Dripcast

Server-less Java programming framework for Device Applications

Ikuo Nakagawa
Dripcast Project
July, 2012
Quick Try...

Collecting GPS information from thousands of cabs & show location, Next, select cabs near from "a customer" who wants a cab
Bibbidi Bobbidi Boo
(also called "The Magic Song")
is a novelty song, written in 1948 by Al Hoffman, Mack David, and Jerry Livingston.
It was introduced in the 1950 Disney film Cinderella, performed by actress Verna Felton.
// an instance we want to handle.
Gadget a = new GadgetImpl();

// store out value to the object, at first.
a.initValue();

// now, verify input against our value.
int v = Integer.parseInt(in.readLine());
if (a.verifyValue(v)) {
    System.out.println("Great! Lucky guess!");
} else {
    System.out.println("No match. Sorry.");
}

interface Gadget {
    void initValue();
    boolean verifyValue(int value);
}

class GadgetImpl implements Gadget {
    private int answer;
    public void initValue() {
        answer = new Random().nextInt(16);
    }
    public boolean verifyValue(int value) {
        return answer == value;
    }
}

Let's make an object to be cloud-enabled.
enable your application to be "cloud-ful"

interface Gadget {
    void initValue();
    boolean verifyValue(int value);
}

class GadgetImpl implements Gadget {
    private int answer;
    public void initValue() {
        answer = new Random().nextInt(16);
    }
    public boolean verifyValue(int value) {
        return answer == value;
    }
}

What you need to do, is only casting "Bibbidi-Babbidi-Boo"!
on Android
Example of "sharing GPS location"

3 minutes implementation for sharing GPS information via cloud. What you need to do, is attach your object with the cloud. That's all.

```java
public class YourActivity implements Activity, LocationListener {
    UUID key = UUID.fromString("your-group-UUID"); // to share an object between group members.
    Map map; // will be associated with a cloud object, e.g., cloud-enabled.
    String id; // to store my phone number.

    public void onCreate() {
        map = B3.createOrAttachInstance(key, NavigableMap.class, "java.util.HashMap");
        id = ((TelephonyManager)getSystemService(TELEPHONY_SERVICE)).getLine1Number();
        ((LocationManager)getSystemService(LOCATION_SERVICE)).requestLocationUpdates(
            LocationManager.GPS_PROVIDER, 0, 0, this);
    }

    public void onLocationChanged(Location location) {
        int x = (int)(location.getLatitude() * 1E6);
        int y = (int)(location.getLongitude() * 1E6);
        map.put(id, x + "," + y);
        // show, your members positions....
        for (Entry entry : map.entrySet()) {
            while (entry != null) {
                // show id and location, e.g.,
                // entry.getKey() and entry.getValue()
                }
            }
        }
    }
```

Getting GPS info of your party members.

Your models are:
- cloud-full computing
- server-less programming

http://dripcast.org
Challenges
Server-less programming model

Past: 3 layer model (client+server+database) as application
Dripcast: devices (by applications) and platform (by service operator)

- GUI / design
- Application Logic
- Server side (Servlet)
- Table / Data (RDB)

via REST/HTTP
via SQL

application provider takes care of all of client+server+database

Past

Dripcast

app. logic and UI by application provider

Service operator provides, resources for you data

Platform (cloud)

Application Logic

simple & easy with server-less model (no need of SQL nor REST/HTTP)

July, 2012
© Dripcast Project
Dripcast clouds in Japan

6 Dripcast clouds are available in Japan, for trial members
Thanks for resources and environment!

A.T.Works, Toyama
HOTnet, Sapporo
Ehime CATV, Matsuyama
u-tokyo, Tokyo
ASTEM, Kyoto
WIDE, Tokyo
Inside Dripcast

basic architecture of Dripcast mechanism
Basic architecture of Dripcast

Client / Device
"Server-less" application on ANDROID or Java
- billions of devices from world wide
- object-key and method call front

Relay
Distribution, Auth., SSO, Security check, etc
- deliver to engines by object-keys
- state-less, so infinitely scalable

Engine
Processing Java method with app. features
- load / store objects + invoke method
- has distribution strategy, ex. consistent hash
- reliable and scalable object store

Store
Storing objects, with simple API
- Java object storage as backend store

parallel & distributed processing

processing engine

object storage

processing + auth.
How to use Dripcast

Let's start to play the Dripcast
Easy to start with standard JDK classes

Cast a magic spell, to attach standard JDK classes.
That's all.

**Create an object**

d.createInstance(key, "java.util.HashMap");

**Access to the object**

Map map;
map = d.attachInstance(key, Map.class);

map.put("name", "Mike");
map.get("name");
Using your own Java libraries

Compile and upload your jar file into the Dripcast cloud. You can attach your own class, as well.

Create an object

d.createInstanc(key, "YourClass");

public interface YourInterface {
    int calculate(int n);
}

public class YourClass implements YourInterface {
    public int calculate(int n) {
        int sum = 0;
        for (int i = 1; i <= n; i++) {
            sum += i;
        }
        return sum;
    }
}

Access the object

YourInterface yi;
yi = d.sttachInstance(key, YourInterface.class);

int sum = yi.calculate(10);
int sum2 = yi.calculate(50);

compile and upload at first

YourClass
Save your objects

Dripcast supports JumboMap (highly scalable Map), by default
You can store Dripcast objects into user's Map.

**Magic Spell**

getUserMap() supported to access user's JumboMap

```java
Map<String, Object> map;
map = d.getUserMap();
map.put("540-xxx-yyyy", "George");
map.put("706-yyy-zzzz", "Mike");
```

JumboMap is very scalable
Map style data store
You can save millions of data objects on the cloud via Map interface
Walk through JumboMap to execute your original TASK, in parallel & distributed manner

**Define Task**

```java
public class YourTask implements WalkTask {
    public void process(UUID key, Object object) {
        // process, here.
    }
}
```

**Apply Task**

```java
WalkTask task = new YourTask();
d.walkTask(d.getUserMap(), task);
```

compile and upload at first

process task through JumboMap
Apply Dripcast to Taxi location management
Now, Collecting GPS info.

Define very simple class to store GPS information.
Associate GPS info with phone-id. it’s easy.

```java
public class YourActivity implements Activity, LocationListener {
    private String id;
    private GPSSensor sensor;

    public void onCreate() {
        id = ((TelephonyManager)getSystemService(TELEPHONY_SERVICE)).getLine1Number();
        ((LocationManager)getSystemService(LOCATION_SERVICE)).requestLocationUpdates(
            LocationManager.GPS_PROVIDER, 0, 0, this);
        Dripcast d = new DripcastBuilder().open();
        ConcurrentMap<String, Object> map = d.getUserMap();
        GPSSensor sensor = map.get(id);
        if (sensor == null) {
            GPSSensor tmp = d.newInstance(GPSSensor.class, GPSSensorImpl.class.getName());
            sensor = map.putIfAbsent(id, tmp);
            if (sensor == null) { sensor = tmp; }
        }

        public void onLocationChanged(Location location) {
            sensor.setLocation(new double[]{location.getAltitude(), location.getLongitude(), location.getAltitude()});
        }
    }

    public void setLocation(double[] values) {
        double[] getLocation() {
            return new double[]{latitude, longitude, altitude};
        }
    }
}
```
public class TabletActivity implements Activity {
    ....

    public void onResume() {
        Dripcast d = new DripcastBuilder().open();
        ConcurrentMap<String, Object> map = d.getUserMap();
        for (String id : map.keySet()) {
            GPSSensor = map.get(id);
            double[] values = sensor.getLocation();
            double latitude = values[0];
            double longitude = values[1];
            double altitude = values[2];
            // show the location of the node.
            // maybe on the Google Map or others.
        }
    }
}

public interface GPSSensor {
    void setLocation(double[] values);
    double[] getLocation();
}

public class GPSSensorImpl implements GPSSensor {
    private double latitude = Double.MAX_VALUE;
    private double longitude = Double.MAX_VALUE;
    private double altitude = Double.MAX_VALUE;
    private long time = -1L;

    public void setLocation(double[] values) {
        latitude = values[0];
        longitude = values[1];
        altitude = values[2];
        time = System.currentTimeMillis();
    }

    public double[] getLocation() {
        return new double[]{latitude, longitude, altitude};
    }
}
public class GPSSensorWalkerTest implements Runnable
{
    public void run()
    {
        Dripcast d = new DripcastBuilder().open();
        Map map = d.newInstance(Map.class, "java.util.HashMap");
        WalkTask task = new GPSSensorWalker(map);
        d.walkUserObjects(task);
        log.warn("There are " + map.size + " objects in Ginza.");
    }
}

// your own implementation for "processing"
public class GPSSensorWalker implements WalkTask
{
    private final Map map;
    GPSSensorWalker(Map map)
    {
        this.map = map;
    }
    public void process(UUID key, Object object)
    {
        if (!(object instanceof GPSSensor)) { return; }
        GPSSensor sensor = (GPSSensor)object;
        double[] values = sensor.getLocation();
        double deltaX = values[0] - 35.671992; // X of Ginza
        double deltaY = values[1] - 139.763926; // Y of Ginza
        double r = Math.sqrt(deltaX * deltaX + deltaY * deltaY);
        if (r < 1.0) {
            map.put(key, values[0] + "," + values[1]);
        }
    }
}

Parallel & distributed processing for stored objects,
You can apply a "process" to your objects, on a cloud.
Dripcast for device applications
Dripcast for "device + cloud" applications

Dripcast mechanism is suitable for both storing device status, and processing those data for analytics

without server-side programming
nor operation
processing device status information as well as storing such data

Virtual space for user
update status (via Dripcast)

Physical space for user
Control physical devices

IEEE1888, etc

HR

control device (via Dripcast)

Applications on a platform (using resources/features on a cloud)

virtual devices on a cloud
Server-less means "focusing into logic"

**device / user**
- data (valuation)
- visualization (UI)
- application logic
  - e.g., business logic

**cloud / servicer**
- resource
- scalability / elasticity
- reliability
- M/W, P/F operation

*Focus, here!*
Visit  
http://dripcast.org

or contact to

info@dripcast.org

for more information.

Thank you.