Archived:J2ME Google Maps API (Português)

Aquivado: Este artigo foi arquivado, pois o conteúdo não é mais considerado relevante para se criar soluções comerciais atuais. Se você achar que este artigo ainda é importante, inclua o template {{ForArchiveReview|*escreva a sua justificativa*}}.

Since this article was written, alternative **Map Tile caching** solutions for Java ME have become available. **Map Tile caching** solutions such as the Nokia Maps API for Java ME have several advantages over **static mapping** solutions such as the static Google Maps API including:

- **Static** mapping services such as the Google Static Maps API or Nokia's RESTful Map API do not cache or tile the images when requested, therefore **each** request involves a round trip to the server. If the map on a mobile application needs to be refreshed at any time, using a caching library will result in a reduction in network traffic after around **three** maps have been displayed. An explanation of this can be found here
- As the name implies, Google's Static Maps API can only retrieve over http static images for a requested coordinate point, image size, image type and zoom level. Newer libraries offer additional functionality out of the box offering dynamic Map content and touch support, where the user can move around his/her current position, zoom in, zoom out, modify the view mode to satellite or translate an address to a coordinate point and show that on the map, among others. This abstraction of the underlying functionality is hidden from the developer,os much less coding is needed in order to achieve the same result .

Additionally the following points apply in favour of Nokia Maps when comparing to Google Maps:

- No legal restrictions of using the API outside a web browser application or need to provide a link to the native Google Maps App (if there is one), or to Google Maps (if there isn't one). See Terms of Service below.
- Higher free daily request limits. Nokia Maps API for Java ME supports up to 50,000 render requests per day and per unique IP address (as of January 2012), for Nokia Developer registered users (free of charge) while the limit for Google's Static Maps API is currently 1000 unique (different) image requests per viewer per day.

An article with the same functionality, written with Nokia Maps API for Java ME, that uses much less code can be found here

Aqui está uma biblioteca simples para consultar o Google Maps com as seguintes características:

- Endereços de geocódigos para suas coordenadas geográficas
- Recuperação de imagens estáticas com tamanho, formato e “zoom” configuráveis.

Para ver um exemplo real desta API, você pode acessar aqui: Java ME Google Maps API sample MIDlet

Pegue sua própria Google Maps API Key

Para usar o seguinte código, você deverá pegar sua própria Google Maps API Key. Se você não tem uma API Key, você pode seguir as instruções aqui: Como usar os dados do Google Maps em aplicações móveis

Use um servidor Proxy para acessar os serviços do Google Maps

Nota: o tema (uso de proxy) provavelmente não é necessário, mesmo assim o abordaremos.
Quando você obtiver uma Google Maps API key, você insere o endereço e essa chave possibilitará acessar os serviços Google Maps. Por esta razão, você deverá configurar o servidor Proxy neste endereço e este receberá requisições HTTP de seus clientes móveis, remetendo-o para os serviços do Google Maps, que responde de volta.

No código abaixo, você deverá colocar a seguinte requisição:

- http://www.yourserver.com/error.html requests to http://maps.google.com/maps/geo
- http://www.yourserver.com/error.html requests to http://maps.google.com/staticmap

Código fonte: classe GoogleMaps

```
import java.io.ByteArrayOutputStream;
import java.io.InputStream;

import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.lcdui.Image;

public class GoogleMaps
{
 String apiKey = null;

 //estas duas propriedades serão usadas com o método de rolagem dos mapas.Você pode remove-lo se não for necessário.
 int offset = 268435456;
 double radius = offset / Math.PI;

 public GoogleMaps(String apiKey)
 {
  this.apiKey = apiKey;
 }

 public double[] geocodeAddress(String address) throws Exception
 {
  byte[] res = loadHttpFile(getGeocodeUrl(address));
```


(C) Copyright Nokia 2013. All rights reserved.

```
 String resString = new String(res, 0, res.length);

 String[] data = split(resString, ',');

 if(data[0].compareTo("200") != 0)
 {
  int errorCode = Integer.parseInt(data[0]);

  throw new Exception("Google Maps Exception: " + getGeocodeError(errorCode));
 }
 else
 {
  return new double[]{
   Double.parseDouble(data[2]),
   Double.parseDouble(data[3])
  };
 }
}
public Image retrieveStaticImage(int width, int height, double lat, double lng, int zoom, String format) throws Exception
{
 byte[] imageData = loadHttpFile(getMapUrl(width, height, lng, lat, zoom, format));

 return Image.createImage(imageData, 0, imageData.length);
}

String getGeocodeError(int errorCode)
{
 switch(errorCode)
 {
 case 400:
  return "Bad request";
 case 500:
  return "Server error";
 case 601:
  return "Missing query";
 case 602:
  return "Unknown address";
 case 603:
  return "Unavailable address";
 case 604:
  return "Unknown directions";
 case 610:
  return "Bad API key";
 case 620:
  return "Too many queries";
 default:
  return "Generic error";
 }
}

String getGeocodeUrl(String address)
{
 return "http://maps.google.com/maps/geo?q=" + urlEncode(address)
  + "&output=csv&key=" + apiKey;
}
String getMapUrl(int width, int height, double lng, double lat, int zoom, String format)
{
 return "http://maps.google.com/maps/api/staticmap?center=" +
  lat + "," + lng + "&format=" + format + "&zoom=" + zoom + "&size=" +
  width + "x" + height + "&key=" + apiKey;
}
String urlEncode(String str)
{
 StringBuffer buf = new StringBuffer();
 char c;
 for(int i = 0; i < str.length(); i++)
 {
  c = str.charAt(i);
  if ((c >= '0' && c <= '9')||
   (c >= 'A' && c <= 'Z')||
   (c >= 'a' && c <= 'z'))
  {
   buf.append(c);
  }
  else
  {
   buf.append("%").append(Integer.toHexString((int) str.charAt(i)));
  }
 }
 return buf.toString();
}
byte[] loadHttpFile(String url) throws Exception
{
 HttpConnection hc = null;

 InputStream is = null;

 byte[] byteBuffer = null;

 try
 {
  hc = (HttpConnection) Connector.open(url);

  hc.setRequestMethod(HttpConnection.GET);

  int ch;

  is = hc.openInputStream();

  int len = (int)hc.getLength();

  if(len > 0)
  {
   byteBuffer = new byte[len];

   is.read(byteBuffer);
  }
  else
  {
   ByteArrayOutputStream bos = new ByteArrayOutputStream();

   while ((ch = is.read()) != -1)
   {
    bos.write(ch);
   }
   byteBuffer = bos.toByteArray();

   bos.close();
  }
```


(C) Copyright Nokia 2013. All rights reserved.

```
  }
  catch(Exception e)
  {
   e.printStackTrace();
  }
  finally
  {
   try
   {
    if(is != null)
     is.close();

    if(hc != null)
     hc.close();
   }
   catch(Exception e2)
   {
    e2.printStackTrace();
   }
  }
  return byteBuffer;
 }
 static String[] split(String s, int chr)
 {
  Vector res = new Vector();

  int curr = 0;
  int prev = 0;

  while((curr = s.indexOf(chr, prev)) >= 0)
  {
   res.addElement(s.substring(prev, curr));

   prev = curr + 1;
  }
  res.addElement(s.substring(prev));

  String[] splitted = new String[res.size()];

  res.copyInto(splitted);

  return splitted;
 }
}
```

Utilidade do método de rolagem para o mapa

Se você necessita de usar rolagem no seu mapa, você precisa calcular o novo centro para sua imagem estática. O método **adjust()** retornará o novo centro de latitude e longitude do mapa, aceitando os seguintes argumentos:

- as atuais coordenadas de **latitute** e **longitude**
- o **deltaX** e **deltaY**, em pixels, do centro do mapa
- o nível de **zoom**

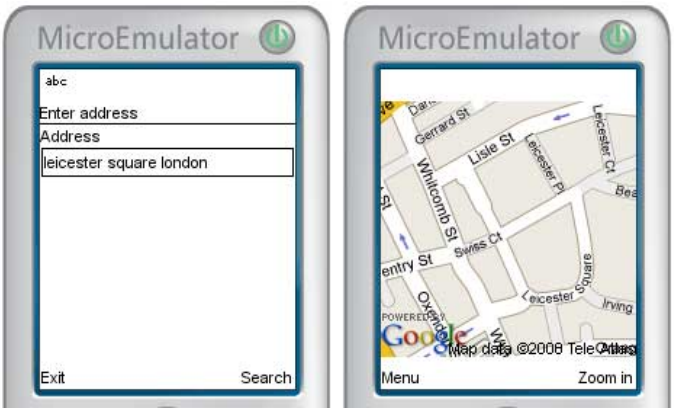
O código original, em JavaScript, está dispónivel em: http://home.provide.net/~bratliff/adjust.js

Nota: para usar o seguinte método, você deverá incluir no seu projeto a biblioteca **MicroFloat**, disponível em: MicroFloat website

```
public double[] adjust(double lat, double lng, int deltaX, int deltaY, int z)
{
 return new double[]{
  XToL(LToX(lat) + (deltaX<<(21-z))),
  YToL(LToY(lng) + (deltaY<<(21-z)))
 };
}
double LToX(double x)
{
 return round(offset + radius * x * Math.PI / 180);
}

double LToY(double y)
{
 return round(
  offset - radius *
  Double.longBitsToDouble(MicroDouble.log(
   Double.doubleToLongBits(
   (1 + Math.sin(y * Math.PI / 180))
   /
   (1 - Math.sin(y * Math.PI / 180))
   )
  )) / 2);
}

double XToL(double x)
{
 return ((round(x) - offset) / radius) * 180 / Math.PI;
}

double YToL(double y)
{
 return (Math.PI / 2 - 2 * Double.longBitsToDouble(
    MicroDouble.atan(
     MicroDouble.exp(Double.doubleToLongBits((round(y)-offset)/radius))
    )
   )) * 180 / Math.PI;
}
double round(double num)
{
 double floor = Math.floor(num);

 if(num - floor >= 0.5)
  return Math.ceil(num);
 else
  return floor;
}
```


(C) Copyright Nokia 2013. All rights reserved.

Código fonte: uso simples



Para usar esta classe, você primeiramente argumenta-a com sua API key:

```
GoogleMaps gMap = new GoogleMaps("API_KEY");
```

Para um endereço em geocódigo, você pode usar o método `geocodeAddress()`:

```
double[] lanLng = gMap.geocodeAddress("Leicester Square, London");
```

Para buscar imagem de um mapa:

```
Image map = gMap.retrieveStaticImage(320, 240, 51.510605, -0.130728, 8, "png32");
```


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.