- 基本流程 示： App.c:文件名/ **main**:函数名/ *文字*:函数说明/ (文字):解释
- AppCrawler.scala
  - **getGlobalEncoding** 方法、**setGlobalEncoding** 方法：略。
  - **main** 方法：所有执行都从 main 函数进去。会先解析参数，内容很多。所有逻辑都在解析参数中，参数类型已经在 **createParser** 的 help("help") text 中给出。调用 **parseParams** 方法。
  - **createParser** 方法：略。
  - **parseParams** 方法：解析参数。根据参数类型，执行不同功能，包括设置基础的 log 等，最后会打印 config。命令行中可以传递 capability 对已有配置文件进行覆盖。接着是各种参数解析。直到最后来执行，一种是遍历一种是报告。

    其中一个命令模式是生成 demo 文件，在 if(config.demo)段中，生成后自动退出。其余功能是遍历，包括未完成的（基于 PO 的自动）生成测试用例源代码的模板 if(config.template)代码块、关于 diff 逻辑的 if(config.candidate)代码块（其功能为如若不为空则 diff）、没有用到的可实现报告重复生成的功能（把所有 log 删掉，只要有 elements.yml 文件和 store 存在，会重新根据文件的点击顺序再次重新生成报告），其可以用来针对修改报告中断言后，预期某些步骤报错后可以重新生成报告。最终调用 **startCrawl**。
  - **parsePath** 方法：略。
  - **startCrawl** 方法：初始化 Crawler 对象，从配置文件里面加载配置，并 start。
  - **addLogFile** 方法：略。
- Crawler.scala
  - **loadPlugins** 方法：增加插件，内默认全加载，也支持他人自行编写插件。
  - **loadConf** 方法：略。
  - **start** 方法：所有 log 都在 AppCrawler 里，通过 *log.addAppender(AppCrawler.fileAppender)*把当前的 log 增加一个输入到总体 log 里面。debug 数据并打印配置文件方便调试。

    若填写了 xpath attribute，则会打印 attribute 并把 Xpath 的表达式放到 XpathUtil 类中，XpathUtil 依赖于 xpath 配置。xpath attribute 是为了表示最后生成 xpath 时为每个标记的元素生成唯一的 xpath 和关键的元素，在此允许配置。调用 **loadPlugins** 方法。

    判断 existDriver，若为空则通过 **setupAppium** 重新建立一个与 Appium 的连接，调用 **setupAppium** 方法。否则使用已经存在的，其通常为空所以多数情况下用不到。会在不再使用独立的命令行或集成到测试用例中时调用，如某 appium 测试用例只需把 driver 显式的赋值过来，从而不需在 appcrawler 中重新连接 appium。.

    通过 log.info 打印相关信息，获取设备屏幕分辨率等。调用 **firstRefresh()**进行第一次刷新。处于 appWhiteList 白名单的 app 可以自动进行遍历，例如测试时分享到了微信，则可以继续在微信中进行遍历（微信添加到白名单中）。不在白名单之内的则会跳出 app 进行回退。系统默认会把当前要测试的 app 添加到白名单中，appNameRecord 代表当前 app，在 **parsePageContext** 解析配置上下文的第一次刷新时进行了初始化。

    判断 testcase 是否为空，不为空则调用 **runSteps**，否则提示 no testcase。接着判断 selectedList 不为空（有 bug，若其设置为空则会包空指针异常，因此还需判断不为 null），则会调用 **crawlWithRetry** 方法。
  - **crawlWithRetry** 方法：爬取对应给定深度，如果超过深度则会产生异常。参数深度覆盖给 maxDepth，清空 urlStack(每个页面都有 activity 对象，把其存入该堆栈中，以字符串格式存入 activity 的 name，退出界面或路径后会弹栈)确保没有内容，并通过 **refreshPage** 刷新页面，接着注册关键按键 **handleCtrlC**。启动第一次，判断 errorCount 报错数量，若有则运行异常捕获逻辑 try(crawl())，若报错则打印 log，对 errorCount+1 并重新执行 try(crawl())，只要 appium 有异常则会重新启动 appium 继续运行。
  - **restart** 方法：略。
  - **firstRefresh** 方法：刷新当前页面，获取当前 pagesource。通过 **getEventElement** 对自己添加标记代表当前已经开始，分别执行 **beforeElementAction**、**doElementAction** 和 **afterElementAction** 来确保三次？逻辑。从而创建第一个 start 事件。确保了整个流程有初始状态。

0_com.xueqiu.android.MainActivity.tag=Start.depth=2.id=Start.name=Start.clicked.png
0_com.xueqiu.android.MainActivity.tag=Start.depth=2.id=Start.name=Start.dom
  - **getEventElement** 方法：略。
  - **runSteps** 方法：有一个自动化的引擎会运行 crawler 下的方法。如果 CrawlerConf.scala 中有 testcase，则会执行 **testcase** 方法：*遇到任何元素则会等待 1000 毫秒*。开发者可以在此函数中追加 case。
  - **setupAppium** 方法：初始化重要变量，防止运行中失败重试没有初始化。打印 afterAllMax 等关键数据。系统是否加载 Util.isLoaded 为 false（其调用 object Util。Util 是 scala 另外一个执行引擎，可动态执行语句。将来 **Util.scala** 需要重构，替换为 bean shell）

    在原有 appium 的 capability 的增加一个 appium，其实是 appium server。可自定义 appium 的地址。如使用 adb 时则不需要这些语句。

    从配置文件里面查找并判断 automationName，支持三种模式。如果是 selenium 则会通过 selenium driver 测试网页；如果是 adb 则启动 adb driver；其他默认为 appium 并启动 appium client。最终通过 *driver=new AppiumClient(url,conf.capability)*指向 Driver 对象。（AppiumDriver 继承与 Selenu

imDrvier，SelenuimDrvier 继承于 ReactWebDriver）该对象来自于 ReactWebDriver，把 API 进行了封装作为了代理类。

Ga.log 为向谷歌分析服务器发送日志，appcrawler 开发者可查询使用的人数。

- **getSchema** 方法、**getUri** 方法、**getUrlElementByMap** 方法：略。
- **handleCtrlC** 方法：一旦按下 Ctrl+C 按键则会执行 **stop** 操作，执行退出逻辑。
- **stop** 方法：stopAll 置为 true（变量再也没有被用过），exitCrawl 置为 true（可能无必要），设置调用的每个插件为停止状态后退出，包括 ReportPlugin 类的 **stop** 方法。
- **refreshPage** 方法：把当前内容进行刷新。填充当前 PageSource 从而给出相应的内容。使用 **parse PageContenxt** 解析当前包名等。
- **parsePageContext** 方法：？，(urlStack 栈，进入或退出某页面就压栈或弹栈该页面的 ActivityName)。如果将要压栈的 ActivityName 存在于 urlStack 中，则会把该 ActivityName 上方的栈全弹出，否则压栈到 urlStack 中。深度也是基于栈来计算的。contentHash 为 DataRecord 类型，其中追加 TData.scala 的 **md5** 方法，判断用户界面是否有变化。
- **cwawl** 方法：伪递归（scala 把递归自动拆成循环，java 则有可能发生栈溢出）。在任何代码中若 exitCrawl 被置为 true 则退出。否则 log.info 打印出 crawl next，作为爬取的标志。nextElement 代表下一个要点击的内容，最初为空。通过 **needExitApp** 判断是否退出 app，若是则置 exitCrawl 为 true。通过判断每次刷新的结果 refreshResult 是否失败，提示刷新失败获取不到 pageSrouce 并进行后退，否则打印 refresh sussces 成功。

  判断 nextElement 是否仍为空，为空则执行算法 **getElementByTriggerActions** 得到元素并赋值给 nextElement。若 Trigger 中无所需元素，则 nextElement 仍为空，执行 **needBackToApp**，判断当前元素是否已经跳出需要测试的 app，从而后退到需要测试的 app。否则若 nextElement 仍为空，执行 **needBackToPage**，判断是否需要返回上一层。否则若 nextElement 仍为空，执行到 **getAvailableElement**，查找正常的元素，得到优先被遍历的第一个元素。

  否则若 nextElement 仍为空，证明所有的元素都已经被遍历过，已经找不到元素需要被遍历，于是判断当前没有被遍历的 size 是否小于 10，输出一次 log，否则判断 afterall 并查看是否有下拉与上滑等动作尝试查找新元素，无论是否有最终都会（本自然段范围）后退 **getBackButton**。

  此时 nextElement 一定有值，先 **fixElementAction** 修正元素动作，如果不跳过，则会顺序执行 **beforeElementAction**、**doElementAction** 和 **afterElementAction** 对元素进行相关操作（记录事件、请求、点击等数据）。否则通过 setElementSkip 会把元素设置为跳过不再执行。最终递归重新遍历 **cwawl** 函数，直到按下 Ctrl+C 调用 **crawlWithRetry** 方法中的 **handleCtrlC** 方法（还没实现按键退出，且还没有其他退出方法）。
- **needExitApp** 方法：判断是否退出 app，包括超时、一直后退超过最大次数也找不到新元素等。
- **getElementByTriggerActions** 方法：配置文件中有 triggerActions（包括 xpath、action 和 times），进行遍历，判断当前页面是否有需要的内容，若有则点击。step.use()，times 次数用过后减一。可合并到 **getAvailableElement** 方法前。
- **needBackToApp** 方法：判断当前的 appNameRecord 是否是在白名单或者是否是需要测试的 app 自身，如果不是则认为已经退出或者跳转到了其他 app。通过 Some(**getEventElement**("BackApp")) 创建事件进行退出。
- **needBackToPage** 方法：urlBlackList 为黑名单，如不遍历当前页面则可以将其加入到黑名单，在测试遍历时如遇到时则可以退出该页面。
- **getAvailableElement** 方法：从 selectedList 中选择所有元素，依次寻找后将所有元素加入到 selectedElements 中。对 selectedElements 进行去重。检查 blackList 黑名单，查找黑名单中是否也有所需元素，若有则把从之前找到的元素 selectedElements 通过 diff 把黑名单中元素去掉，实现使之没有黑名单中的内容。通过 **isSmall** 判断其内元素是过小，若是则跳过并删掉。如果是后退键，则会在所有元素遍历完成之后再遍历进行后退，将其排除过滤。

  若元素已经被点击过则会通过 store.isClicked 排除过滤（需要重构，判断点击次数）。若元素需要被跳过则会通过 store.isSkipped 排除过滤（在 AppCrawler 中，列表页不需要全部遍历，一般只需遍历前面几个，若已经被遍历则会列表后面元素会被跳过）。最终将其行排序，（一般的遍历工具会按照位置顺序点击，在 AppCrawler 中则会）通过 **sortByAttribute** 对所有元素或控件根据属性进行分级排序。再通过 firstList 和 lastList 分别查找其中元素（待优化），进行 diff 后确保了 firstElements 在前，默认的 selectedElements 在中间，lastElements 在最后，接着去重。最终返回 all 的第一个 headOption。
- **isValid** 方法：略。
- **isSmall** 方法：通过比较 element.getX 和 getY 与屏幕尺寸判断是否可见。比较 element.getHeight 和 getWidth 与某一定值判断是否过小。从而排除掉这些元素与控件。
- **sortByAttribute** 方法：首先，判断当前元素在所在页面中的深度 depth（根据页面中控件的子集关系），深度越大的优先级越高。其次，对于导航菜单栏 TabLayout 中的页面（如微信底栏），没有被选中的菜单页面优先级越高；对于已选中的菜单，其同级的未选中的菜单也处于相同的优先级后（算法不完善）。接着，对于列表页面，通过判断 getAncestor.contains 元素的祖先是否包含 List，从而做到列表中的子元素能被优先点击。从而人性化和贴近真实。
- **ifWebViewPage** 方法：略。
- afterUrlRefresh
- beforeElementAction、afterElementAction

- getPredictBackNodes
- getURIElementsByStep
- isEndlessLoop
- **getBackButton**
- **fixElementAction** 方法：交给每个插件来完成，修改当前元素是否需要点击。
- **doElementAcion** 方法：根据原生 urlElement，？，通过 findElementByURI 寻找元素，接着调用。?。
- saveElementScreenshot、saveLog、getBasePathName、saveDom、saveScreen、back：略。

- ReportPlugin.scala
  - **stop** 方法：调用 getCrawler 的 **savelog** 方法，接着通过 **generateReport** 生成报告。
  - **generateReport** 方法：生成报告。
- TData.scala
  - 工具类
  - **md5** 方法：删除一些无用数据，取出 xpath、value、selected 和 text，所有属性连接自恋后的字符串是不是谓一致。
- AppiumClient.scala
  - **findElementByURI** 方法：调用 findElementsById 取到坐标，通过 click()