**HTML**

超文本标记语言（英语：Hyper Text Markup Language，简称：HTML）是一种用于创建网页的标准标记语言。HTML可以建立自己的Web站点，HTML运行在浏览器上，由浏览器来解析。

**1. HTML开头<head>**

|  |
| --- |
| <!DOCTYPE html> // 声明HTML5文件  <html> // 根元素  <head> // 文档的元（meta）  <meta charset="utf-8"> // 声明编码方案  <title>First HTML</title> // 描述文档标题，浏览器标签  </head>  <body> // 可见页面内容  <h1>My First Title</h1> // 标题  <p>My first paragraph</p> // 段落  </body>  </html> |

把内容直接保存成.html文件，用浏览器就能打开

<head>里还可有base，例如

<base href="//www.runoob.com/images/" target="\_blank">

这就指定里根url是www.runoob.com/images/，如果www.runoob.com/images/下有一个图片，则

<img src="logo.png">

就可以引用图片了

以及页面打开方式target="\_blank"是新建页面，不是\_blank的话，点击链接之后页面直接在当前页切换

还有<script>标签用于加载脚本文件，如：JavaScript。

**2. 段落和结构**

<h1></h1>到<h6></h6>表示1 - 6级标题

<p></p>表示一个段落

<br/>表示换行

<hr/>表示横线

居中表示

<p style="text-align:center;">This is a Chapter</p>

改变字体font-family、颜色color、字号font-size

<p style="font-family:Times;color:red;font-size:20px;">Content</p>

**3. 文本编辑**

<!-- 内容 -->表示注释

<b>加粗</b> 表示加粗

<strong>强壮的字体</strong>表示加粗（首选用这个）

<em>强调</em>表示斜体（首选用这个）

<i>斜体</i> 表示斜体

<small>小号字体</small> 表示小号字体

<sub>下标</sub> 表示下标

<sup>上标</sup> 表示上标

<ins>描述插入的文字</ins> 表示插入的文字

<del>描述删除的文字</del> 表示删除的文字

<abbr title="Kalman Filtered Compressive Sensing">KFCS</ abbr>表示缩写

<address>表示地址

<address>

Author: <a href="mailto:haibin.zhao@kit.edu">Haibin Zhao</a><br>

Address: Marienstraße 12, Karlsruhe <br>

Germany <br>

</address>

<cite>引用名词</ cite>引用名词，不含人名

<q>引用句子</ q>引用句子

<blockquote cite="www.a.com">引用段落</blockquote>引用段落

<pre>文本</ pre>内的文本保留空格和空行

<code>一段代码</ code>显示代码，一般配合<pre>使用

**4. 图片**

<img loading="lazy" src="Photo.jpeg" width="258"/>插入图片

里面可以加入alt="替换文字"，当图片无法加载时显示文字

加载gif也是一样的

**5. 超链接**

超链接

<a href="https://.../">显示内容</a>

打开新窗口的超链接

<a href="https://.../" target="\_blank">显示内容</a>

图片超链接：只需要在<a><img/></a>之间插入图片即可

<a href="//www.runoob.com/html/html-tutorial.html">

<img border="0" src="smiley.gif" alt="HTML 教程" width="32" height="32"></a>

**6. 发送邮件**

<a href="mailto:someone@example.com" target="\_blank">发送邮件</a>

给邮件增加内容，规则：

1.写mailto:example.com

2.加一个问号

3.添加内容，内容和内容之间用&连接

内容有：

主题subject=

内容body=

抄送cc=

密送bcc=

4.其中%20表示空格，%0d%0a表示换行

举例

<a href="mailto:hzhao@teco.edu?Subject=Hello%20World!&body=Name:%20%0d%0aThings:%20%0d%0aLinks:%20%0d%0aQuantity:%20%0d%0aContact:%20%0d%0aReason:%20%0d%0aSupervisor:%20" target="\_blank">发送邮件</a>

**7. 页面内部链接**

在内容上加入一个id

<h2><a id="C4">Chapter 4</a></h2>

在需要点击链接的地方加上

<a href="#C4">Go To Chapter 4</a>

**8. 表格**

<table border="1" <!-- border=0 就没有边框了-->

cellpadding="10"> <!-- 单元格边距，可以没有-->

<caption>Monthly savings</caption> <!-- 标题 -->

<tr> <!-- tr表示一行 -->

<th>100</th> <!-- th表示表头 -->

<th>200</th>

<th>300</th>

</tr>

<tr>

<td>100</td> <!-- td表示一列 -->

<td>200</td>

<td>300</td>

</tr>

<tr>

<td>400</td>

<td>500</td>

<td>600</td>

</tr>

</table>

横跨几列

<tr>

<td colspan="2">Hello</td>

<td>500</td>

</tr>

竖跨几行

<tr>

<td rowspan="2">Hello</td>

<td>500</td>

</tr>

下面的行：

<tr>

<td>500</td> <!-- 只有一列，因为第一列被占了 -->

</tr>

**9. 列表**

<ul> <!-- 无序号列表 -->

<li>Coffee</li> <！-- 一个列表项 -->

<li>Milk</li>

</ul>

<ul>写成<ol>就变成了有序列表

<ul>有不同的编号方式style

style="list-style-type:disc"就是圆点

style="list-style-type:circle"就是圆圈

style="list-style-type:square"就是小方框

<ol>有不同的编号方式type，默认是阿拉伯数字。

type="A"就是ABC

type="a"就是abc

type="I"就是罗马数字

type="i"就是小写罗马数字

**10. 交互区域**

填写区域

明文：以填写名字为例

<form action="">

First name: <input type="text" name="firstname"><br>

Last name: <input type="text" name="lastname">

</form>

密码：以填写密码为例

<form action="">

Username: <input type="text" name="user"><br>

Password: <input type="password" name="password">

</form>

文本域

<textarea rows="10" cols="30">

I am a text area.

</textarea>

框框把东西框起来

<form action="">

<fieldset>

<legend>Personal information:</legend> <!-- 内容标记 -->

Name: <input type="text" size="30"><br>

E-mail: <input type="text" size="30"><br>

Date of birth: <input type="text" size="10">

</fieldset>

</form>

填写区域的事件：

选择区域

单选

<form>

<input type="radio" name="sex" value="male">Male<br>

<input type="radio" name="sex" value="female">Female

</form>

复选框

<form>

<input type="checkbox" name="vehicle" value="Bike">I have a bike<br>

<input type="checkbox" name="vehicle" value="Car">I have a car

</form>

下拉选择框

<form action="">

<select name="cars">

<option value="volvo">Volvo</option>

<option value="saab">Saab</option>

<option value="fiat">Fiat</option>

<option value="audi">Audi</option>

<option value="BMW" selected>BMW</option> <!-- 可以设置预选 -->

</select>

</form>

按钮

以发送邮件为例：

<form action="mailto:haibin.zhao@kit.edu?subject=Bestellung", method="post">

名字 <input type="text" name="name" value="MyName"> <br />

<input type="submit" value="发送"><br />

<input type="reset" value="重置">

</form>

动作action是个链接，当表单提交时向何处发送表单数据。

可以是个url

发送邮件时写成mailto的形式

方法method有2种：

get：将表单数据以名称/值对的形式附加到URL中，

post：以 HTTP post 事务的形式发送表单数据。

类型type有好几种

text, submit, reset, button之类的

**11. 颜色**

16进制：#000000

RGB：rgb(0,0,0)

**12. 写入JavaScript代码**

用<script>装起来

<script>

document.write("Hello World!")

</script>

<noscript>抱歉，你的浏览器不支持 JavaScript!</noscript>

<noscript>可以显示如果不支持JS的时候的内容

**CSS**

层叠样式表（英语：Cascading Style Sheets，缩写：CSS）是一种用来为结构化文档（如HTML文档或XML应用）添加样式（字体、间距和颜色等）的计算机语言。

**0. 基本语法**

<head> /\*注释写法\*/

<meta charset="utf-8">

<title> </title>

<style> /\*写在<style>里面\*/

body /\*选择器\*/

{

background-color:#d0e4fe; /\*属性：值；\*/

} /\*声明总是以;结束\*/

h1

{

color:orange;

text-align:center;

}

p

{

font-family:"Times New Roman";

font-size:20px;

}

</style>

</head>

id选择器

<style>

#para1 /\*只对id是para1的生效\*/

{

text-align:center;

color:red;

}

</style>

.......

<body>

<p id="para1">Hello World!</p> /\*设置id\*/

<p>这个段落不受该样式的影响。</p>

</body>

class选择器

定义class

<style>

.center

{

text-align:center;

}

</style>

使用class

<h1 class="center">标题居中</h1>

<p class="center">段落居中。</p>

定义带有类型的class

<style>

p.center /\*只对p有影响\*/

{

text-align:center;

}

</style>

使用带类型的class

<h1 class="center">这个标题不受影响</h1>

<p class="center">这个段落居中对齐。</p>

**1. 调用样式表（Style Sheet）**

可以调用内部和外部样式表。内部样式表正如上面所写，外部样式表如下的方式调用：

<head>

<link rel="stylesheet" type="text/css" href="mystyle.css">

</head>

其中样式表mystyle.css的内容是

hr {color:sienna;}

p {margin-left:20px;}

body {background-image:url("/images/back40.gif");}

**！不要在属性值与单位之间留有空格**

**2. 背景background**

background-color /\*设置背景的颜色\*/

background-color:rgb(0,150,130);

background-image /\*设置背景为图片\*/

background-image:url('/picture.png');

background-repeat /\*图片默认是平铺的，控制图片重复方向\*/

background-repeat:repeat-x; /\*横向重复\*/

background-repeat:repeat-y; /\*纵向重复\*/

background-repeat:no-repeat; /\*不重复\*/

background-position

background-position:right top;

background-attachment /\*背景图像是否固定或者随着页面的其余部分滚动\*/

**3. 文本Text**

比如h1 {text-align:justify;}

color 设置文本颜色

direction 设置文本方向

ltr 默认，left to right

rtl right to left

inherit 规定应该从父元素继承direction属性的值。

letter-spacing 设置字符间距

h1 {letter-spacing:2px}

line-height 设置行高

p.small {line-height:90%}

p.big {line-height:200%}

text-align 对齐元素中的文本

h1 {text-align:center}

h2 {text-align:left}

h3 {text-align:right}

text-decoration 向文本添加修饰

h1.under {text-decoration: underline;} /\*下划线\*/

h1.over {text-decoration: overline;} /\*上划线\*/

p.line {text-decoration: line-through;} /\*删除线\*/

p.blink {text-decoration: blink;} /\*文字闪烁\*/

a.none {text-decoration: none;} /\*去掉所有效果\*/

p.underover {text-decoration: underline overline;} /\*可以加许多效果\*/

text-indent 缩进元素中文本的首行

p {text-indent:50px;}

text-shadow 设置文本阴影

h1{text-shadow: 2px 2px #ff0000;} /\*x方向，y方向，颜色\*/

text-transform 控制元素中的字母

capitalize /\*文本中的每个单词以大写字母开头\*/

uppercase /\*定义仅有大写字母\*/

lowercase /\*定义无大写字母，仅有小写字母\*/

vertical-align 设置元素的垂直对齐

baseline /\*默认，元素放置在父元素的基线上\*/

sub /\*垂直对齐文本的下标\*/

super /\*垂直对齐文本的上标\*/

top /\*把元素的顶端与行中最高元素的顶端对齐\*/

text-top /\*把元素的顶端与父元素字体的顶端对齐\*/

middle /\*把此元素放置在父元素的中部\*/

bottom /\*使元素及其后代元素的底部与整行的底部对齐\*/

text-bottom /\*把元素的底端与父元素字体的底端对齐\*/

length /\*将元素升高或降低指定的高度，可以是负数\*/

% /\*使用 "line-height" 属性的百分比值来排列此元素，允许使用负值\*/

white-space 设置元素中空白的处理方式

pre /\*空白会被浏览器保留，其行为方式类似 HTML 中的 <pre> 标签\*/

nowrap /\*文本不会换行，文本会在在同一行上继续，直到遇到 <br> 标签为止\*/

pre-wrap /\*保留空白符序列，但是正常地进行换行\*/

pre-line /\*合并空白符序列，但是保留换行符\*/

word-spacing 设置字间距

p{word-spacing:30px;}

**4. 字体Font**

字体有2种。特定字体如：宋体，Times。通用字体：Serif，Sans-serif，Monospace等。前者比不多说，后者是满足某些特征的字体

Serif 有棱角的字，如Times

Sans-serif 比较圆润的字，如Arial

Monospace 间距相同的字，如Consolas

font-family 指定文本的字体系列

p{font-family:"Times New Roman", Times, serif;} /\*可以指定多个作为备选，字体带空格时，用双引号包起来\*/

font-size 指定文本的字体大小

font-size: xx-small; /\*代号\*/

font-size: x-small;

font-size: small;

font-size: medium;

font-size: large;

font-size: x-large;

font-size: xx-large;

font-size: larger;

font-size: smaller;

font-size: 12px; /\*绝对尺寸\*/

font-size: 0.8em;

font-size: 80%; /\*相对尺寸\*/

font-style 指定文本的字体样式

normal /\*默认值，浏览器显示一个标准的字体样式\*/

italic /\*浏览器会显示一个斜体的字体样式\*/

oblique /\*浏览器会显示一个倾斜的字体样式，对于没有斜体的字体\*/

font-variant 以小型大写字体或者正常字体显示文本。

normal /\*默认值，浏览器会显示一个标准的字体\*/

small-caps /\*浏览器会显示小型大写字母的字体\*/

font-weight 指定字体的粗细

normal /\*默认值，定义标准的字符\*/

bold /\*定义粗体字符\*/

bolder /\*定义更粗的字符\*/

lighter /\*定义更细的字符\*/

100 - 900 /\*定义由细到粗的字符，400 等同于 normal，而 700 等同于 bold\*/

**5. 链接样式**

a:link - 正常，未访问过的链接

a:link {color:#000000;} /\* 未访问链接\*/

a:visited - 用户已访问过的链接

a:visited {color:#00FF00;} /\* 已访问链接 \*/

a:hover - 当用户鼠标放在链接上时

a:hover {color:#FF00FF;} /\* 鼠标移动到链接上 \*/

a:active - 链接被点击的那一刻

a:active {color:#0000FF;} /\* 鼠标点击时 \*/

**6. 列表**

列表的边缘

ul {

margin: 0;

padding: 0;

list-style-type: none; /\*列表的记号\*/

}

列表的记号

ul {list-style-type:circle;}

ul {list-style-type:square;}

ol {list-style-type:upper-roman;}

ol {list-style-type:lower-alpha;}

还可以用图像，还可以改变记号的位置

**7. Overflow**

文本会超出div的框，可以做出一些选择

visible 默认值。内容不会被修剪，会呈现在元素框之外。

hidden 内容会被修剪，并且其余内容是不可见的。

scroll 内容会被修剪，但是浏览器会显示滚动条以便查看其余的内容。

auto 如果内容被修剪，则浏览器会显示滚动条以便查看其余的内容。

定义div的属性

#overflowTest {

padding: 15px;

width: 80%;

height: 100px;

overflow: scroll;

border: 1px solid #ccc;

}

使用

<div id="overflowTest">

<p>这里的文本内容是可以滚动的，滚动条方向是垂直方向。</p>

<p>这里的文本内容是可以滚动的，滚动条方向是垂直方向。</p>

<p>这里的文本内容是可以滚动的，滚动条方向是垂直方向。</p>

<p>这里的文本内容是可以滚动的，滚动条方向是垂直方向。</p>

<p>这里的文本内容是可以滚动的，滚动条方向是垂直方向。</p>

<p>这里的文本内容是可以滚动的，滚动条方向是垂直方向。</p>

</div>

**8. 居中和对齐**

元素居中：margin:auto;

.center {

margin: auto;

width: 50%; /\*不写或100%都不行\*/

}

文本居中

text-align: center;

图片居中

img {

display: block;

margin: auto;

width: 40%;

}

左右对齐

.right {

position: absolute;

right: 0px; /\*改成left就是居右\*/

width: 300px;

border: 3px solid #73AD21;

padding: 10px;

}

**JavaScript**

JavaScript 是互联网上最流行的脚本语言，这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。JavaScript 是脚本语言，

JavaScript 是一种轻量级的编程语言，

JavaScript 是可插入 HTML 页面的编程代码，

JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。

**0.基本用法**

在HTML里面用<script>把代码包裹起来即可

<script>

document.write("<h1>这是一个标题</h1>");

document.write("<p>这是一个段落。</p>");

</script>

<script>定义函数的位置可以在<head>或者末尾或任何地方。

也可以从外部调用脚本：

<script src="myScript.js"></script>

其中myScript.js的内容是：

function myFunction()

{

document.getElementById("demo").innerHTML="My first JavaScript function.";

}

注释方法

// 单行注释

/\*

多行注释

\*/

**1. 基础**

声明

var a; // 可变，全局变量

let b; // 可变，作用域只在块里(if, while...)

const pi = Math.PI; // 常量

数据类型

String

Number // 包含整数和浮点数

Boolean

Null

Undefined

Object

引号嵌套：外面单引号，里面双引号。或者用\"

类型转换：

var num = 123;

var str = num.toString(); // 方法

var str = String(num); // 函数

var a = "123";

var b = Number(a);

var b = parseInt(a);

var b = parseFloat(a);

字符串拼接："+"

字符串的方法

.length

.replace("a","b")

.toUpperCase()

取余%

自增自减

var a = 1;

a++ // 先使用a再增加

++a // 先增加1再使用a

a = a + 3; 等价于 a += 3;

逻辑运算

与 &&

或 ||

大于 > // >=

小于 < // <=

等于 == // 比较时会进行类型转换

不等于 != // 比较时会进行类型转换

全等 === // 比较时不会进行类型转换

不全等 !== // 比较时不会进行类型转换

三元运算符

boolean? state1 : state2

圆周率

Math.PI

**控制流**

判断

if ()

{ }

else if ()

{ }

else

{ }

循环

while()

{ }

或者

do // 先运行后判断

{ } while()

或者

for ( ; ; )

{ }

**对象（类似于字典）**

声明对象

var obj = new Object();

var obj = {}; // 都可以

定义

obj = {

name : "Haibin",

age : "20",

email : "haibin.zhao@kit.edu",

contact : {

phone : "17663413624",

address : "Marienstraße"

}

};

访问

obj["name"]

修改

obj.name = "Tobias"

增添

obj.wechat = "hahaha"

obj.contact.plt = "76131"

**数组（列表）**

var a = new Array();

var a = [];

定义

a[0] = "dog";

a[1] = "cat";

常用

追加一个

a.push("lion");

删除最后一个

a.pop();

反转

a.reverse();

删除第一个

a.shift();

在开头增加

a.unshift();

读取

for(let i = 0; i < a.length; i++)

{

console.log(a[i]);

}

或者

for(let i in a)

{

console.log(a[i]);

}

**函数**

var a = 0;

function add(x)

{

a += x;

}

add(4);

多个传入量

function add()

{

let sum = 0;

for (let i = 0, j = arguments.length; i < j; i++)

{

sum += arguments[i];

}

return sum;

}

**Node.js**

node可以运行.js文件！

创建服务器

|  |
| --- |
| //读取内置模块http，用于开发服务器  var http = require("http");  // 创建服务器  var server = http.createServer(function(req,res)  {  res.end("<h1>Hello World, this is my first Server!</h1>");  });  // 监听  server.listen(3000, "127.0.0.1");  // 命令行反馈  console.log("Server is running..."); |

浏览器输入127.0.0.1:3000就能看到。3000是端口（port）

node没有根目录，单线程（thread）

设置html的header

|  |
| --- |
| var server = http.createServer(function(req,res)  {  res.setHeader("Content-Type", "text/html; charset=UTF-8");  res.end("Hello World!");  }); |

每次访问server都会调用服务器。每次刷新网页或有人访问，res\_num都会增加。

|  |
| --- |
| var res\_num = 1;  var server = http.createServer(function(req,res)  {  res\_num = res\_num + 1; // PS: 每次访问网页都会产生多次请求  res.setHeader("Content-Type", "text/html; charset=UTF-8");  res.end(res\_num.toString());  }); |

读取文件：

|  |
| --- |
| var fs = require("fs");  var server = http.createServer(function(req,res)  {  res.setHeader("Content-Type", "text/html; charset=UTF-8");  fs.readFile("text.txt", function(err, filecontext) // 读取文件  {  res.end(filecontext); // filecontext就是内容  });  }); |

文件还可以是html文件，就能直接显示页面了

子页面（路由）req.url

|  |
| --- |
| var server = http.createServer(function(req,res)  {  res.setHeader("Content-Type", "text/html; charset=UTF-8");  if (req.url == '/')  {  res.end("Main Page");  }  else if (req.url == '/music')  {  res.end("Music page");  }  else  {  res.end("No such page!");  }  }); |

\*插曲：正则表达式

|  |
| --- |
| var str = "I am 12 years old, and I have 20 apples. My mon is 30 years old. She has 100 oranges.";  var reg = /\d/g; // 正则表达，取出整数  var result = reg.exec(str);  console.log(result); |

/\* 返回的result是一个数组：

[

'1', // 这就是整数

index: 5, // 整数的index

input: 'I am 12 years old, and I have 20 apples. My mon is 30 years old. She has 100 oranges.',

groups: undefined

]

\*/

|  |
| --- |
| result = reg.exec(str); // 再次调用exec则再读下一个数  console.log(result); |

/\* 返回的result是：

[

'2',

index: 6,

input: 'I am 12 years old, and I have 20 apples. My mon is 30 years old. She has 100 oranges.',

groups: undefined

]

\*/

|  |
| --- |
| var reg = /\d+/g; // 有了加号，则直接读完整个整数  var result = reg.exec(str);  console.log(result); |

/\* 返回的result是：

[

'12', // 找到的整数

index: 5,

input: 'I am 12 years old, and I have 20 apples. My mon is 30 years old. She has 100 oranges.',

groups: undefined

]]

\*/

|  |
| --- |
| var reg = /\d+ years old/g; // 指定整数的寻找规则，后面有" years old"  var result = reg.exec(str);  console.log(result); |

/\* 返回的result是：

[

'12 years old', // 这里

index: 5,

input: 'I am 12 years old, and I have 20 apples. My mon is 30 years old. She has 100 oranges.',

groups: undefined

]

\*/

|  |
| --- |
| var reg = /(\d+) years old/g; // 可以把所需部分分离出来  var result = reg.exec(str);  console.log(result); |

/\* 返回的result是：

[

'12 years old',

'12', // 在第二项

index: 5,

input: 'I am 12 years old, and I have 20 apples. My mon is 30 years old. She has 100 oranges.',

groups: undefined

]

\*/

|  |
| --- |
| var reg = /(\d+) years old/g;  var result = reg.exec(str);  console.log(result[1]); // 可以直接取出来 |

/\* 返回的result[1]是：

12

\*/

|  |
| --- |
| var reg = /(\d+) years old/g;  var result;  while (result = reg.exec(str)) // 还可以用while循环来找  {  console.log(result[1]);  } |

/\* 返回的结果是：

12

30

\*/

**调用本地资源**

|  |
| --- |
| const HOST = "127.0.0.1";  const PORT = 3000;  const PHOTO\_PATH = "/photo.jpg";  const HTML\_PATH = '/learn\_html\_2.html'; // 包含一个photo.jpg  var http = require("http");  var fs = require("fs");  var server = http.createServer(function(req, res)  {  if (req.url == '/hello')  {  fs.readFile("."+HTML\_PATH, function(err, data) // 路径要加"."  {  res.end(data); // 可以显示html  })  } |

/\* 因为html包含photo.jpg，所以会调用HOST.PORT/photo.jpg，所以需要加入这个页面

要注意路径的不同：

req.url是"/photo.jpg"，而文件路径是"./photo.jpg"，注意".

如果html里有javascript的文件或者css的文件，也要加到子页面里

\*/

|  |
| --- |
| else if (req.url == '/photo.jpg')  {  fs.readFile("."+PHOTO\_PATH, function(err, data)  {  res.end(data);  });  }  else  {  res.end("Error!");  }  });  server.listen(PORT, HOST);  console.log("Server is running...");  console.log(HOST + ":" + PORT); |

Url模块

|  |
| --- |
| var http = require("http");  var url = require("url");  var server = http.createServer(function(req, res)  {  var urljson = url.parse(req.url);  console.log(urljson);  }); |

输入网址

http://127.0.0.1:3000/1.html?id=123&name=Haibin#abc // #后面的是哈希值，对于后端没有用

显示

Url {

......

search: '?id=123&name=Haibin',

query: 'id=123&name=Haibin',

pathname: '/1.html',

path: '/1.html?id=123&name=Haibin',

href: '/1.html?id=123&name=Haibin'

}

|  |
| --- |
| var urljson = url.parse(req.url);  console.log(urljson.path); // 可以提取内容 |

返回

/1.html?id=123&name=Haibin

可以在parse里面加入true

|  |
| --- |
| var urljson = url.parse(req.url, true); |

则返回

Url {

......

search: '?id=123&name=Haibin',

query: [Object: null prototype] { id: '123', name: 'Haibin' },

pathname: '/1.html',

path: '/1.html?id=123&name=Haibin',

href: '/1.html?id=123&name=Haibin'

}

可以发现query变成一个Object（字典）了

path模块获取扩展名

|  |
| --- |
| var url = require("url");  var path = require("path"); // 导入  var server = http.createServer(function(req, res)  {  var urljson = url.parse(req.url, true);  var pathname = urljson.pathname;  var extname = path.extname(pathname); // 这里  console.log(extname);  res.end("Hello");  }); |

**实践：自动给文件夹里的东西添加路由**

例如文件在nodejs5文件夹中

|  |
| --- |
| const mime = { // 建立一个Header的类型表  '.jpg' : 'image/jpeg', // 可以网上查到  '.jpeg' : 'image/jpeg',  '.png' : 'image/png',  '.html' : 'text/html;charset=URF-8',  '.js' : 'application/x-javascript',  '.css' : 'text/css'  };  var server = http.createServer(function(req, res)  {  var pathname = url.parse(req.url).pathname;  var extname = path.extname(pathname);  if (!extname) // 如果没有类型，就是文件夹  {  pathname = pathname + 'index.html'; // 就进入index.html页面  }  fs.readFile("./nodejs5" + pathname, function(err, data)  {  if (err)  {  res.end("No such file.");  }  if (mime.hasOwnProperty(extname)) // 如果mime里面有对应类型  {  res.setHeader('content-type', mime[extname]); // Header设为这个类型  }  res.end(data); // 显示内容  });  }); |

输入：http://127.0.0.1:3000/photo.jpg

则显示./nodejs5/photo.jpg图片

输入：http://127.0.0.1:3000

则显示/nodejs5/index.html

引用包

假设写了一个模块叫test.js，里面有许多函数，现在要引用它

|  |
| --- |
| require("./test.js") // 引用这个文件  require("test.js") // 引用./node\_modules里的文件，文件夹名字固定！ |

因此可以把.js包都放在node\_modules里，就不用写路径了

node\_modules文件夹可以不在当前文件夹，而在父级文件夹里，也就是说，对于一个大项目，可以把node\_modules放在最外层文件夹里

如果require一个文件夹，则自动引用文件夹下的index.js

npm: Node Package Manager

官方网站：https://www.npmjs.com/

可以去寻找需要的包，下载包的方法：

|  |
| --- |
| $ cd /文件夹  $ npm install 包的名字 |

显然，当别人没有node\_modules的时候，别人就没法运行代码了。因此可以在项目目录（和node\_modules同一级目录）写一个依赖项package.json

|  |
| --- |
| {  "dependencies" :  {  "包名字" : "版本", // 格式  "solarLunar" : "latest" // 举例  }  } |

在package.json这一级目录，直接在命令行运行：

|  |
| --- |
| $ npm install |

即可安装依赖包。

**Express.js**

传统node.js路由不方便，静态资源服务不方便，进而有了express

基本使用

|  |
| --- |
| var express = require('express'); // 调用  var app = express(); // 创建实例  // 下面的都是固定模式的写法  app.get('/', function(req, res)  {  res.send('I am the main page.'); // 这里不是end而是send了  });  app.get('/music', function(req, res)  {  res.send('Music channel.');  });  app.get('/news', function(req, res)  {  res.send('News channel.')  });  app.listen(3000); |

解读：

|  |
| --- |
| app.get('/student/:matrikelnummer', function(req, res) //get对应res.send()  {  res.set('Content-type', 'text/html; charset=UTF-8'); // set  // 尝试获取matrikelnummer的6位整数（查正则表达）  if (/^[\d]{6}$/.test(req.params.matrikelnummer))  {  res.send('Student, matrikelnummer: ' + req.params.matrikelnummer);  }  else  {  res.send("Matrikelnummer not found!");  }  }); |

可以用":"引导变量，便于正则路由，用req.params.变量名得到

Express 静态路由

|  |
| --- |
| app.use(express.static('./nodejs5')); // 指向文件夹即可 |

Express打开文件

|  |
| --- |
| app.get('/page', function(req, res)  {  // sendFile必须要绝对路径  res.sendFile(\_\_dirname + '/learn\_html\_2.html'); // \_\_dirname表示绝对路径  }); |

**Socket.IO**

搭建服务器

|  |
| --- |
| const express = require('express');  const app = express();  const PORT = 3000;  const http = require('http').createServer(); // 用http创建server  const io = require('socket.io')(http); // 引用socketio，还要给出服务器  io.on('connection', function(socket) // 'connection'是一个事件名  // socket是server和client的连接  {  socket.emit('welcome', 'Hello, welcome to the socket.io server!');  // 事件'connection'的CallBack函数  // 'welcome'是发出的事件名  // 'Hello...'是发出的内容  console.log('New client came.')  });  http.listen(PORT, function() // 开始运行server  {  console.log("Server is listening on localhost: " + PORT);  }); |

**注意：所有的IO内容都要写在io.on{ }里面**

**socket.emit( )只会给发过信息来的client回复，例如**

**client1发送了一个"ReadData"给server**

|  |
| --- |
| socket.emit("ReadData", "Hello!"); |

**server收到后：**

|  |
| --- |
| socket.on("ReadData", function (data)  {  socket.emit("Move");  }); |

**这个Move只会被client1接收到，不会被其他clients接收到，如果要传给别的clients，要写**

|  |
| --- |
| socket.on("ReadData", function (data)  {  io.sockets.emit("Move");  }); |

搭建一个client

|  |
| --- |
| const PORT = 3000;  const io = require('socket.io-client'); // 导入socketio的client模块  let socket = io.connect('http://localhost:'+PORT); // 连接server  // 连接server之后，server发出'welcome'的事件  socket.on("welcome", function(data) // 收到'welcome'后，  {  console.log('Received: ' + data); // 回调这一行  }); |

子页面：

|  |
| --- |
| io.of('/games').on('connection', function(socket) // .of就是子页面  {  socket.emit('welcome', 'Hello, welcome to the socket.io server!');  console.log('New client came.')  }); |

对应的client

|  |
| --- |
| let socket = io.connect('http://localhost:' + PORT + '/games'); |

**https的server问题**

必须要用https连接的时候，需要证书和密匙

1. 建立CA(Certificate Authority)证书和钥匙

|  |
| --- |
| $ openssl genrsa -out 名字-ca.key 4096  $openssl req -x509 -new -nodes -key 名字-ca.key -sha256 -days 365 -out 名字ca.crt |

2. 在iPhone上安装证书

a. 把.crt传到手机上

b. 设置 - 通用 - （最下面）描述文件与设备管理 - 安装证书

c. 设置 - 通用 - （最上面）关于本机 - （最下面）证书信任设置 - 信任证书

3. 在Mac上双击证书

a. 选择钥匙串：**登陆**

b. 查看证书：**始终信任**

4. 生成服务器钥匙

|  |
| --- |
| $ openssl genrsa -out 服务器名.key 4096 |

5. 创建配置文件config.cnf，内容：

|  |
| --- |
| [ req ]  prompt = no  default\_bits = 4096  distinguished\_name = req\_distinguished\_name  req\_extensions = req\_ext  [ req\_distinguished\_name ]  countryName = DE  localityName = Karlsruhe  organizationName = KIT  commonName = Haibin  [ req\_ext ]  extendedKeyUsage = serverAuth  subjectAltName = @alt\_names  [alt\_names]  DNS.1 = localhost  IP.1 = 192.168.0.10  IP.2 = 127.0.0.1  IP.3 = 10.12.181.95 |

其中

countryName = DE

localityName = Karlsruhe

organizationName = KIT

commonName = Haibin

比较重要，另外

**[alt\_names]**

**DNS.1 = localhost**

**IP.1 = 192.168.0.10**

**IP.2 = 127.0.0.1**

**IP.3 = 10.12.181.95**

**也非常重要！应当把wifi的ip加进去，方法如下：在命令行里**

|  |
| --- |
| $ ifconfig |

**找到**

**en0下的inet（在我这是上文中的IP.3），加进去**

6. 生成服务器证书

|  |
| --- |
| $ openssl req -new -key 服务器名.key -config config.cnf -out 服务器名.csr |

7.验证CSR

|  |
| --- |
| $ openssl req -in 服务器名.csr -noout -text |

8. 颁发证书

|  |
| --- |
| $ openssl x509 -req -in 服务器名.csr -CA 名字-ca.crt -CAkey 名字-ca.key -CAcreateserial -out 服务器名.crt -days 365 -sha256 -extfile config.cnf -extensions req\_ext |

9.验证证书

|  |
| --- |
| $ openssl x509 -in 服务器名.crt -text -noout |

大功告成！

**SocketIO in Swift**

安装：

Xcode - File - Swift Packages - Add Package Dependency - GitHub

|  |
| --- |
| https://github.com/socketio/socket.io-client-swift |

初始化：

|  |
| --- |
| let socket\_manager = SocketManager(socketURL: URL(string: "https://10.12.181.95:3000/")!) // 自己设置  let socket: SocketIOClient  override init() {  socket = socket\_manager.defaultSocket  socket.on(clientEvent: .connect) {data, ack in  print("socket connected.")  }  socket.connect()  } |

监听信息：

|  |
| --- |
| socket.on("takephoto") {data, act in  self.TakePhoto()  } |

发送信息

|  |
| --- |
| socket.emit("depthData", my\_data, image\_data, instrinsic\_matrix) |

**SocketIO in HTML**

1. 在<head>调用socket

在socket.io官网，Resource下面找到cdn资源，复制下来即可

**必须要是同一个版本的socket**

|  |
| --- |
| <head>  ...  <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.4/socket.io.js" crossorigin="anonymous"></script>  </head> |

2. 在某个地方引用写好的javascript文件

|  |
| --- |
| <body>  <script src="/control.js"></script>  ...  </body> |

3. 在js文件里面写

|  |
| --- |
| var socket = io(); |

就能连接server了

使用方法举例：

|  |
| --- |
| function SendLabel() {  console.log('Hello');  socket.emit(RESET\_LABEL\_AND\_INDEX, label);  } |

**SocketIO in Python**

安装正确的版本：

https://python-socketio.readthedocs.io/en/latest/intro.html

Version Compatibility

例如我的JS Socket是2.0.4，则

|  |
| --- |
| $ pip install "python-socketio[client]"==4.6.0  $ pip install python-engineio==3.14.2 |

在python里：

|  |
| --- |
| import socketio  sio = socketio.Client()  @sio.event()  def connect():  print('connected')  @sio.on('takephoto')  def hehe():  print('take photo!')  sio.connect('http://localhost:3000')  sio.wait() |

**SSH**

Secure Shell（安全外壳协议，简称SSH）是一种加密的网络传输协议，可在不安全的网络中为网络服务提供安全的传输环境。SSH通过在网络中创建安全隧道来实现SSH客户端与服务器之间的连接。SSH最常见的用途是远程登录系统，人们通常利用SSH来传输命令行界面和远程执行命令。

概述

SSH以非对称加密实现身份验证。身份验证有多种途径，例如其中一种方法是使用自动生成的公钥-私钥对来简单地加密网络连接，随后使用密码认证进行登录；另一种方法是人工生成一对公钥和私钥，通过生成的密钥进行认证，这样就可以在不输入密码的情况下登录。任何人都可以自行生成密钥。公钥需要放在待访问的电脑之中，而对应的私钥需要由用户自行保管。认证过程基于生成出来的私钥，但整个认证过程中私钥本身不会传输到网络中。

SSH协议有两个主要版本，分别是SSH-1和SSH-2。无论是哪个版本，核实未知密钥来源都是重要的事情，因为SSH只验证提供用户是否拥有与公钥相匹配的私钥，只要接受公钥而且密钥匹配服务器就会授予许可。这样的话，一旦接受了恶意攻击者的公钥，那么系统也会把攻击者视为合法用户。

密钥管理

在类Unix系统中，已许可登录的公钥通常保存在用户 /home 目录的 ~/.ssh/authorized\_keys 文件中，该文件只由SSH使用。当远程机器持有公钥，而本地持有对应私钥时，登录过程不再需要手动输入密码。另外为了额外的安全性，私钥本身也能用密码保护。

私钥会保存在固定位置，也可以通过命令行参数指定（例如ssh命令的“-i”选项）。ssh-keygen是生成密钥的工具之一。

SSH也支持基于密码的身份验证，此时密钥是自动生成的。若客户端和服务端从未进行过身份验证，SSH未记录服务器端所使用的密钥，那么攻击者可以模仿服务器端请求并获取密码，即中间人攻击。但是密码认证可以禁用，而且SSH客户端在发现新密钥或未知服务器时会向用户发出警告。

**证书，签名，HTTPS**

作者：编程指北

链接：https://www.zhihu.com/question/52493697/answer/1600962734

来源：知乎

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

**1. 什么是加密**

加密就是对明文数据按某种特殊算法进行处理，使其成为不可读的一段代码，通常称为“密文“， 密文通过”密钥“解密后还原出原来的明文，通过这样的途径可以达到保护数据不被非法人窃取、阅读的目的。

定义简单吧？那来看个题，考虑以下哪些属于加密方法：

AES, RSA, SM4

MD5, BASE64

这几种都是日常开发中常用的数据编码技术，但是只有AES、RSA、SM4才能算是加密方法。为什么呢？一个区分的简单方法就是看编码后的数据是否还能还原，能还原的是加密。

MD5实际上是对数据进行有损压缩，无论数据有多长，1KB、1Mb 还是 1G，都会生成固定128位的散列值，并且MD5理论上是不可能对编码后的数据进行还原的，即不可逆。MD5因为其具有不可逆性、单向恒定性（相同的数据多次计算值不变）被广泛应用于文件完整性验证、口令加密以及接下来会讲到的数字签名中。

至于BASE64是否算做加密方法，仁者见仁。在这里不下结论，因为BASE64编码不需要密钥，且编码后的字符串任何人都可以解码出原串，所以一般不认为是加密方法。BASE64常用来做转码，把二进制字节序列转化为ASCII字符序列。

**2. 加密算法的分类**

加密算法按照加解密使用的密钥是否相同，可分为：

对称加密（Symmetric Cryptography）

非对称加密（Asymmetric Cryptography）

2.1 对称加密

对称加密是指加密和解密时使用同一个密钥。

手机屏幕的截图

描述已自动生成

2.2 非对称加密

非对称加密是指加密和解密使用不同的密钥，这两个密钥分别叫做「公钥」、「私钥」。

公钥是可以公开给所有人的，而私钥需要自己保密的。

公钥加密的数据只能用私钥解密：

日程表

描述已自动生成

同理，私钥“加密”的数据只能用公钥“解密”：

手机屏幕的截图

描述已自动生成

大家注意到没，我对私钥“加密”这里打了引号，为什么呢？因为私钥不是用来加密的，准确的说法应该是「私钥签名，公钥验签」。这个问题很多同学都存在误解，认为公私钥都可以用于加密。实际上不是的，至于为什么，后面讲完签名我会解释的。

**3. 故事开始**

为了讲这个故事，小北请来了密码学中常用的学术情侣，Alice和Bob，以及窃听者代表Eve。

我们从Alice、Bob约会的故事展开，来讲讲其中暗藏着哪些危机，又是如何一步步化解的。

3.1 第一回合

九月，一个夜黑风高的晚上，Bob想约Alice出来玩，于是给Alice发了一封邮件：

手机屏幕截图

中度可信度描述已自动生成

但我们都知道网络是不可信的，并且由于消息在网络中是明文传输的，所以黑客可以轻易的截获、篡改甚至冒充Bob。来，我们看看黑客Eve是怎么干的：

图示

描述已自动生成

瞧，Eve轻易的拿到了邮件内容（窃听），并且修改了邮件内容（篡改），甚至说他可以随时冒充Bob给Alice发送邮件（伪装）。如果上图中Eve伪造的内容被Alice接收到了，那么后果可想而知。现实世界中，我们每天都在通过网络进行聊天、转账、浏览不存在网站。如果都是这样明文传输数据，显然毫无安全感。

3.2 第二回合

既然我们不能明文传输，那么Bob和Alice提前商量好密钥，使用对称加密对邮件内容加密不就好了~

图示

描述已自动生成

现在Bob发送的邮件都使用和Alice提前商量好的密钥加密后再传输。由于没有密钥，Eve 就算截获到数据也无法获取邮件的内容，也没法篡改和冒充Bob。因为篡改后的数据必须使用密钥再次加密Alice才能正确解密。那么只要Bob和Alice能够保证密钥不泄露，整个通信就是安全的。

如果密钥泄露，被中间人截获，那么就等同于明文通信。所以我们不能把安全性寄托在人上面。并且这里也存在一个问题，如果两个人不能线下见面，如何在网上安全的交换密钥呢？

这似乎是无解的，因为交换密钥的时候我们必须明文通信，不然对方根本看不懂。但是明文交换即意味着可能泄露。

但是别忘了我们的密码学工具箱里还有一个好东西—「非对称加密」。

Bob和Alice各自生成一对公私钥，因为公钥本来就是公开的，即可以被任何人获取，所以可以通过网络明文交换公钥。然后使用公钥加密邮件内容后发送给对方，接收者使用自己的私钥即可解密。

3.3 第三回合

来看看，在非对称加密体系下，Bob如何给Alice发消息的。

首先Alice需要先生成一对公私钥，私钥只能Alice自己知道，公钥是可以让任何人都知道的，因此可将公钥直接发送给Bob，就算被截获也无所谓。

图示

描述已自动生成

Bob使用Alice的公钥加密邮件内容，加密后的内容只能由Alice的私钥解密，所以就算Eve截获也是徒劳。

反之，如果Alice想给Bob回信，就需要用Bob的公钥加密后发送。

这就解决了密钥交换问题，也保证了邮件内容不会泄露。也就是说现在可以防窃听。

3.4 如何证明Bob是Bob

不知道你注意到没有，这里也存在另外一个问题：

Eve也可以使用Alice的公钥冒充Bob给Alice发邮件，因为Alice的公钥本来就是公开的，任何人都可以获得。由于Eve也可以获得Alice公钥，所以没法防止Eve伪造和篡改，并且对于Alice而言，她无法分辨出邮件到底是Eve发的还是Bob。所以这个问题的本质就是「Alice如何确认邮件来自于Bob」。

那么在生活中，我们如何做这件事呢？那就是让Bob在纸上签名并且按手印，因为指纹和字迹是Bob独有的，其它人很难伪造。所以我们需要在计算机中引入类似的机制：

即只有Bob自己能够产生的独一无二的标志，并且其它人能够验证这个标志确实是属于Bob的。这就是我们今天要讲的主题—「数字签名」。

还记得什么是Bob独有的吗？

对，就是Bob自己的私钥，Bob用自己的私钥对邮件内容计算一个「签名」，将「签名」和邮件内容一起发送出去，接受者Alice可以使用Bob的公钥验证这个签名是否正确，这就叫「验签」。

如果不是Bob的私钥计算的签名，那么Alice用Bob公钥验签将会出错。

图示

描述已自动生成

可以看到，Eve试图使用自己的私钥计算签名然后发送给Alice，但是Alice使用Bob的公钥进行验签时将会出错！

那么Eve可能篡改内容并冒充Bob的签名吗？不可能！因为内容发生改变时，对应的签名也需要重新计算，而签名的生成依赖于私钥，只要Bob的私钥不泄露，签名就不会被冒充。所以使用数字签名，我们能够鉴别消息的发送者，也就是说黑客无法伪装发送者进行发送数据，也无法篡改。

注意：

可以看出我们这里数据是明文传输的，存在窃听风险。但是我们为了阐述数字签名机制是如何运转的，故意将保证信息机密性的机制省略了。

如果想要保证数据的机密性，我们常见的做法是，通信双方通过非对称加密安全交换对称加密的密钥，后续通信过程的数据都使用对称加密保证数据机密性。

并且「签名」的作用本身也不是用来保证数据的机密性，而是用于验证数据来源的防止数据被篡改的，也就是确认发送者的身份。

一般而言，我们不会直接对数据本身直接计算数字签名，为什么呢？

因为数字签名属于非对称加密，非对称加密依赖于复杂的数学运算，包括大数乘法、大数模等等，耗时比较久。

如果数据量大的时候计算数字签名将会比较耗时，所以一般做法是先将原数据进行Hash运算，得到的Hash值就叫做「摘要」。

「摘要」就像人的指纹一样，可以代表一个人，只要内容发生了改变，计算出来的摘要也应该变化。「摘要」最好是不可逆转的，一般使用开头提到的MD5作为Hash函数，MD5输出的结果固定位128位。

为什么「摘要」最好是不可逆转的？

因为既然Alice可以用Bob公钥解开签名，那么理论上其它人，比如Eve也可以使用Bob公钥解开签名拿到数据。

所以我们最好对数据的「摘要」进行签名，这样，Eve就算解开签名，拿到的也是「摘要」，如果摘要是不可逆转的，也就是无法从摘要反推出原文，也就达到了保密的作用。

发送者使用私钥对「摘要」计算数字签名。那么接收者如何验证呢？

图示

描述已自动生成

接受者Alice收到后，取下数字签名，同时用Bob的公钥解密，得到「摘要1」，证明确实是Bob发的。

(画外音：如果使用Bob的公钥验证签名出错，那么签名一定不是Bob的私钥生成的）

再对邮件内容使用相同的散列函数计算「摘要2」，与上面得到的「摘要1」进行对比，两者一致就说明信息未被篡改。

这样两步分证明发送者身份和保证数据未被篡改。

3.5 这就够了吗？

Bob和Alice现在可以依赖于对称加密进行保密通信，也可以依赖于数字签名验证消息是否是对方发送的。但是这一切的根基是建立在Alice持有的公钥确实是Bob的，反之亦然。

什么意思呢？

试想，Eve如果将自己的公钥冒充Bob发送给Alice，然后Alice保存了下来，那以后凡是Bob发送的消息，反而会验证签名失败，被当做冒充者。那你可能会问，为什么Eve可以将自己的公钥发送给Alice，而Alice毫不知情呢？

图示

描述已自动生成

看！我们又回到了最初的起点，只不过这次被篡改的是公钥，之前是消息本身。因为Bob的公钥是直接通过网络发送给Alice的，所以Eve才可以在这一步做手脚，进行篡改，将自己的公钥冒充Bob发送给Alice，也就是发送公钥这一步没有做到：防篡改，防冒充

防篡改怎么和防冒充怎么实现的呢？

我们前面讲了，就是靠数字签名！但是数字签名需要接受者持有发送者公钥，才能进行验签。而我们现在处理的是分发公钥这一步。这像是先有鸡还是先有蛋的问题，现在的问题就是「Bob无法证明它自己是Bob」。

这个是不是似曾相识，以前去办事的时候经常被要求出具「我妈是我妈」这类证明。但是我们自己说“我妈就是我妈”，人家根本不会信呀，需要一个可信第三方出具证明，比如派出所。那么「Alice如何才能确认Bob发送给自己的公钥确实是Bob的，而没有被篡改？」在只有Alice和Bob两人的情况下是没法验证的。所以，我们这里也需要一个第三方帮Bob证明「Bob的公钥就是Bob的公钥」

3.6 数字证书

为了解决这个问题，就引入了「数字证书」，什么叫数字证书呢？

数字证书是指在互联网通讯中标志通讯各方身份信息的一个数字认证，人们可以在网上用它来识别对方的身份。因此数字证书又称为数字标识。数字证书对网络用户在交流中的信息和数据等以加密或解密的形式保证了信息和数据的完整性和安全性。

只要你理解了前面的数字签名，就能理解这里的数字证书，因为我把数字证书叫做「公钥的数字签名」。为什么呢？我们引入数字证书的目的是为了保证公钥不被篡改，即使被篡改了也能识别出来。而防篡改的方法就是数字签名，但是这个签名不能我们自己做，原因说过了，因为我们的公钥还没分发出去，别人无法验证。

所以只能找可信的第三方来帮我们签名，即证书颁布机构（CA），CA会将：证书的颁布机构、有效期、公钥、持有者(subject)等信息用CA的私钥进行签名。

并且将签名结果和这些信息放在一起，这就叫做「数字证书」。

这样，Bob就可以去CA申请一个证书，然后将自己的证书发给Alice，那么Alice如何验证这个证书确实是Bob的呢？

当然是使用CA的公钥进行验签。

注意：

CA的公钥也是需要使用证书来分发的，所以Alice的电脑必须安装CA的证书，证书里包含了CA的公钥。

收到Bob发过来的数字证书后，Alice使用CA的公钥进行验证，验证通过即证明这确实是Bob证书，也就可以使用证书中包含的Bob的公钥，按照之前讨论的流程进行通信。

那么Eve是否可以在中途篡改Bob的证书呢？

答案是不行，因为证书的信息使用CA的私钥进行签名，只要Eve修改了任何一个Bit都会导致最后签名验证不通过。

那Eve可不可以修改证书信息后自己重新计算一次证书的数字签名呢？

也不行，因为证书的数字签名计算依赖于CA的私钥，Eve是拿不到CA的私钥的。如果拿到了，说明什么？整个世界都是不可信的。

3.7 数字证书长啥样

这是我电脑中的自带的证书：

图形用户界面, 文本, 应用程序

描述已自动生成

可以看到，包含了证书持有人的公钥和证书的签名。另外，证书颁发机构是有层级关系的，下级CA的证书是需要由上级CA签名的。换句话说一定存在根证书颁发机构，那么他们的证书是由谁签名的呢？答案是自签，自己给自己认证。这是我电脑中的一个自签的根证书颁发机构：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

为什么根证书可以自签，谁来保证安全？你把钱存在银行，你会担心吗？我们基于对国家的信任，才信任银行，这就是信任链的基础！我们思考问题应该是分层的，如果不认可一个统一的基础，一直套娃下去，那么问题就无解。那还有个问题，如何保证根证书的可靠性？这是操作系统和浏览器预装的，由微软、苹果等操作系统厂商来选择根证书。

3.8 证书不可信？

那么什么情况下浏览器会提示“证书不可信”呢？根据我们上面的分析，下面是可能的原因：

证书不是权威CA颁发。有些企业为了贪图便宜使用盗版的证书，没有经过CA认证。也就是无法使用浏览器内置CA公钥进行验证。

证书过期

上面说了，证书里有一项就是有效期，一般就是一年或者两年的时间。如果证书过期，那么浏览器就会提示“证书不可信”

证书部署错误

可能是服务器证书部署出错，比如证书与域名不匹配，因为证书里有一项是持有人信息的。

好了，饶了一大圈，Bob 终于可以安全的向Alice发出前往红树林的邀请了

附录

非对称加密中公私钥都可以加密，那么什么时候用公钥加密，什么时候用私钥“加密”？

加密场景，那么肯定希望只有我才能解密，别人只能加密。即公钥加密，私钥解密。

签名场景，既然是签名，就希望只能我才能签名，别人只能验证。即私钥签名，公钥验签

什么是数字签名，数字签名的作用是什么？

数字签名就是使用私钥对数据摘要进行签名，并附带和数据一起发送。可以起到防篡改、防伪装、防否认的作用。

为什么要对数据的摘要进行签名，而不是直接计算原始数据的数字签名？

数据可能比较大，签名是使用非对称加密算法，比较耗时防止第三方使用公钥解开签名后，拿到原始数据

什么是数字证书，数字证书存在解决了什么问题？

数字证书就是由CA机构使用自己私钥，对证书申请者的公钥进行签名认证。数字证书解决了如何安全分发公钥的问题，也奠定了信任链的基础。