一步步搭建物联网系统(教你设计物联网系统)

源自毕业论文:基于REST服务的最小物联网系统设计

1 前言

设计物联网系统是件有意思的事情，它需要考虑到软件、硬件、通讯等多个不同方面。通过探索不同的语言，不同的框架，从而形成不同的解决方案。

在这里，我们将对设计物联网系统有一个简单的介绍，并探讨如何设计一个最小的物联网系统。

1.1 目标读者

目标读者: 初入物联网领域，希望对物联网系统有一个大概的认识和把握，并学会掌握一个基础的物联网系统的设计。

- 硬件开发人员，对物联网有兴趣。
- 没有web开发经验
- 几乎为0的linux使用经验
- 想快速用于生产环境
- 对硬件了解有限的开发人员。
- 没接触过51、ARM、Arduino
- 想了解以下内容：
- RESTful与IOT
- CoAP协议
- MQTT

本文档对一些概念(如)只做了一些基本介绍，以方便读者理解。如果您想进一步了解这些知识，会列出一些推荐书目，以供参考。

1.2 不适合人群

- 如果你是在这方面已经有了丰富经验的开发者。

- 当前状态 建议
- 不是为了学习而学习这方面的知识。

1.3 介绍

关于内容的选择，这是一个有意思的话题，因为我们很难判断不同的开发者用的是怎样的语言，用的是怎样的框架。

于是我们便自作主张地选择了那些适合于理论学习的语言、框架、硬件，去除掉其他一些我们不需要考虑的因素，如语法，复杂度等等。当然，这些语言、框架、硬件也是最流行的。

- Arduino: 如果你从头开始学过硬件的话，那么你会爱上它的。
- Raspberry PI: 如果你从头编译过GNU/Linux的话，我想你会爱上她的。
- Python: 简单地来说，你可以方便地使用一些扩展，同时代码就表达了你的想法。
- PHP : 这是一门容易部署的语言，我想你只需要在你的Ubuntu机器上，执行一下脚本就能完成安装了。而且，如果你是一个硬件开发者的话，你会更容易地找到其他开发者。
- Javascript : 考虑到javascript这门语言已经无处不在了，而且会更加流行。所以，在这里CoAP、MQTT等版本是基于Nodejs的。
- HTML、CSS : 这是必须的，同样，他们仍然无处不在。
- GNU/Linux: 作为部署到服务器的一部分——你需要掌握他。当然如果你要用WAMP也是可以的。
- CoAP: 用NodeJS构建IOT CoAP物联网

1.3.1 为什么没有C？

`C都不懂还跑过来干嘛`。

1.3.2 为什么不是JAVA？

大有以下两个原因

- 学习JAVA的人很多，但是它不适合我们集中精力构建与学习，因为无关的代码太多了。
- 之前以及现在，我还是不喜欢JAVA (我更喜欢脚本语言，可以提高工作效率)。

1.4 如何阅读

这只是一个小小的建议，仅针对于在选择阅读上没有经验的读者。

当前状态	建议
软件初学者	从头阅读
硬件开发者	从头阅读
没有web经验的开发者	从第二部分开始

我们会在前面十章里简单介绍一些必要的基础知识，这些知识将会在后面我们构建物联网系统时用到。

某一天，正走在回学校的路上的我突然想到：“未来将会是一个科技的时代——虽然现在也是——只是在未来，科技将会无处不在。如果我们依旧对周围这些无处不在的代码一无所知的话，或许我们会成为黑客帝国之中被控制的普通人。”于是开始想着，有一天人们会像学习一门语言一样开始学习编程，直到又有一天我看到了学习编程如同学习一门语言的说法。这又恰好在我做完最小物联网系统之后，算是一个有趣的时间点，我开始想着像之前做最小物联网系统的那些步骤一样，写一个简单的入门。也可以补充好之前在这个最小物联网系统缺失的那些东西，给那些正在开始试图去解决编程问题的人。

让我们先从身边的语言下手，也就是现在无处不在的html+javascript+css。

2 无处不在的HTML

之所以从html开始，是因为我们不需要配置一个复杂的开发环境，也许你还不知道开发环境是什么东西，不过这也没关系，毕竟这些知识需要慢慢的接触才能有所了解，尤其是对于普通的业余爱好者来说，当然，对于专业选手言自然不是问题。HTML是Web的核心语言，也算是比较基础的语言。

2.1 html的hello,world

Hello,world是一个传统，所以在这里也遵循这个有趣的传统，我们所要做的事情其实很简单，虽然也有一点点hack的感觉。——让我们先来新建一个文并命名为“helloworld.html”。

(PS:大部分人应该都是在windows环境下工作的，所以你需要新建一个文本，然后重命名，或者你需要一个编辑器，在这里我们推荐用**sublime text**。破解不破解，注册不注册都不会对你的使用有太多的影响。)

1. 新建文件
2. 输入
   `hello,world`
3. 保存为->“helloworld.html”，
4. 双击打开这个文件。 正常情况下都应该是用你的默认浏览器打开。只要是一个正常工作的现代浏览器，都应该可以看到上面显示的是“Hello,world”。

这才是最短的hello,world程序，但是呢？在ruby中会是这样子的

```
2.0.0-p353 :001 > p "hello,world"
"hello,world"
    => "hello,world"
2.0.0-p353 :002 >
```

等等，如果你了解过html的话，会觉得这一点都不符合语法规则，但是他工作了，没有什么比安装完Nginx后看到It works!更让人激动了。

遗憾的是，它可能无法在所有的浏览器上工作，所以我们需要去调试其中的bug。

2.1.1 调试hello,world

我们会发现我们的代码在浏览器中变成了下面的代码，如果你和我一样用的是chrome，那么你可以右键浏览器中的空白区域，点击审查元素，就会看到下面的代码。

```
<html>
    <head></head>
    <body>hello,world</body>
</html>
```

这个才是真正能在大部分浏览器上工作的代码，所以复制它到编辑器里吧。

2.1.2 说说hello,world

我很不喜欢其中的<*></*>，但是我也没有找到别的方法来代替它们，所以这是一个设计得当的语言。甚至大部分人都说这算不上是一门真正的语言，不过html的原义是

超文本标记语言

所以我们可以发现其中的关键词是标记——markup，也就是说html是一个markup，head是一个markup，body也是一个markup。

然而，我们真正工作的代码是在body里面，至于为什么是在这里面，这个问题就太复杂了。打个比方来说：

1. 我们所使用的汉语是人类用智慧创造的，我们所正在学的这门语言同样也是人类创造的。
2. 我们在自己的语言里遵循着桌子是桌子，凳子是凳子的原则，很少有人会问为什么。

2.1.3 想用中文？

所以我们也可以把计算机语言与现实世界里用于交流沟通的语言划上一个等号。而我们所要学习的语言，并不是我们最熟悉的汉语语言，所以我们便觉得这些很复杂，但是如果我们试着用汉语替换掉上面的代码的话

```
<语言>
    <头><结束头>
    <身体>你好，世界<结束身体>
<结束语言>
```

这看上去很奇怪，只是因为是直译过去的原因，也许你会觉得这样会好理解一点，但是输入上可就一点儿也不方便，因为这键盘本身就不适合我们去输入汉字，同时也意味着可能你输入的会有问题。

让我们把上面的代码代替掉原来的代码然后保存，打开浏览器会看到下面的结果

```
<语言> <头><结束头> <身体>你好，世界<结束身体> <结束语言>
```

更不幸的结果可能是

```
<璇□▉> <澶�><缁撴潫澶�> <韬□綊>浣犲ソ锛屼笘鐣�<缁撴潫韬□綊> <缁撴潫璇□▉>
```

这是一个编码问题，对中文支持不友好。

我们把上面的代码改为和标记语言一样的结构

```
<语言>
    <头></头>
    <身体>你好，世界</身体>
<结束语言>
```

于是我们看到的结果便是

```
<语言> <头> <身体>你好，世界
```

被chrome浏览器解析成什么样了？

```
<html><head></head><body><语言>
        <头><!--头-->
        <身体>你好，世界<!--身体-->
    <!--语言-->
</body></html>
```

以

结尾的是注释，写给人看的代码，不是给机器看的，所以机器不会去理解这些代码。

但是当我们把代码改成

```
<whatwewanttosay>你好世界</whatwewanttosay>
```

浏览器上面显示的内容就变成了

```
你好世界
```

或许你会觉得很神奇，但是这一点儿也不神奇，虽然我们的中文语法也遵循着标记语言的标准，但是我们的浏览器不支持中文标记。

结论:

1. 浏览器对中文支持不友好。
2. 浏览器对英文支持友好。

刚开始的时候不要对中文编程有太多的想法，这是很不现实的:

1. 现有的系统都是基于英语语言环境构建的，对中文支持不是很友好。
2. 中文输入的速度在某种程度上来说没有英语快。

我们离开话题已经很远了，但是这里说的都是针对于那些不满于英语的人来说的，只有当我们可以从头构建一个中文系统的时候才是可行的，而这些就要将cpu、软件、硬件都包含在内，甚至我们还需要考虑重新设计cpu的结构，在某种程度上来说会有些不现实。或许，需要一代又一代人的努力。忘记那些吧，师夷长之技以治夷。

2.2 其他html标记

添加一个标题，

```
<html>
    <head>
        <title>标题</title>
    </head>
    <body>hello,world</body>
</html>
```

我们便可以在浏览器的最上方看到“标题”二字，就像我们常用的淘宝网，也包含了上面的东西，只是还包括了更多的东西，所以你也可以看懂那些我们可以看到的淘宝的标题。

```
<html>
<head>
    <title>标题</title>
</head>
<body>
hello,world
<h1>大标题</h1>
<h2>次标题</h2>
<h3>...</h3>
<ul>
    <li>列表1</li>
    <li>列表2</li>
</ul>
```

```
</body>
</html>
```

更多的东西可以在一些书籍上看到，这边所要说的只是一次简单的语言入门，其他的东西都和这些类似。

2.2.1 美妙之处

我们简单地上手了一门不算是语言的语言，浏览器简化了这其中的大部分过程，虽然没有C和其他语言来得有专业感，但是我们试着去开始写代码了。我们可能在未来的某一篇中可能会看到类似的语言，诸如python，我们所要做的就是

```
$ python file.py
=>hello,world
```

然后在终端上返回结果。只是因为在我看来学会html是有意义的，简单的上手，然后再慢慢地深入，如果一开始我们就去理解指针，开始去理解类。我们甚至还知道程序是怎么编译运行的时候，在这个过程中又发生了什么。虽然现在我们也没能理解这其中发生了什么，但是至少展示了

1. 中文编程语言在当前意义不大，不现实，效率不高兼容性差
2. 语言的语法是固定的。（ps:虽然我们也可以进行扩充，我们将会在后来支持上述的中文标记。）
3. 已经开始写代码，而不是还在配置开发环境。
4. 随身的工具才是最好的，最常用的code也才是实在的。

2.2.2 更多

我们还没有试着去解决“某商店里的糖一颗5块钱，小明买了3颗糖，小明一共花了多少钱”的问题。也就是说我们学会的是一个还不能解决实际问题的语言，于是我们还需要学点东西，比如javascript,css。我们可以将Javascript理解为解决问题的语言，html则是前端显示，css是配置文件，这样的话，我们会在那之后学会成为一个近乎专业的程序员。我们刚刚学习了一下怎么在前端显示那些代码的行为，于是我们还需要Javascript。

3 无处不在的Javascript

Javascript现在已经无处不在了，也许你正打开的某个网站，他便可能是node.js+json+javascript+mustache.js完成的，虽然你还没理解上面那些是什么，也正是因为你不理解才需要去学习更多的东西。但是你只要知道Javascript已经无处不在了，它可能就在你手机上的某个app里，就在你浏览的网页里，就运行在你IDE中的某个进程里。

3.1 Javascript的Hello,world

这里我们还需要有一个helloworld.html，Javascript是专为网页交互而设计的脚本语言，所以我们一点点来开始这部分的旅途，先写一个符合标准的helloworld.html

```
<!DOCTYPE html>
<html>
    <head></head>
    <body></body>
</html>
```

然后开始融入我们的javascript，向HTML中插入Javascript的方法，就需要用到html中的<script>标签，我们先用页面嵌入的方法来写helloworld。

```
<!DOCTYPE html>
<html>
    <head>
        <script>
            document.write('hello,world');
        </script>
    </head>
    <body></body>
</html>
```

按照标准的写法，我们还需要声明这个脚本的类型

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            document.write('hello,world');
        </script>
    </head>
    <body></body>
</html>
```

没有显示hello,world?试试下面的代码

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <script type="text/javascript">
            document.write('hello,world');
        </script>
    </head>
    <body>
        <noscript>
            disable Javascript
        </noscript>
    </body>
</html>
```

3.2 更js一点

我们需要让我们的代码看上去更像是js，同时是以js结尾。就像C语言的源码是以C结尾的，我们也同样需要让我们的代码看上去更正式一点。于是我们需要在helloworld.html的同一文件夹下创建一个app.js文件，在里面写着

```
document.write('hello,world');
```

同时我们的helloworld.html还需要告诉我们的浏览器js代码在哪里

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript" src="app.js"></script>
    </head>
    <body>
        <noscript>
            disable Javascript
        </noscript>
    </body>
</html>
```

3.2.1 从数学出发

让我们回到第一章讲述的小明的问题，从实际问题下手编程，更容易学会编程。小学时代的数学题最喜欢这样子了——某商店里的糖一个5块钱，小明买了3个糖，小明一共花了多少钱。在编程方面，也许我们还算是小学生。最直接的方法就是直接计算3x5=?

```
document.write(3*5);
```

document.write实际也我们可以理解为输出，也就是往页面里写入3*5的结果，在有双引号的情况下会输出字符串。我们便会在浏览器上看到15，这便是一个好的开始，也是一个糟糕的开始。

3.3 设计和编程

对于实际问题，如果我们只是止于所要得到的结果，很多年之后，我们就成为了code monkey。对这个问题进行再一次设计，所谓的设计有些时候会把简单的问题复杂化，有些时候会使以后的扩展更加简单。这一天因为这家商店的糖价格太高了，于是店长将价格降为了4块钱。

```
document.write(3*4);
```

于是我们又得到了我们的结果，但是下次我们看到这些代码的时候没有分清楚哪个是糖的数量，哪个是价格，于是我们重新设计了程序

```
tang=4;
num=3;
document.write(tang*num);
```

这才能叫得上是程序设计，或许你注意到了";"这个符号的存在，我想说的是这是另外一个标准，我们不得不去遵守，也不得不去fuck。

3.3.1 函数

记得刚开始学三角函数的时候，我们会写

```
sin 30=0.5
```

而我们的函数也是类似于此，换句话说，因为很多搞计算机的先驱都学好了数学，都把数学世界的规律带到了计算机世界，所以我们的函数也是类似于此，让我们做一个简单的开始。

```
function hello(){
    return document.write("hello,world");
}
hello();
```

当我第一次看到函数的时候，有些小激动终于出现了。我们写了一个叫hello的函数，它返回了往页面中写入hello,world的方法，然后我们调用了hello这个函数，于是页面上有了hello,world。

```
function sin(degree){
    return document.write(Math.sin(degree));
```

```
}
sin(30);
```

在这里degree就称之为变量。 于是输出了-0.9880316240928602，而不是0.5，因为这里用的是弧度制，而不是角度制。

`sin(30)`

的输出结果有点类似于sin 30。写括号的目的在于，括号是为了方便解析，这个在不同的语言中可能是不一样的，比如在ruby中我们可以直接用类似于数学中的表达:

```
 2.0.0-p353 :004 > Math.sin 30
=> -0.9880316240928618
2.0.0-p353 :005 >
```

我们可以在函数中传入多个变量，于是我们再回到小明的问题，就会这样去编写代码。

```
function calc(tang,num){
    result=tang*num;
    document.write(result);
}
calc(3,4);
```

但是从某种程度上来说，我们的calc做了计算的事又做了输出的事，总的来说设计上有些不好。

3.3.2 重新设计

我们将输出的工作移到函数的外面，

```
function calc(tang,num){
    return tang*num;
}
document.write(calc(3,4));
```

接着我们用一种更有意思的方法来写这个问题的解决方案

```
function calc(tang,num){
    return tang*num;
}
function printResult(tang,num){
    document.write(calc(tang,num));
}
printResult(3, 4)
```

看上去更专业了一点点，如果我们只需要计算的时候我们只需要调用calc，如果我们需要输出的时候我们就调用printResult的方法。

3.3.3 object和函数

我们还没有说清楚之前我们遇到过的document.write以及Math.sin的语法为什么看上去很奇怪，所以让我们看看他们到底是什么，修改app.js为以下内容

```
document.write(typeof document);
document.write(typeof Math);
```

typeof document会返回document的数据类型，就会发现输出的结果是

`object object`

所以我们需要去弄清楚什么是object。对象的定义是

> 无序属性的集合，其属性可以包含基本值、对象或者函数。

创建一个object，然后观察这便是我们接下来要做的

```
store={};
store.tang=4;
store.num=3;
document.write(store.tang*store.num);
```

我们就有了和document.write一样的用法，这也是对象的美妙之处，只是这里的对象只是包含着基本值，因为

`typeof story.tang="number"`

一个包含对象的对象应该是这样子的。

```
store={};
store.tang=4;
store.num=3;
document.writeln(store.tang*store.num);

var wall=new Object();
wall.store=store;
document.write(typeof wall.store);
```

而我们用到的document.write和上面用到的document.writeln都是属于这个无序属性集合中的函数。

下面代码说的就是这个无序属性集中中的函数。

```
var IO=new Object();
function print(result){
    document.write(result);
};
IO.print=print;
IO.print("a obejct with function");
IO.print(typeof IO.print);
```

我们定义了一个叫IO的对象，声明对象可以用

```
var store={};
```

又或者是

```
var store=new Object{};
```

两者是等价的，但是用后者的可读性会更好一点，我们定义了一个叫print的函数，他的作用也就是document.write，IO中的print函数是等价于print()函数，这也就是对象和函数之间的一些区别，对象可以包含函数，对象是无序属性的集合，其属性可以包含基本值、对象或者函数。

复杂一点的对象应该是下面这样的一种情况。

```
var Person={name:"phodal",weight:50,height:166};
function dream(){
    future;
};
Person.future=dream;
document.write(typeof Person);
document.write(Person.future);
```

而这些会在我们未来的实际编程过程中用得更多。

3.3.4 面向对象

开始之前先让我们简化上面的代码，

```
Person.future=function dream(){
    future;
}
```

看上去比上面的简单多了，不过我们还可以简化为下面的代码。。。

```
var Person=function(){
    this.name="phodal";
    this.weight=50;
    this.height=166;
    this.future=function dream(){
        return "future";
    };
};
var person=new Person();
document.write(person.name+"<br>");
document.write(typeof person+"<br>");
document.write(typeof person.future+"<br>");
document.write(person.future()+"<br>");
```

只是在这个时候Person是一个函数，但是我们声明的person却变成了一个对象一个**Javascript**函数也是一个对象，并且，所有的对象从技术上讲也只不过是函数。这里的“
”是HTML中的元素，称之为DOM，在这里起的是换行的作用，我们会在稍后介绍它，这里我们先关心下this。this关键字表示函数的所有者或作用域，也就是这里的Person。

上面的方法显得有点不可取，换句话说和一开始的

```
document.write(3*4);
```

一样，不具有灵活性，因此在我们完成功能之后，我们需要对其进行优化，这就是程序设计的真谛——解决完实际问题后，我们需要开始真正的设计，而不是解决问题时的编程。

```
var Person=function(name,weight,height){
    this.name=name;
    this.weight=weight;
    this.height=height;
    this.future=function(){
        return "future";
    };
};
var phodal=new Person("phodal",50,166);
document.write(phodal.name+"<br>");
document.write(phodal.weight+"<br>");
document.write(phodal.height+"<br>");
document.write(phodal.future()+"<br>");
```

于是，产生了这样一个可重用的Javascript对象,this关键字确立了属性的所有者。

3.4 其他

Javascript还有一个很强大的特性，也就是原型继承，不过这里我们先不考虑这些部分，用尽量少的代码及关键字来实际我们所要表达的核心功能，这才是这里的核心，其他的东西我们可以从其他书本上学到。

所谓的继承，

```
var Chinese=function(){
    this.country="China";
}

var Person=function(name,weight,height){
    this.name=name;
    this.weight=weight;
    this.height=height;
    this.futrue=function(){
        return "future";
    }
}
Chinese.prototype=new Person();

var phodal=new Chinese("phodal",50,166);
document.write(phodal.country);
```

完整的Javascript应该由下列三个部分组成:

- 核心(ECMAScript)——核心语言功能
- 文档对象模型(DOM)——访问和操作网页内容的方法和接口
- 浏览器对象模型(BOM)——与浏览器交互的方法和接口

我们在上面讲的都是ECMAScript，也就是语法相关的，但是JS真正强大的，或者说我们最需要的可能就是对DOM的操作，这也就是为什么jQuery等库可以流行的原因之一，而核心语言功能才是真正在哪里都适用的，至于BOM，真正用到的机会很少，因为没有完善的统一的标准。

一个简单的DOM示例,

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <noscript>
        disable Javascript
    </noscript>
    <p id="para" style="color:red">Red</p>
</body>
    <script type="text/javascript" src="app.js"></script>
</html>
```

我们需要修改一下helloworld.html添加

```
<p id="para" style="color:red">Red</p>
```

同时还需要将script标签移到body下面，如果没有意外的话我们会看到页面上用红色的字体显示Red，修改app.js。

```
var para=document.getElementById("para");
para.style.color="blue";
```

接着，字体就变成了蓝色，有了DOM我们就可以对页面进行操作，可以说我们看到的绝大部分的页面效果都是通过DOM操作实现的。

3.5 美妙之处

这里说到的Javascript仅仅只是其中的一小小部分，忽略掉的东西很多，只关心的是如何去设计一个实用的app，作为一门编程语言，他还有其他强大的内制函数，要学好需要一本有价值的参考书。这里提到的只是其中的不到20%的东西，其他的80%或者更多会在你解决问题的时候出现。

- 我们可以创建一个对象或者函数，它可以包含基本值、对象或者函数。
- 我们可以用Javascript修改页面的属性，虽然只是简单的示例。
- 我们还可以去解决实际的编程问题。

4 无处不在的CSS

或许你觉得CSS一点儿也不重要，而事实上，如果说HTML是建筑的框架，CSS就是房子的装修。那么Javascript呢，我听到的最有趣的说法是小三——还是先让我们回到代码上来吧。

4.1 CSS

下面就是我们之前说到的代码，css将Red三个字母变成了红色。

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <p id="para" style="color:red">Red</p>
</body>
    <script type="text/javascript" src="app.js"></script>
</html>
```

只是，

```
var para=document.getElementById("para");
para.style.color="blue";
```

将字体变成了蓝色，CSS+HTML让页面有序的工作着，但是Javascript却打乱了这些秩序，有着唯恐世界不乱的精彩，也难怪被冠以小三之名了——或许终于可以理解，为什么以前人们对于Javascript没有好感了——不过这里要讲的是正室，也就是CSS，这时还没有Javascript。

Red Fonts

4.2 关于CSS

这不是一篇专业讲述CSS的书籍，所以我不会去说CSS是怎么来的，有些东西我们既然可以很容易从其他地方知道，也就不需要花太多时间去重复。诸如重构等这些的目的之一也在于去除重复的代码，不过有些重复是不可少的，也是有必要的，而通常这些东西可能是由其他地方复制过来的。

到目前为止我们没有依赖于任何特殊的硬件或者是软件，对于我们来说我们最基本的需求就是一台电脑，或者可以是你的平板电脑，当然也可以是你的智能手机，因为他们都有个浏览器，而这些都是能用的，对于我们的CSS来说也不会有例外的。

CSS(Cascading Style Sheets)，到今天我也没有记得他的全称，CSS还有一个中文名字是层叠式样式表，事实上翻译成什么可能并不是我们关心的内容，我们需要关心的是他能做些什么。作为三剑客之一，它的主要目的在于可以让我们方便灵活地去控制Web页面的外观表现。我们可以用它做出像淘宝一样复杂的界面，也可以像我们的书本一样简单，不过如果要和我们书本一样简单的话，可能不需要用到CSS。HTML一开始就是依照报纸的格式而设计的，我们还可以继续用上面说到的编辑器，又或者是其他的。如果你喜欢DreamWeaver那也不错，不过一开始使用IDE可无助于我们写出良好的代码。

忘说了，CSS也是有版本的，和windows，Linux内核等等一样，但是更新可能没有那么频繁，HTML也是有版本的，JS也是有版本的，复杂的东西不是当前考虑的内容。

4.3 代码结构

对于我们的上面的Red示例来说，如果没有一个好的结构，那么以后可能就是这样子。

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <p style="font-size: 22px;color:#f00;text-align: center;padding-left: 20px;">如果没有一个好的结构</p>
    <p style="  font-size:44px;color:#3ed;text-indent: 2em;padding-left: 2em;">那么以后可能就是这样子。。。。</p>
</body>
</html>
```

虽然我们看到的还是一样的:

如果没有一个好的结构

那么以后可能就是这样子。。。。

No Style

于是我们就按各种书上的建议重新写了上面的代码

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS example</title>
    <style type="text/css">
        .para{
            font-size: 22px;
            color:#f00;
            text-align: center;
            padding-left: 20px;
        }
        .para2{
            font-size:44px;
            color:#3ed;
            text-indent: 2em;
            padding-left: 2em;
        }
    </style>
</head>
<body>
    <p class="para">如果没有一个好的结构</p>
    <p class="para2">那么以后可能就是这样子。。。。</p>
</body>
</html>
```

总算比上面好看也好理解多了，这只是临时的用法，当文件太大的时候，正式一点的写法应该如下所示:

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS example</title>
    <style type="text/css" href="style.css"></style>
</head>
<body>
    <p class="para">如果没有一个好的结构</p>
    <p class="para2">那么以后可能就是这样子。。。。</p>
</body>
</html>
```

我们需要

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS example</title>
    <link href="./style.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <p class="para">如果没有一个好的结构</p>
    <p class="para2">那么以后可能就是这样子。。。。</p>
</body>
</html>
```

然后我们有一个像app.js一样的style.css放在同目录下，而他的内容便是

```
.para{
    font-size: 22px;
    color:#f00;
    text-align: center;
    padding-left: 20px;
}
.para2{
    font-size:44px;
    color:#3ed;
    text-indent: 2em;
    padding-left: 2em;
}
```

这代码和JS的代码有如此多的相似

```
var para={
    font_size:'22px',
    color:'#f00',
    text_align:'center',
    padding_left:'20px',
}
```

而22px、20px以及#f00都是数值，因此：

```
var para={
    font_size:22px,
    color:#f00,
    text_align:center,
    padding_left:20px,
}
```

目测差距已经尽可能的小了，至于这些话题会在以后讨论到，如果要让我们的编译器更正确的工作，那么我们就需要非常多这样的符号，除非你乐意去理解:

```
(dotimes (i 4) (print i))
```

总的来说我们减少了符号的使用，但是用lisp便带入了更多的括号，不过这是一种简洁的表达方式，也许我们可以在其他语言中看到。

```
\d{2}/[A-Z][a-z][a-z]/\d{4}
```

上面的代码，是为了从一堆数据中找出“某日/某月/某年”。如果一开始不理解那是正则表达式，就会觉得那个很复杂。

这门语言可能是为设计师而设计的，但是设计师大部分还是不懂编程的，不过相对来说这门语言还是比其他语言简单易懂一些。

4.4 样式与目标

如下所示，就是我们的样式

```
.para{
    font-size: 22px;
    color:#f00;
    text-align: center;
    padding-left: 20px;
}
```

我们的目标就是

```
如果没有一个好的结构
```

所以样式和目标在这里牵手了，问题是他们是如何在一起的呢？下面就是CSS与HTML沟通的重点所在了:

4.4.1 选择器

我们用到的选择器叫做类选择器，也就是class，或者说应该称之为class选择器更合适。与类选择器最常一起出现的是ID选择器，不过这个适用于比较高级的场合，诸如用JS控制DOM的时候就需要用到ID选择器。而基本的选择器就是如下面的例子:

```
p.para{
    color:#f0f;
}
```

将代码添加到style.css的最下面会发现“如果没有一个好的结构”变成了粉红色，当然我们还会有这样的写法

```
p>.para{
    color:#f0f;
}
```

为了产生上面的特殊的样式，虽然不好看，但是我们终于理解什么叫层叠样式了，下面的代码的重要度比上面高，也因此有更高的优先规则。

而通常我们可以通过一个

```
p{
    text-align:left;
}
```

这样的元素选择器来给予所有的p元素一个左对齐。

还有复杂一点的复合型选择器，下面的是HTML文件

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS example</title>
    <link href="./style.css" rel="stylesheet" type="text/css" />
</head>
```

```
<body>
    <p class="para">如果没有一个好的结构</p>
    <div id="content">
        <p class="para2">那么以后可能就是这样子。。。。</p>
    </div>
</body>
</html>
```

还有CSS文件

```
.para{
    font-size: 22px;
    color:#f00;
    text-align: center;
    padding-left: 20px;
}
.para2{
    font-size:44px;
    color:#3ed;
    text-indent: 2em;
    padding-left: 2em;
}

p.para{
    color:#f0f;
}
div#content p {
    font-size:22px;
}
```

4.5 更有趣的CSS

一个包含了para2以及para_bg的例子

```
    <div id="content">
        <p class="para2 para_bg">那么以后可能就是这样子。。。。</p>
    </div>
```

我们只是添加了一个黑色的背景

```
.para_bg{
    background-color:#000;
}
```

重新改变后的网页变得比原来有趣了很多，所谓的继承与合并就是上面的例子。

我们还可以用CSS3做出更多有趣的效果，而这些并不在我们的讨论范围里面，因为我们讨论的是be a geek。

或许我们写的代码都是那么的简单，从HTML到Javascript，还有现在的CSS，只是总有一些核心的东西，而不是去考虑那些基础语法，基础的东西我们可以在实践的过程中一一发现。但是我们可能发现不了，或者在平时的使用中考虑不到一些有趣的用法或者说特殊的用法，这时候可以通过观察一些精致设计的代码中学习到。复杂的东西可以变得很简单，简单的东西也可以变得很复杂。

5 无处不在的三剑客

这时我们终于了解了我们的三剑客，他们也就这么可以结合到一起了，HTML+Javascript+CSS是这一切的基础。而我们用到的其他语言如PHP、Python、Ruby等等到最后都会变成上面的结果，当然还有Coffeescript之类的语言都是以此为基础，这才是我们需要的知识。

5.1 Hello,Geek

有了一些些基础之后，我们终于能试着去写一些程序了。也是时候去创建一个像样的东西，或许你在一些界面设计方面的书籍看过类似的东西，可能我写得也没有那些内容好，只是这些都是一些过程。过去我们都是一点点慢慢过来的，只是现在我们也是如此，技术上的一些东西，事实上大家都是知道的。就好比我们都觉得我们可以开个超市，但是如果让我们去开超市的话，我们并不一定能赚钱。

学习编程的目的可能不在于我们能找到一份工作，那只是在编程之外的东西，虽然确实也是很确定的。但是除此之处，有些东西也是很重要的。

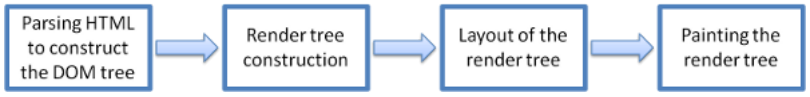
过去总是不理解为什么会一些人会不厌其烦地去回答别人的问题，有时候可能会想是一种能力越大责任越大的感觉，但是有时候在写一些博客或者回答别人的问题的时候我们又重新思考了这些问题，又重新学习了这些技能。所以这里可能说的不是关于编程的东西而是一些编程以外的东西，关于学习或者学习以外的东西。

5.2 从源码学习

过去总觉得学了一种语言的语法便算是学会了一种语言，直到有一天接触运用该语言的项目的时候，虽然也会写上几行代码，但是却不像这种语言的风格。于是这也是这一篇的意义所在了：

5.3 浏览器渲染过程

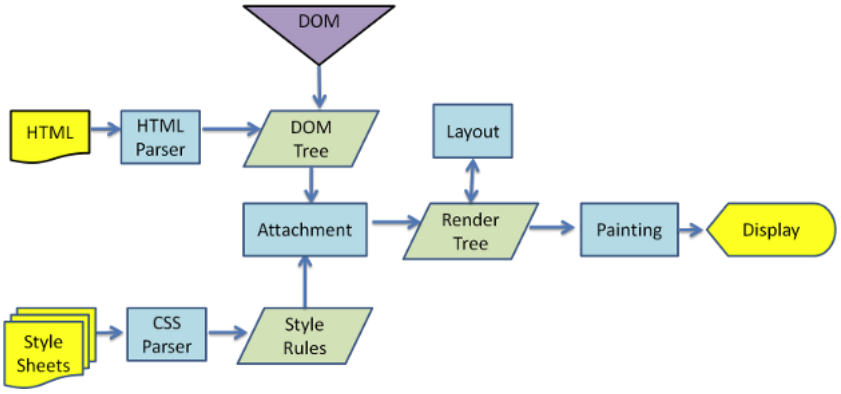
基本的渲染引擎的过程如下图所示:



flow

- 解析HTML去构建DOM树
- 渲染树形结构
- 生成渲染的树形图布局
- 绘制树形图

对于Webkit浏览器来说，他的过程如下所示:



webkitflow

5.3.1 HTML

写好HTML的一个要点在于读别人写的代码，这只是一方面，我们所说的HTML方面的内容可能不够多，原因有很多，很多东西都需要在实战中去解决。读万卷书和行万里路，分不清哪个有重要的意义，但是如果可以同时做好两个的话，成长会更快的。

写好HTML应该会有下面的要点

- 了解标准及遵守绝大多数标准
- 注重可读性，从ID及CLASS的命名
- 关注SEO与代码的联系

或许在这方面我也算不上很了解，不过按笔者的经验来说，大致就是如此。

多数情况下我们的HTML是类似于下面这样子的

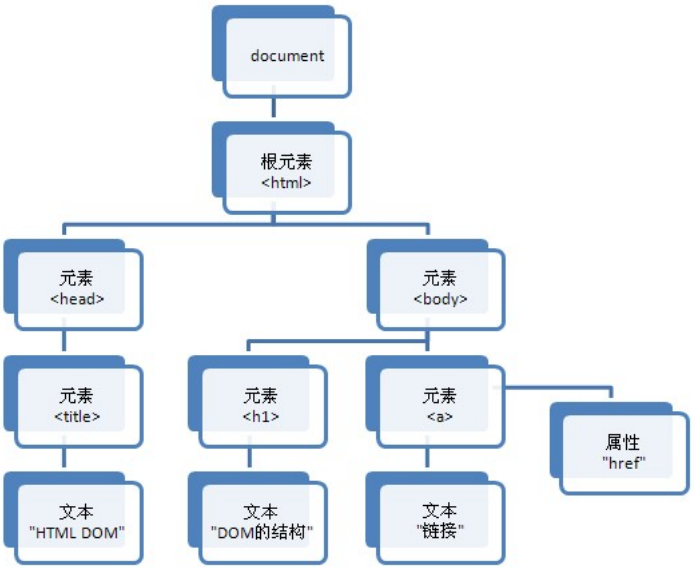
```
<div class="col-md-3 right">
    {% nevercache %}
    {% include "includes/user_panel.html" %}
    {% endnevercache %}
    <div class="panel panel-default">
    <div class="panel-body">
    {% block right_panel %}
    {% ifinstalled mezzanine.twitter %}
    {% include "twitter/tweets.html" %}
    {% endifinstalled %}
    {% endblock %}
    </div>
    </div>
</div>
```

换句话说HTML只是基础，而不是日常用到的。我们的HTML是由template生成的，我们可以借助于mustache.js又或者是angluarjs之类的js库来生成最后的HTML，所以这里只是一个开始。

还需要了解的一部分就是HTML的另外一个重要的部分，DOM树形结构

5.4 DOM树形结构图

DOM是文档对象化模型（Document Object Model）的简称。DOM Tree是指通过DOM将HTML页面进行解析，并生成的HTML tree树状结构和对应访问方法。



DOM Tree

5.4.1 javascript

这里以未压缩的jQuery源码和zepto.js作一个小小的比较，zepto.js是兼容jQuery的，因此我们举几个有意思的函数作一简单的比较，关于源码可以在官网上下载到。

在zepto.js下面判断一个值是否是函数的方面如下，

```
function isFunction(value) { return type(value) == "function" }
```

而在jQuery下面则是这样的

```
isFunction: function( obj ) {
    return jQuery.type(obj) === "function";
}
```

而他们的用法是一样的，都是

```
$.isFunction();
```

jQuery的作法是将诸如isFunction,isArray这些函数打包到jQuery.extend中，而zepto.js的也是这样的，只不过多了一行

```
$.isFunction = isFunction
```

遗憾的是我也没去了解过为什么，之前我也没有看过这些库的代码，所以这个问题就要交给读者去解决了。jQuery里面提供了函数式编程接口，不过jQuery更多的是构建于CSS选择器之上，对于DOM的操作比javascript自身提供的功能强大得多。如果我们的目的在于更好的编程，那么可能需要诸如Underscore.js之类的库。或许说打包自己常用的函数功能为一个库，诸如jQuery

```
function isFunction(value) { return type(value) == "function" }
function isWindow(obj)     { return obj != null && obj == obj.window }
function isDocument(obj)   { return obj != null && obj.nodeType == obj.DOCUMENT_NODE }
function isObject(obj)     { return type(obj) == "object" }
```

我们需要去了解一些故事背后的原因，越来越害怕GUI的原因之一，在于不知道背后发生了什么，即使是开源的，我们也无法了解真正的背后发生什么了。对于不是这个工具、软件的用户来说，开源更多的意义可能在于我们可以添加新的功能，当然还有免费。如果没有所谓的危机感，以及认为自己一直在学习工具的话，可以试着去打包自己的函数，打包自己的库。

```
var calc={
    add: function(a,b){
        return a+b;
    },
    sub: function(a,b){
```

```
        return a-b;
    },
    dif: function(a,b){
        if(a>b){
            return a;
        }else{
            return b;
        }
    }
}
```

然后用诸如jslint测试一下代码。

```
 $ ./jsl -conf jsl.default.conf
JavaScript Lint 0.3.0 (JavaScript-C 1.5 2004-09-24)
Developed by Matthias Miller (http://www.JavaScriptLint.com)

app.js
/Users/fdhuang/beageek/chapter4/src/app.js(15): lint warning: missing semicolon
    }
........^


0 error(s), 1 warning(s)
```

于是我们需要在第15行添加一个分号。

最好的方法还是阅读别人的代码，而所谓的别人指的是一些相对较大的网站的，有比较完善的开发流程，代码质量也不会太差。而所谓的复杂的代码都是一步步构建上去的，罗马不是一天建成的。

有意思的是多数情况下，我们可能会用原型去开发我们的应用，而这也是我们需要去了解和掌握的地方，

```
 function Calc(){

}
Calc.prototype.add=function(a,b){
    return a+b;
};
Calc.prototype.sub=function(a,b){
    return a-b;
};
```

我们似乎在这里展示了更多的Javascript的用法，但是这不是一好的关于Javascript的介绍，有一天我们还要用诸如qunit之类的工具去为我们的function写测试，这时就是一个更好的开始。

如果我们乐意的话，我们也可以构建一个类似于jQuery的框架，以用来学习。

作为一门编程语言来说，我们学得很普通，在某种意义上来说算不上是一种入门。但是如果我们可以在其他的好书在看到的内容，就没有必要在这里进行复述，目的在于一种学习习惯的养成。

5.4.2 CSS

CSS有时候很有趣，但是有时候有很多我们没有意识到的用法，这里以Bootstrap为例，这是一个不错的CSS库。最令人兴奋的是没有闭源的CSS，没有闭源的JS，这也就是前端好学习的地方所在了，不过这是一个开源的CSS库，虽然是这样叫的，但是称之为CSS库显然不合适。

```
 a,
a:visited {
  text-decoration: underline;
}
a[href]:after {
  content: " (" attr(href) ")";
}
abbr[title]:after {
  content: " (" attr(title) ")";
}
a[href^="javascript:"]:after,
a[href^="#"]:after {
  content: "";
}
```

这里有一些有趣的，值得一讲的CSS用法。

- 伪类选择器,如a:visited这样需要其他条件来对元素应用样式，用于已访问的链接。
- 属性选择器,如a[href]这样当a元素存在href这样的属性的时候来寻找应用元素。

其他的还需要去好好了解的就是**CSS**的盒模型，作为CSS的基石之一。

5.5 CSS盒模型图

(ps:以下内容来自于Mozilla Developer NetWorks)

CSS下这些矩形盒子由标准盒模型描述。这个模型描述元素内容占用空间。盒子有四个边界：外边距边界margin edge, 边框边界border edge, 内边距边界padding edge 与 内容边界content edge。

CSS Box Model

内容区域content area 是真正包含元素内容的区域。位于内容边界的内部，它的大小为内容宽度 或 content-box宽及内容高度或content-box高。

如果 box-sizing 为默认值， width, min-width, max-width, height, min-height 与 max-height 控制内容大小。

内边距区域padding area 用内容及可能的边框之间的空白区域扩展内容区域。它位于内边距边界内部，通常有背景——颜色或图片（不透明图片盖住背景颜色）. 它的大小为 padding-box 宽与 padding-box 高。

内边距与内容边界之间的空间可以由 padding-top, padding-right, padding-bottom, padding-left 和简写属性 padding 控制。

边框区域border area 是包含边框的区域，扩展了内边距区域。它位于边框边界内部，大小为 border-box 宽和 border-box 高。由 border-width 及简写属性 border控制。

外边距区域margin area用空白区域扩展边框区域，以分开相邻的元素。它的大小为 margin-box 的高宽。

外边距区域大小由 margin-top, margin-right, margin-bottom, margin-left 及简写属性 margin 控制。

在 外边距合并 的情况下，由于盒之间共享外边距，外边距不容易弄清楚。

最后注意，对于行内非替换元素，其占用空间（行高）由 line-height 决定，即使有内边距与边框。

诸如

```
* {
  margin: 0px;
  padding: 0px;
  font-family: Helvetica;
}
```

这样的通用器用来进行全局选择的工具和我们用于抵消某个body对于子选择器的影响一样值得注意得多。

5.6 笔记

写博客似乎是一个不错的好习惯，作为一个不是很优秀的写手。对于来说，有时候发现原来能教会别人对于自己的能力来说算是一种肯定。有些时候教会别人才算是自己学会的表现，总会在项目上的时候需要自己去复述工作的一个过程，我们需要整理好我们的思路才能带给别人更多的收获。我们的笔记上总会留下自己的学习的一些过程，有些时候我们想要的只是一点点的鼓励，有时是诸如评论一类，有时可能是诸如访问量。更多的可能是我们可以重新整理自己的知识，好好复习一下，以便于好好记住，写出来是一个好的过程。

无处不在的三剑客就这样到了这里，写得似乎很多也很少，但是还是没有做出来一个东西，于是我们朝着这样一个方向前进。

6 GNU/Linux 强大且Free

GNU/Linux

6.1 什么是Linux

Linux是一种自由和开放源码的类UNIX操作系统内核。目前存在着许多不同的Linux发行版，可安装在各种各样的电脑硬件设备，从手机、平板电脑、路由器和影音游戏控制台，到桌上型电脑，大型电脑和超级电脑。 Linux是一个领先的操作系统内核，世界上运算最快的10台超级电脑运行的都是基于Linux内核的操作系统。

Linux操作系统也是自由软件和开放源代码发展中最著名的例子。只要遵循GNU通用公共许可证,任何人和机构都可以自由地使用Linux的所有底层源代码，也可以自由地修改和再发布。严格来讲，**Linux**这个词本身只表示**Linux**内核，但在实际上人们已经习惯了用**Linux**来形容整个基于**Linux**内核，并且使用**GNU**工程各种工具和数据库的操作系统（也被称为**GNU/Linux**）。通常情况下，Linux被打包成供桌上型电脑和服务器使用的Linux发行版本。一些流行的主流Linux发行版本，包括Debian（及其衍生版本Ubuntu），Fedora和openSUSE等。Linux得名于电脑业余爱好者Linus Torvalds。

~~而不是如百度百科所讲的Linux操作系统是UNIX操作系统的一种克隆系统。它诞生于1991年的Linux桌面[1]10月5日（这是第一次正式向外公布的时间）。以后借助于Internet网络，并通过全世界各地计算机爱好者的共同努力，已成为今天世界上使用最多的一种UNIX类操作系统，并且使用人数还在迅猛增长。~~

Linux只是个内核，而不是操作系统，所以在这我们再理解一下操作系统是由什么组成的。

6.2 操作系统

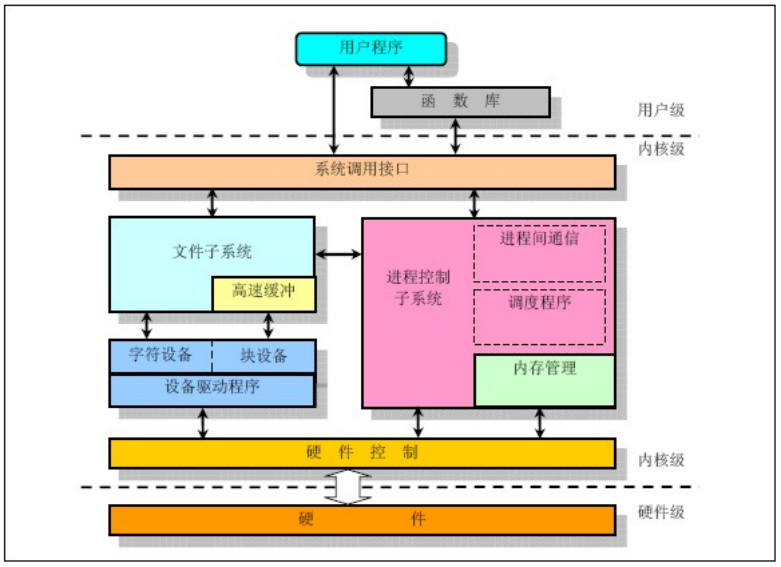
操作系统（英语：Operating System，简称OS）是管理计算机硬件与软件资源的计算机程序，同时也是计算机系统的内核与基石。操作系统需要处理如管理与配置内存、决定系统资源供需的优先次序、控制输入与输出设备、操作网络与管理文件系统等基本事务。操作系统也提供一个让用户与系统交互的操作界面。 操作系统的型态非常多样，不同机器安装的操作系统可从简单到复杂，可从手机的嵌入式系统到超级计算机的大型操作系统。许多操作系统制造者对它涵盖范畴的定义也不尽一致，例如有些操作系统集成了图形用户界面(GUI)，而有些仅使用命令行界面(CLI)，而将GUI视为一种非必要的应用程序。

操作系统位于底层硬件与用户之间，是两者沟通的桥梁。用户可以通过操作系统的用户界面，输入命令。操作系统则对命令进行解释，驱动硬件设备，实现用户要求。以现代标准而言，一个标准PC的操作系统应该提供以下的功能：

- 进程管理（Processing management）
- 内存管理（Memory management）
- 文件系统（File system）
- 网络通信（Networking）
- 安全机制（Security）
- 用户界面（User interface）
- 驱动程序（Device drivers）

而让我们来看一下两者之间的不同之处，这是一张linux的架构图我们可以发现内核只是位于底层。

6.2.1 Linux架构图



Linux Kernel

6.2.1.1 用户模式

应用程序（sh、vi、OpenOffice.org等）

复杂库（KDE、glib等） 简单库（opendbm、sin等）

C库（open、fopen、socket、exec、calloc等）

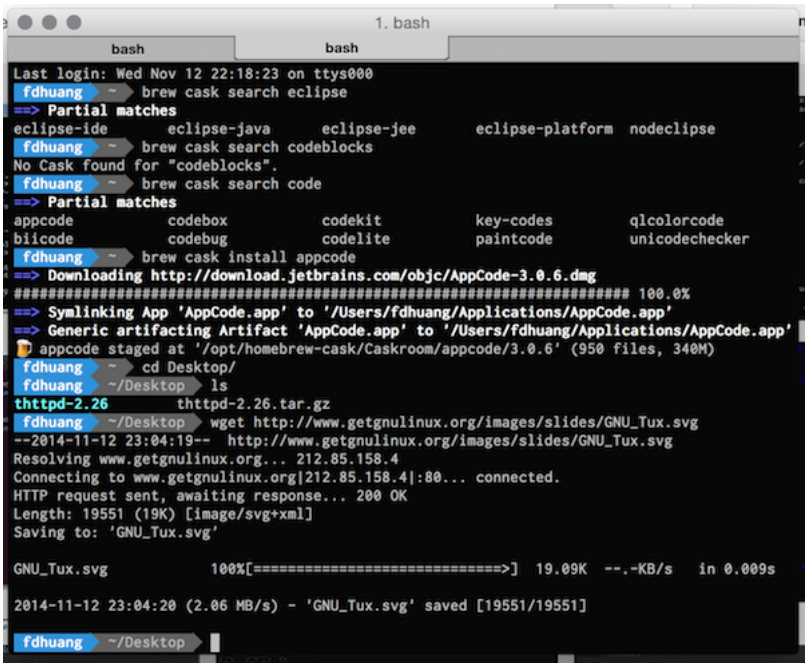
6.2.1.2 内核模式

- 系统中断、调用、错误等软硬件消息
- 内核（驱动程序、进程、网络、内存管理等）
- 硬件（处理器、内存、各种设备）

我们可以发现，由linux内核+shell可以构成一个操作系统，而linux本身只是个内核，也就是图中的内核模式，负责控制系统的这些部分。也就是我们可以发现，Linux内核构成了一个操作系统除用户界面以外的部分，而shell就是这最后的用户界面。

而linux内核以外的部分就是由GNU计划构成的。

6.2.2 Shell



Shell

Shell是系统的用户界面，提供了用户与内核进行交互操作的一种接口。它接收用户输入的命令并把它送入内核去执行。

实际上Shell是一个命令解释器，它解释由用户输入的命令并且把它们送到内核。不仅如此，Shell有自己的编程语言用于对命令的编辑，它允许用户编写由shell命令组成的程序。Shell编程语言具有普通编程语言的很多特点，比如它也有循环结构和分支控制结构等，用这种编程语言编写的Shell程序与其他应用程序具有同样的效果

bash 是一个为GNU计划编写的Unix shell。它的名字是一系列缩写：Bourne-Again SHell — 这是关于Bourne shell（sh）的一个双关语（Bourne again / born again）。Bourne shell是一个早期的重要shell，由史蒂夫·伯恩在1978年前后编写，并同Version 7 Unix一起发布。bash则在1987年由布莱恩·福克斯创造。在1990年，Chet Ramey成为了主要的维护者。

shell将会是我们在GNU/linux中经常用到的经常有到的工具之一，用来操作计算机用的。在迁移到linux之前我们可以试用cygwin来进行模拟：

Cygwin是许多自由软件的集合，最初由Cygnus Solutions开发，用于各种版本的Microsoft Windows上，运行`UNIX类`系统。

6.2.3 GCC

GCC（GNU Compiler Collection，GNU编译器套装），是一套由GNU开发的编程语言编译器。它是一套以GPL及LGPL许可证所发行的自由软件，也是GNU计划的关键部分，亦是自由的类Unix及苹果电脑Mac OS X 操作系统的标准编译器。GCC（特别是其中的C语言编译器）也常被认为是跨平台编译器的事实标准。

GCC原名为GNU C语言编译器（GNU C Compiler），因为它原本只能处理C语言。GCC很快地扩展，变得可处理C++。之后也变得可处理Fortran、Pascal、Objective-C、Java、Ada，以及Go与其他语言。

```
#include <stdio.h>
main()
{
  printf("Hello world\n");
}
~/temp/free> gcc hello.c -o hello
hello.c:2:1: warning: type specifier missing, defaults to 'int' [-Wimplicit-int]
main()
^~~~
1 warning generated.
~/temp/free> ./hello
Hello world
```

同shell一样，对于GNU/linux系统而言,GCC的作用也是无可取代的。当然如果只是一般用途的话，GCC对于一般用户可能没用，但是在些GNU/Linux系统上，我们可能就需要自己编译源码成二进制文件，而没有软件包，因而其重要性是不言而喻的。自然的如果我们自己动手编译GNU/Linux操作系统的话，我们会理解其的重要意义。有兴趣的同学可以试一下：Linux From Scratch (LFS)。

6.2.4 启动引导程序

最后，当我们构成以上的那些之后，我们就需要一个引导程序，以便使系统启动，引导进内核。

启动程序（bootloader）于电脑或其他计算机应用上，是指引导操作系统启动的程序。启动程序启动方式与程序视应用机型种类。例如在普通PC上，引导程序通常分为两部分：第一阶段引导程序位于主引导记录，用于引导位于某个分区上的第二阶段引导程序，如NTLDR、GNU GRUB等。

BIOS 开机完成后，bootloader就接手初始化硬件设备、创建存储器空间的映射，以便为操作系统内核准备好

正确的软硬件环境。

简单的bootloader的虚拟汇编码，如其后的八个指令:

- 0: 将P暂存器的值设为8
- 1: 检查纸带({paper tape)读取器，是否已经可以进行读取
- 2: 如果还不能进行读取, 跳至1
- 3: 从纸带读取器，读取一byte至累加器
- 4: 如为带子结尾，跳至8
- 5: 将暂存器的值，存储至P暂存器中的数值所指定的地址
- 6: 增加P暂存器的值
- 7: 跳至1

但是随着计算机操作系统越来越复杂，位于MBR的空间已经放不下引导操作系统的代码，于是就有了第二阶段的引导程序，而MBR中代码的功能也从直接引导操作系统变成了引导第二阶段的引导程序。

通常在一个GNU/Linux系统中选用GNUGRUB做为引导程序，例如Ubuntu就是用GRUB2。

GNU GRUB（简称“GRUB”）是一个来自GNU项目的启动引导程序。GRUB是多启动规范的实现，它允许用户可以在计算机内同时拥有多个操作系统，并在计算机启动时选择希望运行的操作系统。GRUB可用于选择操作系统分区上的不同内核，也可用于向这些内核传递启动参数。

GNU GRUB的前身为Grand Unified Bootloader。它主要用于类Unix系统；同大多Linux发行版一样，GNU系统也采用GNU GRUB作为它的启动器。Solaris从10 1/06版开始在x86系统上也采用GNU GRUB作为启动器。

以上也就构成了一个简单的操作系统。

6.3 从编译开始

我们以一次编译开始我们的Linux学习之旅。

6.3.1 开始之前

- 如果你没有用过GNU/Linux，我想你需要在虚拟机上安装一个。
- 一个主流的GNU/Linux发行版，如Ubuntu,CentOS,Debian,Mint,OpenSUSE,Fedora等等。
- 学会如何打开shell(ps:bash,zsh,sh等等)。

或者你也可以在Windows上安装Cygwin。

6.3.2 编译Nginx

1.下载这个软件的源码包

```
wget http://nginx.org/download/nginx-1.7.4.tar.gz
```

wget是一个用于下载的软件，当然你也可以用软件，只是用wget似乎会比图形界面快哦。

2.解压软件包

```
tar -vf nginx-1.7.4.tar.gz
```

-vf的意思是Extract，也就是解压，而tar则是这个解压软件的名字。看上去似乎比WinRAR来得复制得多，但是你可以计时一下，从下载到解压完，和你用鼠标比哪个比较快。

3.到nginx目录下

这里需要分两部进行

1).列出所有文件

```
ls -al
```

```
drwxr-xr-x   15 fdhuang  staff   510B Sep  2 13:44 nginx-1.7.4
-rw-r--r--    1 fdhuang  staff   798K Aug  5 21:55 nginx-1.7.4.tar.gz
```

2).到nginx-1.7.4目录

```
cd nginx-1.7.4
```

4.配置nginx

一次简单的配置如下

```
./configure
```

当你熟练以后，你可能和我一样用下面的配置(注意:用下面的代码会出错。)

```
./configure --user=www --group=www --add-module=../ngx_pagespeed-1.8.3.4-beta --add-module=../ngx_cache_purge --prefix=/usr/local/nginx --w
```

过程中可能会提示你其中出了多少错误，而这时你便可以很愉快地去用搜索引擎搜索他们。

5.make

这里就会用到GCC等等。

```
make
```

6.运行

如果运行顺利的话，应该可以直接

```
./objs/nginx
```

6.3.3 其他

1.如果没有wget,make,gcc等命令的时候可以用类似于下面的方法安装，

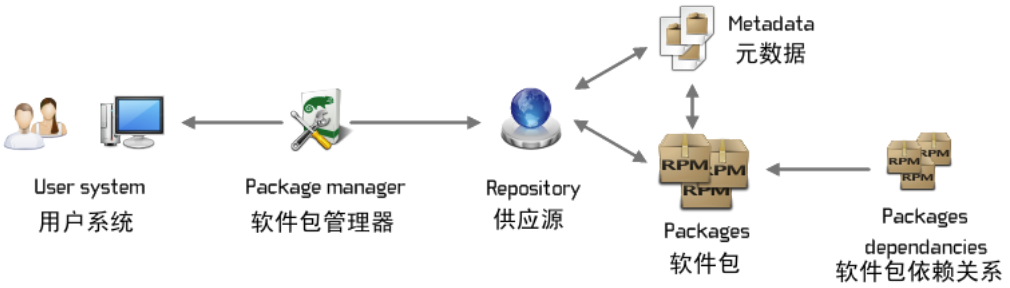
```
sudo apt-get install gcc,make,wget
```

2.正常情况下一个开源项目都会有一个README，会告诉你应该如何去做。

6.4 包管理

GNU/Linux最方便的东西莫过于包管理了。

引自OpenSUSE官网的说明及图片[1]



package management

1. Linux 发行版无非就是一堆软件包 (package) 形式的应用程序加上整体地管理这些应用程序的工具。通常这些 Linux 发行版，包括 openSUSE，都是由成千上万不同的软件包构成的。

2. 软件包: 软件包不止是一个文件，内含构成软件的所有文件，包括程序本身、共享库、开发包以及使用说明等。

3. 元数据 (metadata) 包含于软件包之中，包含软件正常运行所需要的一些信息。软件包安装之后，其元数据就存储于本地的软件包数据库之中，以用于软件包检索。

4. 依赖关系 (dependencies) 是软件包管理的一个重要方面。实际上每个软件包都会涉及到其他的软件包，软件包里程序的运行需要有一个可执行的环境（要求有其他的程序、库等），软件包依赖关系正是用来描述这种关系的。

Linux 下的软件包通常是以下三种格式：

- tgz - tar gzip 文件。这类文件是基本的压缩软件包，可以容纳软件包维护者认为有用的所有的东西。此格式除本身的压缩格式外，并没有有关软件包内容的标准。
- deb - 此格式的软件包常用于 Debian 系统，是标准的 Debian 软件包格式。
- rpm - 此格式由 Red Hat Linux 所创建，并经由 LSB 标准化，现已为众多 Linux 发行版所采用，是一个优秀的软件包格式。openSUSE 即是用此格式。更多信息可以参阅此处。

所以这就需要能自动解决依赖关系的软件包管理器。软件包管理系统就是一个工具集，为系统提供一个统一的安装、升级、删除软件的方式。

6.5 Ubuntu LNMP

在余下的章节中，我们需要去部署，需要去使用Ubuntu。如果在Windows下可以使用LAMP，但是在这里我们只说Ubuntu。开始之前你需要安装好Ubuntu，无论是在虚拟机，还是在真机安装，或者是Docker。

6.5.1 Update软件包列表

`apt-get`是debian，ubuntu发行版的包管理工具。`apt-get update`可以确保我们的软件包列表是最新的，下面是一个简单的更新过程。

打开Terminal或者Konsole等等之类的终端控制台。

```
root@70cdc7a176a5:/# sudo apt-get update
Ign http://archive.ubuntu.com trusty InRelease
```

```
Ign http://archive.ubuntu.com trusty-updates InRelease
Ign http://archive.ubuntu.com trusty-security InRelease
Ign http://archive.ubuntu.com trusty-proposed InRelease
Get:1 http://archive.ubuntu.com trusty Release.gpg [933 B]
Get:2 http://archive.ubuntu.com trusty-updates Release.gpg [933 B]
Get:3 http://archive.ubuntu.com trusty-security Release.gpg [933 B]
Get:4 http://archive.ubuntu.com trusty-proposed Release.gpg [933 B]
Get:5 http://archive.ubuntu.com trusty Release [58.5 kB]
Get:6 http://archive.ubuntu.com trusty-updates Release [62.0 kB]
Get:7 http://archive.ubuntu.com trusty-security Release [62.0 kB]
Get:8 http://archive.ubuntu.com trusty-proposed Release [209 kB]
Get:9 http://archive.ubuntu.com trusty/main Sources [1335 kB]
Get:10 http://archive.ubuntu.com trusty/restricted Sources [5335 B]
Get:11 http://archive.ubuntu.com trusty/universe Sources [7926 kB]
Get:12 http://archive.ubuntu.com trusty/main amd64 Packages [1743 kB]
Get:13 http://archive.ubuntu.com trusty/restricted amd64 Packages [16.0 kB]
Get:14 http://archive.ubuntu.com trusty/universe amd64 Packages [7589 kB]
64% [14 Packages 664 kB/7589 kB 9%]                          58.3 kB/s 1min 58s
```

更新完应该会显示:

```
Fetched 20.5 MB in 5min 22s (63.6 kB/s)
Reading package lists... Done
```

6.5.2 安装MySQL

安装命令

```
sudo apt-get install mysql-server php5-mysql
```

过程:

```
root@70cdc7a176a5:/# sudo apt-get install mysql-server php5-mysql
Reading package lists... 0%
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libaio1 libdbd-mysql-perl libdbi-perl libhtml-template-perl libmysqlclient18
  libterm-readkey-perl libwrap0 lsof mysql-client-5.5 mysql-client-core-5.5
  mysql-common mysql-server-5.5 mysql-server-core-5.5 php5-common php5-json
  psmisc tcpd
Suggested packages:
  libclone-perl libmldbm-perl libnet-daemon-perl libplrpc-perl
  libsql-statement-perl libipc-sharedcache-perl tinyca mailx php5-user-cache
The following NEW packages will be installed:
  libaio1 libdbd-mysql-perl libdbi-perl libhtml-template-perl libmysqlclient18
  libterm-readkey-perl libwrap0 lsof mysql-client-5.5 mysql-client-core-5.5
  mysql-common mysql-server mysql-server-5.5 mysql-server-core-5.5 php5-common
  php5-json php5-mysql psmisc tcpd
0 upgraded, 19 newly installed, 0 to remove and 12 not upgraded.
Need to get 9982 kB of archives.
After this operation, 99.1 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/main libaio1 amd64 0.3.109-4 [6364 B]
Get:2 http://archive.ubuntu.com/ubuntu/ trusty-updates/main mysql-common all 5.5.40-0ubuntu0.14.04.1 [14.1 kB]
Get:3 http://archive.ubuntu.com/ubuntu/ trusty-updates/main libmysqlclient18 amd64 5.5.40-0ubuntu0.14.04.1 [598 kB]
Get:4 http://archive.ubuntu.com/ubuntu/ trusty/main libwrap0 amd64 7.6.q-25 [46.2 kB]
Get:5 http://archive.ubuntu.com/ubuntu/ trusty/main libdbi-perl amd64 1.630-1 [879 kB]
Get:6 http://archive.ubuntu.com/ubuntu/ trusty/main libdbd-mysql-perl amd64 4.025-1 [99.3 kB]
Get:7 http://archive.ubuntu.com/ubuntu/ trusty/main libterm-readkey-perl amd64 2.31-1 [27.4 kB]
Get:8 http://archive.ubuntu.com/ubuntu/ trusty-updates/main mysql-client-core-5.5 amd64 5.5.40-0ubuntu0.14.04.1 [703 kB]
Get:9 http://archive.ubuntu.com/ubuntu/ trusty-updates/main mysql-client-5.5 amd64 5.5.40-0ubuntu0.14.04.1 [1466 kB]
Get:10 http://archive.ubuntu.com/ubuntu/ trusty-updates/main mysql-server-core-5.5 amd64 5.5.40-0ubuntu0.14.04.1 [3215 kB]
47% [10 mysql-server-core-5.5 850 kB/3215 kB 26%]            79.9 kB/s 1min 6s
```

在安装的过程中会要求你输入数据库密码。(默认为空)

6.5.3 安装Nginx

```
echo "deb http://ppa.launchpad.net/nginx/stable/ubuntu $(lsb_release -sc) main" | sudo tee /etc/apt/sources.list.d/nginx-stable.list
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys C300EE8C
sudo apt-get update
sudo apt-get install nginx
```

启动Nginx Server

```
sudo service nginx start
```

6.5.4 安装PHP

sudo apt-get install php5-fpm

安装过程

```
root@70cdc7a176a5:/# sudo apt-get install php5-fpm
Reading package lists... Done
```

```
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libsystemd-daemon0
Suggested packages:
  php-pear
The following NEW packages will be installed:
  libsystemd-daemon0 php5-fpm
0 upgraded, 2 newly installed, 0 to remove and 12 not upgraded.
Need to get 2201 kB of archives.
After this operation, 9326 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu/ trusty-proposed/main libsystemd-daemon0 amd64 204-5ubuntu20.8 [9608 B]
Get:2 http://archive.ubuntu.com/ubuntu/ trusty-updates/universe php5-fpm amd64 5.5.9+dfsg-1ubuntu4.5 [2191 kB]
Fetched 2201 kB in 1min 5s (33.6 kB/s)
Selecting previously unselected package libsystemd-daemon0:amd64.
(Reading database ... 13105 files and directories currently installed.)
Preparing to unpack .../libsystemd-daemon0_204-5ubuntu20.8_amd64.deb ...
Unpacking libsystemd-daemon0:amd64 (204-5ubuntu20.8) ...
Selecting previously unselected package php5-fpm.
Preparing to unpack .../php5-fpm_5.5.9+dfsg-1ubuntu4.5_amd64.deb ...
Unpacking php5-fpm (5.5.9+dfsg-1ubuntu4.5) ...
Processing triggers for ureadahead (0.100.0-16) ...
Setting up libsystemd-daemon0:amd64 (204-5ubuntu20.8) ...
Setting up php5-fpm (5.5.9+dfsg-1ubuntu4.5) ...

Creating config file /etc/php5/fpm/php.ini with new version
php5_invoke: Enable module pdo for fpm SAPI
php5_invoke: Enable module pdo_mysql for fpm SAPI
php5_invoke: Enable module opcache for fpm SAPI
php5_invoke: Enable module json for fpm SAPI
php5_invoke: Enable module mysql for fpm SAPI
php5_invoke: Enable module mysqli for fpm SAPI
invoke-rc.d: policy-rc.d denied execution of start.
Processing triggers for libc-bin (2.19-0ubuntu6.3) ...
Processing triggers for ureadahead (0.100.0-16) ...
```

7 Arduino 极客的玩具

7.1 极客的玩具

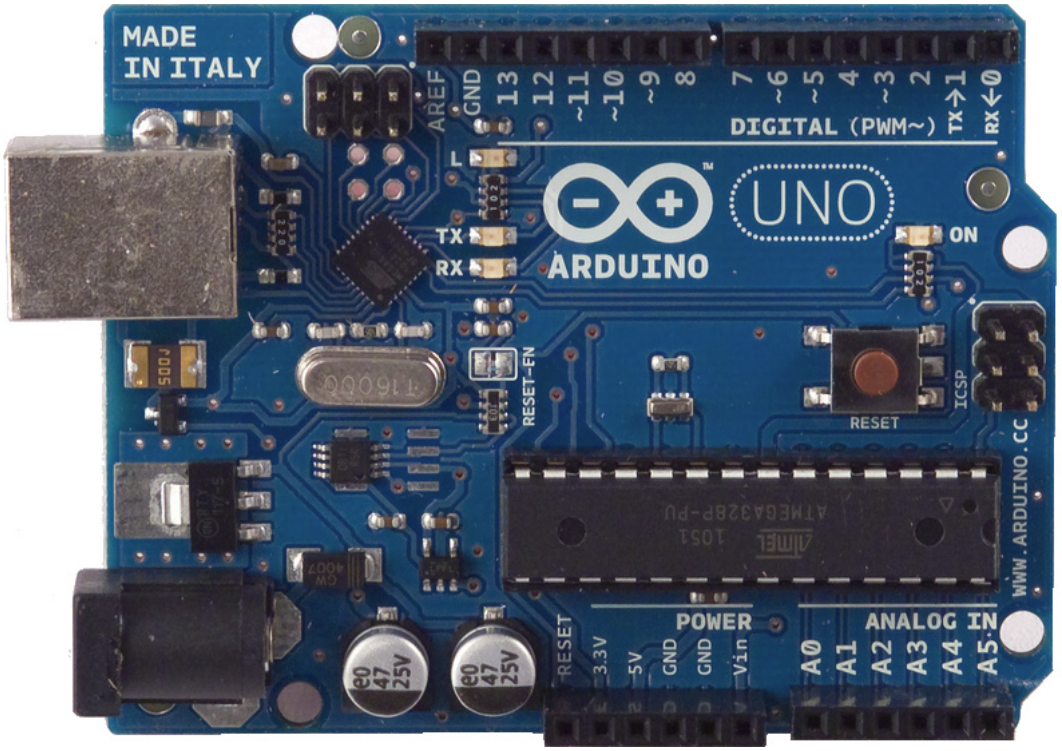
Arduino，是一个开放源代码的单芯片微电脑，它使用了Atmel AVR单片机，采用了基于开放源代码的软硬件平台，构建于开放源代码 simple I/O 接口板，并且具有使用类似Java，C 语言的Processing/Wiring开发环境。

Arduino开发板封装了常用的库到开发环境中，可以让用户在开发产品时，将主要注意力放置于所需要实现的功能上，而不是开发的过程中。在为Arduino写串口程序时，我们只需要用Serial.begin(9600)以9600的速率初始化串口，而在往串口发送数据时，可以用Serial.write('1')的方式向串口发送字串'1'。

Arduino的出现很大程度上降低了电子制作的难度，初学者甚至不懂编程也可以上手Arduino,这也是它的魅力所在。

7.2 硬件熟悉

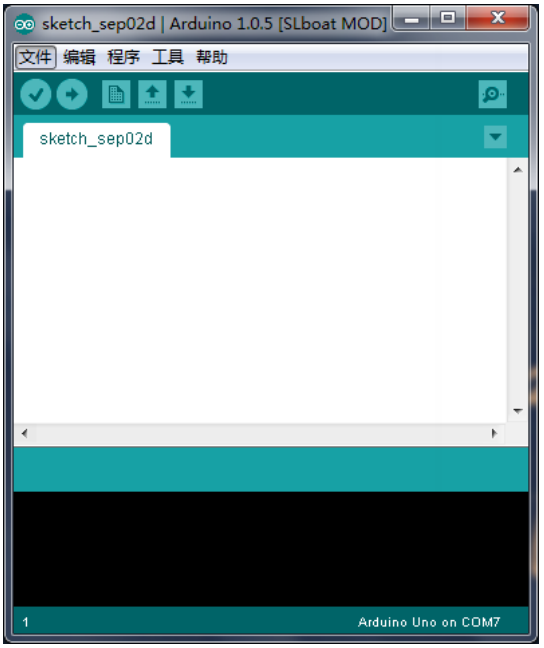
为了满足各种需求，Arduino团队设计了很多款开发板，如UNO、Pro mini、Mega2560、Due、Leonardo、Yún、Pro、Fio、Nano等十几种 开发板和扩展板。最适合初学者的一款是Arduino UNO 。下图是Arduino UNO 的外观图：



UNO

注：后面的程序也是基于Arduino UNO开发板来讲解。

7.3 开发环境



Arduino

开发环境如上图，十分简洁，编写代码需要知道两个基本的函数：

```
void setup(){

}

void loop(){

}
```

`setup()`函数用于初始化（如GPIO初始化，串口初始化，定时器初始化等）特点是只执行一次；`loop()`函数是一个死循环，可以看做C语言的`while(1)`函数。

7.4 点亮一个LED

对初学者来说，点亮led已成为入门必修课，使用Arduino控制led十分简单，并且很容易理解。 使用到的函数：

- pinMode(pin,mode)
- digitalWrite(pin,value)

上一段代码分析：

```
int led=13;

void setup()
{
    pinMode(led,OUTPUT);
}
void loop()
{
    digitalWrite(led,HIGH);
    delay(1000);
    digitalWrite(led,LOW);
    delay(1000);
}
```

该程序实现Arduino单片机13号引脚以1S时间电平翻转，如果外接一个led，就可以看到led以1S的间隔闪烁；函数`pinMode()`有两个参数pin、value，pin参数用来指定引脚号,本程序中设置为13号引脚，mode用于设置引脚模式，有三个值：

- `INPUT`
- `OUTPUT`
- `INPUT_PULLUP`

表示让某一个IO引脚作输入，反之，

- `OUTPUT`则使一个IO引脚做输出
- `INPUT_PULLUP`则配置一个IO引脚具有上拉输入功能(上拉电阻的目的是为了保证在无信号输入时输入端的电平为高电平)，从英文意思也能很直观的看出来。

理解了`pinMode()`函数,`digitalWrite()`就很容易理解啦，value的取值有两个`HIGH`、`LOW`,`HIGH`表示让某一个引脚输出高电平，反之，`LOW`则使某一个引脚输出低电平。 程序中还是用到`delay(ms)`函数，它表示延时多少毫秒，例如延时500 ms ,直接调用`delay(500);`就可以了。

如果你仔细查看我的描述，你会发现我没有讲13号引脚怎么来的，是这样的：Arduino团队为了简化对引脚描述，对每个引脚都进行了编号，以UNO开发板为例，可以发现开发板排座的附近有对应的白颜色的数字，那便是所有的引脚编号，A0~A5是6路ADC输入引脚，0-13表示13路基本IO，数字前面的`~`表示该引脚具有PWM功能。如果要使用某一引脚，只需要知道引脚编号就可编写相应代码进行操作。

例如`digitalWrite(2,LOW)`表示向2号引脚输出低电平。其他操作类似，是不是so easy ~！

7.5 串口通信

使用到的基本函数：

- Serial.begin()
- Serial.write()
- Serial.read()
- Serial.available()

在此项目中需要使用串口，Arduino串口初始化使用`Serial.begin(9600);`,其传输波特率为9600，其他波特率也行，函数位于`setup()`中，之后可以使用`Serial.read()`、`Serial.write()`读入一个字符，输出一个字符，使用`Serial.print()`输出字符串.代码如下:

```
char ch='1';
void setup()
{
    Serial.begin(9600);
}
void loop()
{
    Serial.write(ch);
    while(1)
    {
        if(Serial.available())
```

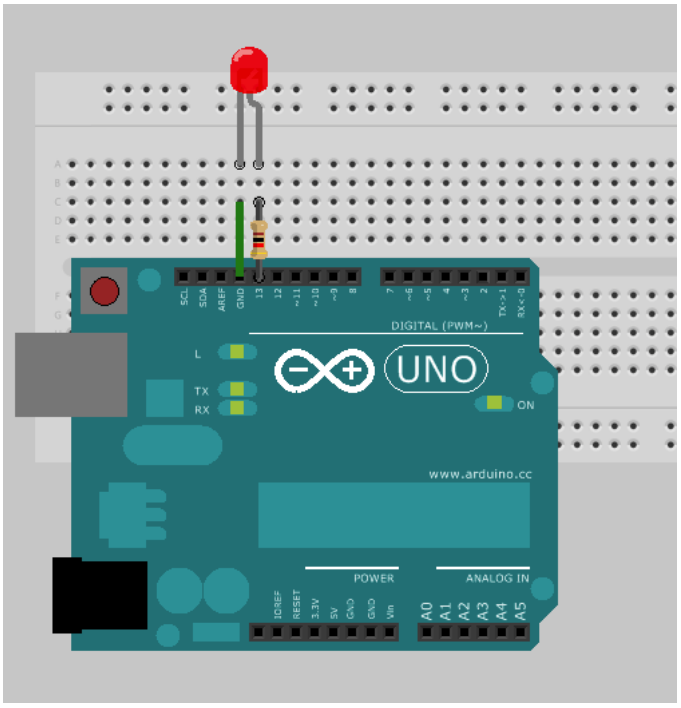
```
        {
            ch = Serial.read();
            Serial.print(ch);
        }
    }
}
```

以上程序实现字符的输出(Serial.write(),Serial.print())和读入(Serial.read())。如果需要了解更多，可以参考：Arduino官网

7.5.1 关于Arduino Setup()

如果你对Arduino的Setup很疑惑的话，可以看看这里。下面Arduino源码目录中的main函数:

```
#include <Arduino.h>

int main(void)
{
    init();
    setup();
    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }
    return 0;
}
```



hwcnt

8 Python 代码如散文

python

作为一门计算机语言来说，Python会有下面的特点。

- 语言学习起来容易
- 解决生活中的实际问题
- 支持多学科

我们可以和其他不是脚本语言的语言进行一个简单的对比，如C，你需要去编译去运行，有时候还需要解决跨平台问题，本来你是在你的Windows上运行得好好的，但是有一天你换了一个Mac电脑的时候，问题变得很棘手，你甚至不知道怎么去解决问题。我没有用过MFC，听说很方便，但是在其他平台下就没有一个好的解决方案。这里可能跑得有点远，但是不同的用户可能在不同的平台上，这也就是脚本语言的优势所在了。

8.1 代码与散文

你可能听过，也可能了解过，不过在这里我们可能不会去讲述那些基础的语法的东西，我们想说的是代码格式的重要性，在html中你可以这样去写你的代码

```
<html><head><title>This is a Title
</title></head><body><div class="content">
<p>flakjfaklfjalfa</p></div>
</body></html>
```

又或者是js的minify，它可能会使你的代码看起来像是这样的:

```
function NolTracker(b,a){this.pvar=b;this.mergeFeatures(a)}
```

可能的是如果是python的话，你可能会遇到下面的问题。。

```
File "steps.py", line 10
  try:
  ^
IndentationError: expected an indented block
```

如果你对JSLint、Lint这类的工具有点印象的话，你也可以认为python集成了这类工具。整洁的代码至少应该看上去要有整洁的衣服，就好像是我们看到的一个人一样，而后我们才会有一个好的印象。更主要的一点是代码是写给人看的，而衣服更多的时候对于像我这样的人来说，他首先应该是要保暖的，其次对于一个懒的人来说。。。

> 程序应该是具有可读性的短文，它将在计算机上执行，从而解决某些问题

我们需要去读懂别人的代码，别人也需要去读懂我们的代码。计算机可以无条件地执行你那未经编排过的程序，但是人就不是如此了。

```
    var calc={add: function(a,b){return a+b;},sub: function(a,b){return a-b;},dif: function(a,b){if(a>b){return a;}else{return b;}}}
```

上面的代码相对于下面的代码可读性没有那么多，但是计算机可以无条件地执行上面的代码。上面的代码对于网络传输来说是好的，但是对于人来说并不是如此，我们需要一些工具来辅助我们去读懂上面的代码。如果代码上写得没有一点可读性，诸如函数命名没有多少实际意义，如果我们把前面的函数就成这样：

```
var c={
    a: function(a,b){
        return a+b;
    },
    s: function(a,b){
        return a-b;
    },
    d: function(a,b){
        if(a>b){
            return a;
        }else{
            return b;
        }
    }
}
```

那么只有在我们理解了这个函数是干什么之后才能理解函数是干什么，而不是光看函数名就可以了。

在Javascript解决一个函数的办法有很多，在其他一些语言如Ruby或者Perl中也是如此，解决问题的办法有很多，对于写代码的人来说是一个享受的过程，但是对于维护的人来说并非如此。而这个和Python的思想不是很一致的是，Python设计的理念是

> 对于特定的问题，只要有一种最好的方法来解决就够了

可读性的代码在今天显得比以前重要的多，以前写程序的时候我们需要去考虑使用汇编或者其他工具来提高程序的效率。

```
        .global _start

        .text
_start:
        # write(1, message, 13)
        mov     $1, %rax                # system call 1 is write
        mov     $1, %rdi                # file handle 1 is stdout
        mov     $message, %rsi          # address of string to output
        mov     $13, %rdx               # number of bytes
        syscall                         # invoke operating system to do the write

        # exit(0)
        mov     $60, %rax               # system call 60 is exit
        xor     %rdi, %rdi              # we want return code 0
        syscall                         # invoke operating system to exit
message:
        .ascii  "Hello, world\n"
```

所以上面的代码的可读性在今天新生一代的程序员来说可能没有那么容易理解。芯片运行的速度越来越快，在程序上我们也需要一个越来越快的解决方案，而所谓的越来越快的解决方案指的不是运行速度上，而是开发速度上。如果你没有办法在同样时间内开发出更好的程序，那么你就可能输给你的竞争对手。

8.1.1 开始之前

我们终于又从一种语言跳到了另外一种语言，我们可能习惯了一种模式，而不敢于去尝试新的东西，这些或许是我们的一些习惯又或者是因为害怕等等。

作为另外一个跨平台能力很强的语言，这里说的是与Javascript、HTML比较，或许你会觉得C算是最好的，但是我们这里讨论更多的是脚本语言，也就是直接可以运行的。在现在主流的大多数移动平台上，python也有良好的支持，如Android,IOS，只是这些算是类Unix系统内核，python还支持之前Nokia的Symbian。

开始之前我们需要确认我们的平台上已经有了python环境，也就是可以运行下面的Hello,World，你可以在网上很多地方看到，最简单的地方还是到官网，又或者是所用移动平台的store下载。

8.1.2 Python的Hello,World

Python的Hello,World有两种形式，作为一种脚本语言来说，Javascript也是一种脚本语言，只是两者之间有太多的不同之处，每个人都会有不同的选择对于一种语言用来作为其的习惯。于是这就是我们的

```
print "Hello,World"
```

当我们把我们的脚本在shell环境下运行时

```
>>>  print "Hello,world"
  File "<stdin>", line 1
    print "Hello,world"
    ^
IndentationError: unexpected indent
>>> print "Hello,world"
Hello,world
>>>
```

如果你没有预料到缩进带来的问题的时候，这就是一个问题了。

和我们在Javascript或者是CSS里面一样，我们也可以用一个文件的方式来写入我们的代码，文件后缀名是py，所以创建一个helloworld.py，输入上面的代码，然后执行

```
python helloworld.py
```

一个理想的结果，或许你试过C语言的helloworld，如果了解过GCC的话应该是可以这样的:

```
./a.out
```

也就是执行编译完后的程序，需要注意的是helloworld.py没有编译，不过也会输出

```
Hello,world
```

8.1.3 我们想要的Hello,World

我们想展示的是如何结合前面学习的内容做一个更有意思的Hello,World。

```
import cherrypy
class HelloWorld(object):
    def index(self):
        return "Hello World!"
    index.exposed = True

cherrypy.quickstart(HelloWorld())
```

8.2 算法

我们需要去了解算法(algorithm)，引经据典的话就是这样子:

> a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer

也就是计算或其他解决问题的操作需要遵循的一个过程或者一套规则，书上还提到的说法是——解决问题的诀窍，让我想起了hack一词。我们总会去想某些东西是否有一个更快的计算方法，有时候在处理某些问题上也显示了一个好的算法的重要性。

8.3 实用主义哲学

(来自于:HyryStudio)

大多数工程师、科学家使用科学计算软件的目的都是为了快速解决其工作中遇到的问题，而不是开发出一套完整的软件。这就是为什么MATLAB这样的商用科学计算软件十分流行的原因。而Python在这一点上实际上和MATLAB十分相似，我们也可以使用Python众多的扩展库快速写出一次性的数据处理、运算的脚本。然而由于Python语言的一些高级特性，以及众多的科学计算之外的扩展库，我们可以将积累下来的一次性脚本进行改造，为它们提供命令行、GUI、数据库、网络等众多接口，最终成为一套完整的工具包或者实用的计算软件。而且由于是开源的自由软件，我们可以在任何系统中安装Python环境，运行我们 的程序。

Python一直保持着很强的实用主义，它通常不会去试着重新开发一整套函数库，而是将现有的开源函数库包装成其扩展库。而Python则通过这些扩展库将众多的开源函数库连接在一起，是名符其实的胶水语言。例如由华盛顿大学的教授主导开发的 Sage ，就是一套以代替MATLAB、Mathematica、Maple等商用科学计算软件为目的的开源系统。它通过Python结合了众多的开源科学计算软件，并通过网页浏览器提供了一个与其交互的记事本文档界面。 Python的科学计算扩展库非常多，不同专业的技术人员都可以找到适合自己的扩展库。下面是我经常会用到的一个非常不完全的列表：

- NumPy + SciPy + matplotlib + IPython : 这几个应该是每位开发者都应具备的扩展库。NumPy提供了多维数组以及众多的处理函数，SciPy提供了各种数值运算功能，matplotlib能绘制 出精美的二维图表，IPython则提供了一个超强的命令行，最新版的IPython还添加于Sage类似的浏览器的记事本界面(notebook)。
- SciKits : 其中包括许多独立的扩展库，作为SciPy的补充。其中 scikit-learn 是一套机器学习库，包含了比较完善的文档以及众多的实例程序。
- Pandas : 以Python世界中 R 的替代品为目标的数据分析库。根据其官方网站的测试，Pandas在许多方面的性能都比R要高。
- ETS : 这是一套Enthought公司开发的函数库，其中的 Mayavi 能很方便地对数据进行三维可视化。
- OpenCV : 这是一套计算机视觉库，目前的最新版本已经提供了十分完备的Python接口，能够调用OpenCV中众多的图像处理、模式识别函数直接对NumPy数组进行处理。

8.4 包管理

关于Python的包管理

- Eggs 格式是 setuptools 引入的一种文件格式，它使用 .egg 扩展名，用于 Python 模块的安装。
- pip 是目前 python 包管理的事实标准，2008年发布。它被用作 easy_install 的替代品，但是它仍有大量的功能建立在 setuptools 组件之上。

8.4.1 python requests

```
Requests 是使用 Apache2 Licensed 许可证的 HTTP 库。用 Python 编写，真正的为人类着想。
```

```
Python 标准库中的 urllib2 模块提供了你所需要的大多数 HTTP 功能，但是它的 API太渣了。它是为另一个时代、另一个互联网所创建的。它需要巨量的工作，甚至包括各种方法覆盖，来
```

以安装requests为例：

命令:

```
sudo pip install requests
```

结果:

```
Downloading/unpacking requests
  Downloading requests-2.4.3-py2.py3-none-any.whl (459kB): 459kB downloaded
Installing collected packages: requests
Successfully installed requests
Cleaning up...
```

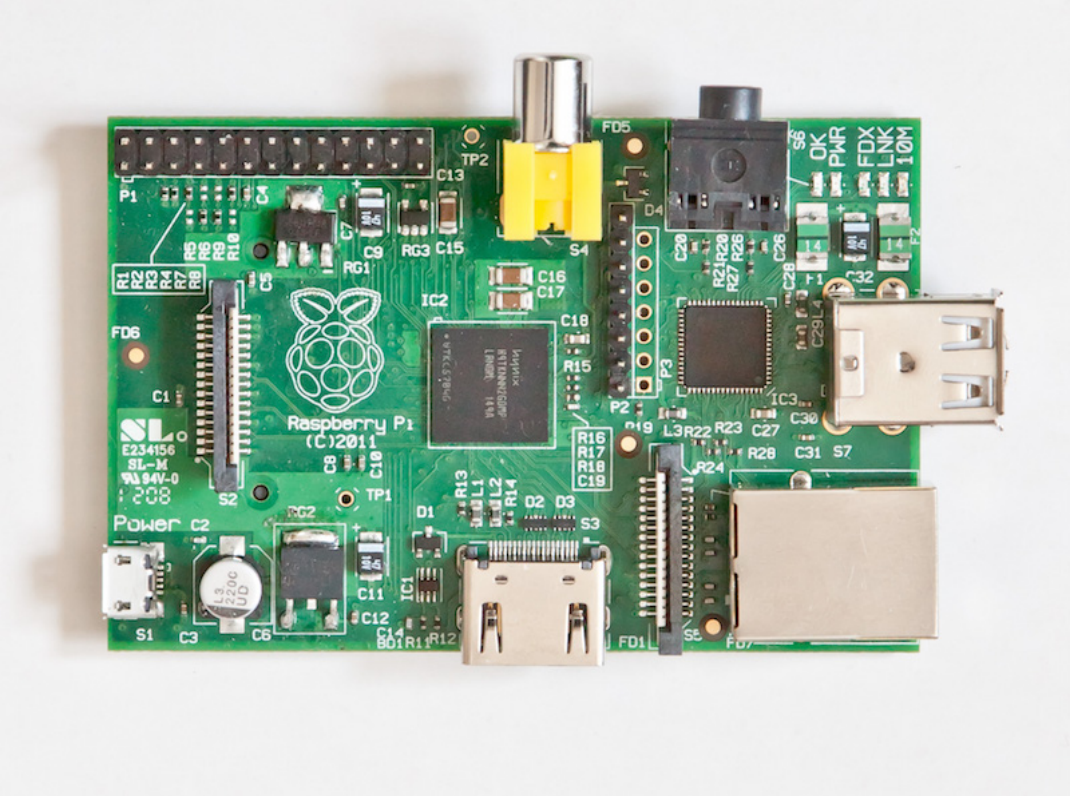
用这个库我们可以做些什么？看看官网的示例:

```
>>> import requests
>>> r = requests.get('https://github.com/timeline.json')
>>> r.json()
```

到现在你会发现我们没有说到任何的Python语法，这不是一本关于Python语法的书，如我们在开头所说的。下面是我们将会在后面用到的代码

```
#!/usr/bin/env python
import requests

url = "http://b.phodal.com/athome/1"
r = requests.get(url)
print r.text
```

9 Raspberry Pi



Raspberry Pi

9.1 Geek的盛宴

Raspberry Pi是一款针对电脑业余爱好者、教师、小学生以及小型企业等用户的迷你电脑，预装Linux系统，体积仅信用卡大小，搭载ARM架构处理器，运算性能和智能手机相仿。在接口方面，Raspberry Pi提供了可供键鼠使用的USB接口，此外还有千兆以太网接口、SD卡扩展接口以及1个HDMI高清视频输出接口，可与显示器或者TV相连。

Linux是一套免费使用和自由传播的类Unix操作系统，是一个基于POSIX和UNIX的多用户、多任务、支持多线程和多CPU的操作系统。它能运行主要的UNIX工具软件、应用程序和网络协议。它支持32位和64位硬件。Linux继承了Unix以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。

Raspberry Pi相比于一般的ARM开发板来说，由于其本身搭载着Linux操作系统，可以用诸如Python、Ruby或Bash来执行脚本，而不是通过编译程序来运行，具有更高的开发效率。

9.2 Raspberry Pi 初始化

今天的Raspbian默认已经安装`openssh-server`，并默认开启了OpenSSH-Server。

接着我们就可以看到系统启动了，要我们输入用户名和密码

```
Raspbian GNU/Linux 7 raspberrypi ttyAMA0

raspberrypi login: pi
Password:
Last login: Sat Apr 26 05:58:07 UTC 2014 on ttyAMA0
Linux raspberrypi 3.10.25+ #622 PREEMPT Fri Jan 3 18:41:00 GMT 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
ls

NOTICE: the software on this Raspberry Pi has not been fully configured. Please run 'sudo raspi-config'
```

然后

```
sudo raspi-config
```

选择第一个，下面就可以继续了

```
Expand Filesystem          Ensures that all of the SD card s
```

接着重启后，便可以扩展SD卡成功。

注: Raspbian与一般的Debian系统使用起来区别不是太大(ps:命令上)，由于CPU是不同的架构，在编译上可能有所区别。通常PC上的软件需要重新编译才能在RPi上运行，所以如果可以用apt-get安装的话，就不要自己编译了。

9.3 Raspberry Pi GPIO

General Purpose Input Output （通用输入/输出）简称为GPIO，或总线扩展器，利用工业标准I2C、SMBus或SPI接口简化了I/O口的扩展。当微控制器或芯片组没有足够的I/O端口，或当系统需要采用远端串行通信或控制时，GPIO产品能够提供额外的控制和监视功能。

GPIO

10 Server 一切皆为服务

10.1 服务器

服务器（Server）指：

- 一个管理资源并为用户提供服务的计算机软件，通常分为文件服务器（能使用户在其它计算机访问文件），数据库服务器和应用程序服务器。
- 运行以上软件的计算机，或称为网络主机（Host）。
- 一般来说，服务器通过网络对外提供服务。可以通过Intranet对内网提供服务，也可以通过Internet对外提供服务。

10.2 Web服务器

WEB服务器也称为WWW(WORLD WIDE WEB)服务器，主要功能是提供网上信息浏览服务。 WWW 是 Internet的多媒体信息查询工具，是 Internet 上近年才发展起来的服务，也是发展最快和目前用的最广泛的服务。正是因为有了WWW工具，才使得近年来 Internet 迅速发展，且用户数量飞速增长。

10.3 LNMP

LNMP

Linux+Nginx+MySQL+PHP

> Nginx (“engine x”) 是一个高性能的 HTTP 和 反向代理 服务器，也是一个 IMAP/POP3/SMTP 代理服务器。 Nginx 是由 Igor Sysoev 为俄罗斯访问量第二的 Rambler.ru站点开发的，第一个公开版本0.1.0发布于2004年10月4日。其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。2011年6月1日，nginx 1.0.4发布。

> MySQL 是一个关系型数据库管理系统，由瑞典MySQL AB公司开发，目前属于Oracle公司。MySQL是最流行的关系型数据库管理系统，在WEB应用方面MySQL是最好的RDBMS(Relational Database Management System：关系数据库管理系统)应用软件之一。MySQL是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

> PHP于1994年由Rasmus Lerdorf创建，刚刚开始是Rasmus Lerdorf为了要维护个人网页而制作的一个简单的用Perl语言编写的程序。这些工具程序用来显示 Rasmus Lerdorf 的个人履历，以及统计网页流量。后来又用C语言重新编写，包括可以访问数据库。他将这些程序和一些表单直译器整合起来，称为 PHP/FI。PHP/FI 可以和数据库连接，产生简单的动态网页程序。

11 Web服务

Web服务是一种服务导向架构的技术，通过标准的Web协议提供服务，目的是保证不同平台的应用服务可以互操作。

根据W3C的定义，Web服务（Web service）应当是一个软件系统，用以支持网络间不同机器的互动操作。网络服务通常是许多应用程序接口（API）所组成的，它们透过网络，例如国际互联网（Internet）的远程服务器端，执行客户所提交服务的请求。

尽管W3C的定义涵盖诸多相异且无法介分的系统，不过通常我们指有关于主从式架构（Client-server）之间根据SOAP协议进行传递XML格式消息。无论定义还是实现，WEB服务过程中会由服务器提供一个机器可读的描述（通常基于WSDL）以辨识服务器所提供的WEB服务。另外，虽然WSDL不是SOAP服务端点的必要条件，但目前基于Java的主流WEB服务开发框架往往需要WSDL实现客户端的源代码生成。一些工业标准化组织，比如WS-I，就在WEB服务定义中强制包含SOAP和WSDL。

WEB服务实际上是一组工具，并有多种不同的方法调用之。三种最普遍的手段是：

- 远程过程调用（RPC）
- 面向服务架构（SOA）
- 表述性状态转移（REST）。

11.1 SOAP VS RESTful

简单对象访问协议是交换数据的一种协议规范，使用在计算机网络Web服务中，交换带结构信息。SOAP为了简化网页服务器从XML数据库中提取数据时，节省去格式化页面时间，以及不同应用程序之间按照HTTP通信协议，遵从XML格式执行资料互换，使其抽象于语言实现、平台和硬件。

12 HTTP 熟悉&陌生

12.1 你所没有深入的HTTP

Internet有两个核心协议: IP和TCP，这样讲述起来似乎会很漫长。

基本概念

> 超文本传输协议 (HTTP-Hypertext transfer protocol) 是一种详细规定了浏览器和万维网服务器之间互相通信的规则，通过因特网传送万维网文档的数据传送协议。

- HTTP是用于客户端与服务端之间的通信。
- 传输层的TCP是基于网络层的IP协议的,而应用层的HTTP协议又是基于传输层的TCP协议的。

注意：HTTP协议只规定了客户端与服务端的通信规则，而没有规定其通讯协议，只是现在的大部分实现都是将TCP作为通讯协议。

12.1.1 打开网页时发生了什么

简单地来说，当我们在浏览器上输入URL的敲下回车的时候。

- 浏览器需要查找域名[2]的IP，从不同的缓存直至DNS服务器。
- 浏览器会给web服务器发送一个HTTP请求
- 服务器“处理”请求
- 服务器发回一个HTML响应
- 浏览器渲染HTML到页面。

在StackOverflow上有一个这样的回答会比较详细。

- browser checks cache; if requested object is in cache and is fresh, skip to #9
- browser asks OS for server's IP address
- OS makes a DNS lookup and replies the IP address to the browser
- browser opens a TCP connection to server (this step is much more complex with HTTPS)
- browser sends the HTTP request through TCP connection
- browser receives HTTP response and may close the TCP connection, or reuse it for another request
- browser checks if the response is a redirect (3xx result status codes), authorization request (401), error (4xx and 5xx), etc.; these are handled differently from normal responses (2xx)
- if cacheable, response is stored in cache
- browser decodes response (e.g. if it's gzipped)
- browser determines what to do with response (e.g. is it a HTML page, is it an image, is it a sound clip?)
- browser renders response, or offers a download dialog for unrecognized types

忽略一些细节便剩下了

1. 从浏览器输入URL
2. 浏览器找到服务器，服务器返回HTML文档
3. 从对应的服务器下载资源

说说第一步，开始时我们输入的是URI(统一资源标识符,Uniform Resource Identifier)，它还有另外一个名字叫统一资源定位器(URL[3],Uniform Resource Locator)。

12.1.2 URL组成

网址算是URL的一个俗称，让我们来看看一个URL的组成，以HTTP版IOT中的URL为例。

```
http://b.phodal.com/athome/1
```

开始之前，我们需要标出URL的80端口以及json文件的全称，那么上面的网址就会变成

```
http://b.phodal.com:80/athome/1.json
```

那么对于这个URL的就有下面几部分组成

- `http://` http说的是这个URL用的是HTTP协议，至于`//`是一个分隔符，用法和C语言中的`;`一样。这样的协议还可以是coap,https,ftp等等。
- `b` 是子域名，一个域名在允许的情况下可以有不限数量的子域名。
- `phodal.com` 代表了一个URL是phodal.com下面的域名
- `80` 80是指80端口，默认的都是80，对于一个不是80端口的URL应该是这样的`http://iot-coap.phodal.com:8896/`
- `athome` 指的是虚拟目录部分，或者文件路径
- `1.json`看上去就是一个文件名，然而也代表着这是一个资源。

对就一个稍微复杂点的例子就是

```
http://ebook.designiot.cn/#你所没有深入的http
```

这里的`#`后面是锚部分，如果你打开这个URL就会发现会直接跳转到相应的锚部分，对就于下面这样的一个例子来说

```
http://www.phodal.com/search/?q=iot&type=blog
```

`?`后面的`q=iot&type=blog`的部分是参数部分，通常用于查询或者、搜索。

12.2 一次HTTP GET请求

当我们打开最小物联网系统的一个页面时，如http://b.phodal.com/athome/1.json

我们在浏览器上看到的结果是

```
[
  {
    "id": 1,
    "temperature": 19,
    "sensors1": 31,
    "sensors2": 7.5,
    "led1": 0
  }
]
```

只是我们看到的是结果，忽略了这其中的过程，于是我们用curl[4]命令来看看详细的情况。

```
curl -I -s http://b.phodal.com/athome/1.json
```

出于某种原因考虑，删去了其中一些元素，剩下下面这些。

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Fri, 05 Sep 2014 15:05:49 GMT

[{"id":1,"temperature":19,"sensors1":31,"sensors2":7.5,"led1":0}]
```

我们用curl命令向服务器发起了GET请求，服务器返回了上面的结果。

12.2.1 HTTP响应

一个HTTP响应由三部分组成

- 状态行(状态码)
- 消息报头(响应报头)
- 响应正文(消息体)

12.2.1.1 HTTP响应 状态码

在上面的结果中，状态行是

```
HTTP/1.1 200 OK
```

返回的状态码是200，OK是状态码的原因短语。

如果是一个跳转的页面，它就可能是下面的结果:

```
HTTP/1.0 301 MOVED PERMANENTLY
Date: Mon, 08 Sep 2014 12:04:01 GMT
Content-Type: text/html; charset=utf-8
```

HTTP Status有五种状态，而这五种状态又有所细分，提一下这五种状态，详细可参见
http://zh.wikipedia.org/wiki/HTTP%E7%8A%B6%E6%80%81%E7%A0%81

- 1xx消息
- 2xx成功
- 3xx重定向
- 4xx客户端错误
- 5xx服务器错误

如

- 200 ok - 成功返回状态，对应，GET,PUT,PATCH,DELETE.
- 201 created - 成功创建。
- 304 not modified - HTTP缓存有效。
- 400 bad request - 请求格式错误。
- 401 unauthorized - 未授权。
- 403 forbidden - 鉴权成功，但是该用户没有权限。
- 404 not found - 请求的资源不存在
- 405 method not allowed - 该http方法不被允许。
- 410 gone - 这个url对应的资源现在不可用。
- 415 unsupported media type - 请求类型错误。
- 422 unprocessable entity - 校验错误时用。

- 429 too many request - 请求过多。

HTTP Method	Operation Performed

12.2.1.2 HTTP响应 响应报头

在这次响应中，返回了两个报头，即

```
Content-Type: application/json
Date: Fri, 05 Sep 2014 15:05:49 GMT
```

Content-Type和Date，在这里的Context-Type是application/json，而通常情况下我们打开一个网站时，他的Content-Type应该是text/html。

```
Content-Type: text/html;
```

Content-Type是最重要的报头。

12.2.1.3 HTTP响应 响应正文

正文才是我们真正想要的内容，上面的都是写给浏览器看的，一般的人不会去关注这些。

```
HTTP/1.1 200 OK
Server: phodal.com/0.17.5
Content-Type: application/json

[{"id":1,"temperature":19,"sensors1":31,"sensors2":7.5,"led1":0}]
```

通常这是以某种格式写的，在这里是以JSON写的，而对于一个网站的时候则是HTML，如:

```
<html>
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

那么这次GET请求返回的就是:

```
HTTP/1.0 200 OK
Date: Mon, 08 Sep 2014 12:04:01 GMT
Content-Type: text/html; charset=utf-8

<html>
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    [{"id":1,"temperature":19,"sensors1":31,"sensors2":7.5,"led1":0}]
</body>
</html>
```

虽然与第一次请求的结果在游览器上看似乎是一样的(ps:可能有微小的差异)，然而其本质是不同的。

推荐及参考书目：

- 《Web性能权威指南》
- 《图解HTTP》
- 《RESTful Web Services Cookbook》
- 《RESTful Web APIs》

13 设计RESTful API

> REST从资源的角度来观察整个网络，分布在各处的资源由URI确定，而客户端的应用通过URI来获取资源的表征。获得这些表征致使这些应用程序转变了其状态。随着不断获取资源的表征，客户端应用不断地在转变着其状态，所谓表征状态转移。

因为我们需要的是一个Machine到Machine沟通的平台，需要设计一个API。而设计一个API来说，RESTful是很不错的一种选择，也是主流的选择。而设计一个RESTful服务，的首要步骤便是设计资源模型。

13.0.1 资源

互联网上的一切信息都可以看作是一种资源。

HTTP Method	Operation Performed

HTTP Method	Operation Performed
GET	Get a resource (Read a resource)
POST	Create a resource
PUT	Update a resource
DELETE	Delete Resource

13.1 设计RESTful API

设计RESTful API是一个有意思的话题。下面是一些常用的RESTful设计原则:

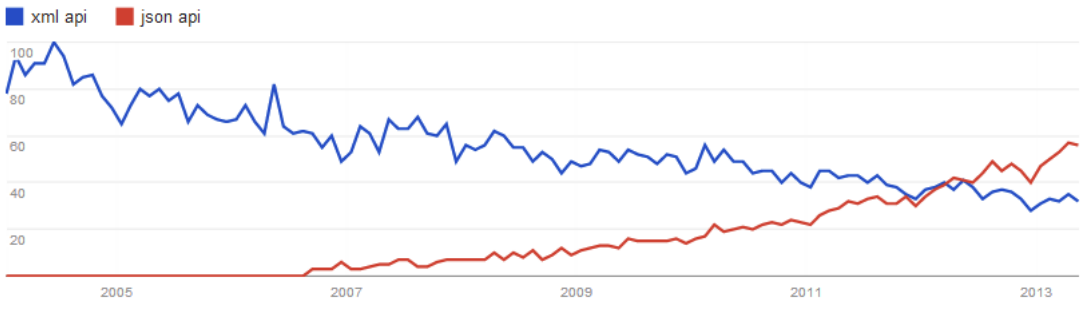
13.2 REST关键目标

- 组件间交互的可伸缩性
- 接口的通用性
- 组件的独立部署
- 通过中间组件来减少延迟、实施安全策略和封装已有系统

13.3 判断是否是 RESTful的约束条件

- 客户端-服务器分离
- 无状态
- 可缓存
- 多层系统
- 统一接口
- 随需代码（可选）

13.4 JSON



xml-vs-json

14 环境准备

14.1 Laravel

Laravel是一套简洁、优雅的PHP Web开发框架(PHP Web Framework)。它可以让你从面条一样杂乱的代码中解脱出来；它可以帮你构建一个完美的网络APP，而且每行代码都可以简洁、富于表达力。

- RESTful 路由: 通过简单的闭包就能响应HTTP请求。帮你快速开始构建非凡的应用。
- 强大的数据操纵能力: Laravel自带了强大的Eloquent ORM 和迁移工具。能够完美的与MySQL、Postgres、SQL Server 和 SQLite协同工作。
- 优雅的模版引擎: PHP代码或轻量级的Blade模版引擎都可无缝融合。Blade模版可以继承，并且拥有极快的解析速度。相信你会喜欢它的。
- 为明天做准备: 构建大型的企业级应用或者只是提供简单的JSON API；书写强大的控制器或轻巧的RESTful路由，Laravel适应所有级别的开发工作。
- 可靠的基石: Laravel 的基石是数个Symfony组件，这些经过千锤百炼、可靠的组件为你的应用提供坚实的基础。
- 基于Composer管理器: Composer 是一套帮你管理第三方扩展包的工具。能够让你迅速在 Packagist 中找到需要的扩展包。

- 强大的社区支持: 无论你是一个PHP新手还是经验丰富的架构师，都能在社区中找到需要的知识。你可以在IRC中讨论Idea，或者在论坛中发布问题。
- 测试、重构: Laravel 从开始就将测试作为重点功能。我们提供了灵活的IoC容器，集成了PHPUnit 测试工具。不用担心，这些都很容易上手。

14.1.1 为什么是 Laravel

- 因为个人喜爱，你也可以用 Ruby On Rails来搭建这样一个功能，或者是Java。
- PHP在我的服务器上运行得挺不错的，而且我又不需要重新去写配置那些配置。
- Laravel 可以简单的开发我们所需要的功能，换句话说他是 PHP 世界的 Ruby On Rails。

这里不会再重述之前的问题，这里只是将需要的步骤一个个写下来，然后丢到这里好好说一下。至于RESTful是什么，前面已经介绍了，就不再重复了。那么下面，我们就用Laravel来搭建一个平台给物联网用的。

14.2 安装 Laravel

14.2.1 GNU/Linux安装Composer

GNU/Linux Ubuntu/OpenSUSE下可以执行

```
$ curl -sS https://getcomposer.org/installer | php
```

14.2.1.1 Windows安装Composer

请直接下载

Composer-Setup

14.2.1.2 Mac OS

1.安装Composer

```
brew install homebrew/php/composer
```

2.安装Laravel

```
composer global require "laravel/installer=~1.1"
```

3.创建Laravel工程

```
composer create-project laravel/laravel your-project-name --prefer-dist
```

14.2.1.3 Mac OS

1.下载laravel.phar

```
wget http://laravel.com/laravel.phar
```

2.重命名

```
mv laravel.phar laravel
```

3.移动到bin中

```
sudo mv laravel /usr/local/bin
```

4.创建项目

```
laravel new blog
```

14.3 MySQL

14.3.1 安装MySQL

```
出于某些原因，我建议用MariaDB替换MySQL，如果你"真正"需要mysql，将mariadb替换为mysql
```

ps: 在下文中我会继续用MySQL，而不是MariaDB，MairaDB是MySQL的一个分支，真正的开源分支。

Ubuntu/Debian/Mint

```
$ sudo apt-get install mariadb-server
```

Fedora/Centos

```
$ sudo yum install mariadb-server
```

openSUSE

```
$ sudo zypper install mariadb-server
```

Mac OS

```
$ brew install mariadb
```

14.3.2 配置MySQL

修改database.php

```
app/config/database.php
```

要修改的就是这个

```
'mysql' => array(
    'driver'    => 'mysql',
    'host'      => 'localhost',
    'database'  => 'iot',
    'username'  => 'root',
    'password'  => '940217',
    'charset'   => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix'    => '',
),
```

如果你已经有phpmyadmin，似乎对你来说已经很简单了，如果没有的话，就直接用

```
$ mysql -uroot -p
```

来创建一个新的

```
CREATE DATABASE IF NOT EXISTS iot default charset utf8 COLLATE utf8_general_ci;
```

数据库的目的在于存储数据等等的闲话这里就不多说了，创建一个RESTful的目的在于产生下面的JSON格式数据，以便于我们在Android、Java、Python、jQuery等语言框架或者平台上可以调用，最主要的是可以直接用Ajax来产生更炫目的效果。

```
{
"id": 1,
"temperature": 14,
"sensors1": 12,
"sensors2": 12,
"led1": 0
}
```

15 创建REST服务

15.1 数据库迁移

这个名字是源自于Ruby On Rails在那时候的印象，不直接使用MySQL的目的在于让我们可以专注于过程。

15.1.1 创建表

表的概念，类似于在Excel中的表，如果你真实不懂数据库。 让我们创建一个athomes的表，为什么是athomes，因为以前在写android程序的时候就叫的是athome，忽略掉这些次要的因素吧。

```
$ php artisan migrate:make create_athomes_table
```

打开 app/database/migrations/***create_athomes_table.php***这里的是由日期和某些东西组成的，修改生成的代码为下面。

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateAthomesTable extends Migration {
    public function up()
    {
        Schema::create('athomes', function(Blueprint $table)
        {
            $table->increments('id');
            $table->float('temperature');
            $table->float('sensors1');
            $table->float('sensors2');
            $table->boolean('led1');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::drop('athomes');
    }
}
```

意思大致就是id是自加的，也就是我们在localhost/athome/{id}，当我们创建一个新的数据的时候，会自动加上去，最后一个timestamps批的是时间，会包含创建时间和修改时间。 剩下的temperature,sensors1,sensors2是小数，以及只有真和假的led1。

Verb	Path	Action	Route Name

15.1.2 数据库迁移

我们只是写了我们需要的数据的格式而并没有丢到数据库里，

```
$ php artisan migrate
```

这个就是我们执行迁移的命令，如果你用phpmyadmin可以直接打开查看，没有的话，可以。

```
$ mysql -uroot -p
use iot;
select * from athomes;
```

就可以看到我们写的东西，那么接下来就是创建RESTful服务了

15.2 创建RESTful

用下面的代码实现我们称之为Athomes控制器的创建

```
$ php artisan controller:make AthomesController
```

就会在app/controllers下面生成下面的代码

```
class AthomesController extends \BaseController {
    public function index() {}
    public function create()  {}
    public function store() {}
    public function show($id) {}
    public function edit($id) {}
    public function update($id) {}
    public function destroy($id) {}
}
```

15.3 Laravel Resources

上面的代码过于沉重，请让我用 Ctrl+C 来带来点知识吧。

Verb	Path	Action	Route Name
GET	/resource	index	resource.index
GET	/resource/create	create	resource.create
POST	/resource	store	resource.store
GET	/resource/{resource}	show	resource.show
GET	/resource/{resource}/edit	edit	resource.edit
PUT/PATCH	/resource/{resource}	update	resource.update
DELETE	/resource/{resource}	destroy	resource.destroy

所以我们只需要专注于创建 create, edit, show, destory 等等。好吧，你可能没有耐心了，但是在修改这个之前我们需要先在app/model 加个 class

```
class Athomes extends Eloquent {
    protected $table = 'athomes';
}
```

如果你想要的只是控制器Athomes的代码的话。。

```
class AthomesController extends \BaseController {
    public $restful=true;
    protected $athome;
    public function __construct(Athomes $athome)
    {
        $this-->athome = $athome ;
     }
    public function index()
    {
        $maxid=Athomes::all();
        return Response::json($maxid);
    }
    public function create()
    {
        $maxid=Athomes::max('id');
        return View::make('athome.create')->with('maxid',$maxid);
    }
```

```
    public function store()
    {
        $rules = array(
            'led1'=>'required',
            'sensors1' => 'required|numeric|Min:-50|Max:80',
            'sensors2' => 'required|numeric|Min:-50|Max:80',
            'temperature' => 'required|numeric|Min:-50|Max:80'
        );
        $validator = Validator::make(Input::all(), $rules);
        if ($validator->fails()) {
            return Redirect::to('athome/create')
                ->withErrors($validator)
                ->withInput(Input::except('password'));
        } else {
            $nerd = new Athomes;
            $nerd->sensors1       = Input::get('sensors1');
            $nerd->sensors2       = Input::get('sensors2');
            $nerd->temperature    = Input::get('temperature');
            $nerd->led1           = Input::get('led1');
            $nerd->save();
            Session::flash('message', 'Successfully created athome!');
            return Redirect::to('athome');
        }
    }
    public function show($id)
    {
        $myid=Athomes::find($id);
        $maxid=Athomes::where('id','=',$id)
                        ->select('id','temperature','sensors1','sensors2','led1')
                        ->get();
        return Response::json($maxid);
    }
    public function edit($id)
    {
        $athome = Athomes::find($id);
        return View::make('athome.edit')
            ->with('athome', $athome);
    }
    public function update($id)
    {
        $rules = array(
            'led1'=>'required|',
            'sensors1' => 'required|numeric|Min:-50|Max:80',
            'sensors2' => 'required|numeric|Min:-50|Max:80',
            'temperature' => 'required|numeric|Min:-50|Max:80'
        );
        $validator = Validator::make(Input::all(), $rules);
        if ($validator->fails()) {
            return Redirect::to('athome/' . $id . '/edit')
                ->withErrors($validator);
        } else {
            $nerd = Athomes::find($id);
            $nerd->sensors1       = Input::get('sensors1');
            $nerd->sensors2       = Input::get('sensors2');
            $nerd->temperature    = Input::get('temperature');
            $nerd->led1           = Input::get('led1');
            $nerd->save();
            Session::flash('message', 'Successfully created athome!');
            return Redirect::to('athome');
        }
    }
    public function destroy($id)
    {
        $athome = Athomes::find($id);
        $athome->delete();
        if(is_null($athome))
        {
             return Response::json('Todo not found', 404);
        }
        Session::flash('message', 'Successfully deleted the nerd!');
        return Redirect::to('athome');
    }
}
```

希望你能读懂，没有的话，继续。

下面这部分来自于之前的博客，这里就不多加论述了。 这个也就是我们要的模板，

15.3.1 修改Create()

```
 public function create()
{
    $maxid=Athomes::max('id');
    return View::make('athome.create')->with('maxid',$maxid);
}
```

这里需要在app/views/创建一个athome里面创建一个create.blade.php，至于maxid，暂时还不需要，后面会用到show。如果只需要模板，可以简化为

```
public function create()
{
    return View::make('athome.create');
}
```

这里只是对其中代码的进行一下说明。

15.3.2 创建表单

15.3.2.1 创建表单之前

由于使用到了bootstrap以及bootstrap-select，记得添加css。

```
<link rel="stylesheet" type="text/css" href="<?= url('css/bootstrap.min.css') ?>" />
<link rel="stylesheet" type="text/css" href="<?= url('css/bootstrap-select.min.css') ?>" />
```

以及javascript

```
<script type="text/javascript" src="<?= url('js/jquery.min.js')?>"></script>
<script type="text/javascript" src="<?= url('js/bootstrap.min.js') ?>"></script>
<script type="text/javascript" src="<?= url('js/bootstrap-select.min.js') ?>"></script>
<script>
$('.selectpicker').selectpicker();
</script>
```

15.3.2.2 创建表单

这里用到的是之前提到的那个作者写下的，稍微修改了一下。

```
<div class="row-fluid">
  {{ HTML::ul($errors->all()) }}
  {{ Form::open(array('url' => 'athome')) }}
     <div class="form-group">
          {{ Form::label('led1', '开关1') }}
          {{ Form::select('led1',array('关','开'),$selected=NULL,array('class'=>'selectpicker')) }}
      </div>
      <div class="form-group">
          {{ Form::label('sensors1', 'sensors1') }}
          {{ Form::text('sensors1', Input::old('sensors1'), array('class' => 'form-control')) }}
      </div>
      <div class="form-group">
          {{ Form::label('sensors2', 'sensors2') }}
          {{ Form::text('sensors2', Input::old('sensors2'), array('class' => 'form-control')) }}
      </div>
      <div class="form-group">
          {{ Form::label('temperature', 'temperature') }}
          {{ Form::text('temperature', Input::old('temperature'), array('class' => 'form-control')) }}
      </div>
      {{ Form::submit('Create!', array('class' => 'btn btn-primary')) }}
  {{ Form::close() }}
  </div>
```

开关一开始打算用 checkbox，加上 bootstrap-switch 实现 ON OFF 弱弱地觉得还是没掌握好的节奏，所以最后用 select 来实现。

还需要修改一下之前的 create()，添加一行

```
return Redirect::to('athome');
```

也就是添加完后，重定向到首页查看，最后例子给出的 create 如下

```
public function store()
{
    $rules = array(
        'led1'=>'required',
        'sensors1' => 'required|numeric|Min:-50|Max:80',
        'sensors2' => 'required|numeric|Min:-50|Max:80',
        'temperature' => 'required|numeric|Min:-50|Max:80'
    );
    $validator = Validator::make(Input::all(), $rules);
    if ($validator->fails()) {
        return Redirect::to('athome/create')
            ->withErrors($validator);
    } else {
        // store
        $nerd = new Athomes;
        $nerd->sensors1       = Input::get('sensors1');
        $nerd->sensors2       = Input::get('sensors2');
        $nerd->temperature    = Input::get('temperature');
        $nerd->led1           = Input::get('led1');
        $nerd->save();
        Session::flash('message', 'Successfully created athome!');
```

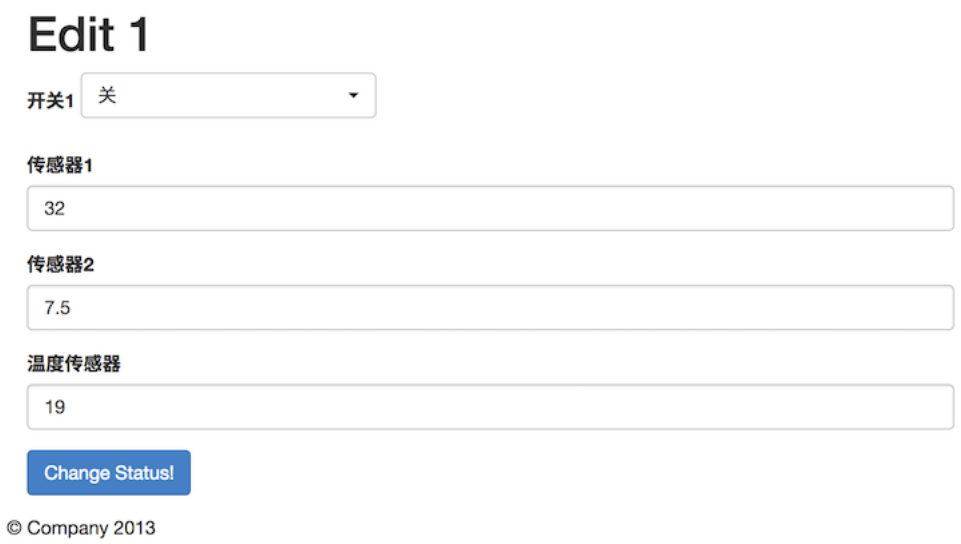
```
        return Redirect::to('athome');
    }
}
```

15.3.3 编辑模板

完整的 blade 模板文件

```
<!DOCTYPE html lang="zh-cn">
<html>
    <head>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <meta name="keywords" content="">
        <meta name="viewport" content="width=device-width">
        <meta name="description" content="">
        <title>@yield('title')</title>
        <link rel="stylesheet" type="text/css" href="<?= url('css/bootstrap.min.css') ?>" />
        <link rel="stylesheet" type="text/css" href="<?= url('css/bootstrap-select.min.css') ?>" />
        <link rel="stylesheet" href="<?= url('css/justified-nav.css') ?>" type="text/css" media="screen" />
    </head>
<body>
<div class="container">
<div class="container">
  <div class="row-fluid">
<h1>Edit {{ $athome->id }}</h1>
<!-- if there are creation errors, they will show here -->
{{ HTML::ul($errors->all()) }}
{{ Form::model($athome, array('route' => array('athome.update', $athome->id), 'method' => 'PUT')) }}
        <div class="form-group">
            {{ Form::label('led1', '开关1') }}
            {{ Form::select('led1',array('关','开'),$selected=NULL,array('class'=>'selectpicker')) }}
        </div>
        <div class="form-group">
            {{ Form::label('sensors1', '传感器1') }}
            {{ Form::text('sensors1', Input::old('sensors1'), array('class' => 'form-control')) }}
        </div>
        <div class="form-group">
            {{ Form::label('sensors2', '传感器2') }}
            {{ Form::text('sensors2', Input::old('sensors2'), array('class' => 'form-control')) }}
        </div>
        <div class="form-group">
            {{ Form::label('temperature', '温度传感器') }}
            {{ Form::text('temperature', Input::old('temperature'), array('class' => 'form-control')) }}
        </div>
    {{ Form::submit('Edit the Nerd!', array('class' => 'btn btn-primary')) }}
{{ Form::close() }}
    </div>
</div>
<div class="footer">
        <p>© Company 2013</p>
      </div>
</div>
</div>
<script type="text/javascript" src="<?= url('js/jquery.min.js')?>"></script>
<script type="text/javascript" src="<?= url('js/bootstrap.min.js') ?>"></script>
<script type="text/javascript" src="<?= url('js/bootstrap-select.min.js') ?>"></script>
<script>
 $('.selectpicker').selectpicker();
 </script>
<script type="text/javascript" src="<?= url('js/log.js') ?>"></script>
</body>
</html>
```

效果图:



Blade Edit

最后效果见:http://b.phodal.com/

16 前端显示

16.1 库与车轮子

在多数的情况下我们都没有理由也没有必要去重新发明我们的车轮，在这时使用库会是一个比较好的做法。

16.2 库

16.2.1 jQuery

> Jquery是继prototype之后又一个优秀的Javascript库。它是轻量级的js库 ，它兼容CSS3，还兼容各种浏览器（IE 6.0+, FF 1.5+, Safari 2.0+, Opera 9.0+），jQuery2.0及后续版本将不再支持IE6/7/8浏览器。jQuery使用户能更方便地处理HTML（标准通用标记语言下的一个应用）、events、实现动画效果，并且方便地为网站提供AJAX交互。jQuery还有一个比较大的优势是，它的文档说明很全，而且各种应用也说得很详细，同时还有许多成熟的插件可供选择。jQuery能够使用户的html页面保持代码和html内容分离，也就是说，不用再在html里面插入一堆js来调用命令了，只需要定义id即可。

在我们的代码里用到了jQuery，然而这是一种简单而且快速有速地方法。

16.2.2 jQuery Mobile

> jQuery Mobile是jQuery 在手机上和平板设备上的版本。jQuery Mobile不仅会给主流移动平台带来jQuery核心库，而且会发布一个完整统一的jQuery移动UI框架。支持全球主流的移动平台。jQuery Mobile开发团队说：能开发这个项目，我们非常兴奋。移动Web太需要一个跨浏览器的框架，让开发人员开发出真正的移动Web网站。

整个展示页面由三部分组成，即`header`,`content`,`footer`。而我们主要需要关心的是content，也就是真下的内容。

我们只需要结合`HighChart`来设计我们的content就可以了，下面是一个简单地`HighChart`示例:

```
 <div id="Tchart" style="min-width:800px;width:100%;height:300px"></div>
</div>
```

剩下的事就由`HighChart`来做。

最后代码如下所示

```
<body>
    <div data-role="page">

        <div data-role="header" class="ui-bar ui-bar-b" class="jqm-header">
            <h1>基础控制</h1>
        </div>
```

```
        <div data-role="collapsible" data-collapsed="false">
            <h3>查看温度情况</h3>
        <div>
        <div id="Tchart" style="min-width:800px;width:100%;height:300px"></div>
        </div>
    </div>
    <div data-role="footer" class="ui-bar ui-bar-b" data-position="fixed">
        <h4><a href="http://www.phodal.com">Power by Phodal</a></h4>
    </div>
</body>
```

可以看到上面的代码与一般的HTML不同的地方是`data-role`,`data-collapsed`,`data-position`，而这些是jQuery Mobile所拥有的data属性。

data-role参数表： rold丨详细 ———–丨–- page丨页面容器，其内部的mobile元素将会继承这个容器上所设置的属性 header丨页面标题容器，这个容器内部可以包含文字、返回按钮、功能按钮等元素 footer丨页面页脚容器，这个容器内部也可以包含文字、返回按钮、功能按钮等元素 content丨页面内容容器，这是一个很宽容的容器，内部可以包含标准的html元素和jQueryMobile元素 controlgroup丨将几个元素设置成一组，一般是几个相同的元素类型 fieldcontain丨区域包裹容器，用增加边距和分割线的方式将容器内的元素和容器外的元素明显分隔 navbar丨功能导航容器，通俗的讲就是工具条 listview丨列表展示容器，类似手机中联系人列表的展示方式 list-divider丨列表展示容器的表头，用来展示一组列表的标题，内部不可包含链接 button丨按钮，将链接和普通按钮的样式设置成为jQueryMobile的风格 none丨阻止框架对元素进行渲染，使元素以html原生的状态显示，主要用于form元素。

同上，我们也可以在网上找到其他相对就的属性。

16.3 网站前台显示

16.3.1 Highcharts

Highcharts有以下的特点

- 兼容性：兼容当今所有的浏览器，包括 iPhone、IE 和火狐等等；
- 对个人用户完全免费；
- 纯JS，无BS；
- 支持大部分的图表类型：直线图，曲线图、区域图、区域曲线图、柱状图、饼装图、散布图；
- 跨语言：不管是 PHP、Asp.net 还是 Java 都可以使用，它只需要三个文件：一个是Highcharts 的核心文件 highcharts.js，还有 a canvas emulator for IE 和 Jquery类库或者 MooTools 类库；
- 提示功能：鼠标移动到图表的某一点上有提示信息；
- 放大功能：选中图表部分放大，近距离观察图表；
- 易用性：无需要特殊的开发技能，只需要设置一下选项就可以制作适合自己的图表；
- 时间轴：可以精确到毫秒;

在这里只需将需要处理的数据存储到数组中，便可以将其渲染成为图形，下面的温度走势图便是基于Highcharts的结果:

Temperture

先看看最后代码如下所示:

```
var dataLength = [];

function drawTemp() {
    var zero = [];
    $.getJSON('/athome/', function(json) {
        var items = [];
        dataLength.push(json.length);
        $.each(json, function(key, val) {
            zero.push(val.temperature);
```

```
        });
        chart = new Highcharts.Chart({
            color: {
                linearGradient: {
                    x1: 0,
                    x2: 0,
                    y1: 0,
                    y1: 1
                },
                stops: [
                    [0, '#003399'],
                    [1, '#3366AA']
                ]
            },
            chart: {
                renderTo: 'Tchart',
                type: 'spline'
            },
            title: {
                text: '本月温度情况'
            },
            subtitle: {
                text: ''
            },
            xAxis: {
                categories: [],
                title: {
                    text: ''
                }
            },
            yAxis: {
                title: {
                    text: '温度 (°C)'
                }
            },
            tooltip: {
                backgroundColor: '#FCFFC5',
                borderColor: 'black',
                borderRadius: 10,
                borderWidth: 1,
                enabled: true,
                formatter: function() {
                    return '<b>' + this.series.name + '</b><br/>' + this.x + ': ' + this.y + '°C';
                }
            },
            legend: {
                layout: 'vertical',
                align: 'right',
                verticalAlign: 'top',
                x: -10,
                y: 100,
                borderWidth: 0
            },
            plotOptions: {
                line: {
                    dataLabels: {
                        enabled: true
                    },
                    enableMouseTracking: false
                }
            },
            series: [{
                name: '本月',
                data: zero
            }, {
                name: '对比',
                data: [26.0]
            }]
        });
    });
};

function showTemper() {
    var length = dataLength[0];
    $.ajax({
        url: '/athome/' + length,
        type: 'GET',
        dataType: 'json',
        async: true,
        timeout: 1000,
        error: function() {},
        success: function(sdata) {
            $('.temperStatus').empty();
            $('.temperStatus').append(sdata.temperature);
        }
    });
};
```

```
$(document).ready(function() {
    setInterval("drawTemp();", 5000);
    setInterval("showTemper();", 800);
    drawTemp();
    showTemper();
});
```

我们先把`HighChart`部分的代码删去，就变成下面的内容：

```
 var dataLength = [];

function drawTemp() {
    var zero = [];
    $.getJSON('/athome/', function(json) {
        var items = [];
        dataLength.push(json.length);
        $.each(json, function(key, val) {
            zero.push(val.temperature);
        });
    });
};

function showTemper() {
    var length = dataLength[0];
    $.ajax({
        url: '/athome/' + length,
        type: 'GET',
        dataType: 'json',
        async: true,
        timeout: 1000,
        error: function() {},
        success: function(sdata) {
            $('.temperStatus').empty();
            $('.temperStatus').append(sdata.temperature);
        }
    });
};

$(document).ready(function() {
    setInterval("drawTemp();", 5000);
    setInterval("showTemper();", 800);
    drawTemp();
    showTemper();
});
```

代码看上去是由两部分组成的`drawTemperature`和`showTemperature`

16.3.2 实时数据

剥离后的Ajax部分代码如下所示，主要用的是jQuery框架的getJSON来实现的

```
 var dataLength = [];
function drawTemp() {
    var zero = [];
    $.getJSON('/athome/', function(json) {
        var items = [];
        dataLength.push(json.length);
        $.each(json, function(key, val) {
            zero.push(val.temperature);
        });
};
```

实际上，我们做的只是从/athome/下面获取数据，再将数据堆到数组里面，再把这部分放到图形中。等等，什么是Ajax?

AJAX = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）。 AJAX 不是新的编程语言，而是一种使用现有标准的新方法。 AJAX 是与服务器交换数据并更新部分网页的艺术，在不重新加载整个页面的情况下。 JSON我们前面也已经了解过了，看看getJSON吧。

jQuery. getJSON

方法定义：jQuery.getJSON(url, data, callback)

通过get请求得到json数据

·url用于提供json数据的地址页 ·data(Optional)用于传送到服务器的键值对 ·callback(Optional)回调函数，json数据请求成功后的处理函数 我想你似乎应该懂得了一点，就是在不刷新网页的同时，用javascript获取数据放到图表上，就这么简单。

17 RESTful的CoAP协议

17.1 CoAP: 嵌入式系统的REST

引自维基百科上的介绍，用的是谷歌翻译。。。

> 受约束的应用协议（COAP）是一种软件协议旨在以非常简单的电子设备，使他们能够在互联网上进行交互式通信中使用。它特别针对小型低功率传感器，开关，阀门和需要被控制或监督远程，通过标准的Internet网络类似的组件。 COAP是一个应用层协议，该协议是用于在资源受限的网络连接设备，例如无线传感器网络节点使用。 COAP被设计为容易地转换为HTTP与Web简化集成，同时也能满足特殊的要求，例如多播支持，非常低的开销，和简单性。多播，低开销，以及简单性是因特网极其重要物联网（IOT）和机器对机器（M2M）设备，这往往是积重难返，有太多的内存和电源，比传统的互联网设备有。因此，效率是非常重要的。 COAP可以在支持UDP或UDP的模拟大多数设备上运行。

简单地来说，CoAP是简化了HTTP协议的RESTful API，因而也只提供了REST的四个方法，即PUT,GET,POST和DELETE。对于微小的资源受限，在资源受限的通信的IP的网络，HTTP不是一种可行的选择。它占用了太多的资源和太多的带宽。而对于物联网这种嵌入式设备来说，关于资源与带宽，是我们需要优先考虑的内容。

- CoAP采用了二进制报头，而不是文本报头(text header)
- CoAP降低了头的可用选项的数量。
- CoAP减少了一些HTTP的方法
- CoAP可以支持检测装置

17.2 CoAP 命令行工具

为了测试测试我们的代码是否是正确工作，我们需要一个CoAP的命令行工具。目前有两个不错的工具可以使用。

- CoAP-cli，一个基于NodeJS的CoAP命令行工具，其核心是基于Node-CoAP库。
- libcooap，一个用C写的CoAP命令行工具。
- FireFox Copper， 一个Firefox的插件。

17.2.1 Node CoAP CLI

安装命令如下

```
npm install coap-cli -g
```

17.2.1.1 CoAP命令行

在coap-cli中，一共有四个方法。分别表示REST的四种不同的方式:

```
Commands:

get                     performs a GET request
put                     performs a PUT request
post                    performs a POST request
delete                  performs a DELETE request
```

在这里，我们用coap://vs0.inf.ethz.ch/来作一个简单的测试

```
coap get coap://vs0.inf.ethz.ch/
(2.05)  ************************************************************
I-D
```

测试一下现在的最小的物联网系统CoAP版

```
coap get coap://iot-coap.phodal.com/id/1
(2.05)  [{"id":1,"value":"is id 1","sensors1":19,"sensors2":20}]
```

17.2.2 libcoap

17.2.2.1 mac os libcoap安装

Mac OS下可以直接用

```
brew install libcoap
```

17.2.2.2 Ubuntu libcoap安装

Ubuntu GNU/Linux下

17.2.2.3 Windows libcoap安装

Windows 下

安装完libcoap，我们可以直接用自带的两个命令

```
coap-client
coap-server
```

1.用coap-server启一个CoAP服务

```
coap-server
```

2.客户端获取数据

```
coap-client -m get coap://localhost
```

返回结果

```
v:1 t:0 tkl:0 c:1 id:37109
This is a test server made with libcoap (see http://libcoap.sf.net)
Copyright (C) 2010--2013 Olaf Bergmann <bergmann@tzi.org>
```

17.2.3 Firefox Copper

为了能访问coap://localhost/，于是我们便需要安装一个Firefox并安装一个名为Copper的插件。

1. 下载地址: https://addons.mozilla.org/en-US/firefox/addon/copper-270430/
2. 作为测试我们同样可以访问 coap://vs0.inf.ethz.ch:5683/

17.3 CoAP Hello,World

接着我们便开始试试做一个简单的CoAP协议的应用:

这里用到的是一个Nodejs的扩展Node-CoAP

> node-coap is a client and server library for CoAP modelled after the http module.

Node-CoAP是一个客户端和服务端的库用于CoAP的模块建模。创建一个package.json文件，添加这个库

```
{
  "dependencies":{
    "coap": "0.7.2"
  }
}
```

接着执行

```
npm install
```

就可以安装好这个库。如果遇到权限问题，请用

```
sudo npm install
```

接着，创建这样一个app.js

```
const coap        = require('coap')
    , server  = coap.createServer()

server.on('request', function(req, res) {
  res.end('Hello ' + req.url.split('/')[1] + '\n')
})

server.listen(function() {
  console.log('server started')
})
```

执行

```
node app.js
```

便可以在浏览器上访问了，因为现在什么也没有，所以什么也不会返回。

接着下来再创建一个client端的js，并运行之

```
const coap  = require('coap')
    , req   = coap.request('coap://localhost/World')

req.on('response', function(res) {
  res.pipe(process.stdout)
})

req.end()
```

就可以在console上输出

```
Hello World
```

也就达到了我们的目的，用CoAP协议创建一个服务，接着我们应该用它创建更多的东西，如产生JSON数据，以及RESTful。和HTTP版的最小物联网系统一样，CoAP版的最小物联网系统也是要返回JSON的。

17.4 CoAP 数据库查询

17.4.1 Node Module

这说里NodeJS Module的意义是因为我们需要在别的地方引用到db_helper这个库，也就是下一小节要的讲的内容。

这样我们就可以在server.js类似于这样去引用这个js库。

```
var DBHelper = require('./db_helper.js');
DBHelper.initDB();
```

而这样调用的前提是我们需要去声明这样的module，为了方便地导出函数功能调用。

```
function DBHelper(){
}
DBHelper.initDB = function(){};
module.exports = DBHelper;
```

17.4.2 Node-Sqlite3

这次我们用的是SQLite3(你可以用MySQL，出于安全考虑用SQLite3，SQLite3产生的是一个文件)。一个简单的initDB函数

```
var db = new sqlite3.Database(config["db_name"]);
var create_table = 'create table if not exists basic (' + config["db_table"] + ');';

db.serialize(function() {
    db.run(create_table);
    _.each(config["init_table"], function(insert_data) {
        db.run(insert_data);
    });
});
db.close();
```

首先从配置中读取db_name，接着创建table，然后调用underscore的each方法，创建几个数据。配置如下所示

```
config = {
    "db_name": "iot.db",
    "db_table": "id integer primary key, value text, sensors1 float, sensors2 float",
    "init_table":[
        "insert or replace into basic (id,value,sensors1,sensors2) VALUES (1, 'is id 1', 19, 20);",
        "insert or replace into basic (id,value,sensors1,sensors2) VALUES (2, 'is id 2', 20, 21);"
    ],
    "query_table":"select * from basic;"
};
```

而之前所提到的url查询所做的事情便是

```
DBHelper.urlQueryData = function (url, callback) {
    var db = new sqlite3.Database("iot.db");
    var result = [];
    console.log("SELECT * FROM basic where " + url.split('/')[1] + "=" + url.split('/')[2]);
    db.all("SELECT * FROM basic where " + url.split('/')[1] + "=" + url.split('/')[2], function(err, rows) {
        db.close();
        callback(JSON.stringify(rows));
    });
};
```

将URL传进来，便解析这个参数，接着再放到数据库中查询，再回调回结果。这样我们就可以构成之前所说的查询功能，而我们所谓的post功能似乎也可以用同样的方法加进去。

17.4.3 查询数据

简单地记录一下在IoT-CoAP中一次获取数据地过程。

先看看在示例中的Get.js的代码，这关乎在后面server端的代码。

```
const coap       = require('coap')
    ,requestURI = 'coap://localhost/'
    ,url        = require('url').parse(requestURI + 'id/1/')
    ,req        = coap.request(url)
    ,bl         = require('bl');

req.setHeader("Accept", "application/json");
req.on('response', function(res) {
  res.pipe(bl(function(err, data) {
    var json = JSON.parse(data);
```

```
    console.log(json);
  }));

});
req.end();
```

const定义数据的方法，和我们在其他语言中有点像。只是这的const主要是为了程序的健壮型,减少程序出错，当然这不是javascript的用法。

我们构建了一个请求的URL

```
coap://localhost/id/1/
```

我们对我们的请求添加了一个Header，内容是Accept，值是'application/json'也就是JSON格式。接着，便是等待请求回来，再处理返回的内容。

判断请求的方法

在这里先把一些无关的代码删除掉，并保证其能工作，so，下面就是简要的逻辑代码。

```
var coap            = require('coap');
var server          = coap.createServer({});
var request_handler = require('./request_handler.js');

server.on('request', function(req, res) {
    switch(req.method){
        case "GET": request_handler.getHandler(req, res);
            break;
    }
});

server.listen(function() {
    console.log('server started');
});
```

创建一个CoAP服务，判断req.method，也就是请求的方法，如果是GET的话，就调用request_handler.getHandler(req, res)。而在getHandler里，判断了下请求的Accept

```
request_helper.getHandler = function(req, res) {
    switch (req.headers['Accept']) {
        case "application/json":
            qh.returnJSON(req, res);
            break;
        case "application/xml":
            qh.returnXML(req, res);
            break;
    }
};
```

如果是json刚调用returnJSON,

Database与回调

而这里为了处理回调函数刚分为了两部分

```
query_helper.returnJSON = function(req, res) {
    DBHelper.urlQueryData(req.url, function (result) {
        QueryData.returnJSON(result, res);
    });
};
```

而这里只是调用了

```
DBHelper.urlQueryData = function (url, callback) {
    var db = new sqlite3.Database(config["db_name"]);
    console.log("SELECT * FROM basic where " + url.split('/')[1] + "=" + url.split('/')[2]);
    db.all("SELECT * FROM basic where " + url.split('/')[1] + "=" + url.split('/')[2], function(err, rows) {
        db.close();
        callback(JSON.stringify(rows));
    });
};
```

这里调用了node sqlite3去查询对应id的数据，用回调处理了数据无法到外部的问题，而上面的returnJSON则只是返回最后的结果，code以及其他的内容。

```
QueryData.returnJSON = function(result, res) {
    if (result.length == 2) {
        res.code = '4.04';
        res.end(JSON.stringify({
            error: "Not Found"
        }));
    } else {
        res.code = '2.05';
        res.end(result);
```

```
    }
};
```

当resulst的结果为空时，返回一个404，因为没有数据。这样我们就构成了整个的链，再一步步返回结果。

在IoT-CoAP中我们使用到了一个Block2的东西，于是便整理相关的一些资料，作一个简单的介绍，以及在代码中的使用。

17.5 CoAP Block

CoAP是一个RESTful传输协议用于受限设备的节点和网络。基本的CoAP消息是一个不错的选择对于小型载荷如

- 温度传感器
- 灯光开关
- 楼宇自动化设备

然而，有时我们的应用需要传输更大的有效载荷，如——更新固件。与HTTP，TCP做繁重工作将大型有效载荷分成多个数据包，并确保他们所有到达并以正确的顺序被处理。

CoAP是同UDP与DLTS一样是基于数据报传输的，这限制了资源表示(resource representation)的最大大小，使得传输不需要太多的分割。虽然UDP支持通过IP分片传输更大的有效载荷，且仅限于64KiB，更重要的是，并没有真正很好地约束应用和网络。

而不是依赖于IP分片，这种规范基本COAP了对“块”选项，用于传输信息从多个资源区块的请求 - 响应对。在许多重要的情况下，阻止使服务器能够真正无状态：服务器可以处理每块分开传输，而无需建立连接以前的数据块传输的其他服务器端内存。

综上所述，块(Block)选项提供了传送一个最小的在分块的方式更大的陈述。

17.5.1 CoAP POST

看看在IoT CoAP中的post示例。

```
const coap     = require('coap')
      ,request  = coap.request
      ,bl       = require('bl')
      ,req = request({hostname: 'localhost',port:5683,pathname: '',method: 'POST'});

req.setOption('Block2',  [new Buffer('1'),new Buffer("'must'"), new Buffer('23'), new Buffer('12')]);
req.setHeader("Accept", "application/json");
req.on('response', function(res) {
    res.pipe(bl(function(err, data) {
        console.log(data);
        process.exit(0);
    }));

});

req.end();
```

Block2中一共有四个数据，相应的数据结果应该是

```
{ name: 'Block2', value: <Buffer 31> }
{ name: 'Block2', value: <Buffer 27 6d 75 73 74 27> }
{ name: 'Block2', value: <Buffer 32 33> }
{ name: 'Block2', value: <Buffer 31 32> }
```

这是没有解析的Block2，简单地可以用

```
_.values(e).toString()
```

将结果转换为

Block2,1 Block2,‘must’ Block2,23 Block2,12

接着按“,”分开，

```
_.values(e).toString().split(',')[1]
```

就有

```
[ '1', '\'must\'', '23', '12' ]
```

便可以很愉快地将其post到数据库中了，

在做IoT-CoAP的过程中只支持JSON，查阅CoAP的草稿时发现支持了诸多的Content Types。

17.5.2 CoAP Content Types

以下文字来自谷歌翻译:

Media type	Encoding	Id.	Reference	Identifier

互联网媒体类型是通过HTTP字符串标识，如“application/xml”。该字符串是由一个顶层的类型“applicaion”和子类型的“XML”。为了尽量减少使用这些类型的媒体类型来表示的开销消息有效载荷，COAP定义一个标识符编码方案互联网媒体类型的子集。预计这桌将可扩展标识符的值的IANA维护。内容类型选项被格式化为一个8位无符号整数。初始映射到一个合适的互联网媒体类型标识符表所示。复合型高层次类型（multipart和不支持消息）。标识符值是从201-255保留的特定于供应商的，应用程序特定的或实验使用和不由IANA。

下面是HTTP的标识符及类型

Internet media type	Identifier
text/plain (UTF-8)	0
text/xml (UTF-8)	1
text/csv (UTF-8)	2
text/html (UTF-8)	3
image/gif	21
image/jpeg	22
image/png	23
image/tiff	24
audio/raw	25
video/raw	26
application/link-format [I-D.ietf-core-link-format]	40
application/xml	41
application/octet-stream	42
application/rdf+xml	43
application/soap+xml	44
application/atom+xml	45
application/xmpp+xml	46
application/exi	47
application/x-bxml	48
application/fastinfoset	49
application/soap+fastinfoset	50
application/json	51

而在CoAP中只有简单地几个

Media type	Encoding	Id.	Reference
text/plain; charset=utf-8	-	0	[RFC2046][RFC3676][RFC5147]
application/link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/octet-stream	-	42	[RFC2045][RFC2046]
application/exi	-	47	[EXIMIME]
application/json	-	50	[RFC4627]

简单地说就是：

`诸如application/json的Content Types在CoAP中应该是50`。如上表所示的结果是其对应的结果，这样的话可以减少传递的信息量。

17.6 CoAP JSON

于是在一开始的时候首先支持的便是“application/json”这样的类型。

首先判断请求的header

```
request_helper.getHandler = function(req, res) {
    switch (req.headers['Accept']) {
        case "application/json":
            qh.returnJSON(req, res);
            break;
        case "application/xml":
            qh.returnXML(req, res);
            break;
    }
};
```

再转至相应的函数处理，而判断的依据则是Accept是不是“application/json”。

```
registerFormat('text/plain', 0)
registerFormat('application/link-format', 40)
registerFormat('application/xml', 41)
registerFormat('application/octet-stream', 42)
registerFormat('application/exi', 47)
registerFormat('application/json', 50)
```

对应地我们需要在一发出请求的时候设置好Accept，要不就没有办法返回我们需要的结果。

```
req.setHeader("Accept", "application/json");
```

返回**JSON**

在给IoT CoAP添加了JSON支持之后，变得非常有意思，至少我们可以获得我们想要的结果。在上一篇中我们介绍了一些常用的工具——CoAP 命令行工具集。

CoAP客户端代码

开始之前我们需要有一个客户端代码，以便我们的服务端可以返回正确的数据并解析

```
var coap = require('coap');
var requestURI = 'coap://localhost/';
var url = require('url').parse(requestURI + 'id/1/');
console.log("Request URL: " + url.href);
var req = coap.request(url);
var bl = require('bl');

req.setHeader("Accept", "application/json");
req.on('response', function(res) {
  res.pipe(bl(function(err, data) {
    var json = JSON.parse(data);
    console.log(json);
  }));

});

req.end();
```

代码有点长内容也有点多，但是核心是这句话：

```
req.setHeader("Accept", "application/json");
```

这样的话，我们只需要在我们的服务端一判断，

```
if(req.headers['Accept'] == 'application/json') {
    //do something
};
```

这样就可以返回数据了

CoAP Server端代码

Server端的代码比较简单，判断一下

```
if (req.headers['Accept'] == 'application/json') {
        parse_url(req.url, function(result){
            res.end(result);
        });
        res.code = '2.05';
    }
```

请求的是否是JSON格式，再返回一个205，也就是Content，只是这时设计是请求一个URL返回对应的数据。如

```
coap://localhost/id/1/
```

这时应该请求的是ID为1的数据，即

```
[ { id: 1, value: 'is id 1', sensors1: 19, sensors2: 20 }]
```

而parse_url只是从数据库从读取相应的数据。

```
function parse_url(url ,callback) {
    var db = new sqlite3.Database(config["db_name"]);
    var result = [];
    db.all("SELECT * FROM basic;", function(err, rows) {
        callback(JSON.stringify(rows));
    })
}
```

并且全部都显示出来，设计得真是有点不行，不过现在已经差不多了。

17.7 使用IoT-CoAP构建物联网

(注意：windows系统npm install失败时，需要自己建立一个C:and Settings[USERNAME]Data文件)

```
npm install iot-coap
```

1.新建**index.js**

注意: 如果已经存在一个index.js文件，请将下面内容添加到文件末尾(create index.js, and add)

```
var iotcoap         = require('iot-coap');

iotcoap.run();
iotcoap.rest.run();
```

注意:在db配置可以选择mongodb和sqlite3，替换所需要的数据库即可。(you can choice db on iot.js with 'sqlite' or 'mongodb')

2.创建**iot.js**

```
exports.config  = {
    "db_name": "iot.db",
    "mongodb_name": "iot",
    "mongodb_documents": "iot",
    "db": "mongodb",
    "table_name": "basic",
    "keys":[
        "id",
        "value",
        "sensors1",
        "sensors2"
    ],
    "db_table": "id integer primary key, value text, sensors1 float, sensors2 float",
    "mongodb_init":[
        {
            id: 1,
            value: "is id 1",
            sensors1: 19,
            sensors2: 20
        },
        {
            id: 2,
            value: "is id 2",
            sensors1: 20,
            sensors2: 21
        }
    ],
    "init_table":[
        "insert or replace into basic (id,value,sensors1,sensors2) VALUES (1, 'is id 1', 19, 20);",
        "insert or replace into basic (id,value,sensors1,sensors2) VALUES (2, 'is id 2', 20, 21);"
    ],
    "query_table":"select * from basic;",
    "rest_url": "/id/:id",
    "rest_post_url": "/",
    "rest_port": 8848
};
```

3.运行(run)

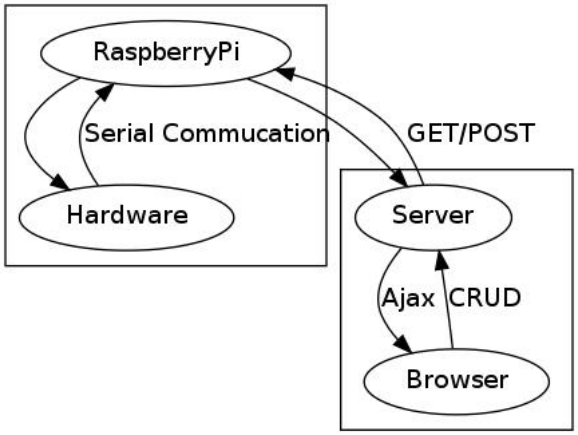
```
node index.js
```

show:

```
coap listening at coap://0.0.0.0:5683
restify listening at http://0.0.0.0:8848
```

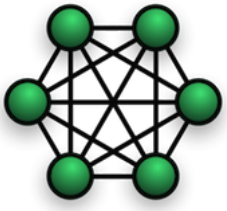
18 简单物联网

到这时，我们算搭建了一个简单的REST服务了。接着我们可以简单的做一个最小的物联网系统，将我们的单片机、MCU等等连上网。



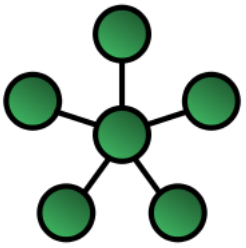
硬件结构图

考虑到如果我们只是单一连接各个节点，那么系统的结构图，同下所示



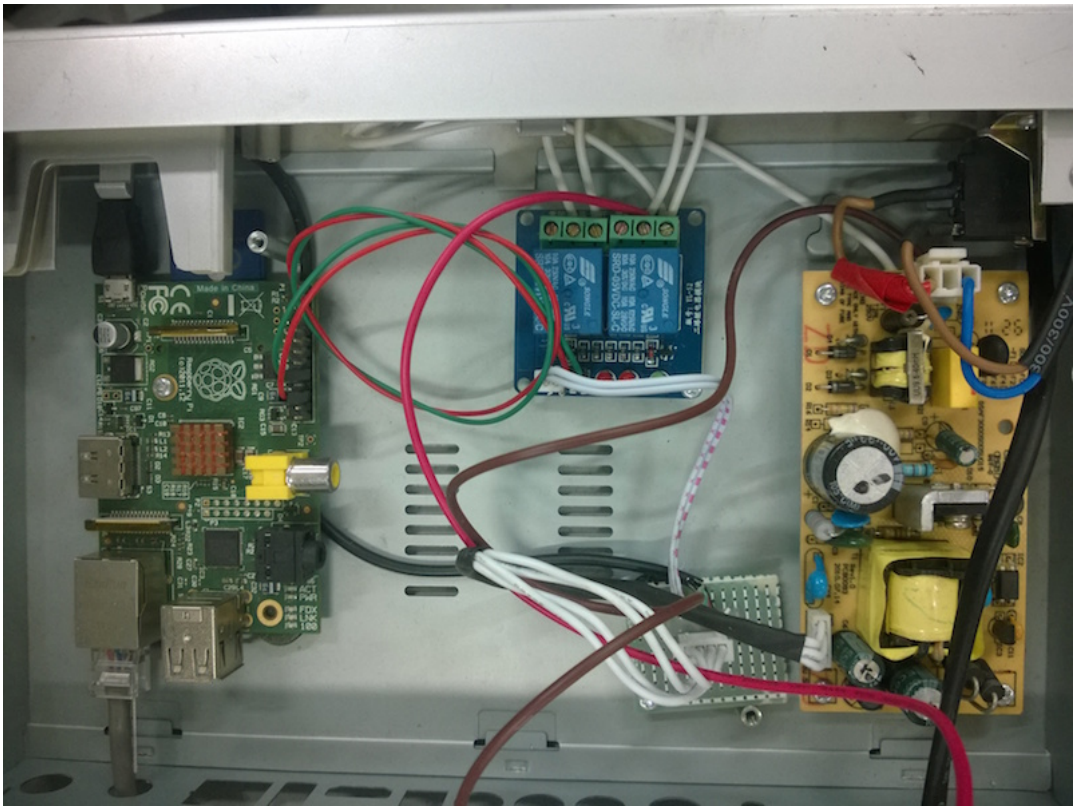
全连接

下面的星形结构图类似于我们在接下来所要构建的系统



星形结构图

一个用于控制真实电器的硬件实物图



简单实物图

18.1 硬件通信

18.1.1 串口通信

Arduino与Raspberry Pi通过串口通信的方式实现通信，相互传输所需要的数据，Raspberry Pi将资源传于互联网上对应的接口，接口可以在互联网上被访问。Laravel框架构架于服务器之上，将Raspbery Pi获取过来的数据存储于MySQL数据，再以REST服务的方式共享数据，互联网上的其他设备便可以通过网络来访问这些设备。Ajax用于将后台的数据以不需要刷新的方式传递到网站前台，通过HighCharts框架显示给终端用户。

18.1.1.1 Python

1.在Windows中的串口通常是`COM1`,`COM0`等等

```
ser=serial.Serial("COM0",9600)
```

2.Mac OS系统中位于/dev目录下，名字类似于`tty.usbmodem1451`。

```
    serial.Serial("/dev/tty.usbmodem1451",9600)
```

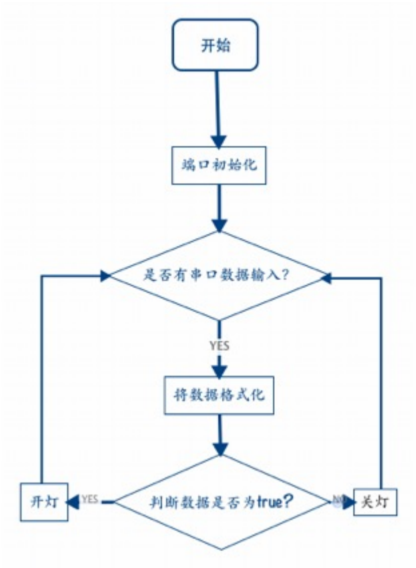
3.在Linux内核的系统中虚拟串口用的节点是ttyACM，位于/dev目录下。

```
    serial.Serial("/dev/ttyACM0",9600)
```

串行接口是一种可以将接受来自CPU的并行数据字符转换为连续的串行数据流发送出去，同时可将接受的串行数据流转换为并行的数据字符供给CPU的器件。一般完成这种功能 的电路，我们称为串行接口电路。

便是打开这个设备，以9600的速率传输数据。

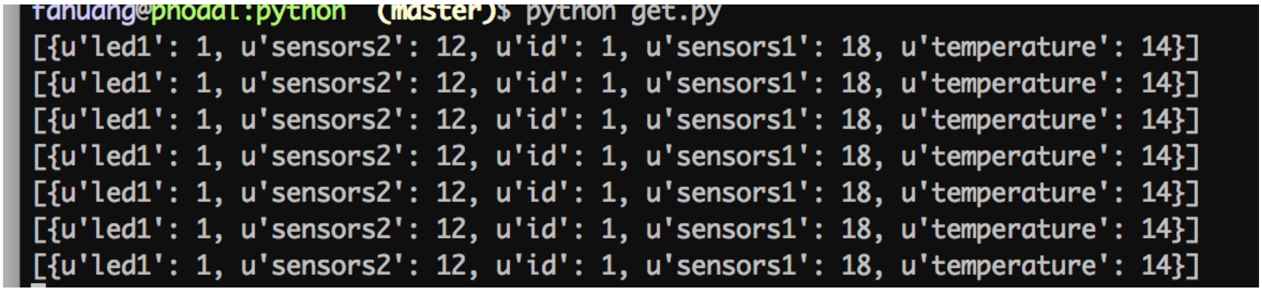
程序框架如下所示:



Raspberry Pi

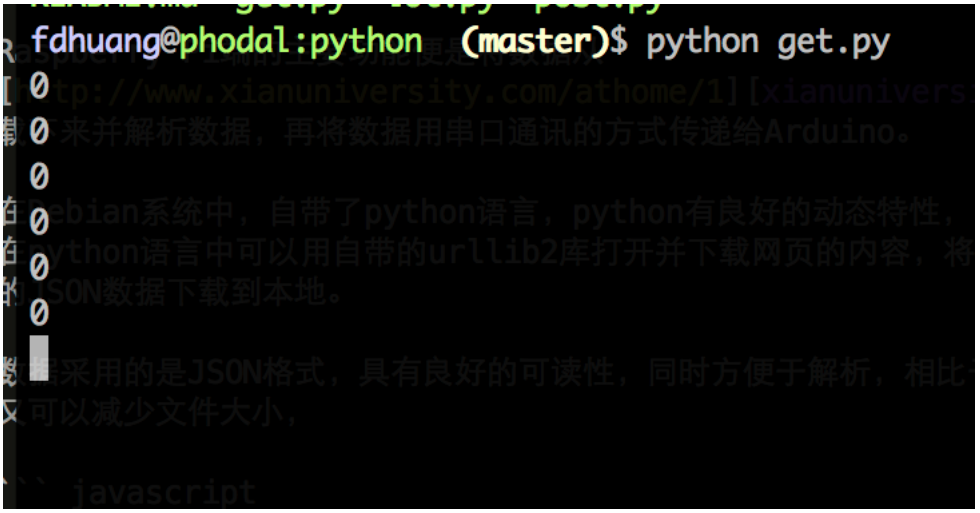
代码如下:

```
 import json
import urllib2
import serial
import time

url="http://www.xianuniversity.com/athome/1"

while 1:
    try:
        date=urllib2.urlopen(url)
        result=json.load(date)
        status=result[0]["led1"]
        ser=serial.Serial("/dev/ttyACM0",9600)
        if status==1 :
            ser.write("1")
        elif status==0:
            ser.write("0")
        time.sleep(1)
    except urllib2.URLError:
        print "Bad URL or timeout"
```



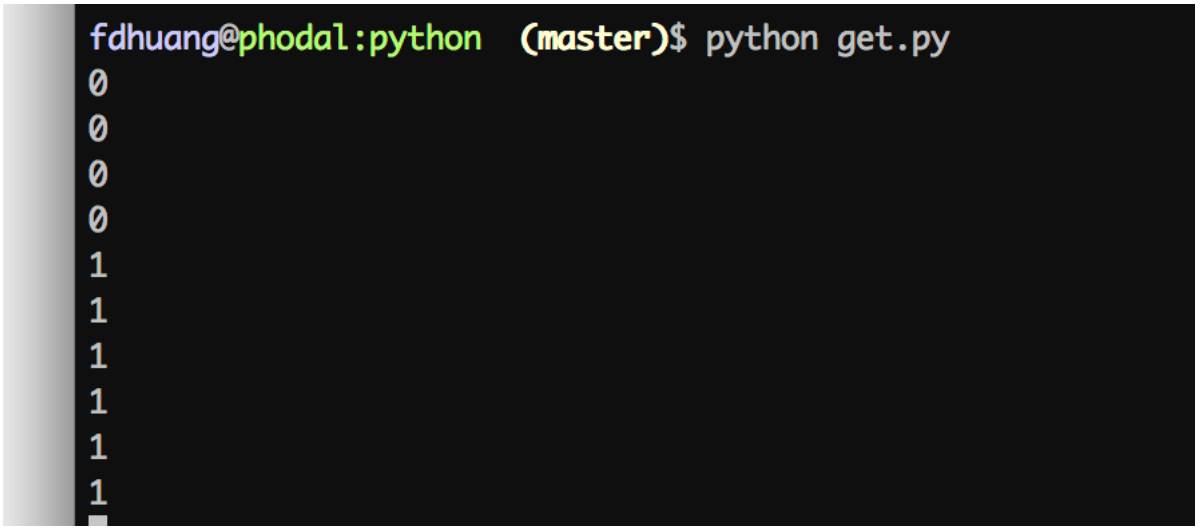
*python*返回*json*数据

系统还需要对上面的数据进行处理，只拿其中的结果



*python*处理完后的结果

当改变led的状态后，便可以得到下面的结果



改变状态后的结果

18.1.1.2 Ruby

如果你用的是Ruby的话，可以尝试使用`serialport`

安装

```
sudo gem install serialport
```

代码大致如下

```
require 'serialport'
sp = SerialPort.new "/dev/ACM0", 9600
sp.write "1"
```

注意: 根据相关的系统修改相关的代码。

18.2 硬件

18.2.1 Arduino

这样我们在我们的Arduino上所要做的便是，读取串口的结果并控制IO口。

```
int ledPort=13;

void setup() {
  Serial.begin(9600);
  pinMode(ledPort,OUTPUT);
}
```

```
int serialData;
void loop() {
  String inString = "";
  while (Serial.available()> 0)
  {
    int inChar = Serial.read();
    if (isDigit(inChar)) {
      inString += (char)inChar;
    }
    serialData=inString.toInt();
    Serial.print(serialData);
  }
  if(serialData==1){
    digitalWrite(ledPort,HIGH);
  }else{
    digitalWrite(ledPort,LOW);
  }
}
```

如果结果是1的话，就让13口为高电平，也就是让灯亮起来。

18.2.2 继电器

> 继电器（英文名称：relay）是一种电控制器件，是当输入量（激励量）的变化达到规定要求时，在电气输出电路中使被控量发生预定的阶跃变化的一种电器。它具有控制系统（又称输入回路）和被控制系统（又称输出回路）之间的互动关系。通常应用于自动化的控制电路中，它实际上是用小电流去控制大电流运作的一种“自动开关”。故在电路中起着自动调节、安全保护、转换电路等作用。

在这里我们可以默认为我们想要为单片机的5V电压控制220V的电器。

最后我们便可以通过些来控制灯的开和关。

19 Android简单示例

由于在某些嵌入式系统中使用的是Android系统，这里给出一个简单的Android App的示例，具体代码可以从clone自 https://github.com/phodal/iot-android

代码说明，经过测试的版本有

- Android 2.3
- Android 4.0.4

机型有

- HTC G1 (android 2.3)
- Motor xt300 (android 2.3)
- Sony ST25I (android 4.0.4)
- MI2

应该可以在大部分的手机上工作。

19.1 调用Web Services GET

这里我们参考一篇文章来调用Web Services——Calling Web Services in Android using HttpClient

19.1.1 创建RESTClient

在这里我们首先会定义四个REST方法GET、POST、PUT、DELETE

```
public void Execute(RequestMethod method) throws Exception {
    switch (method) {
    case GET: {
        // add parameters
        String combinedParams = "";
        if (!params.isEmpty()) {
            combinedParams += "?";
            for (NameValuePair p : params) {
                String paramString = p.getName() + "="
                        + URLEncoder.encode(p.getValue(), HTTP.UTF_8);
                if (combinedParams.length() > 1) {
```

```
                    combinedParams += "&" + paramString;
                } else {
                    combinedParams += paramString;
                }
            }
        }

        HttpGet request = new HttpGet(url + combinedParams);
        request.addHeader("Accept-Encoding", "gzip");

        // add headers
        for (NameValuePair h : headers) {
            request.addHeader(h.getName(), h.getValue());
        }

        executeRequest(request, url);
        break;
    }
    case POST: {
        HttpPost request = new HttpPost(url);
        request.addHeader("Accept-Encoding", "gzip");

        // add headers
        for (NameValuePair h : headers) {
            request.addHeader(h.getName(), h.getValue());
        }
        if (!data.equals("")) {
            request.setEntity(new StringEntity(data, HTTP.UTF_8));
        }

        if (!params.isEmpty()) {
            request.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8));
        }

        executeRequest(request, url);
        break;
    }
    case PUT: {
        HttpPut request = new HttpPut(url);
        request.addHeader("Accept-Encoding", "gzip");

        // add headers
        for (NameValuePair h : headers) {
            request.addHeader(h.getName(), h.getValue());
        }
        if (!data.equals("")) {
            request.setEntity(new StringEntity(data, HTTP.UTF_8));
        }

        if (!params.isEmpty()) {
            request.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8));
        }

        executeRequest(request, url);
        break;
    }
    case DELETE: {
        HttpDelete request = new HttpDelete(url);
        request.addHeader("Accept-Encoding", "gzip");

        // add headers
        for (NameValuePair h : headers) {
            request.addHeader(h.getName(), h.getValue());
        }

        executeRequest(request, url);
        break;
    }
    }
}
```

这四个方法最后都执行executeRequest来获取响应结果。

```
protected void executeRequest(HttpUriRequest request, String url) {

        HttpParams httpParameters = new BasicHttpParams();
        HttpConnectionParams.setConnectionTimeout(httpParameters,
                timeoutConnection);
        HttpConnectionParams.setSoTimeout(httpParameters, timeoutSocket);

        HttpProtocolParams.setUseExpectContinue(httpParameters, false);
        request.setParams(httpParameters);

        setOauth(request);

        DefaultHttpClient client = new DefaultHttpClient();
```

```
        HttpResponse httpResponse;

        try {
            httpResponse = client.execute(request);
            responseCode = httpResponse.getStatusLine().getStatusCode();
            message = httpResponse.getStatusLine().getReasonPhrase();

            HttpEntity entity = httpResponse.getEntity();

            if (entity != null) {
                InputStream instream = httpResponse.getEntity().getContent();
                Header contentEncoding = httpResponse
                        .getFirstHeader("Content-Encoding");

                if (contentEncoding != null
                        && contentEncoding.getValue().equalsIgnoreCase("gzip")) {
                    instream = new GZIPInputStream(instream);
                }

                // instream = entity.getContent();
                response = convertStreamToString(instream);

                // Closing the input stream will trigger connection release
                instream.close();
            }

        } catch (ClientProtocolException e) {
            client.getConnectionManager().shutdown();
            e.printStackTrace();
        } catch (IOException e) {
            client.getConnectionManager().shutdown();
            e.printStackTrace();
        }
    }
```

接着，我们便可以执行getResponse()函数来获取结果。

19.2 使用REST Client获取结果

使用RESTClient时，便可以用下面的示例

```
RestClient client = new RestClient(tUrl);
try {
    client.Execute(RequestMethod.GET);
    if (client.getResponseCode() != 200) {
        //do something
    }
    //JSONArray jArray = new JSONArray(client.getResponse());
} catch (Exception e) {
    //do something
}
```

而这时，我们只需要对相应的数据进行处理就可以了，如

```
JSONArray jArray = new JSONArray(client.getResponse());
JSONObject jObj=jArray.getJSONObject(0);
vshow.setText(jObj.toString());

outputJSON(jObj);
```

将他转换为String，接着在Android端上显示最后的结果。

20 尾声

20.1 路

20.2 其他

意见及建议: https://github.com/phodal/designiot/issues

邮箱: h@phodal.com

1. https://zh.opensuse.org/index.php?title=%E8%BD%AF%E4%BB%B6%E5%8C%85%E7%AE%A1%E7%90%86&variant=zh↩

2. 形如http://www.phodal.com↩

3. URL 是 URI 的子集↩

4. curl是利用URL语法在命令行方式下工作的开源文件传输工具。↩