外 文 翻 译

毕业设计题目：基于 **Control 4** 主机与 **Zigbee** 外扩 **AP** 设备的智能家庭安防系统

原文 1： JSON-RPC 2.0 Specification

译文 2： JSON-RPC 2.0 规范

原文2： Zigbee Wireless Sensor Network in Environmental Monitoring Applications

译文2： Zigbee无线传感器网络在环境监测中的应用

原文 1:

JSON-RPC 2.0 Specification

Origin Date:

2010-03-26 (based on the 2009-05-24 version)

Updated:

2013-01-04

Author:

JSON-RPC Working Group <json-rpc@googlegroups.com>

Table of Contents

1 Overview

JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over http, or in many various message passing environments. It uses JSON (RFC 4627) as data format.

It is designed to be simple!

2 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and

“OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Since JSON-RPC utilizes JSON, it has the same type system (see http://www.json.org or RFC 4627). JSON can represent four primitive types (Strings, Numbers, Booleans, and Null) and two structured types (Objects and Arrays). The term “Primitive” in this specification references any of those four primitive JSON types. The term “Structured” references either of the structured JSON types. Whenever this document refers to any JSON type, the first letter is always capitalized: Object, Array, String, Number, Boolean, Null. True and False are also capitalized.

All member names exchanged between the Client and the Server that are considered for matching of any kind should be considered to be case-sensitive. The terms function, method, and procedure can be assumed to be interchangeable.

The Client is defined as the origin of Request objects and the handler of Response objects.
The Server is defined as the origin of Response objects and the handler of Request objects.

One implementation of this specification could easily fill both of those roles, even at the same time, to other different clients or the same client. This specification does not address that layer of complexity.

3 Compatibility

JSON-RPC 2.0 Request objects and Response objects may not work with existing JSON-RPC 1.0 clients or servers. However, it is easy to distinguish between the two versions as 2.0 always has a member named “jsonrpc” with a String value of “2.0” whereas 1.0 does not. Most 2.0 implementations should consider trying to handle 1.0 objects, even if not the peer-to-peer and class hinting aspects of 1.0.

4 Request object

A rpc call is represented by sending a Request object to a Server. The Request object has the following members:

jsonrpc

A String specifying the version of the JSON-RPC protocol. MUST be exactly “2.0”.

method

A String containing the name of the method to be invoked. Method names that begin with the word rpc followed by a period character (U+002E or ASCII 46) are reserved for rpc-internal methods and extensions and MUST NOT be used for anything else.

Params

A Structured value that holds the parameter values to be used during the invocation of the method. This member MAY be omitted.

Id

An identifier established by the Client that MUST contain a String, Number, or NULL value if included. If it is not included it is assumed to be a notification. The value SHOULD normally not be Null [1] and Numbers SHOULD NOT contain fractional parts [2]

The Server MUST reply with the same value in the Response object if included. This member is used to correlate the context between the two objects.

[1] The use of Null as a value for the id member in a Request object is discouraged, because this specification uses a value of Null for Responses with an unknown id. Also, because JSON-RPC 1.0 uses an id value of Null for Notifications this could cause confusion in handling.

[2] Fractional parts may be problematic, since many decimal fractions cannot be represented exactly as binary fractions.

4.1 Notification

A Notification is a Request object without an “id” member. A Request object that is a Notification signifies the Client’s lack of interest in the corresponding Response object, and as such no Response object needs to be returned to the client. The Server MUST NOT reply to a Notification, including those that are within a batch request.

Notifications are not confirmable by definition, since they do not have a Response object to be returned. As such, the Client would not be aware of any errors (like e.g. “Invalid params”,”Internal error”).

4.2 Parameter Structures

If present, parameters for the rpc call MUST be provided as a Structured value. Either by-position through an Array or by-name through an Object.

- by-position: params MUST be an Array, containing the values in the Server expected order.
- by-name: params MUST be an Object, with member names that match the Server expected parameter names. The absence of expected names MAY result in an error being generated. The names MUST match exactly, including case, to the method's expected parameters.

5 Response object

When a rpc call is made, the Server MUST reply with a Response, except for in the case of Notifications. The Response is expressed as a single JSON Object, with the following members:

jsonrpc

A String specifying the version of the JSON-RPC protocol. MUST be exactly "2.0".

result

This member is REQUIRED on success.

This member MUST NOT exist if there was an error invoking the method.

The value of this member is determined by the method invoked on the Server.

Error

This member is REQUIRED on error.

This member MUST NOT exist if there was no error triggered during invocation.

The value for this member MUST be an Object as defined in section 5.1.

id

This member is REQUIRED.

It MUST be the same as the value of the id member in the Request Object.

If there was an error in detecting the id in the Request object (e.g. Parse error/Invalid Request), it MUST be Null.

Either the result member or error member MUST be included, but both members MUST NOT be included.

5.1 Error object

When a rpc call encounters an error, the Response Object MUST contain the error member with a value that is a Object with the following members:

code

A Number that indicates the error type that occurred.

This MUST be an integer.

Message

A String providing a short description of the error.

The message SHOULD be limited to a concise single sentence.

Data

A Primitive or Structured value that contains additional information about the error.
This may be omitted.
The value of this member is defined by the Server (e.g. detailed error information, nested errors etc.).

The error codes from and including -32768 to -32000 are reserved for pre-defined errors. Any code within this range, but not defined explicitly below is reserved for future use. The error codes are nearly the same as those suggested for XML-RPC at the following

code	**message**	**meaning**
-32700	Parse error	Invalid JSON was received by the server. An error occurred on the server while parsing the JSON text.
-32600	Invalid Request	The JSON sent is not a valid Request object.
-32601	Method not found	The method does not exist / is not available.
-32602	Invalid params	Invalid method parameter(s).
-32603	Internal error	Internal JSON-RPC error.
-32000 to -32099	Server error	Reserved for implementation-defined server-errors.

The remainder of the space is available for application defined errors.

6 Batch

To send several Request objects at the same time, the Client MAY send an Array filled with Request objects.

The Server should respond with an Array containing the corresponding Response objects, after all of the batch Request objects have been processed. A Response object SHOULD exist for each Request object, except that there SHOULD NOT be any Response objects for notifications. The Server MAY process a batch rpc call as a set of concurrent tasks, processing them in any order and with any width of parallelism.

The Response objects being returned from a batch call MAY be returned in any order within the Array. The Client SHOULD match contexts between the set of Request objects and the resulting set of Response objects based on the id member within each Object.

If the batch rpc call itself fails to be recognized as an valid JSON or as an Array with at least one value, the response from the Server MUST be a single Response object. If there are no Response objects contained within the Response array as it is to be sent to the client, the server MUST NOT return an empty Array and should return nothing at all.

7 Examples

Syntax:

```
→ data sent to Server
← data sent to Client
```

rpc call with positional parameters:

```
→ {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "i
d": 1}
← {"jsonrpc": "2.0", "result": 19, "id": 1}

→ {"jsonrpc": "2.0", "method": "subtract", "params": [23, 42], "i
d": 2}

← {"jsonrpc": "2.0", "result": -19, "id": 2}
```

rpc call with named parameters:

```
→ {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahe
nd": 23, "minuend": 42}, "id": 3}

← {"jsonrpc": "2.0", "result": 19, "id": 3}



→ {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend":
 42, "subtrahend": 23}, "id": 4}

← {"jsonrpc": "2.0", "result": 19, "id": 4}
```

a Notification:

```
→ {"jsonrpc": "2.0", "method": "update", "params": [1,2,3,4,5]}
→ {"jsonrpc": "2.0", "method": "foobar"}
```

rpc call of non-existent method:

```
→ {“jsonrpc”: “2.0”, “method”: “foobar”, “id”: “1”}
← {“jsonrpc”: “2.0”, “error”: {“code”: -32601, “message”: “Meth
od not found”}, “id”: “1”}
```

rpc call with invalid JSON:

```
→ {“jsonrpc”: “2.0”, “method”: “foobar, “params”: “bar”, “baz}
← {“jsonrpc”: “2.0”, “error”: {“code”: -32700, “message”: “Pars
e error”}, “id”: null}
```

rpc call with invalid Request object:

```
→ {“jsonrpc”: “2.0”, “method”: 1, “params”: “bar”}
← {“jsonrpc”: “2.0”, “error”: {“code”: -32600, “message”: “Inva
lid Request”}, “id”: null}
```

rpc call Batch, invalid JSON:

```
→ [

  {“jsonrpc”: “2.0”, “method”: “sum”, “params”: [1,2,4], “id”:
“1”},

  {“jsonrpc”: “2.0”, “method”

]

← {“jsonrpc”: “2.0”, “error”: {“code”: -32700, “message”: “Pars
e error”}, “id”: null}
```

rpc call with an empty Array:

```
→ []

← {“jsonrpc”: “2.0”, “error”: {“code”: -32600, “message”: “Inva
lid Request”}, “id”: null}
```

rpc call with an invalid Batch (but not empty):

```
→ [1]
← [

  {“jsonrpc”: “2.0”, “error”: {“code”: -32600, “message”: “Inval
id Request”}, “id”: null}

]
```

rpc call with invalid Batch:

```
→ [1,2,3]
← [

  {“jsonrpc”: “2.0”, “error”: {“code”: -32600, “message”: “Inval
id Request”}, “id”: null},

  {“jsonrpc”: “2.0”, “error”: {“code”: -32600, “message”: “Inval
id Request”}, “id”: null},

  {“jsonrpc”: “2.0”, “error”: {“code”: -32600, “message”: “Inval
id Request”}, “id”: null}

]
```

rpc call Batch:

```
→ [
        {“jsonrpc”: “2.0”, “method”: “sum”, “params”: [1,2,4], “i
d”: “1”},
        {“jsonrpc”: “2.0”, “method”: “notify_hello”, “params”:
[7]},
        {“jsonrpc”: “2.0”, “method”: “subtract”, “params”: [42,23],
 “id”: “2”},
        {“foo”: “boo”},
        {“jsonrpc”: “2.0”, “method”: “foo.get”, “params”: {“name”:
 “myself”}, “id”: “5”},
        {“jsonrpc”: “2.0”, “method”: “get_data”, “id”: “9”}
    ]
← [
        {“jsonrpc”: “2.0”, “result”: 7, “id”: “1”},
        {“jsonrpc”: “2.0”, “result”: 19, “id”: “2”},
        {“jsonrpc”: “2.0”, “error”: {“code”: -32600, “message”: “I
nvalid Request”}, “id”: null},
        {“jsonrpc”: “2.0”, “error”: {“code”: -32601, “message”: “M
ethod not found”}, “id”: “5”},
        {“jsonrpc”: “2.0”, “result”: [“hello”, 5], “id”: “9”}
    ]
```

rpc call Batch (all notifications):

```
→ [
        {“jsonrpc”: “2.0”, “method”: “notify_sum”, “params”: [1,2,
4]},
        {“jsonrpc”: “2.0”, “method”: “notify_hello”, “params”: [7]}
```

```
    ]
← //Nothing is returned for all notification batches
```

8 Extensions

Method names that begin with rpc. Are reserved for system extensions, and MUST NOT be used for anything else. Each system extension is defined in a related specification. All system extensions are OPTIONAL.

译文 1：

JSON-RPC 2.0 规范

Table of Contents

1.概述

JSON-RPC 是一个无状态且轻量级的远程过程调用(RPC)协议。 本规范主要定义了一些数据结构及其相关的处理规则。它允许运行在基于 socket,http 等诸多不同消息传输环境的同一进程中。其使用 JSON（RFC 4627）作为数据格式。
它为简单而生!

2.约定

文档中关键字"MUST"、"MUST NOT"、"REQUIRED"、"SHALL"、"SHALL NOT"、"SHOULD"、"SHOULD NOT"、"RECOMMENDED"、"MAY"和 "OPTIONAL" 将在 RFC 2119 中得到详细的解释及描述。
由于 JSON-RPC 使用 JSON，它具有与其相同的类型系统(见 http://www.json.org 或 RFC 4627)。JSON 可以表示四个基本类型(String、Numbers、Booleans 和 Null)和两个结构化类型(Objects 和 Arrays)。 规范中，术语“Primitive”标记那 4 种原始类型，“Structured”标记两种结构化类型。任何时候文档涉及 JSON 数据类型，第一个字母都必须大写：Object，Array，String，Number，Boolean，Null。包括 True 和 False 也要大写。
在客户端与任何被匹配到的服务端之间交换的所有成员名字应是区分大小写的。 函数、方法、过程都可以认为是可以互换的。
客户端被定义为请求对象的来源及响应对象的处理程序。
服务端被定义为响应对象的起源和请求对象的处理程序。

该规范的一种实现为可以轻而易举的填补这两个角色,即使是在同一时间,同一客户端或其他不相同的客户端。 该规范不涉及复杂层。

3.兼容性

JSON-RPC 2.0 的请求对象和响应对象可能无法在现用的 JSON-RPC 1.0 客户端或服务端工作，然而我们可以很容易在两个版本间区分出 2.0，总会包含一个成员命名为 “jsonrpc” 且值为“2.0”， 而 1.0 版本是不包含的。大部分的 2.0 实现应该考虑尝试处理 1.0 的对象，即使不是对等的也应给其相关提示。

4.请求对象

发送一个请求对象至服务端代表一个 rpc 调用， 一个请求对象包含下列成员：

jsonrpc

指定 JSON-RPC 协议版本的字符串，必须准确写为“2.0”

method

包含所要调用方法名称的字符串，以 rpc 开头的方法名，用英文句号（U+002E or ASCII 46）连接的为预留给 rpc 内部的方法名及扩展名，且不能在其他地方使用。

params

调用方法所需要的结构化参数值，该成员参数可以被省略。

id

已建立客户端的唯一标识 id，值必须包含一个字符串、数值或 NULL 空值。如果不包含该成员则被认定为是一个通知。该值一般不为 NULL[1]，若为数值则不应该包含小数[2]。

服务端必须回答相同的值如果包含在响应对象。 这个成员用来两个对象之间的关联上下文。

[1] 在请求对象中不建议使用 NULL 作为 id 值，因为该规范将使用空值认定为未知 id 的请求。另外，由于 JSON-RPC 1.0 的通知使用了空值，这可能引起处理上的混淆。

[2] 使用小数是不确定性的，因为许多十进制小数不能精准的表达为二进制小数。

4.1 通知

没有包含“id”成员的请求对象为通知， 作为通知的请求对象表明客户端对相应的响应对象并不感兴趣，本身也没有响应对象需要返回给客户端。服务端必须不回复一个通知，包含那些批量请求中的。

由于通知没有返回的响应对象，所以通知不确定是否被定义。同样，客户端不会意识到任何错误（例如参数缺省，内部错误）。

4.2 参数结构

rpc 调用如果存在参数则必须为基本类型或结构化类型的参数值，要么为索引数组，要么为关联数组对象。

- 索引：参数必须为数组，并包含与服务端预期顺序一致的参数值。
- 关联名称：参数必须为对象，并包含与服务端相匹配的参数成员名称。没有在预期中的成员名称可能会引起错误。名称必须完全匹配，包括方法的预期参数名以及大小写。

5.响应对象

当发起一个 rpc 调用时，除通知之外，服务端都必须回复响应。响应表示为一个 JSON 对象，使用以下成员：

jsonrpc

指定 JSON-RPC 协议版本的字符串，必须准确写为“2.0”

result

该成员在成功时必须包含。

当调用方法引起错误时必须不包含该成员。
服务端中的被调用方法决定了该成员的值。

error
该成员在失败是必须包含。
当没有引起错误的时必须不包含该成员。
该成员参数值必须为 5.1 中定义的对象。

id
该成员必须包含。
该成员值必须于请求对象中的 id 成员值一致。
若在检查请求对象 id 时错误（例如参数错误或无效请求），则该值必须为空值。
响应对象必须包含 result 或 error 成员，但两个成员必须不能同时包含。

5.1 错误对象

当一个 rpc 调用遇到错误时，返回的响应对象必须包含错误成员参数，并且为带有下列成员参数的对象：

code
使用数值表示该异常的错误类型。 必须为整数。

message
对该错误的简单描述字符串。 该描述应尽量限定在简短的一句话。

data
包含关于错误附加信息的基本类型或结构化类型。该成员可忽略。 该成员值由服务端定义（例如详细的错误信息，嵌套的错误等）。
-32768 至-32000 为保留的预定义错误代码。在该范围内的错误代码不能被明确定义，保留下列以供将来使用。错误代码基本与 XML-RPC 建议的一样，url：http://xmlrpc-epi.sourceforge.net/specs/rfc.fault_codes.php

code	message	meaning
-32700	Parse error 语法解析错误	服务端接收到无效的 json。该错误发送于服务器尝试解析 json 文本
-32600	Invalid Request 无效请求	发送的 json 不是一个有效的请求对象。
-32601	Method not found 找不到方法	该方法不存在或无效
-32602	Invalid params 无效的参数	无效的方法参数。
-32603	Internal error 内部错误	JSON-RPC 内部错误。
-32000 to -32099	Server error 服务端错误	预留用于自定义的服务器错误。

除此之外剩余的错误类型代码可供应用程序作为自定义错误。

6.批量调用

当需要同时发送多个请求对象时，客户端可以发送一个包含所有请求对象的数组。
当批量调用的所有请求对象处理完成时，服务端则需要返回一个包含相对应的响应对象数组。每个响应对象都应对应每个请求对象，除非是通知的请求对象。服务端可以并发的，以任意顺序和任意宽度的并行性来处理这些批量调用。

这些相应的响应对象可以任意顺序的包含在返回的数组中，而客户端应该是基于各个响应对象中的 id 成员来匹配对应的请求对象。
若批量调用的 rpc 操作本身非一个有效 json 或一个至少包含一个值的数组，则服务端返回的将单单是一个响应对象而非数组。若批量调用没有需要返回的响应对象，则服务端不需要返回任何结果且必须不能返回一个空数组给客户端。

7.示例

Syntax:

```
--> data sent to Server
<-- data sent to Client
```

带索引数组参数的 rpc 调用:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
<-- {"jsonrpc": "2.0", "result": 19, "id": 1}

--> {"jsonrpc": "2.0", "method": "subtract", "params": [23, 42], "id": 2}
<-- {"jsonrpc": "2.0", "result": -19, "id": 2}
```

带关联数组参数的 rpc 调用:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahend": 23, "minuend": 42}, "id":
3}
<-- {"jsonrpc": "2.0", "result": 19, "id": 3}

--> {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend": 42, "subtrahend": 23}, "id":
4}
<-- {"jsonrpc": "2.0", "result": 19, "id": 4}
```

通知:

```
--> {"jsonrpc": "2.0", "method": "update", "params": [1,2,3,4,5]}
--> {"jsonrpc": "2.0", "method": "foobar"}
```

不包含调用方法的 rpc 调用:

```
--> {"jsonrpc": "2.0", "method": "foobar", "id": "1"}
<-- {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id": "1"}
```

包含无效 json 的 rpc 调用:

```
--> {"jsonrpc": "2.0", "method": "foobar, "params": "bar", "baz]
<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse error"}, "id": null}
```

包含无效请求对象的 rpc 调用:

```
--> {"jsonrpc": "2.0", "method": 1, "params": "bar"}
<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
```

包含无效 json 的 rpc 批量调用:

```
--> [
      {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},
        {"jsonrpc": "2.0", "method"
    ]
<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse error"}, "id": null}
```

包含空数组的 rpc 调用:

```
--> []
<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
```

非空且无效的 rpc 批量调用:

```
--> [1]
<-- [
    {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
    ]
```

无效的 rpc 批量调用:

```
--> [1,2,3]
<-- [
    {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null},
    {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null},
    {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
    ]
```

rpc 批量调用:

```
--> [
    {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},
    {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]},
    {"jsonrpc": "2.0", "method": "subtract", "params": [42,23], "id": "2"},
    {"foo": "boo"},
    {"jsonrpc": "2.0", "method": "foo.get", "params": {"name": "myself"}, "id": "5"},
    {"jsonrpc": "2.0", "method": "get_data", "id": "9"}
    ]
<-- [
    {"jsonrpc": "2.0", "result": 7, "id": "1"},
    {"jsonrpc": "2.0", "result": 19, "id": "2"},
    {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null},
    {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id": "5"},
    {"jsonrpc": "2.0", "result": ["hello", 5], "id": "9"}
    ]
```

所有都为通知的 rpc 批量调用:

```
--> [
    {"jsonrpc": "2.0", "method": "notify_sum", "params": [1,2,4]},
    {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]}
]
```

```
<-- //Nothing is returned for all notification batches
```

7.扩展

以 rpc 开头的方法名预留作为系统扩展，且必须不能用于其他地方。每个系统扩展都应该有相关规范文档，所有系统扩展都应是可选的。

原文 2：

Zigbee Wireless Sensor Network in Environmental Monitoring Applications

I. ZIGBEE TECHNOLOGY

Zigbee is a wireless standard based on IEEE802. 15. 4 that was developed to address the unique needs of most wireless sensing and control applications. Technology is low cost, low power, a low data rate, highly reliable, highly secure wireless networking protocol targeted towards automation and remote control applications. It's depicts two key performance characteristics – wireless radio range and data transmission rate of the wireless spectrum. Comparing to other wireless networking protocols such as Bluetooth, Wi-Fi, UWB and so on, shows excellent transmission ability in lower transmission rate and highly capacity of network.

A. Zigbee Framework

Framework is made up of a set of blocks called layers. Each layer performs a specific set of services for the layer above. As shown in Fig.1. The IEEE 802. 15. 4 standard defines the two lower layers: the physical (PHY) layer and the medium access control (MAC) layer. The Alliance builds on this foundation by providing the network and security layer and the framework for the application layer.

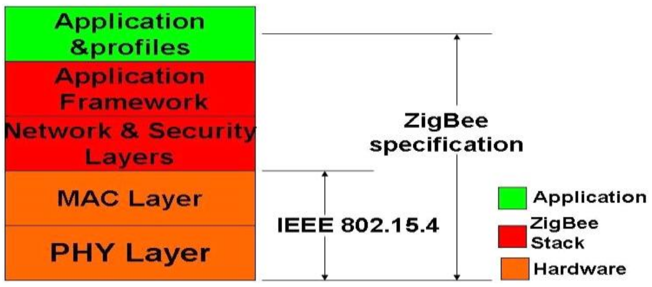

Fig.1 Framework

The IEEE 802. 15. 4 has two PHY layers that operate in two separate frequency ranges: 868/915 MHz and 2. 4GHz. Moreover, MAC sub-layer controls access to the radio channel using a CSMA-CA mechanism. Its responsibilities may also include transmitting beacon frames, synchronization, and providing a reliable transmission mechanism.

B. Zigbee's Topology

The network layer supports star, tree, and mesh topologies, as shown in Fig.2. In a star topology, the network is controlled by one single device called coordinator. The coordinator is responsible for initiating and maintaining the devices on the network. All other devices, known as end devices, directly communicate with the coordinator. In mesh and tree topologies, the coordinator is responsible for starting the network and for choosing certain key network parameters, but the network may be extended through the use of routers. In tree networks, routers move data and control messages through the network using a hierarchical routing strategy. Mesh networks allow full peer-to-peer communication.

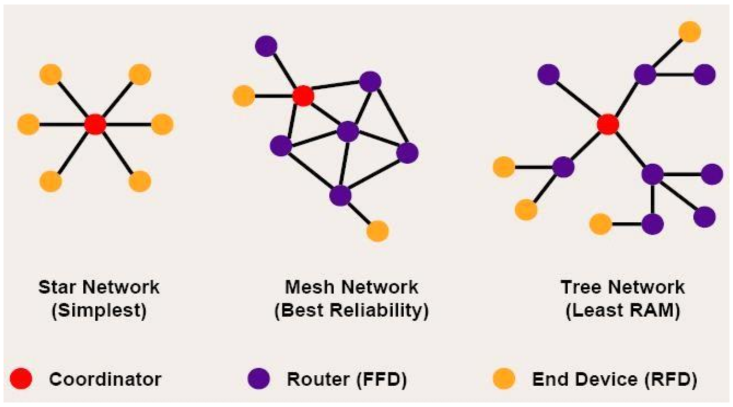

Fig.2 Mesh topologies

Fig.3 is a network model, it shows that supports both single-hop star topology constructed with one coordinator in the center and the end devices, and mesh topology. In the network, the intelligent nodes are composed by Full Function Device (FFD) and Reduced Function Device (RFD). Only the FFN defines the full functionality and can become a network coordinator. Coordinator manages the network, it is to say that coordinator can start a network and allow other devices to join or leave it. Moreover, it can provide binding and address-table services, and save messages until they can be delivered.

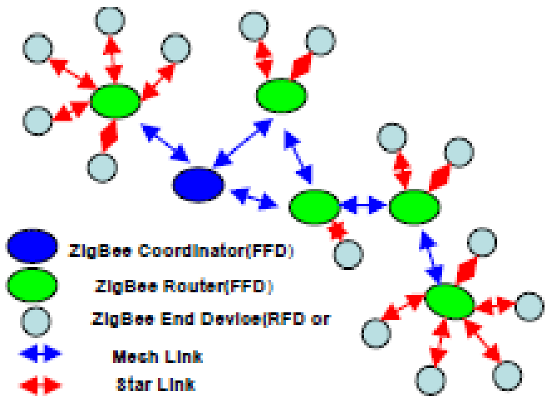

Fig.3 Zigbee network model

II. THE GREENHOUSE ENVIRONMENTAL MONITORING SYSTEM DESIGN

Traditional agriculture only use machinery and equipment which isolating and no communicating ability. And farmers have to monitor crops' growth by themselves. Even if some people use electrical devices, but most of them were restricted to simple communication between control computer and end devices like sensors instead of wire connection, which couldn't be strictly defined as wireless sensor network. Therefore, by through using sensor networks and, agriculture could become more automation, more networking and smarter.

In this project, we should deploy five kinds of sensors in the greenhouse basement. By through these deployed sensors, the parameters such as temperature in the greenhouse, soil temperature, dew point, humidity and light intensity can be detected real time. It is key to collect different parameters from all kinds of sensors. And in the greenhouse, monitoring the vegetables growing conditions is the top issue. Therefore, longer battery life and lower data rate and less complexity are very important. From the introduction about above, we know that meet the requirements for reliability, security, low costs and low power.

A. System Overview

The overview of Greenhouse environmental monitoring system, which is made up by one sink node (coordinator), many sensor nodes, workstation and database. Mote node and sensor node together composed of each collecting node. When sensors collect parameters real time, such as temperature in the greenhouse, soil temperature, dew point, humidity and light intensity, these data will be offered to A/D converter, then by through quantizing and encoding become the digital signal that is able to transmit by wireless sensor communicating node. Each wireless sensor communicating node has ability of transmitting, receiving function.

In this WSN, sensor nodes deployed in the greenhouse, which can collect real time data and transmit data to sink node (Coordinator) by the way of multi-hop. Sink node complete the task of data analysis and data storage. Meanwhile, sink node is connected with GPRS/CDMA can provide remote control and data download service. In the monitoring and controlling room, by running greenhouse management software, the sink node can periodically receives the data from the wireless sensor nodes and displays them on monitors.

B. Node Hardware Design

Sensor nodes are the basic units of WSN. The hardware platform is made up sensor nodes closely related to the specific application requirements. Therefore, the most important work is the nodes design which can perfect implement the function of detecting and transmission as a WSN node, and perform its technology characteristics. Fig.4 shows the universal structure of the WSN nodes. Power module provides the necessary energy for the sensor nodes. Data collection module is used to receive and convert signals of sensors. Data processing and control module's functions are node device control, task scheduling, and energy computing and so on. Communication module is used to send data between nodes and frequency chosen and so on.

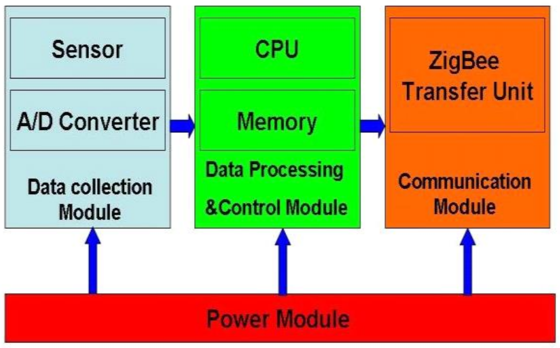

Fig.4 Universal structure of the wsn nodes

In the data transfer unit, the module is embedded to match the MAC layer and the NET layer of the protocol. We choose CC2430 as the protocol chips, which integrated the CPU, RF transceiver, net protocol and the RAM together. CC2430 uses an 8 bit MCU (8051), and has 128KB programmable flash memory and 8KB RAM. It also includes A/D converter, some Timers, AES128 Coprocessor, Watchdog Timer, 32K crystal Sleep mode Timer, Power on Reset, Brown out Detection and 21 I/Os. Based on the chips, many modules for the protocol are provided. And the transfer unit could be easily designed based on the modules.

As an example of a sensor end device integrated temperature, humidity and light, the design is shown in Fig. 5.

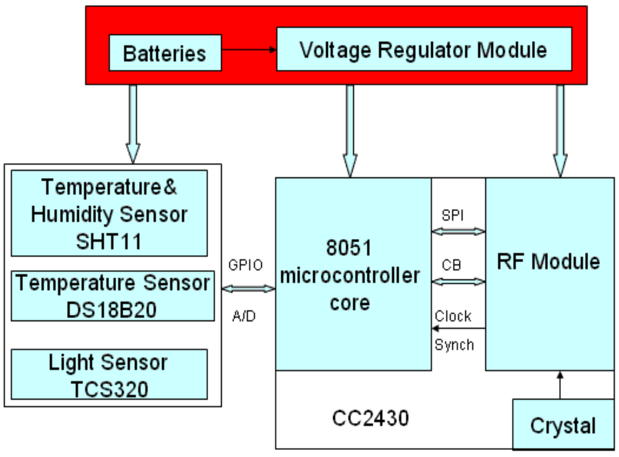

Fig.5 The hardware design of a sensor node

The SHT11 is a single chip relative humidity and temperature multi sensor module comprising a calibrated digital output. It can test the soil temperature and humidity. The DS18B20 is a digital temperature sensor, which has 3 pins and data pin can link MSP430 directly. It can detect temperature in greenhouse. The TCS320 is a digital light sensor. SHT11, DS18B20 and TCS320 are both digital sensors with small size and low power consumption. Other sensor nodes can be obtained by changing the sensors.

The sensor nodes are powered from onboard batteries and the coordinator also allows to be powered from an external power supply determined by a jumper.

C. Node Software Design

The application system consists of a coordinator and several end devices. The general structure of the code in each is the same, with an initialization followed by a main loop.

The software flow of coordinator, upon the coordinator being started, the first action of the application is the initialization of the hardware, liquid crystal, stack and application variables and opening the interrupt. Then a network will be formatted. If this net has been formatted successfully, some network information, such as physical address, net ID, channel number will be shown on the LCD. Then program will step into application layer and monitor signal. If there is end device or router want to join in this net, LCD will shown this information, and show the physical address of applying node, and the coordinator will allocate a net address to this node. If the node has been joined in this network, the data transmitted by this node will be received by coordinator and shown in the LCD.

The software flow of a sensor node, as each sensor node is switched on, it scans all channels and, after seeing any beacons, checks that the coordinator is the one that it is looking for. It then performs a synchronization and association. Once association is complete, the sensor node enters a regular loop of reading its sensors and putting out a frame containing the sensor data. If sending successfully, end device will step into idle state; by contrast, it will collect data once again and send to coordinator until sending successfully.

D. Greenhouse Monitoring Software Design

We use VB language to build an interface for the test and this greenhouse sensor network software can be installed and launched on any Windows-based operating system. It has 4 dialog box selections: setting controlling conditions, setting Timer, setting relevant parameters and showing current status. By setting some parameters, it can perform the functions of communicating with port, data collection and data viewing.

译文 2：

Zigbee 无线传感器网络在环境监测中的应用

I. Zigbee 技术

Zigbee 是一种基于 IEEE802.15.4 的无线标准上被开发用来满足大多数无线传感和控制应用的独特需求。Zigbee 技术是低成本，低功耗，低数据速率，高可靠性，高度安全的无线网络协议实现自动化和远程控制应用的目标。它描述了两个关键的性能特点—无线射频范围和无线频谱的数据传输速率。相较于其他如蓝牙，Wi-Fi 技术，超宽带等无线网络协议，Zigbee 虽然传输速率慢但传输容量大的特点向我们展示了他出色的传输能力。

A、技术框架

Zigbee的框架是由一组层组成的。上述层中每一层都要执行一组特定的服务任务。图1所示。在IEEE802.15.4标准定义了两个较低层：物理层（PHY）和媒体接入控制（MAC）层。Zigbee联盟建立在网络层和安全层及应用层框架提供的基础上。

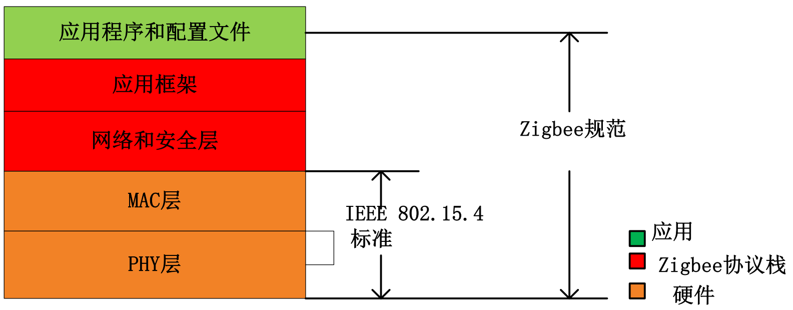

图1 技术框架

在IEEE802.15.4有两个PHY层，它们在两个不同的频率范围操作：868/915兆赫和2.4GHz。此外，MAC子层控制访问无线电频道使用的CSMA- CA的机制。它的功能还可以包括信标帧传输，同步，并提供一个可靠的传输机制。

B、Zigbee技术的拓扑

Zigbee网络层支持星形，树形和网状形拓扑结构，如图2所示。在星型拓扑结构中，网络是由一个叫做Zigbee协调器的单一设备控制的。 Zigbee协调器负责发起和维护网络上的设备。所有其他装置，称为终端设备，直接与Zigbee协调器相连通。在网状和树状拓扑结构中，Zigbee协调器的作用是启动网络，并选择一些重要的网络参数，但网络可以通过Zigbee路由器扩展。在树状网络中，路由器

将通过使用分层路由策略移动数据和控制消息。网状网络允许完全对等的对等通信。

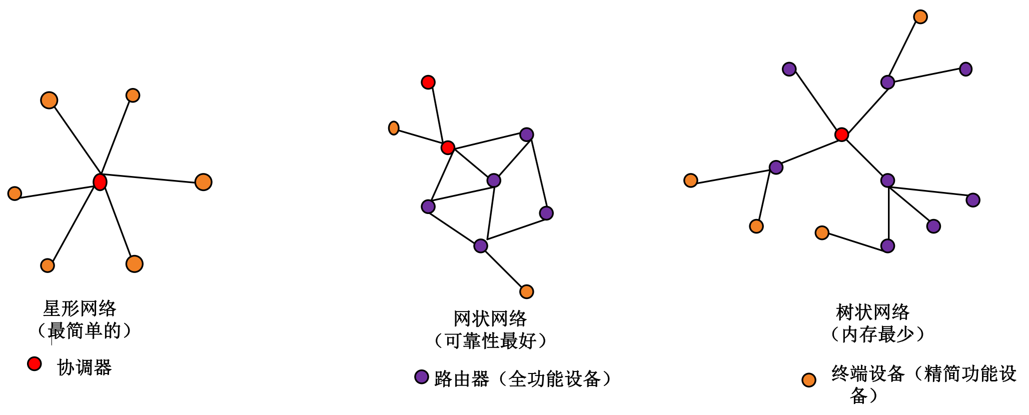

图2 技术的拓扑

图 3 是一个 Zigbee 网络模型，它表明 Zigbee 支持协调器中心的单跳星形拓扑结构和终端设备，以及网状拓扑构造。在 Zigbee 网络中，智能节点由全功能设备（FFD）和精简功能设备（RFD）组成。只有 FFN 定义了完整的 Zigbee 功能，并且可成为网络协调器。协调器管理网络，也就是说，协调器可以启动网络，并允许其他设备加入或离开它。此外，它可以提供绑定和地址表服务，并保存，直到他们能传递信息。

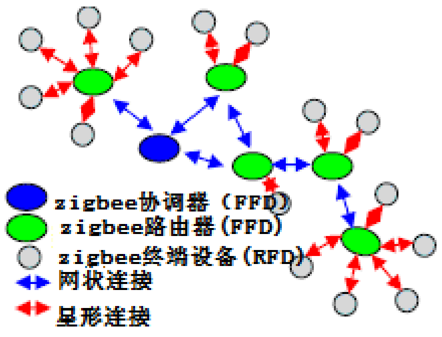

图3 Zigbee网络模型

II.温室环境监测的系统设计

传统农业只使用孤立和没有沟通能力的机器和设备。农民们必须自己亲自监控作物的生长。即使有些人用电气设备，但他们中大多只限于控制计算机和终端设备的简单通信，此终端设备像传感器而不是像线相连接的传感器，严格上说不能被定义为无线传感器网络。因此，通过使用传感器网络和Zigbee，农业可能变得更加自动化，更加的网络化和智能化。

在这个项目中，我们要在温室的地下室部署五种传感器。通过这些部署的传感器，如温室的温度，土壤温度，露点，湿度和光照强度的参数可以实时检测。它的关键是从各种不同的传感器来收集不同的参数。而在温室，监测蔬菜的长势是首要问题。因此，延长电池的寿命，减小数据速率和降低复杂度是非常重要的。从上述关于 Zigbee 的介绍，我们知道 Zigbee 满足了可靠性，安全性，低成本，低功耗的要求。

A、系统概述

温室环境监测系统是由一个接收器节点（协调器），许多传感器节点，工作站和数据库组成的。莫特节点和传感器节点共同组成了每个收集节点。当传感器参数进行实时采集，如温室温度，土壤温度，露点，湿度和光照强度，这些数据将提供给的 A / D 转换器，然后透过量化和编码成为数字信号，它能通过无线传感器通信节点传送。每个无线传感器通信节点有传送和接收的能力。

在这种传感器网络中，传感器节点部署在温室，它可以采集实时数据和通过多跳方式传送数据到接收器节点（协调器）。接收器节点完成了数据分析和贮存的任务。同时，接收器节点与 GPRS/CDMA 连接可以提供远程控制和数据下载服务。在监控室通过运行温室管理软件，接收器节点可以定期收到来自无线传感器节点和在监视器上显示这些数据。

B、节点的硬件设计

传感器节点是无线传感器网络的基本单位。硬件平台是由密切相关的具体应用要求的传感器节点组成的。因此，最重要的工作是节点设计，可以完美执行无线传感器网络的传送和监测功能，，并体现了Zigbee的技术特点。图4显示了无线传感器网络节点的普遍结构。电源模块为传感器节点提供了必要的能量。数据采集模块被用来接收和转换传感器的信号。数据处理和控制模块的功能是节点设备控制，任务调度，能量计算等。通讯模块被用来在节点与频率选择之间传送数据等。

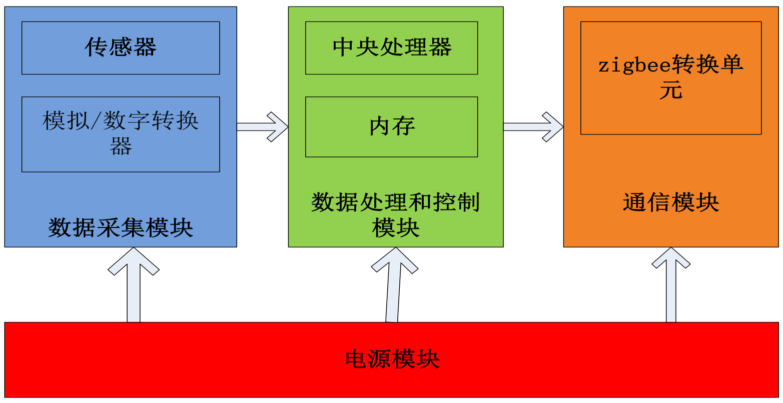

图 4 无线传感器网络节点的通用结构

在数据传输单元，Zigbee 模块是嵌入式的用来相匹 Zigbee 协议的 MAC 层和 NET 层。我们选择 CC2430 作为 zigbee 协议的芯片，它把 CPU，射频收发器，网络协议和 RAM 集合在一起。CC2430 运用一个 8 比特的微控制器（8051），并具有 128KB 可编程闪存和 8KB 的 RAM。它还包括 A / D 转换，某些计时器，AES128 协处理器，看门狗定时器，32K 的晶体休眠模式定时器，上电复位，掉电检测和 21 个 I / O 操作系统。基于主芯片、为 Zigbee 协议提供许多模块。在那些模块的基础上 Zigbee 传输单元可以很容易地被设计出来。

以一个集成温度、湿度和光照的传感器终端设备为例，设计如图 5 所示。

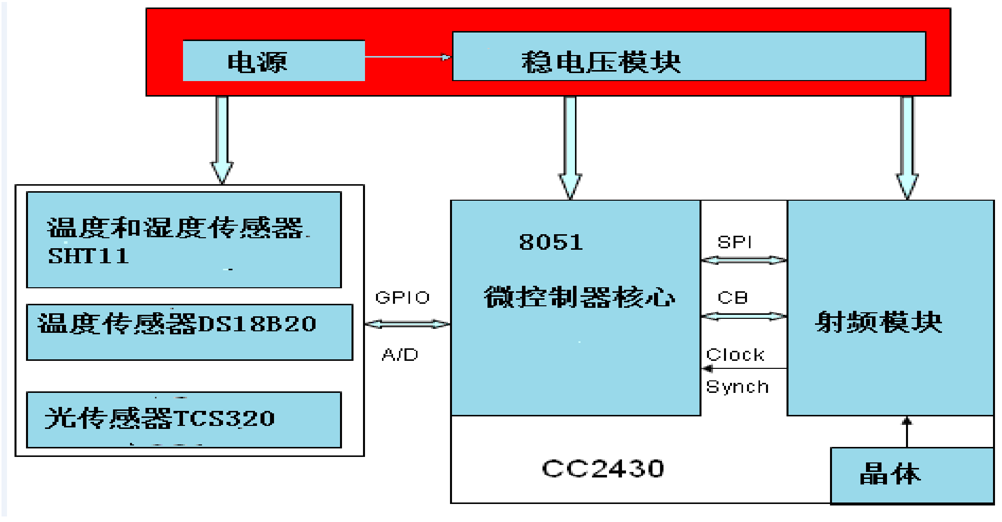

图 5 传感器节点的硬件设计

该 SHT11 是一种相对于湿度和温度的多传感器模块包括校准的数字输出的单芯片。它可以测试土壤温度和湿度。 DS18B20 的数字温度传感器，它有 3 个引脚，并且数据引脚可以直接连接 MSP430。它可以检测温室的温度。TCS320 是一种数字光传感器。DS18B20 和 TCS320 SHT11,都是数字传感器具有体积小、功耗低的特点。其他传感器节点可以通过改变传感器获得。

传感器节点由供电板载电池供电，协调器还允许通过跳线由外部电源跳线确供电。

C、节点的软件设计

应用系统由一个协调器和几个终端设备组成。每个代码的一般结构是相同的，一个主循环后初始化。

协调器软件流程，经协调器开始，应用程序的第一步是硬件，液晶，栈和应用程序变量的初始化并且开放中断。然后，一个网络将被格式化。如果这个网络已被格式化成功，一些网络信息，如物理地址，网络 ID，通道号，将会显示在液晶显示屏上。然后，程序将进入应用层和监测 Zigbee 信号。如果有终端设备或路由器想要加入这一网络，液晶显示屏将显示此信息，并显示了应用节点的物理地址，协调员将分配一个网络地址到该节点。如果节点已加入了这个网络，数据由此节点传送，将由协调器接收，并且显示在液晶显示器上。

一个传感器节点软件流程，当每个传感器节点被打开或者在遇到任何航标后一个正在被寻找的协调器被检测到时，它会扫描所有频道。然后执行同步和连接。一旦完成连接，传感器节点便会进入阅读传感器和输出包含节点数据框架的定期循环、如果发送成功，终端设备将进入空闲状态，相反，它会再次收集数据并且发送到协调器，直到发送成功。

D、温室监控软件的设计

我们用 VB 语言为测试来构建一个界面，这温室传感器网络的软件，可以安装任何基于 Windows 操作系统。它有 4 个对话框选择：设置控制条件，设置定时器，设定相关参数，并显示当前的状态。通过设置一些参数，它可以执行与港口沟通的功能，数据收集和数据浏览。