OkHttp使用进阶 译自OkHttp Github官方教程

没有使用过OkHttp的，可以先看OkHttp使用介绍

英文版原版地址

Recipes · square/okhttp Wiki

同步get

下载一个文件，打印他的响应头，以string形式打印响应体。
响应体的 `string()` 方法对于小文档来说十分方便、高效。但是如果响应体太大（超过1MB），应避免适应 `string()` 方法 ，因为他会将把整个文档加载到内存中。
对于超过1MB的响应body，应使用流的方式来处理body。

```
private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    Request request = new Request.Builder()
        .url("http://publicobject.com/helloworld.txt")
        .build();

    Response response = client.newCall(request).execute();
    if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

    Headers responseHeaders = response.headers();
    for (int i = 0; i < responseHeaders.size(); i++) {
      System.out.println(responseHeaders.name(i) + ": " + responseHeaders.value(i));
    }
    System.out.println(response.body().string());
}
```

异步get

在一个工作线程中下载文件，当响应可读时回调Callback接口。读取响应时会阻塞当前线程。OkHttp现阶段不提供异步api来接收响应体。

```
private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    Request request = new Request.Builder()
        .url("http://publicobject.com/helloworld.txt")
        .build();

    client.newCall(request).enqueue(new Callback() {
      @Override public void onFailure(Request request, Throwable throwable) {
        throwable.printStackTrace();
      }

      @Override public void onResponse(Response response) throws IOException {
        if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

        Headers responseHeaders = response.headers();
        for (int i = 0; i < responseHeaders.size(); i++) {
          System.out.println(responseHeaders.name(i) + ": " + responseHeaders.value(i));
        }

        System.out.println(response.body().string());
      }
    });
}
```

提取响应头

典型的HTTP头 像是一个 `Map<String, String>` :每个字段都有一个或没有值。但是一些头允许多个值，像Guava的Multimap。例如：HTTP响应里面提供的 `Vary` 响应头，就是多值的。OkHttp的api试图让这些情况都适用。
当写请求头的时候，使用 `header(name, value)` 可以设置唯一的name、value。如果已经有值，旧的将被移除，然后添加新的。使用 `addHeader(name, value)` 可以添加多值（添加，不移除已有的）。
当读取响应头时，使用 `header(name)` 返回最后出现的name、value。通常情况这也是唯一的name、value。如果没有值，那么 `header(name)` 将返回null。如果想读取字段对应的所有值，使用 `headers(name)` 会返回一个list。
为了获取所有的Header，Headers类支持按index访问。

```
private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    Request request = new Request.Builder()
        .url("https://api.github.com/repos/square/okhttp/issues")
        .header("User-Agent", "OkHttp Headers.java")
        .addHeader("Accept", "application/json; q=0.5")
        .addHeader("Accept", "application/vnd.github.v3+json")
        .build();
    Response response = client.newCall(request).execute();
    if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

    System.out.println("Server: " + response.header("Server"));
    System.out.println("Date: " + response.header("Date"));
    System.out.println("Vary: " + response.headers("Vary"));
}
```

Post方式提交String

使用HTTP POST提交请求到服务。这个例子提交了一个markdown文档到web服务，以HTML方式渲染markdown。因为整个请求体都在内存中，因此避免使用此api提交大文档（大于1MB）。

```
public static final MediaType MEDIA_TYPE_MARKDOWN
  = MediaType.parse("text/x-markdown; charset=utf-8");

private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    String postBody = ""
        + "Releases\n"
        + "--------\n"
        + "\n"
        + " * _1.0_ May 6, 2013\n"
        + " * _1.1_ June 15, 2013\n"
        + " * _1.2_ August 11, 2013\n";

    Request request = new Request.Builder()
        .url("https://api.github.com/markdown/raw")
        .post(RequestBody.create(MEDIA_TYPE_MARKDOWN, postBody))
        .build();

    Response response = client.newCall(request).execute();
    if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

    System.out.println(response.body().string());
}
```

Post方式提交流

以流的方式POST提交请求体。请求体的内容由流写入产生。这个例子是流直接写入Okio的BufferedSink。你的程序可能会使用 `OutputStream` ，你可以使用 `BufferedSink.outputStream()` 来获取。

```
public static final MediaType MEDIA_TYPE_MARKDOWN
      = MediaType.parse("text/x-markdown; charset=utf-8");

private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    RequestBody requestBody = new RequestBody() {
      @Override public MediaType contentType() {
        return MEDIA_TYPE_MARKDOWN;
      }

      @Override public void writeTo(BufferedSink sink) throws IOException {
        sink.writeUtf8("Numbers\n");
        sink.writeUtf8("-------\n");
        for (int i = 2; i <= 997; i++) {
          sink.writeUtf8(String.format(" * %s = %s\n", i, factor(i)));
        }
      }

      private String factor(int n) {
        for (int i = 2; i < n; i++) {
          int x = n / i;
          if (x * i == n) return factor(x) + " × " + i;
        }
        return Integer.toString(n);
      }
    };

    Request request = new Request.Builder()
        .url("https://api.github.com/markdown/raw")
        .post(requestBody)
        .build();

    Response response = client.newCall(request).execute();
    if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

    System.out.println(response.body().string());
}
```

Post方式提交文件

以文件作为请求体是十分简单的。

```
public static final MediaType MEDIA_TYPE_MARKDOWN
  = MediaType.parse("text/x-markdown; charset=utf-8");

private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    File file = new File("README.md");

    Request request = new Request.Builder()
        .url("https://api.github.com/markdown/raw")
        .post(RequestBody.create(MEDIA_TYPE_MARKDOWN, file))
        .build();

    Response response = client.newCall(request).execute();
    if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

    System.out.println(response.body().string());
}
```

Post方式提交表单

使用 `FormEncodingBuilder` 来构建和HTML `<form>` 标签相同效果的请求体。键值对将使用一种HTML兼容形式的URL编码来进行编码。

```java
private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    RequestBody formBody = new FormEncodingBuilder()
        .add("search", "Jurassic Park")
        .build();
    Request request = new Request.Builder()
        .url("https://en.wikipedia.org/w/index.php")
        .post(formBody)
        .build();

    Response response = client.newCall(request).execute();
    if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

    System.out.println(response.body().string());
}
```

Post方式提交分块请求

`MultipartBuilder` 可以构建复杂的请求体，与HTML文件上传形式兼容。多块请求体中每块请求都是一个请求体，可以定义自己的请求头。这些请求头可以用来描述这块请求，例如他的 `Content-Disposition` 。如果 `Content-Length` 和 `Content-Type` 可用的话，他们会被自动添加到请求头中。

```java
private static final String IMGUR_CLIENT_ID = "...";
private static final MediaType MEDIA_TYPE_PNG = MediaType.parse("image/png");

private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    // Use the imgur image upload API as documented at https://api.imgur.com/endpoints/image
    RequestBody requestBody = new MultipartBuilder()
        .type(MultipartBuilder.FORM)
        .addPart(
            Headers.of("Content-Disposition", "form-data; name=\"title\""),
            RequestBody.create(null, "Square Logo"))
        .addPart(
            Headers.of("Content-Disposition", "form-data; name=\"image\""),
            RequestBody.create(MEDIA_TYPE_PNG, new File("website/static/logo-square.png")))
        .build();

    Request request = new Request.Builder()
        .header("Authorization", "Client-ID " + IMGUR_CLIENT_ID)
        .url("https://api.imgur.com/3/image")
        .post(requestBody)
        .build();

    Response response = client.newCall(request).execute();
    if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

    System.out.println(response.body().string());
}
```

使用Gson来解析JSON响应

Gson是一个在JSON和Java对象之间转换非常方便的api。这里我们用Gson来解析Github API的JSON响应。
注意：`ResponseBody.charStream()` 使用响应头 `Content-Type` 指定的字符集来解析响应体。默认是UTF-8。

```
private final OkHttpClient client = new OkHttpClient();
private final Gson gson = new Gson();

public void run() throws Exception {
    Request request = new Request.Builder()
        .url("https://api.github.com/gists/c2a7c39532239ff261be")
        .build();
    Response response = client.newCall(request).execute();
    if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

    Gist gist = gson.fromJson(response.body().charStream(), Gist.class);
    for (Map.Entry<String, GistFile> entry : gist.files.entrySet()) {
      System.out.println(entry.getKey());
      System.out.println(entry.getValue().content);
    }
}

static class Gist {
    Map<String, GistFile> files;
}

static class GistFile {
    String content;
}
```

响应缓存

为了缓存响应，你需要一个你可以读写的缓存目录，和缓存大小的限制。这个缓存目录应该是私有的，不信任的程序应不能读取缓存内容。
一个缓存目录同时拥有多个缓存访问是错误的。大多数程序只需要调用一次 `new OkHttp()` ，在第一次调用时配置好缓存，然后其他地方只需要调用这个实例就可以了。否则两个缓存示例互相干扰，破坏响应缓存，而且有可能会导致程序崩溃。
响应缓存使用HTTP头作为配置。你可以在请求头中添加 `Cache-Control: max-stale=3600` ,OkHttp缓存会支持。你的服务通过响应头确定响应缓存多长时间，例如使用 `Cache-Control: max-age=9600` 。

```
private final OkHttpClient client;

public CacheResponse(File cacheDirectory) throws Exception {
    int cacheSize = 10 * 1024 * 1024; // 10 MiB
    Cache cache = new Cache(cacheDirectory, cacheSize);

    client = new OkHttpClient();
    client.setCache(cache);
}

public void run() throws Exception {
    Request request = new Request.Builder()
        .url("http://publicobject.com/helloworld.txt")
        .build();

    Response response1 = client.newCall(request).execute();
    if (!response1.isSuccessful()) throw new IOException("Unexpected code " + response1);

    String response1Body = response1.body().string();
    System.out.println("Response 1 response:          " + response1);
    System.out.println("Response 1 cache response:    " + response1.cacheResponse());
    System.out.println("Response 1 network response:  " + response1.networkResponse());

    Response response2 = client.newCall(request).execute();
    if (!response2.isSuccessful()) throw new IOException("Unexpected code " + response2);

    String response2Body = response2.body().string();
    System.out.println("Response 2 response:          " + response2);
    System.out.println("Response 2 cache response:    " + response2.cacheResponse());
    System.out.println("Response 2 network response:  " + response2.networkResponse());

    System.out.println("Response 2 equals Response 1? " + response1Body.equals(response2Body));
}
```

扩展

在这一节还提到了下面一句：
There are cache headers to force a cached response, force a network response, or force the network response to be validated with a conditional GET.

我不是很懂cache，平时用到的也不多，所以把Google在Android Developers一段相关的解析放到这里吧。

Force a Network Response

In some situations, such as after a user clicks a 'refresh' button, it may be necessary to skip the cache, and fetch data directly from the server. To force a full refresh, add the no-cache directive:

```
connection.addRequestProperty("Cache-Control", "no-cache");
```

If it is only necessary to force a cached response to be validated by the server, use the more efficient max-age=0 instead:

```
connection.addRequestProperty("Cache-Control", "max-age=0");
```

Force a Cache Response

Sometimes you'll want to show resources if they are available immediately, but not otherwise. This can be used so your application can show something while waiting for the latest data to be downloaded. To restrict a request to locally-cached resources, add the only-if-cached directive:

```
try {
     connection.addRequestProperty("Cache-Control", "only-if-cached");
     InputStream cached = connection.getInputStream();
     // the resource was cached! show it
  catch (FileNotFoundException e) {
     // the resource was not cached
 }
}
```

This technique works even better in situations where a stale response is better than no response. To permit stale cached responses, use the max-stale directive with the maximum staleness in seconds:

```
int maxStale = 60 * 60 * 24 * 28; // tolerate 4-weeks stale
connection.addRequestProperty("Cache-Control", "max-stale=" + maxStale);
```

以上信息来自：HttpResponseCache - Android SDK | Android Developers

取消一个Call

使用 `Call.cancel()` 可以立即停止掉一个正在执行的call。如果一个线程正在写请求或者读响应，将会引发 `IOException` 。当call没有必要的时候，使用这个api可以节约网络资源。例如当用户离开一个应用时。不管同步还是异步的call都可以取消。
你可以通过tags来同时取消多个请求。当你构建一请求时，使用 `RequestBuilder.tag(tag)` 来分配一个标签。之后你就可以用 `OkHttpClient.cancel(tag)` 来取消所有带有这个tag的call。

```
private final ScheduledExecutorService executor = Executors.newScheduledThreadPool(1);
private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    Request request = new Request.Builder()
        .url("http://httpbin.org/delay/2") // This URL is served with a 2 second delay.
        .build();

    final long startNanos = System.nanoTime();
    final Call call = client.newCall(request);

    // Schedule a job to cancel the call in 1 second.
    executor.schedule(new Runnable() {
      @Override public void run() {
        System.out.printf("%.2f Canceling call.%n", (System.nanoTime() - startNanos) / 1e9f);
        call.cancel();
        System.out.printf("%.2f Canceled call.%n", (System.nanoTime() - startNanos) / 1e9f);
      }
    }, 1, TimeUnit.SECONDS);

    try {
      System.out.printf("%.2f Executing call.%n", (System.nanoTime() - startNanos) / 1e9f);
      Response response = call.execute();
      System.out.printf("%.2f Call was expected to fail, but completed: %s%n",
          (System.nanoTime() - startNanos) / 1e9f, response);
    } catch (IOException e) {
      System.out.printf("%.2f Call failed as expected: %s%n",
          (System.nanoTime() - startNanos) / 1e9f, e);
    }
}
```

超时

没有响应时使用超时结束call。没有响应的原因可能是客户点链接问题、服务器可用性问题或者这之间的其他东西。OkHttp支持连接，读取和写入超时。

```
private final OkHttpClient client;

public ConfigureTimeouts() throws Exception {
    client = new OkHttpClient();
    client.setConnectTimeout(10, TimeUnit.SECONDS);
    client.setWriteTimeout(10, TimeUnit.SECONDS);
    client.setReadTimeout(30, TimeUnit.SECONDS);
}

public void run() throws Exception {
    Request request = new Request.Builder()
        .url("http://httpbin.org/delay/2") // This URL is served with a 2 second delay.
        .build();

    Response response = client.newCall(request).execute();
    System.out.println("Response completed: " + response);
}
```

每个call的配置

使用 `OkHttpClient` ，所有的HTTP Client配置包括代理设置、超时设置、缓存设置。当你需要为单个call改变配置的时候，clone 一个 `OkHttpClient` 。这个api将会返回一个浅拷贝（shallow copy），你可以用来单独自定义。下面的例子中，我们让一个请求是500ms的超时、另一个是3000ms的超时。

```
private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    Request request = new Request.Builder()
        .url("http://httpbin.org/delay/1") // This URL is served with a 1 second delay.
        .build();

    try {
      Response response = client.clone() // Clone to make a customized OkHttp for this request.
          .setReadTimeout(500, TimeUnit.MILLISECONDS)
          .newCall(request)
          .execute();
      System.out.println("Response 1 succeeded: " + response);
    } catch (IOException e) {
      System.out.println("Response 1 failed: " + e);
    }

    try {
      Response response = client.clone() // Clone to make a customized OkHttp for this request.
          .setReadTimeout(3000, TimeUnit.MILLISECONDS)
          .newCall(request)
          .execute();
      System.out.println("Response 2 succeeded: " + response);
    } catch (IOException e) {
      System.out.println("Response 2 failed: " + e);
    }
}
```

处理验证

这部分和HTTP AUTH有关。
相关资料：HTTP AUTH 那些事 - 王绍全的博客 - 博客频道 - CSDN.NET

OkHttp会自动重试未验证的请求。当响应是 `401 Not Authorized` 时， `Authenticator` 会被要求提供证书。Authenticator的实现中需要建立一个新的包含证书的请求。如果没有证书可用，返回null来跳过尝试。

```
public List<Challenge> challenges()
Returns the authorization challenges appropriate for this response's code. If the response code is
401 unauthorized, this returns the "WWW-Authenticate" challenges. If the response code is 407
proxy unauthorized, this returns the "Proxy-Authenticate" challenges. Otherwise this returns an
empty list of challenges.
```

当需要实现一个 `Basic` challenge，使用 `Credentials.basic(username, password)` 来编码请求头。

```
private final OkHttpClient client = new OkHttpClient();

public void run() throws Exception {
    client.setAuthenticator(new Authenticator() {
      @Override public Request authenticate(Proxy proxy, Response response) {
        System.out.println("Authenticating for response: " + response);
        System.out.println("Challenges: " + response.challenges());
        String credential = Credentials.basic("jesse", "password1");
        return response.request().newBuilder()
            .header("Authorization", credential)
            .build();
      }

      @Override public Request authenticateProxy(Proxy proxy, Response response) {
        return null; // Null indicates no attempt to authenticate.
      }
    });

    Request request = new Request.Builder()
        .url("http://publicobject.com/secrets/hellosecret.txt")
        .build();

    Response response = client.newCall(request).execute();
    if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

    System.out.println(response.body().string());
}
```

微博：@GavinCT 关注作者获取最新动态
GitHub：https://github.com/GavinCT

如果您觉得此博客对您有用，欢迎通过支付宝对我进行小额赞助。

分类: Android网络

标签: Android开源库使用 , OkHttp

好文要顶 关注我 收藏该文

GavinCT
关注 - 9
粉丝 - 41
+加关注

2 0

« 上一篇：Evernote Markdown Sublime实现
» 下一篇：OkHttp使用介绍

posted @ 2014-09-30 13:43 GavinCT 阅读(57475) 评论(7) 编辑 收藏

评论列表

#1楼 2015-01-06 17:41 **lzyickobe**

你好，博主，感谢你的OKHttp的分享。我在这里转载了你的这篇文章，lzyblog.com

支持(0) 反对(0)

#2楼[楼主] 2015-01-06 17:55 **GavinCT**

@ lzyickobe
烦请申明一下这个OkHttp2.0有Bug，暂时不推荐在产品中使用。

在国内使用OkHttp会因为这个问题导致部分酷派手机用户无法联网，所以对于大众app来说，需要等待这个bug修复后再使用。或者尝试使用OkHttp的老版本。
截止到目前，OkHttp一直没有修复，并把修复计划延迟到了OkHttp2.3中。

支持(1) 反对(0)

#3楼 2015-01-30 12:59 **傅傅傅傅傅先生**

@ GavinCT
现在的版本好像是2.2的，那不是意味着现在还不能正式使用呀~还有其他bug么，国产手机的可能会有很多意料不到的问题产生

支持(0) 反对(0)

#4楼[楼主] 2015-02-02 10:56 **GavinCT**

@ 傅傅傅傅傅先生
可以跟进这个issue 目前还没有解决
https://github.com/square/okhttp/issues/1114

支持(0) 反对(0)

#5楼 2015-06-03 13:16 **西北野狼**

用2.4的jar包，各种崩溃中，我擦

支持(0) 反对(0)

#6楼 2015-06-26 15:53 passerbywhu

@ 西北野狼
不过在build.gradle中直接引入okhttp2.4写不了缓存。不知道什么原因。换成了旧的不知道啥版本的okhttp和1.3版本的okio就好了。。。囧。

支持(0) 反对(0)

#7楼 2015-06-26 16:11 passerbywhu

@ 西北野狼
而且在网络通畅的情况下，如果设置maxStale(365, TimeUnit.DAYS)，取回来的还是缓存。。

支持(0) 反对(0)