出fastboot设备

```
fastboot devices
```

2. 重启相关

```
fastboot reboot          #重启手机
fastboot reboot-bootloader #重启到bootloader模式,其实就是再
```

3. 擦除相关 (erase)

```
fastboot erase boot      #擦除boot分区(擦了引导就没了,会卡在第一屏)
fastboot erase recovery  #擦除recovery分区
fastboot erase system    #擦除system分区(擦了系统就没了,会卡在启动界面)
fastboot erase userdata  #擦除userdata分区(可擦,清空数据用)
fastboot erase cache     #擦除cache分区(可擦,清空数据用)
```

4. 写入分区 (flash)

```
fastboot flash boot boot.img    #写入boot分区
fastboot flash init_boot ****.img #写入initboot 安卓13用来刷机
fastboot flash recovery recovery.img    写入recovery分区
fastboot flash system system.img    #写入system分区
```

win11 wsa安卓相关

```
adb connect 127.0.0.1:58526
adb shell settings get global http_proxy
adb shell "settings put global http_proxy `ip route list match 0
adb shell settings put global http_proxy :0
```

- 树莓派使用记录
 - 系统安装
 - 校验包，解压
 - xz烧录命令(该命令尝试失败)
 - img格式镜像烧录命令如下（亲测成功）
 - 连接wifi
 - 卸载软件
 - 基础命令
 - aptitude
 - 基地64位系统
 - 启用CecOS-CaaS容器云
 - 桌面版系统单声道改为立体声输出
 - 启用和运行Docker服务
 - 安装chromium
 - 安装smb
 - smb共享
 - 程序快捷目录
 - docker 镜像安装
 - 安装Alist
 - php
 - xteve
 - aria2
 - aria2 webUi控制台
 - mysql
 - 安装owncloud
 - vim不能右键粘贴

树莓派使用记录

系统安装

校验包，解压

```
sha1sum 2013-09-25-wheezy-raspbian.zip
unzip 2013-09-25-wheezy-raspbian.zip
```

查看当前哪些设备已经挂载, `df -h` , 插入u盘或sd卡再执行一次 为了防止在写入镜像的时候有其他读取或写入，我们需要卸载设备。两个分区都要卸载。

```
umount /dev/sdb1  
umount /dev/sdb2
```

xz烧录命令(该命令尝试失败)

```
sudo xz -cd kali-2017.3-rpi3-nexmon.img.xz> /dev/sdb
```

查看烧录进度 `sudo pkill -USR1 -n -x xz`

img格式镜像烧录命令如下（亲测成功）

```
sudo dd bs=4M if=2013-09-25-wheezy-raspbian.img of=/dev/sdb
```

查看烧录进度 `sudo pkill -USR1 -n -x dd`

连接wifi

```
## To use this file, you should run command "systemctl disable n

#country=CN
#ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
#update_config=1

## WIFI 1 (Do not uncomment this line!)

network={
    ssid="coolxiaomi"
    psk="coolxiaomi"
    priority=1
    id_str="wifi-1"
}

## WIFI 2 (Do not uncomment this line!)

network={
    ssid="wsj0051"
    psk="752838157w"
    priority=2
    id_str="wifi-2"
}
```

卸载软件

卸载但不删除配置

```
apt-get remove packagename
```

卸载并删除配置

```
apt-get purge packagename
```

基础命令

安装软件 `apt-get install softname1 softname2 softname3.....`

卸载软件 `apt-get remove softname1 softname2 softname3.....`

卸载并清除配置 `apt-get remove -purge softname1`

更新软件信息数据库 `apt-get update`

进行系统升级 `apt-get upgrade`

搜索软件包 `apt-cache search softname1 softname2 softname3.....`

安装deb软件包 `dpkg -i xxx.deb`

删除软件包 `dpkg -r xxx.deb`

连同配置文件一起删除 `dpkg -r -purge xxx.deb`

查看软件包信息 `dpkg -info xxx.deb`

查看文件拷贝详情 `dpkg -L xxx.deb`

查看系统中已安装软件包信息 `dpkg -l`

重新配置软件包 `dpkg-reconfigure xxx`

清除所有已删除包的残余配置文件

```
dpkg -l |grep ^rc|awk '{print $2}' |sudo xargs dpkg -P
```

`dpkg`安装的可以用`apt`卸载，反之亦可。

aptitude

`aptitude update` 更新可用的包列表

`aptitude upgrade` 升级可用的包

`aptitude dist-upgrade` 将系统升级到新的发行版

`aptitude install pkgname` 安装包

`aptitude remove pkgname` 删除包

`aptitude purge pkgname` 删除包及其配置文件

`aptitude search string` 搜索包

`aptitude show pkgname` 显示包的详细信息

`aptitude clean` 删除下载的包文件

`aptitude autoclean` 仅删除过期的包文件

基地64位系统

启用CecOS-CaaS容器云

```
...  
  
开机自动启动 CecOS CaaS容器云 服务  
systemctl enable cecos-caas.service  
  
启动 CecOS CaaS容器云 服务  
systemctl start cecos-caas.service  
  
#####  
  
停止 CecOS CaaS容器云 服务  
systemctl stop cecos-caas.service  
  
禁止 CecOS CaaS容器云 服务开机启动  
systemctl disable cecos-caas.service  
...
```

桌面版系统单声道改为立体声输出

编辑文件 /usr/share/pulseaudio/alsa-mixer/profile-sets/default.cc

在下面的内容每行最前面添加 ";" 符号，以注释掉以下内容：

...

```
[Mapping analog-mono]
device-strings = hw:%f
channel-map = mono
paths-output = analog-output analog-output-lineout analog-output
paths-input = analog-input-front-mic analog-input-rear-mic analo
priority = 7
```

...

下面是注释后的内容：

...

```
;[Mapping analog-mono]
;device-strings = hw:%f
;channel-map = mono
;paths-output = analog-output analog-output-lineout analog-outpu
;paths-input = analog-input-front-mic analog-input-rear-mic anal
;priority = 7
```

...

最后重启系统生效。

启用和运行Docker服务

默认没有启用 Docker服务，需要手动启动。

开机自动启动Docker服务

```
systemctl enable docker.service
```

启动Docker服务

```
systemctl start docker.service
```

停止Docker服务

```
systemctl stop docker.service
```

禁止Docker服务开机启动

```
systemctl disable docker.service
```

安装chromium

[下载地址](#) 使用 `sudo dpkg -i *.deb` 安装, 按顺序安装

1. chromium-codecs-ffmpeg-extra
2. chromium-browser
3. chromium-chromedriver

安装smb

```
sudo apt-get install samba  
sudo apt-get install samba-common
```

smb共享

接下来修改 Samba 的配置文件, 打开/etc/samba/smb.conf, 在末尾添加以下内容:

```
[share] # 共享名称  
comment = my share disk # 描述信息  
path = /home/pi/share # 共享的目录  
browserable = yes # 访问权限  
writable = yes # 写入权限
```

smb重启

```
sudo service smbd restart
```

挂载硬盘

```
sudo df -h  
sudo mount /dev/sda1 /home/pi/share
```

设置硬盘的开机挂载

```
sudo vi /etc/fstab  
/dev/sda1 /home/pi/share ext4 defaults 0 0
```

程序快捷目录

```
/usr/share/applications
```

docker 镜像安装

打开树莓派终端，输入下面命令获得超级用户权限

```
sudo -i
```

安装Alist

```
curl -fsSL "https://alist.nn.ci/v3.sh" | bash -s install
```

php

```
docker run -d --restart unless-stopped --privileged=true -p 5678  
docker cp /home/pi/公共/yy.php php-env:/var/www/html/
```

xteve

```
docker run -d \  
  --name=xteve_guide2go \  
  --net=host \  
  --log-opt max-size=10m \  
  --log-opt max-file=3 \  
  -e TZ="Asia/Shanghai" \  
  -v /home/pi/appdata/xteve:/root/.xteve:rw \  
  -v /home/pi/appdata/xteve/_config:/config:rw \  
  -v /home/pi/appdata/xteve/_guide2go:/guide2go:rw \  
  -v /tmp/xteve:/tmp/xteve:rw \  
  -v /home/pi/appdata/tvheadend/data:/TVH \  
  jjm2473/xteve_guide2go
```

aria2

```
docker run -d \
    --name aria2-pro \
    --restart unless-stopped \
    --log-opt max-size=1m \
    -e PUID=$UID \
    -e PGID=$GID \
    -e UMASK_SET=022 \
    -e RPC_SECRET=wsj0051 \
    -e RPC_PORT=6800 \
    -p 6800:6800 \
    -e LISTEN_PORT=6888 \
    -p 6888:6888 \
    -p 6888:6888/udp \
    -v $PWD/appdata/aria2/aria2-config:/config \
    -v $PWD/downloads:/downloads \
    p3terx/aria2-pro
```

aria2 webUi控制台

```
docker run -d \
    --name ariang \
    --log-opt max-size=1m \
    --restart unless-stopped \
    -p 6880:6880 \
    p3terx/ariang
```

mysql

- 安装

```
sudo docker run --name mysql \
    -v /home/pi/mysql:/var/lib/mysql \
    -e MYSQL_ROOT_PASSWORD=pi \
    -e MYSQL_DATABASE=nextcloud \
    -e MYSQL_USER=nextcloud \
    -e MYSQL_PASSWORD=password \
    -p 3306:3306 \
    --restart=always \
    -d mysql/mysql-server
```

- 赋予本地文件夹读写权限

```
sudo chmod -R 777 /home/mysql
```

- 进入mysql容器，并登陆mysql

```
docker exec -it mysql bash  
mysql -u root -p
```

- 创建用户，赋予权限

```
Create user 'nextcloud'@'%' identified by '123456';  
grant all privileges on nextcloud.* to 'nextcloud'@'%';
```

- 开启远程访问权限

```
use mysql;  
select host,user from user;  
alter user 'root'@'%' identified with mysql_native_password;  
flush privileges;
```

- 开机启动

```
sudo docker update mysql --restart=always
```

安装owncloud

1. 在docker中安装私有云，在树莓派终端输入下面命令。（注意：为了避免出错，下面用到的所有命令以及要修改的内容也给大家提供了文档版本，可以在树莓派爱好者基地微信公众号发送【私有云安装】获得）

```
docker run -d -p 8888:80 --name nextcloud -v /data/nextcloud
```

2. 修改配置文件

```

```shell
mkdir /data/nextcloud/config
sudo docker cp nextcloud:/var/www/html/config/config.sample.php
sudo docker cp /data/nextcloud/config/config.php nextcloud:/var
nano /data/nextcloud/config/config.sample.php
```

修改`trusted_domains`里的值

```txt
1 => preg_match('/cli/i',php_sapi_name())?'127.0.0.1':$_SERVER['
```

```

1. 安装nextcloud客户端

```
apt install nextcloud-desktop-l10n nextcloud-desktop -y
```

vim不能右键粘贴

2. 编辑 vim 的默认配置文件

```
vim /usr/share/vim/vim80/defaults.vim
```

3. 转至第 70 行, 找到:

```

if has('mouse')

set mouse=a

endif

```

4. 将 `set mouse=a` 改为: `set mouse-=a`

5. 输入 `:wq` 保存即可生效。

- [Termux](#)
 - [访问手机存储](#)
 - [创建软链](#)
 - [修改为清华源](#)
 - [安装基本工具](#)
 - [修改终端配色](#)
 - [zsh插件autosuggestions](#)
 - [拷贝到 plugins 目录下](#)
 - [Termux快捷按键](#)
 - [ssh服务](#)
 - [修改启动问候语](#)
 - [重复执行问题](#)
 - [修改neofetch配置](#)
 - [npm安装http-server](#)
 - [使用ecj termux-tools dx编译java文件](#)
 - [手机通知栏时间打开时分秒](#)
 - [使用atilo安装linux](#)
 - [配置apache java环境](#)
 - [电脑ssh连接termux](#)
 - [一键安装tmoe-linux](#)
 - [备份与恢复](#)
 - [参考链接](#)

Termux

访问手机存储

```
termux-setup-storage
```

执行上面的命令以后，会跳出一个对话框，询问是否允许 Termux 访问手机存储，点击"允许"。

创建软链

直接跳转到手机内存卡对应目录的快捷方式

```
ln -s /data/data/com.termux/files/home/storage/shared/ws0051 ws
```

修改为清华源

使用如下命令行替换官方源为 [TUNA 镜像源](#)

```
sed -i 's@^(deb.*stable main\)$@#\1\ndeb https://mirrors.tuna.t  
sed -i 's@^(deb.*games stable\)$@#\1\ndeb https://mirrors.tuna.  
sed -i 's@^(deb.*science stable\)$@#\1\ndeb https://mirrors.tun  
apt update && apt upgrade
```

安装基本工具

```
pkg update  
pkg upgrade  
pkg install vim curl wget git unzip unrar
```

修改终端配色

```
sh -c "$(curl -fsSL https://github.com/Cabbagec/termux-ohmyzsh/r
```

脚本运行后会提示选择背景色和字体

```
Enter a number, leave blank to not to change: 14  
Enter a number, leave blank to not to change: 6
```

设置色彩样式:

输入 `chcolor` 命令更换色彩样式, 或者执行 `~/.termux/colors.sh` 命令

设置字体

运行 `chfont` 命令更换字体, 或者执行 `~/.termux/fonts.sh` 命令

zsh插件autosuggestions

根据用户的平时使用习惯, 终端会自动提示接下来可能要输入的命令, 这个实际使用效率还是比较高的:

拷贝到 plugins 目录下

```
git clone https://github.com/zsh-users/zsh-autosuggestions $ZSH_
```

在 `~/.zshrc` 中配置:

```
plugins=(插件名称 zsh-autosuggestions)
```

Termux快捷按键

```
mkdir $HOME/.termux;  
echo "extra-keys = [ \  
    ['ESC','<','>','BACKSLASH','=','^','$','(',')','{','}','[',']', 'ENTER'  
    ['TAB','&',';','/','~','%','*','HOME','UP','END','PGUP'], \  
    ['CTRL','FN','ALT','|','-','+', 'QUOTE','LEFT','DOWN','RIGHT'  
    ]  
" >> $HOME/.termux/termux.properties
```

ssh服务

```
pkg install openssh
```

安装完成后, sshd 服务默认没有启动, 所以得手动启动下:

```
sshd
```

停止 ssh 服务

```
pkill sshd
```

因为手机上面低的端口有安全限制, 所以这里 openssh 默认的 sshd 默认的服务端口号为 8022 执行 passwd 命令可以直接修改密码 连接方式:

```
ssh 192.168.31.145 -p 8022
```

修改启动问候语

```
vim $PREFIX/etc/motd
```

如果没有安装vim的话会有提示，根据提示安装：`pkg install vim`

修改启动语为sh脚本方式

```
cd $PREFIX/etc  
vim motd
```

motd内容修改为脚本文件，内容为：

```
#!/$PREFIX/bin/bash  
neofetch
```

修改后保存并退出，执行以下命令

```
mv motd profile.d/motd.sh
```

重复执行问题

如果启动后出现触发两次，将sh文件执行语句放进.zshrc下

```
mv $PREFIX/etc/profile.d/motd.sh .  
echo "$PREFIX/bin/bash ~/motd.sh" >> ~/.zshrc
```

修改neofetch配置

```
cd .config/neofetch  
vim config.conf
```

可以修改展示的信息，颜色，修改 `ascii_distro="linux"` 将默认的安卓换为linux

npm安装http-server

```
npm install -g http-server
```

然后，运行 Server。

```
http-server
```

正常情况下，命令行会提示 Server 已经在 8080 端口运行了，并且还会提示外部可以访问的 IP 地址。

使用ecj termux-tools dx编译java文件

1. 更新资源

```
pkg update & pkg upgrade
```

2. 安装所需软件

```
pkg install ecj termux-tools dx
```

3. 创建java文件Hello.java

```
public class Hello{  
    public static void main(String[] args){  
        System.out.println("Hello World!");  
    }  
}
```

4. 编译java文件

```
ecj Hello.java
```

5. 生成安卓虚拟机文件

```
dx --dex --output=Hello.dex Hello.class
```

6. 安卓虚拟机运行程序

```
dalvikvm -cp Hello.dex Hello
```

7. 更简单方式，创建shell脚本 `vim ecj.sh`

```
#!/usr/bin/sh
ecj "$1.java"
dx --dex --output="$1.dex" "$1.class"
dalvikvm -cp "$1.dex" "$1"
```

8. 执行shell编译java

```
sh ecj.sh Hello
```

手机通知栏时间打开时分秒

手机通知栏的时间没有精确到秒，手机root后可以打开

1. 使用一键方式安装adb工具
2. 执行以下命令：

```
pkg install tsu
adb shell
settings put secure clock_seconds 1
```

使用atilo安装linux

在Termux安装Linux的bash脚本

```
echo "deb [trusted=yes] https://yadominjinta.github.io/files/ t
pkg in atilo-cn
```

安装debian

```
atilo pull debian
```

运行debian

```
atilo run debian
```

配置apache java环境

1. 在usr/local下创建java文件夹，将下载好的tar文件移动到java目录下，对应手机目录

```
为 /data/data/com.termux/files/home/.atilo/debian/usr/local/j  
ava
```

2. 解压jdk和tomcat

```
cd /usr/local/java  
tar xzf jdk-8u241-linux-arm64-vfp-hflt.tar.gz  
tar xzf apache-tomcat-9.0.31.tar.gz
```

3. 以下内容拷进 /etc/profile 文件末尾

```
JAVA_HOME=/usr/local/java/jdk1.8.0_241  
PATH=$JAVA_HOME/bin:$PATH  
CLASSPATH=$JAVA_HOME/jre/lib/ext:$JAVA_HOME/lib/tools.jar  
export PATH JAVA_HOME CLASSPATH
```

4. 保存退出后执行以下命令让配置生效，使用 java -version 查看环境变量是否配置成功

```
source /etc/profile
```

电脑ssh连接termux

```
sshd
```

SSH 默认配置文件的路径为 \$PREFIX/etc/ssh/sshd_config

```
passwd
```

Termux 的 ssh 和常规 Linux 不太一样，连接的时候不需要指定用户名。也可以使用公钥连接

```
cat id_rsa.pub > authorized_keys
```

一键安装tmoe-linux

```
. <(curl -L gitee.com/mo2/linux/raw/2/2)
```

备份与恢复

进入目录

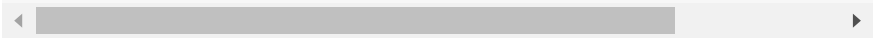
```
cd /data/data/com.termux/files
```

备份配置文件为 `termux-backup.tar.gz`：

```
tar -zcf /sdcard/termux-backup.tar.gz home usr
```

解压提取之前备份的内容，覆盖现存的文件并删除之前的备份文件：

```
tar -zxf /sdcard/termux-backup.tar.gz --recursive-unlink --prese
```



参考链接

1. [国光-Termux](#)
2. [简书](#)
3. [tmoe](#)

- 未分类
 - [Jenkins](#)
 - [maven安装本地jar](#)
 - [vscode 正则匹配空白行](#)
 - [BATOCERA](#)
 - [康佳电视 - LED55K36U 刷机](#)
 - [windows记录](#)
 - [cmd查看并设置代理](#)
 - [上帝模式](#)
 - [浏览器主页被篡改修复](#)
 - [高分辨率远程桌面低分辨率屏幕内容过小解决办法](#)
 - [隐藏文件](#)
 - [路径](#)
 - [Openwrt路由器](#)
 - [查看openwrt设备的cpu架构](#)
 - [查看openwrt已连接网络ip](#)
 - [openwrt关闭led灯](#)

未分类

Jenkins

- [插件下载地址](#)
 - [hpi插件地址](#)
 - [Theme插件](#)
- [页面自定义](#)
 - [修改jar包 jenkins/WEB-INF/lib/jenkins-core-1.651.3.jar 文件中的 lib/layout/layout.jelly](#)

maven安装本地jar

```
mvn install:install-file -Dfile=junit-4.8.1.jar -DgroupId=junit
```

vscode 正则匹配空白行

```
^\s*(?=\r?$)\n
```

BATOCERA

配置文件

```
wifi.enabled=1
wifi.ssid=wifi1-name
wifi.key=*****
wifi2.ssid=wifi2-name
wifi2.key=*****
wifi3.ssid=wifi1-name
wifi3.key=*****
```

康佳电视 - LED55K36U 刷机

1. 电视型号：

- 主程序软件：99015826
- 屏幕参数：99016671

2. 刷机步骤

- 主程序软件，包名：M638Upgrade.bin
 - 将对应的刷机文件放到u盘根目录（u盘要fat32格式，尽量不要用内存卡，刷机文件不要改名）
 - 断电，按住音量+加键(有些机型是信源键)，开电

3. 若刷机后出现双屏

- 刷屏幕参数，包名：M638PanelUpgrade.bin
 - 按遥控器的音量减键不放，然后通电。

windows记录

cmd查看并设置代理

1. 设置socks代理

```
set ALL_PROXY socks5://127.0.0.1:10808
```

2. 配置http代理

```
set http_proxy=http://127.0.0.1:10809
set https_proxy=http://127.0.0.1:10809
```

3. 查看代理

```
netsh winhttp show proxy
```

4. 清除代理

```
netsh winhttp reset proxy
```

上帝模式

新建文件夹，重命名为 上帝模式.{ED7BA470-8E54-465E-825C-99712043E01C}

浏览器主页被篡改修复

任务栏浏览器主页被篡改，右键属性就会看到目标后，有一串网址，删除解决，路径： C:\Users\用户名\AppData\Roaming\Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar

高分辨率远程桌面低分辨率屏幕内容过小解决办法

1. 修改注册表：用运行-regedit编辑注册表，找到： HKEY_LOCAL_MACHINE > SOFTWARE > Microsoft > Windows > CurrentVersion > SideBySide 新建 DWORD，命名 PreferExternalManifest，并双击设置值为 1。
2. 创建文件mstsc.exe.manifest，并放到该路径下： C:\Windows\System32

隐藏文件

```
attrib +s +a +h +r E:\hide  
attrib -s -a -h -r E:\hide
```

路径

1. win11当前壁纸路径 C:\Users\用户名\AppData\Roaming\Microsoft\Windows\Themes\CachedFiles\
2. hosts文件路径 C:\Windows\System32\drivers\etc
3. windows保护历史记录删除 C:\ProgramData\Microsoft\Windows Defender\Scans\History\Service\DetectionHistory

4. 开机启动 C:\ProgramData\Microsoft\Windows\Start
Menu\Programs\Startup

Openwrt路由器

查看openwrt设备的cpu架构

```
cat /etc/os-release |grep ARCH
```

查看openwrt已连接网络ip

使用DHCP客户端查看mac地址、ip信息

```
cat /tmp/dhcp.leases
```

通过arp查看ip、mac地址、端口

```
cat /proc/net/arp
```

openwrt关闭led灯

保存为 /etc/rc.d/S99turnoffled

```
#!/bin/ash
for i in `ls /sys/class/leds`
do cd /sys/class/leds
cd $i
echo 0 > brightness
done
```