基于 REST 服务的最小物联网系统设计

目录

引言

你可以将这个系统当成是你的毕业设计，或者用它来控制你想控制的东西，总之你可以用它来做一个最小的物联网系统。

不过，在这里可能没有那么复杂的功能，因为强调的是最小。

BareMinimum，这也是为什么我没有改 Arduino 上面的工程名的原因，因为它是最小的，(PS：大家都懂的，如果玩硬件)。物联网，这个东西一直很复杂，也不是很复杂，只是从硬件到软件涉及到的东西过多了，不止一点点。当然写在本文的方案也有很多，不止这一个，只是这个算是基本的最小的，仅此而已。(转载保留 Phodal's Blog)

关于

源码:https://github.com/gmszone/iot

文档可能没有足够的详细，因为剩下的部分都可以 Google 到，这里就不写详细了。

框架:

- PHP Laravel
- jQuery (Javascript 主要用于 Ajax)
- jQuery Mobile（可选）(我觉得我有点懒，于是从原来做的项目直接拿了出来）
- Bootstrap (可选）(其实没有多大实际用处，只是因为好看和 jQuery Mobile 一样)

语言:

Processing/C/C++ : Arduino

Python : Raspberry Pi 或者与之相近设备都可以使用，实现与 Arduino 串口通信

PHP 我学得不是很好，因为 Laravel 没有让我学好，但是让我能做想做的事。

相关文章及专栏

- CSDN - Laravel 专栏

1. Laravel RESTful 快速部署指南（一）
2. Laravel RESTful 快速部署指南（二）
3. Laravel RESTful 快速部署指南（三）

相关知识

软件

- RESTful
- Ajax
- JSON

硬件

- 硬件编程
- 串口通信

关于服务器

- Nginx 需要配置，具体配置可以参照 Github 上面的代码
- LNMP 直接用上面的会比较简单，但是可能也会遇到一些问题。
- Phpmyadmin 最好需要有这个，如果不是很精通 MySQL

补充说明

Arduino 不是必需的，只要你懂得如何用你的芯片进行串口通信。

考虑到 Raspberry Pi 的成本可能会有点高，你可以试着用 OpenWRT Linux，主要用在路由器用的，上面可以跑 Python。或者等等过些时候的小米路由器，可以加这个在上面。

如果你没有服务器没有 Raspberry Pi，那就找个路由器来当服务器吧，相关文章如下:

Openwrt python,openwrt 上使用 Python

对了，如果你觉得哪里有问题记得在 Github 上提出来，而不是在原文。

注意

！ 请尽可能少用我的网站做测试

如何开始

```
$ git clone https://github.com/gmszone/iot.git
$ cp iot/rest
```

创建一个新的数据库，如 iot 编辑 app/config/database.php

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

配置 nginx，添加，详细可参考 nginx 下面的配置

```
# include /etc/nginx/includes/enforce_non_www;
if ($host ~* ^www\.(.*))
{
set $host_without_www $1;
rewrite ^/(.*)$ $scheme://$host_without_www/$1 permanent;
}

# Check if file exists
if (!-e $request_filename)
{
rewrite ^/(.*)$ /index.php?/$1 last;
break;
}
```

测试

```
$ sudo python python/get.py
```

再根据需要修改端口，视真实的端口而修改。

关于物联网

物联网（Internet of Things，缩写 IOT）是一个基于互联网、传统电信网等信息承载体，让所有能够被独立寻址的普通物理对象实现互联互通的网络。

物联网一般为无线网，由于每个人周围的设备可以达到一千至五千个，所以物联网可能要包含 500 万亿至一千万亿个物体，在物联网上，每个人都可以应用电子标签将真实的物体上网联结，在物联网上都可以查找出它们的具体位置。通过物联网可以用中心计算机对机器、设备、人员进行集中管理、控制，也可以对家庭设备、汽车进行遥控，以及搜寻位置、防止物品被盗等各种应用。

简单的来说 Internet 是一个由计算机组成的网络，那么物联网就是一个由物体 (Things) 组成的网络，只不过其依赖于 Internet，是 Internet 的一部分。

最小物联网系统

这个也就是我们要讨论的主题了，我们要做的最小物联网系统其实也就相当于是一个平台。我们可以上传我们各种物体的信息，同时给予这些物体一些属性，我们也可以通过网络来控制这些物体，而他们之间也可以相互控制。因此，我们需要给他们提供一个网络，这就是 RESTful 的由来。

所以我们也稍微了解一下 RESTful 吧。

RESTful

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（Representational State Transfer）。

我们的世界是由资源来组成的，一个物体也就相当于是一个资源，以这种方式来构建我们的物联网系统，在目前来说是再好不过的一个方案了。

REST 架构就是希望能够统一这一类的 Hypermedia Controls, 赋予他们标准的, 高度可扩展的标准语义及表现形式, 使得甚至无人工干预的机器与机器间的通用交互协议边的可能.

这个也就是我们的目的了，物联网最后的核心就是使物体与物体之间的交互成为可能。

那么，这里也就解释了为什么我们要用 RESTful 来做这个最小系统的原因了。

最小系统中的 RESTful

例如，一个简单的例子，列举所有物体状态，

```
$ GET http://localhost/athome
```

呈现某一特定状态，

```
$ GET http://localhost/athome/1/
```

剩下的部分这里就不多说了，多说无益，可以自己谷歌去。

接着我们要讨论的就是系统框架

系统框架

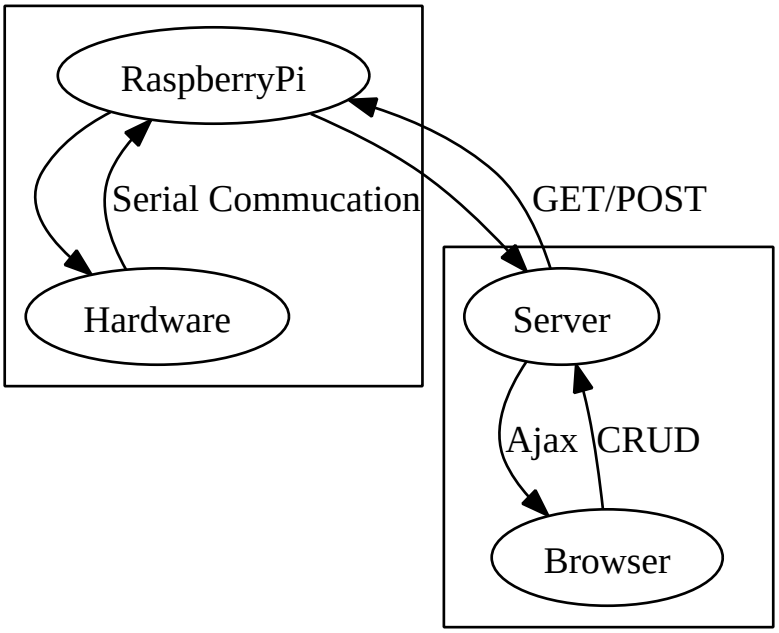

图 1: 系统框架

为什么是 Raspberry Pi

Raspberry Pi 在这里只是充当了数据的发送和接收，虽然我们可以直接将 Raspberry Pi 作为控制的对象，但是将这个从中剥离来讲清楚系统的结构会更加简单。从而，可以让我们把核心注意力聚焦在要解决的问题上，也就是数据传送，每个部分都可以简单地从系统剥离出来，用另外的事物来替换。

为什么是 Arduino

这个问题的答案和上面是一样的，只是因为有些搞物联网是从软件过来的，对于他们来说去理解端口的难道可能有点大。所以，我们在简化系统设计的同时，也把系统的代码简化了。因为 Arduino 足够的简单，我们可以关心问题的本质，而不是如何去编程。

为什么是 Ajax

AJAX 即 ``Asynchronous JavaScript and XML''（异步的 JavaScript 与 XML 技术），指的是一套综合了多项技术的浏览器端网页开发技术。

这里的目的只是在于演示如何运用这些数据，使它具有他应有的价值，而不在于技术本身。当然 ajax 不是必需的，如果你需要的只是用来控制这个灯。

为什么是 Laravel

只是因为个人喜爱，你也可以用 Ruby On Rails 来搭建这样一个功能，或者是 Java。只不过 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。同时 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。

安装 Laravel

这个就比较简单了，不过在那之前你要有 gi t 以及安装了 php 环境，这个在 linux 上面比较好实现，可以用 Raspberry Pi 或者是你的电脑来做这个，不一定用用上你的服务器。

```
$ git clone https://github.com/laravel/laravel
```

先 clone 这个 git，如果你没有安装好 PHP，请安装好，and go on。

```
$ cd laravel
```

laravel 用到了 php 的包管理工具 composer，于是我们还需要用到 composer，与 Laravel 相比也算是一个优雅的工具。

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

这里推荐的是 linux 系统，如果你是 *nix 都是可以的 (ps:mac os x 属于 unix 分支)，除了 windows，所以如果是 windows，请直接下载

Composer-Setup

然后让我们安装所需要的那些包

```
$ php composer.phar install
```

当然这里用的是比较通用的，如果你是 *nix，有支持可以直接

```
$ composer install
```

配置 MySQL

这里并不会列举 MySQL 的安装方法，如果你是 openSUSE，可以

```
$ zypper install mysql
```

这个也可以，不过最近我尽量到迁移到 MariaDB 了。

```
$ zypper install mariadb
```

当然，最简单的方法是直接上官网。这里说的是修改 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 bbs 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
}
```

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。

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，忽略掉这些将要的因素吧。

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

打开 app/database/***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。

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

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 {

    /**
     * Display a listing of the resource.
     *
     * @return Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @return Response
     */
    public function store()
    {
```

```
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  int  $id
     * @return Response
     */
    public function update($id)
    {
        //
    }
```

```
    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return Response
     */
    public function destroy($id)
    {
        //
    }

}
```

Laravel Resources

上面的代码过于沉重，请让我用 Ctrl+C 来带来点知识吧。

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 {

    /**
     * Display a listing of the resource.
     *
     * @return Response
     */
    public $restful=true;

    protected $athome;
```

```
    public function __construct(Athomes $athome)
    {
        $this--->athome = $athome ;
     }

    public function index()
    {
        $maxid=Athomes::all();
        return Response::json($maxid);
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return Response
     */
    public function create()
    {
        $maxid=Athomes::max('id');
       return View::make('athome.create')->with('maxid',$maxid);
    }

    /**
     * Store a newly created resource in storage.
     *
     * @return Response
     */
    public function store()
    {
        // validate
         // read more on validation at http://laravel.com/docs/
validation
        $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);

        // process the login
        if ($validator->fails()) {
            return Redirect::to('athome/create')
                ->withErrors($validator)
                ->withInput(Input::except('password'));
        } 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();

            // redirect
        Session::flash('message', 'Successfully created athome!');
            return Redirect::to('athome');
        }
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return Response
     */
    public function show($id)
    {
        $myid=Athomes::find($id);
        $maxid=Athomes::where('id','=',$id)
                ->select('id','temperature','sensors1','sensors2','led1')
                        ->get();
```

```
        return Response::json($maxid);
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return Response
     */
    public function edit($id)
    {
        // get the nerd
        $athome = Athomes::find($id);

        // show the edit form and pass the nerd
        return View::make('athome.edit')
            ->with('athome', $athome);
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  int  $id
     * @return Response
     */
    public function update($id)
    {
        // validate
         // read more on validation at http://laravel.com/docs/
validation
        $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);

        // process the login
        if ($validator->fails()) {
            return Redirect::to('athome/' . $id . '/edit')
                ->withErrors($validator);
        } else {
            // store
            $nerd = Athomes::find($id);
            $nerd->sensors1       = Input::get('sensors1');
            $nerd->sensors2       = Input::get('sensors2');
            $nerd->temperature    = Input::get('temperature');
            $nerd->led1           = Input::get('led1');
            $nerd->save();

            // redirect
        Session::flash('message', 'Successfully created athome!');
            return Redirect::to('athome');
        }
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return Response
     */
    public function destroy($id)
    {
        // delete
        $athome = Athomes::find($id);
        $athome->delete();
        if(is_null($athome))
        {
             return Response::json('Todo not found', 404);
        }
```

```
        // redirect
      Session::flash('message', 'Successfully deleted the nerd!');
        return Redirect::to('athome');
    }

}
```

希望你能读懂，没有的话，关注下一节。

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，

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。

创建表单

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>
```

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'=>'

      </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' =>
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');
    }
}
```

编辑 edit

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), 'r

        <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' => 'for
control')) }}
        </div>

        <div class="form-group}
            {{ Form::label('sensors2', '传感器2') }}
        {{ Form::text('sensors2', Input::old('sensors2'), array('class' => 'for
control')) }}
        </div>

        <div class="form-group}
            {{ Form::label('temperature', '温度传感器') }}
        {{ Form::text('temperature', Input::old('temperature'), array('class' =
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>
```

最后效果见:http://b.phodal.com/

代码位置:http://b.phodal.com/js/app.js

我觉得似乎我把这个代码写长了，但是我不是故意，只是必需的。先观察 Ajax 部分：

Ajax

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。

HighChart

再省去一部分，摘自我原来的博客

HIGHCHARTS Highcharts 是一个制作图表的纯 Javascript 类库，主要特性如下：

- 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;

不过因为项目原因，所以可能不会再使用这个，只对个人免费，现在的考虑是基于 D3 做一个新的。

官方示例代码

```
$(function () {
        $('#container').highcharts({
            title: {
                text: 'Monthly Average Temperature',
                x: -20 //center
            },
            subtitle: {
                text: 'Source: WorldClimate.com',
                x: -20
            },
            xAxis: {
           categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                    'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
```

```
            },
            yAxis: {
                title: {
                    text: 'Temperature (°C)'
                },
                plotLines: [{
                    value: 0,
                    width: 1,
                    color: '#808080'
                }]
            },
            tooltip: {
                valueSuffix: '°C'
            },
            legend: {
                layout: 'vertical',
                align: 'right',
                verticalAlign: 'middle',
                borderWidth: 0
            },
            series: [{
                name: 'Tokyo',
           data: [7.0, 6.9, 9.5, 14.5, 18.2, 21.5, 25.2, 26.5, 23.3, 18.3, 13.9, 9
            }, {
                name: 'New York',
           data: [-0.2, 0.8, 5.7, 11.3, 17.0, 22.0, 24.8, 24.1, 20.1, 14.1, 8.6, 2
            }, {
                name: 'Berlin',
           data: [-0.9, 0.6, 3.5, 8.4, 13.5, 17.0, 18.6, 17.9, 14.3, 9.0, 3.9, 1.0
            }, {
                name: 'London',
           data: [3.9, 4.2, 5.7, 8.5, 11.9, 15.2, 17.0, 16.6, 14.2, 10.3, 6.6, 4.8
            }]
        });
    });
```

我承认我也不想看这些代码，但是这样子似乎使原文的长度变长了。大部分人也省得去查找了。

所以我们要做的只是用数组代替 data

jQuery Mobile

在首页上看到的那个效果是 jQuery Mobile。。

这里写的数据通讯指的是两部分，一部分是与服务器，一部分是与单片机。这样设计的另外一个原因是，更好的分层，能让我们更好的理解这个系统。负责这个功能的这里用的是 Raspberry Pi，或者是你的 PC 两者都可以，我想你也看到了之前的代码。那么先让我们看看与服务器通信的这部分。

服务器通讯

示例中的代码是这样子的，如果你没有看懂的话，那么等等。

```
import json,urllib2

url="http://b.phodal.com/athome/1"
while True:
    status=json.load(urllib2.urlopen(url))[0]['led1']
```

GET

看看 get.py 的代码，这个是没有压缩的，换句话说，会比较好理解一点

```
import json
import urllib2

url="http://b.phodal.com/athome/1"

while 1:
        date=urllib2.urlopen(url)
        result=json.load(date)
        status=result[0]['led1']
        print status
```

这里做的事情有两件，一件是从服务器 GET，另外一个就是解析 JSON 数据。

如果你用的是 *nix，应该就自带 curl 了，可以试着用下面的命令来 GET

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

那么应该返回的是下面的结果

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

用在 python 里面就是

```
urllib2.open("http://b.phodal.com/athome/1")
```

数据解析

python 带有 json 解析模块，我们在这里只需要用 json.load() 来解析获取下面的 date 就可以了

```
result=json.load(date)
```

解析完的 result 相当于是 C 语言里面的数组，在这里相当于是一个二维数组，我们只需要 result[0][`led1']，在 python 里面叫做字典，意思就是和字典一样。

```
"led1":0
```

led1 的值是 0，所以 result[0]['led1] 的值是 0, 如果你用过 Ruby，那么这个和其中的 Hash 差不多。

因此在这里我们拿到了服务器上面的控制状态的指令，也就是 0。我们还需要传给单片机，也就是 Arduino。。

在我们完成了前面的几部分之后，我们也需要把这最后一部分解决，这里更多的是硬件，Arduino 的存在可以让硬件更简单。

Arduino

Arduino 是一款便捷灵活、方便上手的开源电子原型平台，包含硬件（各种型号的 arduino 板）和软件（arduino IDE)。它适用于艺术家、设计师、爱好者和对于 ``互动" 有兴趣的朋友们。

那么让我们先来看看我们写的代码。

```
void setup() {
  Serial.begin(9600);
  pinMode(13,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(13,HIGH);
  }else{
    digitalWrite(13,LOW);
  }
}
```

这个代码看上去似乎会有点复杂，但是让我们看点基础的，也就是由 Arduino 来控制一个 LED 的亮和灭。

```
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
 digitalWrite(led, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
```

```
 digitalWrite(led, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

这个也就是来自于官方的示例程序，而我们要做的东西也和这个差不多，只是这个是自动的，上面那个是由串口通信来实现的。

串口通信

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。

简单地来说，我们諳就是用这个来实现通信的，用之前的 Raspberry Pi 发送 1 和 0 给 Arduino。那么我们在 Arduino 上就只是接受和执行，这个由 loop 里面的 if 来执行

初始化串口

如果你真心不喜欢 51 上的复杂的串口，那么我想 Arduino 又是解放双手的东西了。

```
Serial.begin(9600);
```

这个就是串口初始化，速率为 9600。

串口读取

```
while (Serial.available()<0)
{
int inChar = Serial.read();
if (isDigit(inChar)) {
  inString += (char)inChar;
}
serialData=inString.toInt();
```

用于读取的就是这么一行

```
int inChar=Serial.read()
```

而下面的部分则是刚我们接收到的数据转换为 1, 由于接到的为 char 类型，那么我们需要转为转为 int 进行判断。

为什么不直接用**'1'** 只是为了写给需要的同学用的，也可以直接在上面用if(serialData==`1')，上面写可以让后期扩展的时候方便一点。

加上之前的部分，我们算是把开源的地方做了一个遍，因为 Windows Phone 需要在 Windows 8 上开发的原因，加上我没有 Macbook 以及 iPhone，所以在这里只会有一个 Android 的示例。当然，原因上也是一样的，相信这些也不会很难。

原理上和 Raspberry Pi 的原理很像，也就是 GET 数据，然后解析，也和服务端差不多。当然在最开始的代码里有拨打电话、发短信等等功能，只是我们似着简化系统为我们想要的理想化模型。

源码地址Home-Anywhere ##Android 开发 ##

写在这里的原因是，因为我也不太擅长，所以也给不了多少指导。只是我试着去写过这样一个程序，有了几个版本，所以算是知道怎样去开发，但是相比较于专业于我的人还是有很多不足，所以希望懂得的人给些建议和意见。

浅析

我们需要的库和在 Raspberry Pi 上的类似，如果你不需要的话，可以看看之前的文章：

最小物联网系统（七）------与服务器通讯

因为 CSDN 上发这些文章已经没有足够的必要，在之前的部分文章实在上是针对这部分写的，只是在自己的博客上梳理了一遍。

我们还要做的事情就是有一个 RESTful 的库，以及解析 JSON 用的。

于是就有了下面两个

RESTclient 这个类的原文在calling-web-services-in-android-using-httpclient，专门用于 REST 用的，如果熟悉的人我想一看就知道了。

GSON 这个库来自于 Google，一个不错的库。

所以我们就构成了开发所需的两部分基础。

Android

关于 Android 开发环境的配置这个网上有，最简单的办法是直接下载一个 Android Studio。

下面只是列举一些代码以及可能会遇到的问题。

Android 4.0 Web 问题

如在源码里看到的那样，

```
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
        .detectDiskReads()
        .detectDiskWrites()
      .detectNetwork()  // or .detectAll() for all detectable problems
        .penaltyLog()
        .build());
StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
        .detectLeakedSqlLiteObjects()
        .detectLeakedClosableObjects()
        .penaltyLog()
        .penaltyDeath()
        .build());
```

这部分用于 Android 4.0 的网络，2.* 可以不需要。

JSONObject 以及 JSONArray

会产生下面这些代码的原因是下载下来的 JSON 数据是类似于二维数组，所以需要转换，下面的代码有些丑陋，但是可能工作得很好。

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

handlerData 的由来

```
public GsonBuilder gsonb = new GsonBuilder();
public Gson gson = gsonb.create();
typePhoData phoData;

public handlerData(JSONObject jObj){
    phoData = gson.fromJson(jObj.toString(),
            typePhoData.class);
}
public int get_id(){
```

```
    return phoData.id;
}

public double get_sensors1(){
    return phoData.sensors1;
}

public double get_sensors2(){
    return phoData.sensors2;
}

public double get_temperature(){
    return phoData.temperature;
}

public int get_led1(){
    return phoData.led1;
}

public class typePhoData{
    public int led1;
    public double temperature;
    public double sensors1;
    public double sensors2;
    public int    id;
}
```

在某些程度上，我好像将这些代码给复杂化了，直接放在原文里可能会好一点，不过造成这种错觉的主要原因可能是受 Java 语言的影响，不过从软件工程的某些角度上来说，这样应该会好一点。其他的:

- typePhoData 的命名可能有些不尽人意，但是暂时没有想到一个合适的
- 用过几天 Ruby 后，似乎这个不算是一个问题
- 如果你要修改的话，相信这个接口也不难，也许比原来的简单，前提是你看过原来的代码。

整理完闭。

REST POST

如果你需要 POST，又懒得去看原文，那么 POST 代码在下面，只是因为我暂时没有时间去研究 Android 里面的这些，以及怎样继续这个项目，因为最小的话，似乎已经不再需要添加任何东西了。

```
RestClient clientPost = new RestClient(url);
clientPost.AddParam("temperature", "23.1");
clientPost.AddParam("led", "true");
clientPost.AddParam("title", "from android");
clientPost.AddParam("more", "nEW tESET");
try {
        clientPost.Execute(RequestMethod.POST);
        if(client.getResponseCode()!=200){
                vshow.setText(clientPost.getErrorMessage());
        }
        String response2 = clientPost.getResponse();
        vshow.setText(response2.toString());
} catch (Exception e) {
        vshow.setText(e.toString());
}
```

大致上是类似的，注意一下都是字符就行了。