An introduction to Firebase on Raspberry Pi

Microprocessor Systems and Applications

Copyright © 2018 Claudiu Groza

GITHUB.COM/CLAUDIUGROZA/MSA

March 2018

Contents

1. Firebase

1.1 Problem statement

The following sections will illustrate some basic operations that can be achieved by working with a real time database service. We warm up with an example that fetches values from a node which contains information related to temperature and humidity. Afterwards, we demonstrate how to push the current state of a button to an upstream node. Finally, we take a look on how to listen for different events using a web client script.

1.2 Firebase API

Firebase realtime database provides an application programming interface which allows developers to store and synchronise data across multiple clients [Fira]. In order to use any of the provided platform services, you first need to setup a Firebase project.

1.2.1 Library support

Firebase provides a REST inferface out of the box. However, we knowingly skip the steps that involves composing plain HTTP requests and use a third party wrapper library [Firb]. By making this decision you will have more time to spend on implementing desired functionalities.

Now, make sure you execute the following commands to setup the environment we need:

```
$ sudo pip install requests==1.1.0
$ sudo pip install python-firebase
```

1.3 Reading data

The current section shows an example of fetching data contained into a database node. We may assume the node contains temperature and humidity data. The information is uploaded by another application, but our initially defined purpose is to read that information.

We start by importing the packages we need in our implementation:

```
from firebase import firebase
import time
```

The next step involves Firebase instance configuration:

```
# generated root for your project
FIREBASE_ROOT = 'https://ms-iot.firebaseio.com'
# initialize Firebase database instance
firebase = firebase.FirebaseApplication(FIREBASE_ROOT, None)
```

You may not believe but we are actually prepared to read data from an arbitrary specified node:

```
while True:
    # execute a GET request on the node
    result = firebase.get('/sensor', None)
    # log the result
    print result
    # wait 1 second between two consecutive requests
    time.sleep(1)
```

Make sure you execute the script and observe how environment values change over time.

1.4 Posting data

We are now familiar of how to read data stored into a Firebase node. This section shows how we can write data to a specific node. The following example demonstrates only the part of writing the button state to Firebase. You are fully responsible of actually reading the state of the button and passing the value to the function we provide.

We do not define the initialization part again because it is likely the same as the previous section has shown.

In order to uniquely identify each of the boards, we consider the host name attribute to be the key of the node we use to write data:

```
# find a local unique identifier
BOARD_NAME = socket.gethostname()

def write_button_state(state):
```

```
    # build the dict to be pushed
    value = {BOARD_NAME : state}
    # prefer update of the node
    result = firebase.patch('/boards', value)
    print result

# mock a value and then push it
write_button_state('UNKNOWN')
```

1.5 Observing data

The true power of Firebase database comes from the ability to listen (observe) change events to a node. We will take a look on that by using a JavaScript web client.

Below is the complete part that exemplifies an approach of observing data contained by a node and then reacting to the nature of any change:

```
// init the node reference path
var buttons = new Firebase('https://ms-iot.firebaseio.com/boards');

// refresh the list in HTML
function refreshList(children) {
    var sections = '';
    for (var i = 0; i < children.length; i++) {
        sections += '<li>' + children[i].key + ': ' + children[i].state
                    + '</li>';
};
    document.getElementById('buttons').innerHTML = sections;
};

// listen for value changes on children
buttons.on("value", function(snapshot) {
    var data = snapshot.val();
    var children = [];
    for (var key in data) {
      state = data[key];
      if (key.trim().length > 0) {
          children.push({
              state: state,
key: key })
} }
    refreshList(children);
});
```

We are ready to test our solution from end to end. Open the *app.html* file via a modern

web browser and then trigger a state change. You can simulate a change by directly modifying the node via Firebase console or use the script provided in the previous chapter.

2. Assignments

1. Extend *write_button_state.py* script such that listens for push-button events. Once an event has occured then it reads the new state and pushes the value to Firebase node. Make sure to use the following literals for button states:
   - PUSHED: state indicates the button was pressed
   - RELEASED: state indicates the button is not pressed

3. Bibliography

3.1 References

[Fira] *Firebase platform*. https://firebase.google.com. [Online; accessed March-2018]. 2018 (cited on page 5).

[Firb] *Firebase python wrapper library*. https://https://pypi.python.org/pypi/python-firebase/1.2. [Online; accessed March-2018]. 2018 (cited on page 5).

3.2 Image credits

- First page illustration.
  http://www.northeastern.edu/levelblog/2018/01/25/guide-iot-careers
- Chapter header background.
  https://blogs.microsoft.com/iot/2015/03/17/simplifying-iot/