



关于国土工作室

我们(国土工作室)是一支专注于 Android 平台企业级应用开发的技术团队,对娱乐多媒体应用有着深刻的理解及研发能力,致力服务于企业用户。为音视频等娱乐多媒体网站、门户网站、SNS、论坛、电子商务等传统网络应用向移动互联网发展提供解决方案和技术支持,为企业提供 Android 培训服务等多种业务。

我们尤其擅长于提供从 Android 客户端到服务端的一站式解决方案和技术支持,服务端可以采用 Java EE,也可以采用轻量级流行的 LAMP 技术体系。目前,研发出了比 KU6、优酷更加强大和完善的 Android 视频网站娱乐多媒体客户端软件,并在持续升级中。

目前,我们正在务实而卓有成效的与音视频等娱乐多媒体网站、门户网站、SNS、论坛、电子商务等传统网络服务商合作,发展迅速,渴望有志之士的加入,和我们一起为成为世界最好的 Android 软件开发和咨询、培训公司而奋斗,为移动互联网和智能手机时代贡献力量!

联系我们

电话:15711060468

Email:guoshiandroid@gmail.com

博客:<http://www.cnblogs.com/guoshiandroid/>





注意: 该文档参考和使用了网络上的很多免费开放的内容, 并以免费开放的方式发布, 希望为移动互联网和智能手机时代贡献绵薄之力! 可以随意转载, 但不得使用该文档谋利。

另外: 国士工作室已免费发布原创教程《大话企业级 Android 开发》, 请访问国士工作室博客 <http://www.cnblogs.com/guoshiandroid/> 获取教程。

- 2007年底 Google 宣布举办总奖金高达1000万美元的开发者大奖赛, 鼓励程序开发者在 Android 上写出实用而又具有创意的应用程序;
- 2009年5月27日, 在 Google 的 I/O 开发者聚会上, Google 发布了总奖金接近2000万美元的第二次大奖赛的消息, 开发者们开始了新一轮的较量;

Android 是 Google 于2007年11月5日宣布的基于 Linux 平台的开源手机操作系统的名称, 该平台由操作系统、中间件、用户界面和应用软件组成, 号称是首个为移动终端打造的真正开放和完整的移动软件。

Android 一出生就被打上了富二代的胎记, 不仅仅是因为诞生于当今的网络霸主 Google, 更主要还有一个空前强大和壮观的开放手机联盟 OHA (Open Handset Alliance) 提供全力的支持。OHA 是什么? OHA 涵盖了中国移动、T-Mobile、Sprint 等移动运营商, 包括 HTC、Motorola、三星等手机制造商, 有 Google 为代表的手机软件商, 还有 Inter、Nvidia 为标志的底层硬件厂商和 Astonishing Tribe 等商业运作公司, 该组织声称组织的所有成员都会基于 Android 来开发新的手机业务。

但是, 要成为 Android 高手并不是一件容易的事情。并不是很多人想象的能够飞快的写出几行漂亮的代码去解决一些困难的问题就是 Android 高手了。真正的 Android 高手需要考虑的问题远远不是写些漂亮的代码就足够的。下面是成为一名真正的 Android 高手必须掌握和遵循的一些准则:

- 1, 学会懒惰
- 2, 精通 Android 体系架构、MVC、常见的设计模式、控制反转 (IoC)
- 3, 编写可重用、可扩展、可维护、灵活性高的代码
- 4, 高效的编写高效的代码
- 5, 学会至少一门服务器端开发技术

一: 学会懒惰

没搞错吧? 竟然让程序开发人员学会懒惰? 程序开发人员可能是世界上最为忙碌的一类人啦! 对, 没错, 学会懒惰! 正因为程序开发人员忙碌, 正因为程序开发人员可能会在客户无限变化的需求之下没日没夜的加班, 所以要学会懒惰, 这样, 你就可以把更多的时





间浪费在美好的事物身上!

如何懒惰:

- 1, Don't Reinvent the Wheel (不要重复发明轮子)。
- 2, Inventing the Wheel(发明轮子)。

1, Don't Reinvent the Wheel (不要重复发明轮子)。

“轮子理论”，也即“不要重复发明轮子”，这是西方国家的一句谚语，原话是：Don't Reinvent the Wheel。“不要重复发明轮子”意思是企业中任何一项工作实际上都别人做过，我们所需要的就是找到做过这件事情的人。拿到软件领域中就是指有的项目或功能，别人已经做过，我们需要用的时候，直接拿来用即可，而不要重新制造。

Android 号称是首个为移动终端打造的真正开放和完整的移动软件。Android 发布后不久 Google 公司就发布了操作系统核心 (Kernel) 与部分驱动程序的源代码，到目前位置除了 Google Map 等 Google 公司的核心组件没有开放源代码外，Android 基本完成了完全的开源，这就极大的促进了 Android 的普及和移植。受到 Android 开放行为和开源精神的影响，在世界各地，有成千上万的程序员喜欢和别人分享自己的聪明才智和自己编写的代码。你可以在 Google 的 Android 讨论组或者 Google 搜索引擎上搜索到很多优秀的程序代码。这样做并不是鼓励大家整天等着让别人为你编写代码，而是你可以“站在伟人的肩膀上”，充分发扬“拿来主义”，聪明地应用别人的程序代码可以节省你大量的时间。

下面笔者为大家介绍几个通用的类，这些类来自笔者平日的收集，如果你能把它们加入到你自己的类库中，迟早你会发现自己在进行 Android 开发的时候受益无穷：

- 1) 从输入流中获取数据并以字节数组返回，这种输入流可以来自 Android 本地也可以来自网络。

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

public class StreamTool {
    /**
     * 从输入流获取数据
     * @param inputStream
     * @return
     * @throws Exception
     */
}
```





```
public static byte[] readInputStream(InputStream inputStream) throws Exception {
    byte[] buffer = new byte[1024]; //你可以根据实际需要调整缓存大小
    int len = -1;
    ByteArrayOutputStream outSteam = new ByteArrayOutputStream();
    while( (len = inputStream.read(buffer)) != -1 ){
        outSteam.write(buffer, 0, len);
    }
    outSteam.close();
    inputStream.close();
    return outSteam.toByteArray();
}
}
```

- 2) 通过 Android 客户端上传数据到服务器：可以上传简单的表单，也可以方便的上传带有附件的文件，此类远远比 Android 自身的 HttpClient 更高效、更易于使用：

```
import java.io.DataOutputStream;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
```





```
public class HttpRequester {

    /**
     * 直接通过 HTTP 协议提交数据到服务器,实现如下面表单提交功能:
     *
     *          <FORM          METHOD=POST
ACTION="http://192.168.0.200:8080/ssi/fileload/test.do" enctype="multipart/form-data">
        <INPUT TYPE="text" NAME="name">
        <INPUT TYPE="text" NAME="id">
        <input type="file" name="imagefile"/>
        <input type="file" name="zip"/>
    </FORM>
     * @param actionUrl 上传路径(注: 避免使用 localhost 或127.0.0.1这样的路径
测试, 因为它会指向手机模拟器, 你可以使用 http://www.itcast.cn 或
http://192.168.1.10:8080这样的路径测试)
     * @param params 请求参数 key 为参数名,value 为参数值
     * @param file 上传文件
     */
    public static String post(String actionUrl, Map<String, String> params, FormFile[]
files) {
        try {
            String BOUNDARY = "-----7d4a6d158c9"; //数据分隔线
            String MULTIPART_FORM_DATA = "multipart/form-data";

            URL url = new URL(actionUrl);
            HttpURLConnection conn = (HttpURLConnection)
url.openConnection();

            conn.setConnectTimeout(5* 1000);
            conn.setDoInput(true);//允许输入
            conn.setDoOutput(true);//允许输出
            conn.setUseCaches(false);//不使用 Cache
```





```
conn.setRequestMethod("POST");
conn.setRequestProperty("Connection", "Keep-Alive");
conn.setRequestProperty("Charset", "UTF-8");
conn.setRequestProperty("Content-Type", MULTIPART_FORM_DATA
+ "; boundary=" + BOUNDARY);

StringBuilder sb = new StringBuilder();
for (Map.Entry<String, String> entry : params.entrySet()) { //构建表单
字段内容

        sb.append("--");
        sb.append(BOUNDARY);
        sb.append("\r\n");
                sb.append("Content-Disposition: form-data; name=\"" +
entry.getKey() + "\"\r\n\r\n");
        sb.append(entry.getValue());
        sb.append("\r\n");
    }

        DataOutputStream outputStream = new
DataOutputStream(conn.getOutputStream());
        outputStream.write(sb.toString().getBytes()); //发送表单 字段数据
        for (FormFile file : files) { //发送文件数据

            StringBuilder split = new StringBuilder();
            split.append("--");
            split.append(BOUNDARY);
            split.append("\r\n");
                    split.append("Content-Disposition: form-data; name=\"" +
file.getFormname() + "\"; filename=\"" + file.getFilename() + "\"\r\n\r\n");
            split.append("Content-Type: " + file.getContentType() + "\r\n\r\n");
            outputStream.write(split.toString().getBytes());
            if (file.getInStream() != null) {
                byte[] buffer = new byte[1024];
```





```
        int len = 0;
        while((len = file.getInputStream().read(buffer))!=-1){
            outputStream.write(buffer, 0, len);
        }
        file.getInputStream().close();
    }else{
        outputStream.write(file.getData(), 0, file.getData().length);
    }
    outputStream.write("\r\n".getBytes());
}
byte[] end_data = ("--" + BOUNDARY + "--\r\n").getBytes();//数据结
束标志

outStream.write(end_data);
outStream.flush();
int cah = conn.getResponseCode();
if (cah != 200) throw new RuntimeException("请求 url 失败");
InputStream is = conn.getInputStream();
int ch;
StringBuilder b = new StringBuilder();
while( (ch = is.read()) != -1 ){
    b.append((char)ch);
}
outStream.close();
conn.disconnect();
return b.toString();
} catch (Exception e) {
    throw new RuntimeException(e);
}
}
```





```
/**
 * 提交数据到服务器
 * @param actionUrl 上传路径(注: 避免使用 localhost 或127.0.0.1这样的路径
测试, 因为它会指向手机模拟器, 你可以使用 http://www.itcast.cn 或
http://192.168.1.10:8080这样的路径测试)
 * @param params 请求参数 key 为参数名,value 为参数值
 * @param file 上传文件
 */
public static String post(String actionUrl, Map<String, String> params, FormFile
file) {
    return post(actionUrl, params, new FormFile[]{file});
}

public static byte[] postFromHttpClient(String path, Map<String, String> params,
String encode) throws Exception {
    List<NameValuePair> formparams = new ArrayList<NameValuePair>();//用
于存放请求参数
    for(Map.Entry<String, String> entry : params.entrySet()){
        formparams.add(new BasicNameValuePair(entry.getKey(),
entry.getValue()));
    }
    UrlEncodedFormEntity entity = new UrlEncodedFormEntity(formparams,
"UTF-8");
    HttpPost httppost = new HttpPost(path);
    httppost.setEntity(entity);
    HttpClient httpclient = new DefaultHttpClient();//看作是浏览器
    HttpResponse response = httpclient.execute(httppost);//发送 post 请求

    return StreamTool.readInputStream(response.getEntity().getContent());
}
```





```
/**
 * 发送请求
 * @param path 请求路径
 * @param params 请求参数 key 为参数名称 value 为参数值
 * @param encode 请求参数的编码
 */
public static byte[] post(String path, Map<String, String> params, String encode)
throws Exception {
    //String params = "method=save&name="+ URLEncoder.encode(" 老毕 ",
"UTF-8")+ "&age=28&";//需要发送的参数
    StringBuilder parambuilder = new StringBuilder("");
    if(params!=null && !params.isEmpty()){
        for(Map.Entry<String, String> entry : params.entrySet()){
            parambuilder.append(entry.getKey()).append("=")
                .append(URLEncoder.encode(entry.getValue(),
encode)).append("&");
        }
        parambuilder.deleteCharAt(parambuilder.length()-1);
    }
    byte[] data = parambuilder.toString().getBytes();
    URL url = new URL(path);
    HttpURLConnection conn = (HttpURLConnection)url.openConnection();
    conn.setDoOutput(true);//允许对外发送请求参数
    conn.setUseCaches(false);//不进行缓存
    conn.setConnectTimeout(5 * 1000);
    conn.setRequestMethod("POST");
    //下面设置 http 请求头
    conn.setRequestProperty("Accept", "image/gif, image/jpeg, image/pjpeg,
image/pjpeg, application/x-shockwave-flash, application/xaml+xml,
application/vnd.ms-xpsdocument, application/x-ms-xbap, application/x-ms-application,
application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*");
```





```
conn.setRequestProperty("Accept-Language", "zh-CN");
conn.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible; MSIE 8.0;
Windows NT 5.2; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR
3.0.04506.30; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)");
conn.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
conn.setRequestProperty("Content-Length", String.valueOf(data.length));
conn.setRequestProperty("Connection", "Keep-Alive");

//发送参数
DataOutputStream          outputStream          =          new
DataOutputStream(conn.getOutputStream());
outputStream.write(data);//把参数发送出去
outputStream.flush();
outputStream.close();
if(conn.getResponseCode()==200){
    return StreamTool.readInputStream(conn.getInputStream());
}
return null;
}
}
```

2, Inventing the Wheel(发明轮子)。

发明轮子? 不错, 发明轮子! 我们不仅要发明轮子, 更要成为努力成为世界上发明轮子的主导力量, 唯有这样, 才能谈的上中华名族软件大业的真正强大。在 **Android**, 要发明轮子, 就是我们要主动的是解决一些世界上他人未解决的难题或者创造新的编程框架或者对 **Android** 进行深度的改造以适合自己的业务发展需要。Google 发布了 **Android** 后不久, 中国移动便投入了大量的人力和物力, 在 **Android** 的基础上创建融入自己业务并开发、封装了新的功能的和框架的 OMS, 这是 **Android** 中发明轮子的一个非常重要的例子。可能你会说, 这发明轮子也太难了吧, 别急, 我们慢慢来, 开发一个框架特定领域的框架吧! 你可能会一脸无辜的说, 开发一个框架是说的那么容易吗? 当然不是啦。但是也并非不可能,





首先, 我们分析一下框架的魅力的源泉, 看看 Spring、Struts 等 Java EE 框架, 在看看 .NET 框架, 当然也可以看看发展的如火如荼、层出不穷的 PHP 框架, 她们的强大和魅力的源泉都在于: IoC(Inversion of Control)。

Don't call us, we'll call you (别找我, 我会来找你的)。我们下面就自己发明一个轮子的模型, 实际展示一个框架最初核心的类, 让你一饱眼福:

- 1) 下面的类是文件下载类, 支持文件的多线程断点续传, 使用该类的即可安全、高效的下载任何类型的二进制文件:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.RandomAccessFile;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Properties;
import java.util.UUID;
import java.util.concurrent.ConcurrentHashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import cn.itcast.service.FileService;

import android.content.Context;
import android.util.Log;
/**
 * 文件下载器
 */
public class FileDownloader {
    private Context context;
    private FileService fileService;
```





```
private static final String TAG = "FileDownloader";
/* 已下载文件大小 */
private int downloadSize = 0;
/* 原始文件大小 */
private int fileSize = 0;
/* 线程数 */
private DownloadThread[] threads;
/* 下载路径 */
private URL url;
/* 本地保存文件 */
private File saveFile;
/* 下载记录文件 */
private File logFile;
/* 缓存各线程最后下载的位置*/
private Map<Integer, Integer> data = new ConcurrentHashMap<Integer, Integer>();
/* 每条线程下载的大小 */
private int block;
private String downloadUrl;//下载路径
/**
 * 获取线程数
 */
public int getThreadSize() {
    return threads.length;
}
/**
 * 获取文件大小
 * @return
 */
public int getFileSize() {
```





```
        return fileSize;
    }
    /**
     * 累计已下载大小
     * @param size
     */
    protected synchronized void append(int size) {
        downloadSize += size;
    }
    /**
     * 更新指定线程最后下载的位置
     * @param threadId 线程 id
     * @param pos 最后下载的位置
     */
    protected void update(int threadId, int pos) {
        this.data.put(threadId, pos);
    }
    /**
     * 保存记录文件
     */
    protected synchronized void saveLogFile() {
        this.fileService.update(this.downloadUrl, this.data);
    }
    /**
     * 构建文件下载器
     * @param downloadUrl 下载路径
     * @param fileSaveDir 文件保存目录
     * @param threadNum 下载线程数
     */
    public FileDownloader(Context context, String downloadUrl, File fileSaveDir, int
```





```
threadNum) {
    try {
        this.context = context;
        this.downloadUrl = downloadUrl;
        fileService = new FileService(context);
        this.url = new URL(downloadUrl);
        if(!fileSaveDir.exists()) fileSaveDir.mkdirs();
        this.threads = new DownloadThread[threadNum];
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setConnectTimeout(6*1000);
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Accept", "image/gif, image/jpeg, image/pjpeg,
image/pjpeg, application/x-shockwave-flash, application/xhtml+xml,
application/vnd.ms-xpsdocument, application/x-ms-xbap, application/x-ms-application,
application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*");
        conn.setRequestProperty("Accept-Language", "zh-CN");
        conn.setRequestProperty("Referer", downloadUrl);
        conn.setRequestProperty("Charset", "UTF-8");
        conn.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible; MSIE 8.0;
Windows NT 5.2; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR
3.0.04506.30; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)");
        conn.setRequestProperty("Connection", "Keep-Alive");
        conn.connect();
        printResponseHeader(conn);
        if (conn.getResponseCode()==200) {
            this.fileSize = conn.getContentLength();//根据响应获取文件大小
            if (this.fileSize <= 0) throw new RuntimeException("1无法获知文件大
小 ");

            String filename = getFileName(conn);
            this.saveFile = new File(fileSaveDir, filename);/* 保存文件 */
        }
    }
}
```





```
        Map<Integer, Integer> logdata = fileService.getData(downloadUrl);
        if(logdata.size()>0){
            for(Map.Entry<Integer, Integer> entry : logdata.entrySet())
                data.put(entry.getKey(), entry.getValue()+1);
        }
        this.block = this.fileSize / this.threads.length + 1;
        if(this.data.size()==this.threads.length){
            for (int i = 0; i < this.threads.length; i++) {
                this.downloadSize += this.data.get(i+1)-(this.block * i);
            }
            print("已经下载的长度"+ this.downloadSize);
        }
    }else{
        throw new RuntimeException("2服务器响应错误 ");
    }
} catch (Exception e) {
    print(e.toString());
    throw new RuntimeException("3连接不到下载路径 ");
}
}
/**
 * 获取文件名
 */
private String getFileName(URLConnection conn) {
    String filename = this.url.toString().substring(this.url.toString().lastIndexOf("/") +
1);
    if(filename==null || "".equals(filename.trim())){//如果获取不到文件名称
        for (int i = 0;; i++) {
            String mine = conn.getHeaderField(i);
            if (mine == null) break;
```





```
if("content-disposition".equals(conn.getHeaderFieldKey(i).toLowerCase())){
    Matcher m =
    Pattern.compile(".*filename=(.*)").matcher(mine.toLowerCase());
    if(m.find()) return m.group(1);
}
}
filename = UUID.randomUUID()+ ".tmp";//默认取一个文件名
}
return filename;
}

/**
 * 开始下载文件
 * @param listener 监听下载数量的变化,如果不需要了解实时下载的数量,可以设置为 null
 * @return 已下载文件大小
 * @throws Exception
 */
public int download(DownloadProgressListener listener) throws Exception {
    try {
        if(this.data.size() != this.threads.length){
            this.data.clear();
            for (int i = 0; i < this.threads.length; i++) {
                this.data.put(i+1, this.block * i);
            }
        }
        for (int i = 0; i < this.threads.length; i++) {
            int downLength = this.data.get(i+1) - (this.block * i);
            if(downLength < this.block && this.data.get(i+1)<this.fileSize){ //该线程未完成下载时,继续下载
```





```
        RandomAccessFile        randOut        =        new
RandomAccessFile(this.saveFile, "rw");

        if(this.fileSize>0) randOut.setLength(this.fileSize);

        randOut.seek(this.data.get(i+1));

        this.threads[i] = new DownloadThread(this, this.url, randOut,
this.block, this.data.get(i+1), i+1);

        this.threads[i].setPriority(7);

        this.threads[i].start();

    }else{

        this.threads[i] = null;

    }

}

this.fileService.save(this.downloadUrl, this.data);

boolean notFinish = true;//下载未完成

while (notFinish) {// 循环判断是否下载完毕

    Thread.sleep(900);

    notFinish = false;//假定下载完成

    for (int i = 0; i < this.threads.length; i++){

        if (this.threads[i] != null && !this.threads[i].isFinish()) {

            notFinish = true;//下载没有完成

            if(this.threads[i].getDownLength() == -1){//如果下载失败,再
重新下载

                RandomAccessFile        randOut        =        new
RandomAccessFile(this.saveFile, "rw");

                randOut.seek(this.data.get(i+1));

                this.threads[i] = new DownloadThread(this, this.url,
randOut, this.block, this.data.get(i+1), i+1);

                this.threads[i].setPriority(7);

                this.threads[i].start();

            }

        }

    }

}
```





```
        }
    }
    if(listener!=null) listener.onDownloadSize(this.downloadSize);
}
fileService.delete(this.downloadUrl);
} catch (Exception e) {
    print(e.toString());
    throw new Exception("下载失败");
}
return this.downloadSize;
}
/**
 * 获取 Http 响应头字段
 * @param http
 * @return
 */
public static Map<String, String> getHttpResponseHeader(HttpURLConnection http) {
    Map<String, String> header = new LinkedHashMap<String, String>();
    for (int i = 0; i++) {
        String mine = http.getHeaderField(i);
        if (mine == null) break;
        header.put(http.getHeaderFieldKey(i), mine);
    }
    return header;
}
/**
 * 打印 Http 头字段
 * @param http
 */
public static void printResponseHeader(HttpURLConnection http){
```





```
Map<String, String> header = getHttpHeader(http);
for(Map.Entry<String, String> entry : header.entrySet()){
    String key = entry.getKey()!=null ? entry.getKey()+ ":" : "";
    print(key+ entry.getValue());
}

private static void print(String msg){
    Log.i(TAG, msg);
}

public static void main(String[] args) {
    /* FileDownloader loader = new FileDownloader(context,
"http://browse.babasport.com/ejb3/ActivePort.exe",
        new File("D:\\androidsoft\\test"), 2);
    loader.getFileSize();//得到文件总大小
    try {
        loader.download(new DownloadProgressListener(){
            public void onDownloadSize(int size) {
                print("已经下载: "+ size);
            }
        });
    } catch (Exception e) {
        e.printStackTrace();
    }
}*/
}
```

2) 下面的类是真正支持下载的线程类:





```
import java.io.InputStream;

import java.io.RandomAccessFile;

import java.net.HttpURLConnection;

import java.net.URL;

import android.util.Log;

public class DownloadThread extends Thread {

    private static final String TAG = "DownloadThread";

    private RandomAccessFile saveFile;

    private URL downUrl;

    private int block;

    /* 下载开始位置 */

    private int threadId = -1;

    private int startPos;

    private int downLength;

    private boolean finish = false;

    private FileDownloader downloader;

    public DownloadThread(FileDownloader downloader, URL downUrl,
RandomAccessFile saveFile, int block, int startPos, int threadId) {

        this.downUrl = downUrl;

        this.saveFile = saveFile;

        this.block = block;

        this.startPos = startPos;

        this.downloader = downloader;

        this.threadId = threadId;

        this.downLength = startPos - (block * (threadId - 1));

    }

}
```





```
@Override
public void run() {
    if(downLength < block){//未下载完成
        try {
            HttpURLConnection http = (HttpURLConnection)
downUrl.openConnection();
            http.setRequestMethod("GET");
            http.setRequestProperty("Accept", "image/gif, image/jpeg,
image/pjpeg, image/pjpeg, application/x-shockwave-flash, application/xaml+xml,
application/vnd.ms-xpsdocument, application/x-ms-xbap, application/x-ms-application,
application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*");
            http.setRequestProperty("Accept-Language", "zh-CN");
            http.setRequestProperty("Referer", downUrl.toString());
            http.setRequestProperty("Charset", "UTF-8");
            http.setRequestProperty("Range", "bytes=" + this.startPos + "-");
            http.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible;
MSIE 8.0; Windows NT 5.2; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET
CLR 3.0.04506.30; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)");
            http.setRequestProperty("Connection", "Keep-Alive");

            InputStream inStream = http.getInputStream();
            int max = 1024 * 1024;
            byte[] buffer = new byte[max];
            int offset = 0;
            print("线程 " + this.threadId + "从位置"+ this.startPos+ "开始下
载 ");
            while (downLength < block && (offset = inStream.read(buffer, 0,
max)) != -1) {
                saveFile.write(buffer, 0, offset);
                downLength += offset;
                downloader.update(this.threadId, block * (threadId - 1) +
downLength);
            }
        }
    }
}
```





```
        downloader.saveLogFile();
        downloader.append(offset);
        int spare = block-downLength;//求剩下的字节数
        if(spare < max) max = (int) spare;
    }
    saveFile.close();
    inStream.close();
    print("线程 " + this.threadId + "完成下载 ");
    this.finish = true;
    this.interrupt();
} catch (Exception e) {
    this.downLength = -1;
    print("线程"+ this.threadId+ ":"+ e);
}
}
}
private static void print(String msg){
    Log.i(TAG, msg);
}
/**
 * 下载是否完成
 * @return
 */
public boolean isFinish() {
    return finish;
}
/**
 * 已经下载的内容大小
 * @return 如果返回值为-1,代表下载失败
 */
```





```
public long getDownLength() {  
    return downLength;  
}  
}
```

- 3) 下面为监听器接口，会实时显示下载的大小，在实际使用的时候建议采用匿名类的方式构建此接口：

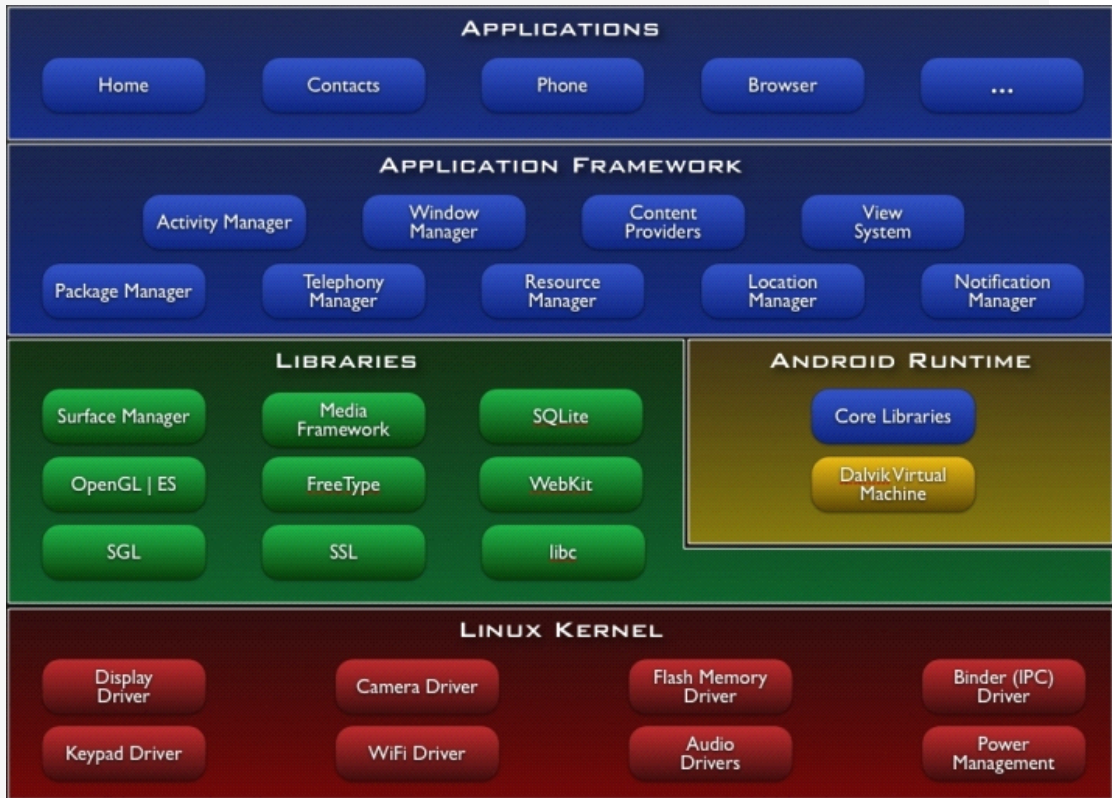
```
public interface DownloadProgressListener {  
    public void onDownloadSize(int size);  
}
```

上面的三个文件在一起就构建起了一个迷你型的 Android 下载框架，这个下载框架可以用于下载任何类型的二进制文件，以后需要下载的时候直接使用即可。其中 IoC 非常直接的体现就是 DownloadProgressListener，在使用的时候只需要只需要传入该接口一个实现实例即可自动的获取实时的下载长度。

二：精通 Android 体系架构、MVC、常见的设计模式、控制反转（IoC）

1，请看某个著名的 IT 公司一则招聘信息的其中一条要求：“熟悉 Android 系统架构及相关技术，1年以上实际 Android 平台开发经验；”，里面非常明确的说道要求熟练 Android 系统架构，这从某种程度上说明了对 Android 体系架构的理解的重要性，下面我们看看 Android 体系结构图，该图源自 Android 的文档：





很明显，上图包含四个主要的层次：

Linux Kernel: 负责硬件的驱动程序、网络、电源、系统安全以及内存管理等功能。

Libraries 和 Android Runtime: Libraries: 即 C/C++ 函数库部分，大多数都是开放源代码的函数库，例如 WebKit，该函数库负责 Android 网页浏览器的运行，例如标准的 C 函数库 Libc、OpenSSL、SQLite 等，当然也包括支持游戏开发 2D SGL 和 3D OpenGL | ES，在多媒体方面有 MediaFramework 框架来支持各种影音和图形文件的播放与显示，例如 MPEG4、H.264、MP3、AAC、AMR、JPG 和 PNG 等众多多媒体文件格式。Android 的 Runtime 负责解释和执行生成的 Dalvik 格式的字节码。

Application Framework (应用软件架构), Java 应用程序开发人员主要是使用该层封装好的 API 进行快速开发。

Applications: 该层是 Java 的应用程序层，Android 内置的 Google Maps、E-mail、即时通信工具、浏览器、MP3 播放器等处于该层，Java 开发人员开发的程序也处于该层，而且和内置的应用程序具有平等的位置，可以调用内置的应用程序，也可以替换内置的应用程序。

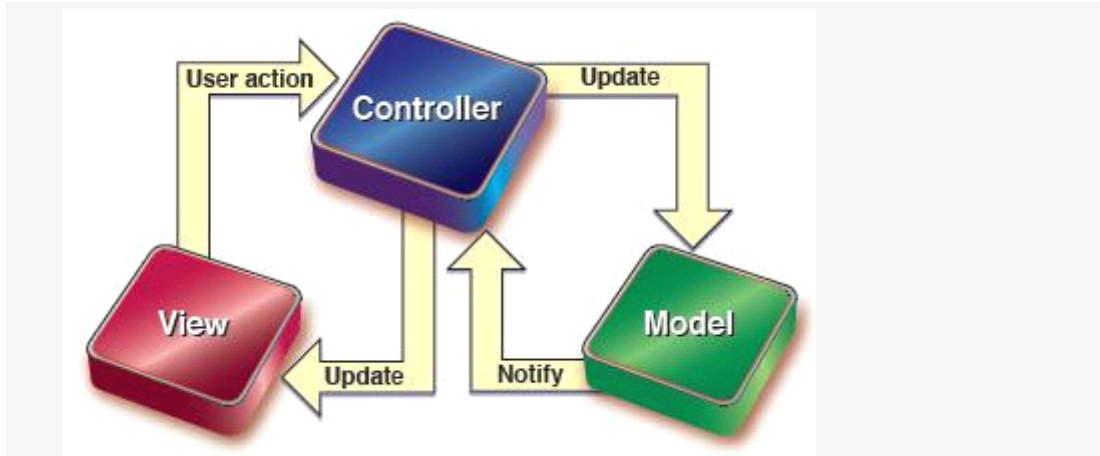
上面的四个层次，下层为上层服务，上层需要下层的支持，调用下层的 service，这种严格分层的方式带来的极大的稳定性、灵活性和可扩展性，使得不同层的开发人员可以按照规范专心特定层的开发。





Android 应用程序使用框架的 API 并在框架下运行,这就带来了程序开发的高度一致性, 另一方面也告诉我们, 要想写出优质高效的程序就必须对整个 Application Framework 进行非常深入的理解。精通 Application Framework, 你就可以真正的理解 Android 的设计和运行机制, 也就更能够驾驭整个应用层的开发。

2, Android 的官方建议应用程序的开发采用 MVC 模式。何谓 MVC? 先看看下图



MVC 是 Model,View,Controller 的缩写, 从上图可以看出 MVC 包含三个部分:

- 模型 (Model) 对象: 是应用程序的主体部分, 所有的业务逻辑都应该写在该层。
- 视图 (View) 对象: 是应用程序中负责生成用户界面的部分。也是在整个 MVC 架构中用户唯一可以看到的一层, 接收用户的输入, 显示处理结果。
- 控制器 (Control) 对象: 是根据用户的输入, 控制用户界面数据显示及更新 Model 对象状态的部分, 控制器更重要的一种导航功能, 想用用户出发的相关事件, 交给 M 哦得了处理。

Android 鼓励弱耦合和组件的重用, 在 Android 中 MVC 的具体体现如下:

- 1) 视图层 (View): 一般采用 XML 文件进行界面的描述, 使用的时候可以非常方便的引入, 当然, 如何你对 Android 了解的比较的多了话, 就一定可以想到在 Android 中也可以使用 JavaScript+HTML 等的方式作为 View 层, 当然这里需要进行 Java 和 JavaScript 之间的通信, 幸运的是, Android 提供了它们之间非常方便的通信实现。
- 2) 控制层 (Controller): Android 的控制层的重任通常落在了众多的 Activity 的肩上, 这句话也就暗含了不要在 Activity 中写代码, 要通过 Activity 交割 Model 业务逻辑层处理, 这样做的另外一个原因是 Android 中的 Activity 的响应时间是 5s, 如果耗时的操作放在这里, 程序就很容易被回收掉。
- 3) 模型层 (Model): 对数据库的操作、对网络等的操作都应该在 Model 里面处理, 当然对业务计算等操作也是必须放在的该层的。





3, 设计模式和 IoC(控制反转)

毫无疑问, Android 之所以能够成为一个开放的气象万千的系统, 与设计模式的精妙应用是分不开的, 只要你稍微用心观察, 就会发现在 Android 中到处都是 A 设计模式或者设计模式的联合运用, 一下的设计模式是您想游刃有余的驾驭 Android 必须掌握的:

- Template Method 模式
- Factory Method 模式
- Observer 模式
- Abstract Factory 模式
- Adapter 模式
- Composite 模式
- Strategy 模式
- State 模式
- Proxy 模式
- Bridge 模式
- Iterator 模式
- Mediator 模式
- Façade 模式

Android 框架魅力的源泉在于 IoC, 在开发 Android 的过程中你会时刻感受到 IoC 带来的巨大方便, 就拿 Activity 来说, 下面的函数是框架调用自动调用的:

```
protected void onCreate(Bundle savedInstanceState) ;
```

不是程序编写者主动去调用, 反而是用户写的代码被框架调用, 这也就反转了! 当然 IoC 本身的内涵远远不止这些, 但是从这个例子中也可以窥视出 IoC 带来的巨大好处。此类的例子在 Android 随处可见, 例如说数据库的管理类, 例如说 Android 中 SAX 的 Handler 的调用等。有时候, 您甚至需要自己编写简单的 IoC 实现, 上面展示的多线程现在就是一个说明。

三: 编写可重用、可扩展、可维护、灵活性高的代码

Android 应用程序的开发是使用 Java 编写, 在架构上使用 MVC, 鼓励组件之间的若耦合。开发出编写可重用、可扩展、可维护、灵活性高的代码需要经历遵循以下原则:

- "开-闭"原则(OCP): 一个软件实体应当对扩展开放, 对修改关闭。这个原则说的





是，在设计一个模块的时候，应当使这个模块可以在不被修改的前提下被扩展。换言之，应当可以在不必修改源代码的情况下改变这个模块的行为。

- 里氏代换原则 (LSP): 一个软件实体如果使用的是一个基类的话，那么一定使用于其子类，而且它根本不能察觉出基类对象和子类对象的区别。
- 依赖倒转原则(DIP): 要依赖于抽象,不要依赖于具体。
- 接口隔离原则 (ISP): 使用多个专门的接口比使用单一的总接口要好。一个类对另外一个类的依赖性应当是建立在最小的接口上的。
- 合成/聚合复用原则 (CARP): 又称合成复用原则(CRP),就是在一个新的对象里面使用一些已有的对象，使之成为新对象的一部分；新的对象通过向这些对象的委派达到复用已有功能的目的。简而言之就是：要尽量使用合成/聚合，尽量不要使用继承。
- 迪米特法则 (LoD): 又称最少知识原则 (LKP)，就是说一个对象应当对其他对象尽可能少的了解。狭义的迪米特法则是指如果两个类不必彼此直接通信,那么这两个类就不应当发生直接的相互作用.如果其中一个类需要调用另一个类的方法的话,可以通过第三者转发这个调用.。广义的迪米特法则是指一个模块设计得好坏的一个重要的标志就是该模块在多大的程度上将自己的内部数据与实现有关的细节隐藏起来。信息的隐藏非常重要的原因在于,它可以使各个子系统之间脱耦,从而允许它们独立地被开发,优化,使用阅读以及修改.。

灵活的使用设计模式可以在面对千变万化的业务需求是编写出可重用、可扩展、可维护、灵活性高的代码。

当然，由于 Android 是运行在移动设备上的，而移动设备的处理能力是有限的，所以有时间必须在编写可重用、可扩展、可维护、灵活性高的代码与高效的代码之间做出适当的平衡。

四：高效的编写高效的代码

高效快速的编写代码和编写高效率执行的代码很多时候都是对立的死敌，很多时候，你想快速的开发，代码的执行效率往往就会慢下来；你想编写高效的代码，开发速度就会慢下来。

不重复发明轮子和发明新的轮子是高效的编写高效的代码的正确是道路。

关于高效的代码，下面网络的一篇文章，直接转载（不知道是哪位哥们写的）如下：

“现代的手持设备，与其说是电话，更像一台拿在手中的电脑。但是，即使是“最快”的手持设备，其性能也赶不上一台普通的台式电脑。





这就是为什么我们在书写 Android 应用程序的时候要格外关注效率。这些设备并没有那么快, 并且受电池电量的制约。这意味着, 设备没有更多的能力, 我们必须把程序写的尽量有效。

本文讨论了很多能让开发者使他们的程序运行更有效的方法, 遵照这些方法, 你可以使你的程序发挥最大的效力。

对于占用资源的系统, 有两条基本原则:

1. 不要做不必要的事
2. 不要分配不必要的内存

所有下面的内容都遵照这两个原则。

有些人可能马上会跳出来, 把本节的大部分内容归于“草率的优化”(xing: 参见[The Root of All Evil]), 不可否认微优化 (micro-optimization。xing: 代码优化, 相对于结构优化) 的确会带来很多问题, 诸如无法使用更有效的数据结构和算法。但是在手持设备上, 你别无选择。假如你认为 Android 虚拟机的性能与台式机相当, 你的程序很有可能一开始就占用了系统的全部内存 (xing: 内存很小), 这会让你的程序慢得像蜗牛一样, 更遑论做其他的操作了。

Android 的成功依赖于你的程序提供的用户体验。而这种用户体验, 部分依赖于你的程序是响应快速而灵活的, 还是响应缓慢而僵化的。因为所有的程序都运行在同一个设备之上, 都在一起, 这就如果在同一条路上行驶的汽车。而这篇文档就相当于你在取得驾照之前必须要学习的交通规则。如果大家都按照这些规则去做, 驾驶就会很顺畅, 但是如果你不这样做, 你可能会车毁人亡。这就是为什么这些原则十分重要。

当我们开门见山、直击主题之前, 还必须要提醒大家一点: 不管 VM 是否支持实时 (JIT) 编译器 (xing: 它允许实时地将 Java 解释型程序自动编译成本机机器语言, 以使程序执行的速度更快。有些 JVM 包含 JIT 编译器。), 下面提到的这些原则都是成立的。假如我们有目标完全相同的两个方法, 在解释执行时 foo() 比 bar() 快, 那么编译之后, foo() 依然会比 bar() 快。所以不要寄希望于编译器可以拯救你的程序。

避免建立对象

世界上没有免费的对象。虽然 GC 为每个线程都建立了临时对象池, 可以使创建对象的代价变得小一些, 但是分配内存永远都比不分配内存的代价大。

如果你在用户界面循环中分配对象内存, 就会引发周期性的垃圾回收, 用户就会觉得界面像打嗝一样一顿一顿的。

所以, 除非必要, 应尽量避免尽力对象的实例。下面的例子将帮助你理解这条原则:

当你从用户输入的数据中截取一段字符串时, 尽量使用 substring 函数取得原始数据的一个子串, 而不是为子串另外建立一份拷贝。这样你就有一个新的 String 对象, 它与原始数据共





享一个 char 数组。

如果你有一个函数返回一个 String 对象，而你确切的知道这个字符串会被附加到一个 StringBuffer，那么，请改变这个函数的参数和实现方式，直接把结果附加到 StringBuffer 中，而不要再建立一个短命的临时对象。

一个更极端的例子是，把多维数组分成多个一维数组。

int 数组比 Integer 数组好，这也概括了一个基本事实，两个平行的 int 数组比(int,int)对象数组性能要好很多。同理，这试用于所有基本类型的组合。

如果你想用一种容器存储(Foo,Bar)元组，尝试使用两个单独的 Foo[]数组和 Bar[]数组，一定比(Foo,Bar)数组效率更高。(也有例外的情况，就是当你建立一个 API，让别人调用它的时候。这时候你要注重对 API 借口的设计而牺牲一点儿速度。当然在 API 的内部，你仍要尽可能的提高代码的效率)

总体来说，就是避免创建短命的临时对象。减少对象的创建就能减少垃圾收集，进而减少对用户体验的影响。

使用本地方法

当你在处理字符串的时候，不要吝惜使用 String.indexOf(), String.lastIndexOf()等特殊实现的方法 (specialty methods)。这些方法都是使用 C/C++实现的，比起 Java 循环快 10 到 100 倍。

使用实类比接口好

假设你有一个 HashMap 对象，你可以将它声明为 HashMap 或者 Map:

```
Map myMap1 = new HashMap();  
HashMap myMap2 = new HashMap();  
哪个更好呢?
```

按照传统的观点 Map 会更好些，因为这样你可以改变他的具体实现类，只要这个类继承自 Map 接口。传统的观点对于传统的程序是正确的，但是它并不适合嵌入式系统。调用一个接口的引用会比调用实体类的引用多花费一倍的时间。

如果 HashMap 完全适合你的程序，那么使用 Map 就没有什么价值。如果有些地方你不能确定，先避免使用 Map，剩下的交给 IDE 提供的重构功能好了。(当然公共 API 是一个例外：一个好的 API 常常会牺牲一些性能)

用静态方法比虚方法好

如果你不需要访问一个对象的成员变量，那么请把方法声明成 static。虚方法执行的更快，因为它可以被直接调用而不需要一个虚函数表。另外你也可以通过声明体现出这个函数的调用不会改变对象的状态。





不用 `getter` 和 `setter`

在很多本地语言如 C++ 中，都会使用 `getter`（比如：`i = getCount()`）来避免直接访问成员变量（`i = mCount`）。在 C++ 中这是一个非常好的习惯，因为编译器能够内联访问，如果你需要约束或调试变量，你可以在任何时候添加代码。

在 Android 上，这就不是个好主意了。虚方法的开销比直接访问成员变量大得多。在通用的接口定义中，可以依照 OO 的方式定义 `getters` 和 `setters`，但是在一般的类中，你应该直接访问变量。

将成员变量缓存到本地

访问成员变量比访问本地变量慢得多，下面一段代码：

Java 代码

```
1 for (int i = 0; i < this.mCount; i++)
2     dumpItem(this.mItems[i]);
```

最好改成这样：

Java 代码

```
3 int count = this.mCount;
4 Item[] items = this.mItems;
5 for (int i = 0; i < count; i++)
6     dumpItems(items[i]);
```

（使用 "this" 是为了表明这些是成员变量）

另一个相似的原则是：永远不要在 `for` 的第二个条件中调用任何方法。如下面方法所示，在每次循环的时候都会调用 `getCount()` 方法，这样做比你在一个 `int` 先把结果保存起来开销大很多。

Java 代码

```
7 for (int i = 0; i < this.getCount(); i++)
8     dumpItems(this.getItem(i));
```

同样如果你要多次访问一个变量，也最好先为它建立一个本地变量，例如：

Java 代码

```
9 protected void drawHorizontalScrollBar(Canvas canvas, int width, int height)
10 {
11     if (isHorizontalScrollBarEnabled()) {
```





```
11     int size = mScrollBar.getSize(false);
12     if (size <= 0) {
13         size = mScrollBarSize;
14     }
15     mScrollBar.setBounds(0, height - size, width, height);
16
mScrollBar.setParams(computeHorizontalScrollRange(), computeHorizontalScroll
Offset(), computeHorizontalScrollExtent(), false);
17     mScrollBar.draw(canvas);
18 }
19 }
```

这里有 4 次访问成员变量 `mScrollBar`，如果将它缓存到本地，4 次成员变量访问就会变成 4 次效率更高的栈变量访问。

另外就是方法的参数与本地变量的效率相同。

使用常量

让我们来看看这两段在类前面的声明：

Java 代码

```
20 static int intVal = 42;
21 static String strVal = "Hello, world!";
```

必以其会生成一个叫做 `<clinit>` 的初始化类的方法，当类第一次被使用的时候这个方法会被执行。方法会将 42 赋给 `intVal`，然后把一个指向类中常量表引用赋给 `strVal`。当以后要用到这些值的时候，会在成员变量表中查找到他们。

下面我们做些改进，使用“`final`”关键字：

Java 代码

```
22 static final int intVal = 42;
23 static final String strVal = "Hello, world!";
```

现在，类不再需要 `<clinit>` 方法，因为在成员变量初始化的时候，会将常量直接保存到类文件中。用到 `intVal` 的代码被直接替换成 42，而使用 `strVal` 的会指向一个字符串常量，而不是使用成员变量。

将一个方法或类声明为“`final`”不会带来性能的提升，但是会帮助编译器优化代码。举例说，如果编译器知道一个“`getter`”方法不会被重载，那么编译器会对其采用内联调用。






你也可以将本地变量声明为"final", 同样, 这也不会带来性能的提升。使用"final"只能使本地变量看起来更清晰些(但是也有些时候这是必须的, 比如在使用匿名内部类的时候)(xing: 原文是 or you have to, e.g. for use in an anonymous inner class)

谨慎使用 **foreach**

foreach 可以用在实现了 Iterable 接口的集合类型上。foreach 会给这些对象分配一个 iterator, 然后调用 hasNext()和 next()方法。你最好使用 foreach 处理 ArrayList 对象, 但是对其他集合对象, foreach 相当于使用 iterator。

下面展示了 foreach 一种可接受的用法:

Java 代码 

```
24 public class Foo {
25     int mSplat;
26     static Foo mArray[] = new Foo[27];
27
28     public static void zero() {
29         int sum = 0;
30         for (int i = 0; i < mArray.length; i++) {
31             sum += mArray[i].mSplat;
32         }
33     }
34
35     public static void one() {
36         int sum = 0;
37         Foo[] localArray = mArray;
38         int len = localArray.length;
39         for (int i = 0; i < len; i++) {
40             sum += localArray[i].mSplat;
41         }
42     }
43
44     public static void two() {
45         int sum = 0;
46         for (Foo a: mArray) {
47             sum += a.mSplat;
48         }
49     }
50 }
```





在 zero()中, 每次循环都会访问两次静态成员变量, 取得一次数组的长度。
retrieves the static field twice and gets the array length once for every iteration through the loop.

在 one()中, 将所有成员变量存储到本地变量。
pulls everything out into local variables, avoiding the lookups.

two()使用了在 java1.5 中引入的 foreach 语法。编译器会将数组的引用和数组的长度保存到本地变量中, 这对访问数组元素非常好。但是编译器还会在每次循环中产生一个额外的对本地变量的存储操作(对变量 a 的存取)这样会比 one()多出 4 个字节, 速度要稍微慢一些。

综上所述: foreach 语法在运用于 array 时性能很好, 但是运用于其他集合对象时要小心, 因为它会产生额外的对象。

避免使用枚举

枚举变量非常方便, 但不幸的是它会牺牲执行的速度和并大幅增加文件体积。例如:

```
public class Foo {public enum Shrubbery { GROUND, CRAWLING, HANGING }}
```

会产生一个 900 字节的 .class 文件(Foo\$Shubbery.class)。在它被首次调用时, 这个类会调用初始化方法来准备每个枚举变量。每个枚举项都会被声明成一个静态变量, 并被赋值。然后将这些静态变量放在一个名为"\$VALUES"的静态数组变量中。而这么一大堆代码, 仅仅是为了使用三个整数。

这样:

Shrubbery shrub = Shrubbery.GROUND;会引起一个对静态变量的引用, 如果这个静态变量是 final int, 那么编译器会直接内联这个常数。

一方面说, 使用枚举变量可以让你的 API 更出色, 并能提供编译时的检查。所以在通常的时候你毫无疑问应该为公共 API 选择枚举变量。但是当性能方面有所限制的时候, 你就应该避免这种做法了。

有些情况下, 使用 ordinal()方法获取枚举变量的整数值会更好一些, 举例来说, 将:

Java 代码

```
51 for (int n = 0; n < list.size(); n++) {  
52     if (list.items[n].e == MyEnum.VAL_X)// do stuff 1  
53     else if (list.items[n].e == MyEnum.VAL_Y)// do stuff 2  
54 }
```





替换为:

Java 代码

```
55 int valX = MyEnum.VAL_X.ordinal();
56 int valY = MyEnum.VAL_Y.ordinal();
57 int count = list.size();
58 MyItem items = list.items();
59 for (int n = 0; n < count; n++) {
60     int valItem = items[n].e.ordinal();
61     if (valItem == valX) // do stuff 1
62     else if (valItem == valY) // do stuff 2
63 }
```

会使性能得到一些改善，但这并不是最终的解决之道。

将与内部类一同使用的变量声明在包范围内

请看下面的类定义:

Java 代码


```
64 public class Foo {
65     private int mValue;
66     public void run() {
67         Inner in = new Inner();
68         mValue = 27;
69         in.stuff();
70     }
71
72     private void doStuff(int value) {
73         System.out.println("Value is " + value);
74     }
75
76     private class Inner {
77         void stuff() {
78             Foo.this.doStuff(Foo.this.mValue);
79         }
80     }
81 }
```





这其中的关键是，我们定义了一个内部类(Foo\$Inner)，它需要访问外部类的私有域变量和函数。这是合法的，并且会打印出我们想要的结果"Value is 27"。

问题是在技术上来讲（在幕后）Foo\$Inner 是一个完全独立的类，它要直接访问 Foo 的私有成员是非法的。要跨越这个鸿沟，编译器需要生成一组方法：

Java 代码 

```
82 static int Foo.access$100(Foo foo) {
83     return foo.mValue;
84 }
85
86 static void Foo.access$200(Foo foo, int value) {
87     foo.doStuff(value);
88 }
```

内部类在每次访问"mValue"和"doStuff"方法时，都会调用这些静态方法。就是说，上面的代码说明了一个问题，你是在通过接口方法访问这些成员变量和函数而不是直接调用它们。在前面我们已经说过，使用接口方法（getter、setter）比直接访问速度要慢。所以这个例子就是在特定语法下面产生的一个“隐性的”性能障碍。

通过将内部类访问的变量和函数声明由私有范围改为包范围，我们可以避免这个问题。这样做可以让代码运行更快，并且避免产生额外的静态方法。（遗憾的是，这些域和方法可以被同一个包内的其他类直接访问，这与经典的 OO 原则相违背。因此当你设计公共 API 的时候应该谨慎使用这条优化原则）

避免使用浮点数

在奔腾 CPU 出现之前，游戏设计者做得最多的就是整数运算。随着奔腾的到来，浮点运算处理器成为了 CPU 内置的特性，浮点和整数配合使用，能够让你的游戏运行得更顺畅。通常在桌面上，你可以随意的使用浮点运算。

但是非常遗憾，嵌入式处理器通常没有支持浮点运算的硬件，所有对"float"和"double"的运算都是通过软件实现的。一些基本的浮点运算，甚至需要毫秒级的时间才能完成。

甚至是整数，一些芯片有对乘法的硬件支持而缺少对除法的支持。这种情况下，整数的除法和取模运算也是有软件来完成的。所以当你在使用哈希表或者做大量数学运算时一定要





小心谨慎。”

五，学会至少一门服务器端开发技术

可能有朋友会问：学习 Android 应用程序开发为什么还需要学习学会至少一门服务器端开发技术呢？答案如下：一方面 Android 号称是首个为移动终端打造的真正开放和完整的移动软件。作为一种移动终端，必须与服务器端结合才能发挥巨大的作用。简言之，需要：云端+云的方式。Android 是为移动互联网时代量身打造的，移动互联网时代的服务模式是“手机终端+互连网络+应用软件”，移动互联网时代应用技术之一的 Android 只是用于开发移动终端软件，而服务端技术用于开发互连网络应用，所以未来移动互联网时代软件的主流应用模式将是“手机客户端+互连网络应用服务端”，这种模式要求做移动互联网开发的程序员不但要掌握像 Android 这样的手机终端软件技术还要掌握开发互连网络应用的服务端技术。目前，软件企业普遍存在这样的问题，做移动互联网开发 Android 终端软件的程序员不了解 web 应用技术，而做 web 应用的程序员不了解移动终端技术，这样就导致了客户端与服务端在衔接上出现了问题。目前的现状是：既掌握移动互联网 Android 终端技术，又掌握 web 应用技术的程序员比较稀缺，随着中国步入移动互联网时代，企业对这种移动互联网时代综合性人才的需求很旺盛。如果不了解 web 应用技术，最终会遇到了技术和发展的瓶颈；另一方面，Google 联合 OHA 推出的真正优势之一也在于和互联网结合，Google 的用意之一也是想开辟新的终端去使用 Google 的优势服务。

服务器端开发技术目前主流的有 Sun 的 Java EE、微软的 .NET，开源的以 PHP 和 MySQL 为代表的 LAMP 体系，我们该选择哪一种呢？从理论上讲，很多人倾向于选择 Java EE，毕竟它们都是使用 Java 作为开发语言的，但是很多人面对 Java EE 众多的框架就望而生畏，其实在学习 Java EE 的时候可以从 Struts 入手，随着业务的需求逐步深入。当然，选择微软的 .NET 也行，毕竟该技术体系也占有很大 市场份额。其实，笔者认为，选择 LAMP 可以是会获得最高的“性价比”的，一方面 PHP 是现在 Web 方面的主流语言，大多数新型的网站尤其是创业性质的网站一般都会选用 PHP 作为服务端开发语言，另一方面，前面也说过，Android 是为移动互联而生的，两者达到了完美的契合。

如果您精通 Android，又精通 LAMP、Java EE、.NET，请联系笔者：

官方博客：<http://www.cnblogs.com/guoshiandroid/>

官方 Email：guoshiandroid@gmail.com

官方讨论 QQ 群：65882321

