



项目分析与改进

UI Automator Python Wrapper 与 JsonRPCServer

网页版请访问 fancylear.github.io

建议您使用1080p分辨率

Sep-24 李阳

提纲

Outlines

- 简介 – 是什么
- 使用 – 怎么用
- 功能 – Python wrapper的功能
- 结构 – Python wrapper和JsonRPCServer的结构与原理
- 流程与逻辑
- JSONRPC原理 – JSONRPC原理
- 功能改进 – 与作者沟通并尝试改进
- 资源 – 所有项目资源以及HTML5幻灯片可以访问 fancylear.github.io 博客上获取
- 参考

所有链接用橘红色标明, 所有指令用蓝色标明

简介

Introduction

➤ 手动测试->繁琐,局限->自动化测试->[UIAutomator](#)

➤ 优点： 1 / 多种定位UI元素的方式

2 / 精确模拟多种操作

3 / 可以自定义Watcher来解决测试过程中的意外，如电话/ANR对话框

4 / 使用Java，对Android开发者友好

缺点： 1 / 需求Android 4.1以上 (API >= 16)

2 / 需要编译运行，部署和修改较繁琐 (Java带来的缺点)

简介

Introduction

- ▶ **Java**导致了部分缺点->繁琐,局限->用**Python**代替并引入远程调试->**Python Wrapper**
- ▶ 1 / Python代替Java (无编译环节,直接执行/修改)
- ▶ 2 / 引用了**jsonrpcserver**, 把手机变为HTTP服务器 (省去部署,在PC端可执行)
- ▶ ...

使用

How to use

► 安装

```
$ pip install uiautomator
```

(使用pip插件安装, 这是一个python library, 包含一个python写的wrapper和编译好的rpcserver类)

```
requires = [  
    "urllib3>=1.7.1"  
]  
test_requires = [  
    'nose>=1.0',  
    'mock>=1.0.1',  
    'coverage>=3.6'  
]
```

依赖于如图 所示的python library

► 手动启动RPCServer方法

► 1/ push jar文件 (libs目录下的) 到 /data/local/tmp/

► 2/ 开启服务器 `adb shell uiautomator runtest bundle.jar uiautomator-stub.jar -c com.github.uiautomatorstub.Stub`

► 3/ 端口映射PC-手机 `adb forward tcp:9008 tcp:9008`

► 4/ 检查是否正常启动 `curl -d '{"jsonrpc":"2.0","method":"deviceInfo","id":1}' localhost:9008/jsonrpc/0`



功能

Functions

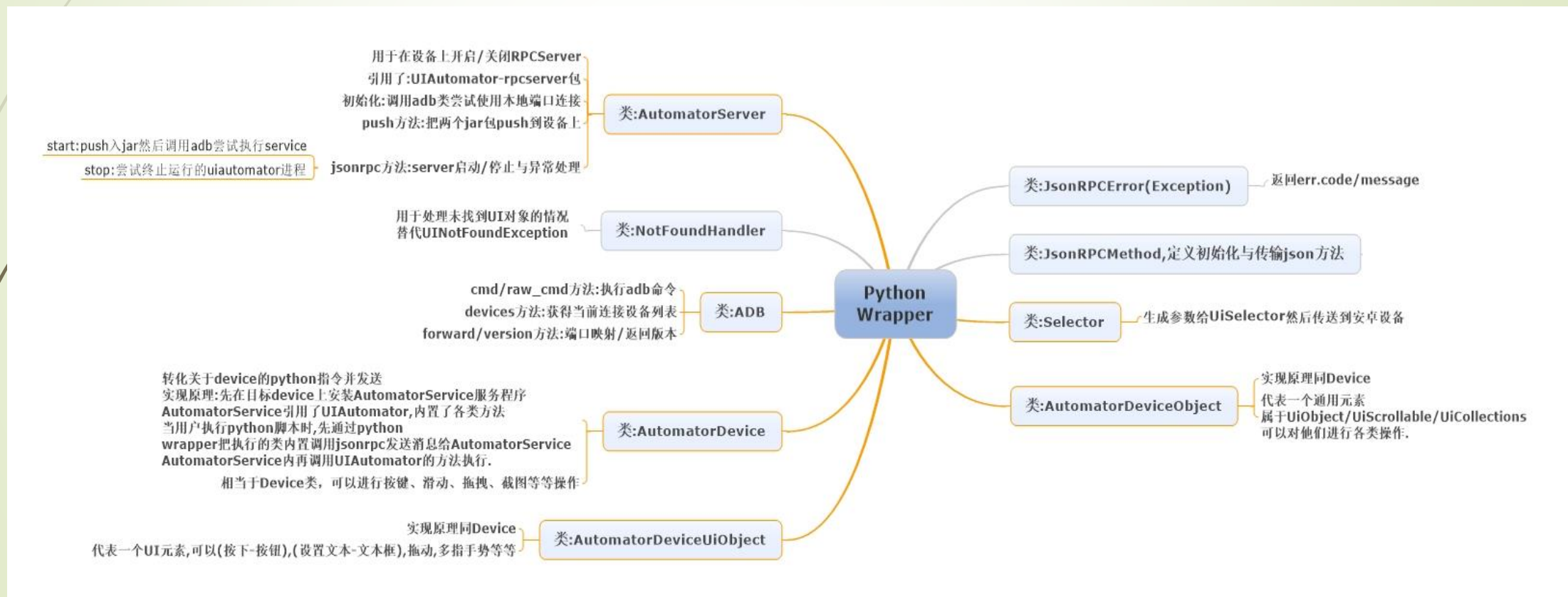
这一部分做成了HTML5页面，请访问 [HTML5幻灯片](#)

注：HTML5功能介绍Slide中大量引用了原作者的解释

结构

Structure

Python Wrapper项目结构



结构

Structure

- 图中对所有类功能进行了分析说明，这里详细说明相对重要的python wrapper原理部分
- 上张结构图中标为“橘红色”的类属于python wrapper
- 下面我们来看“press”方法在python wrapper中如何定义的

```
def press(self):
    """
    press key via name or key code. Supported key name includes:
    home, back, left, right, up, down, center, menu, search, enter,
    delete(or del), recent(recent apps), volume_up, volume_down,
    volume_mute, camera, power.
    Usage:
    d.press.back() # press back key
    d.press.menu() # press home key
    d.press(89)   # press keycode
    """
    @param_to_property(
        key=["home", "back", "left", "right", "up", "down", "center",
            "menu", "search", "enter", "delete", "del", "recent",
            "volume_up", "volume_down", "volume_mute", "camera", "power"]
    )
    def _press(key, meta=None):
        if isinstance(key, int):
            return self.server.jsonrpc.pressKeyCode(key, meta) if meta else self.server.jsonrpc.pressKeyCode(key)
        else:
            return self.server.jsonrpc.pressKey(str(key))
    return _press
```


结构

Structure

- ▶ 可以看到，实际上调用的是jsonrpc类，并调用“pressKeyCode”方法
- ▶ 我们再看一下jsonrpc类，直接返回并调用了JsonRPCMethod类
- ▶ 在异常处理中其内部包含了启动服务器的逻辑(server.start/stop)

```
def jsonrpc(self):  
    return self.jsonrpc_wrap(timeout=int(os.environ.get("jsonrpc_timeout", 90)))  
  
def jsonrpc_wrap(self, timeout):  
    server = self  
    ERROR_CODE_BASE = -32000  
  
    def _JsonRPCMethod(url, method, timeout, restart=True):  
        _method_obj = JsonRPCMethod(url, method, timeout)  
  
    def wrapper(*args, **kwargs):  
        URLError = urllib3.exceptions.HTTPError if os.name == "nt" else urllib2.URLError  
        try:  
            return _method_obj(*args, **kwargs)  
        except (URLError, socket.error, HTTPException) as e:  
            if restart:  
                server.stop()  
                server.start()  
            return _JsonRPCMethod(url, method, timeout, False)(*args, **kwargs)  
        else:
```

注：server.start/stop代码内容为调用本项目的adb类输入手动启动指令

结构

Structure

- ▶ 可以看到JsonRPCMethod根据传递来的参数“pressKeyCode”生成(dump)json消息并发送给HTTP服务器(使用“post” http请求)
- ▶ 下面我们看服务器端（手机），如何执行

```
class JsonRPCMethod(object):  
  
    if os.name == 'nt':  
        pool = urllib3.PoolManager()  
  
    def __init__(self, url, method, timeout=30):  
        self.url, self.method, self.timeout = url, method, timeout  
  
    def __call__(self, *args, **kwargs):  
        if args and kwargs:  
            raise SyntaxError("Could not accept both *args and **kwargs as JSONRPC parameters.")  
        data = {"jsonrpc": "2.0", "method": self.method, "id": self.id()}  
        if args:  
            data["params"] = args  
        elif kwargs:  
            data["params"] = kwargs  
        if os.name == "nt":  
            res = self.pool.urlopen("POST",  
                                    self.url,  
                                    headers={"Content-Type": "application/json"},  
                                    body=json.dumps(data).encode("utf-8"),  
                                    timeout=self.timeout)  
        jsonresult = json.loads(res.data.decode("utf-8"))
```

当操作系统为Windows时，生成json消息，发送“方法名”

注：server.start/stop代码内容为调用本项目的adb类输入手动启动指令

结构

Structure

- ▶ 此时服务器端(手机)已经启动AutomatorService,直接调用里面的pressKeyCode方法
- ▶ (来自AutomatorServiceImpl.java)
- ▶ 在手机上执行,可以看到, AutomatorService是直接引用了UIAutomator中的方法
- ▶ 至此, pressKeyCode方法在机器上成功执行。

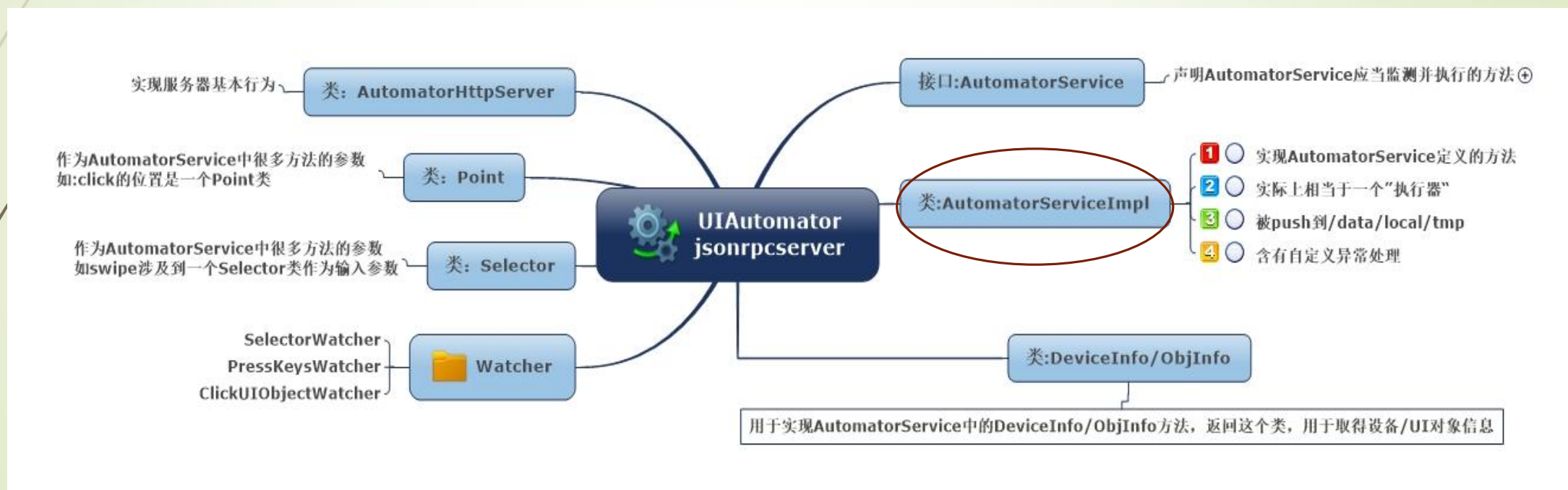
```
/**
 * Simulates a short press using a key code. See KeyEvent.
 *
 * @param keyCode the key code of the event.
 * @return true if successful, else return false
 */
@Override
public boolean pressKeyCode(int keyCode) {
    return UiDevice.getInstance().pressKeyCode(keyCode);
}
```

- ▶ 关于rpcserver是如何启动的,会在后面说明

结构

Structure

JsonRPCServer项目结构，下面对HttpServer部分展开一下



结构

Structure

- ▶ AutomatorService是这里面最重要的类，用于处理jsonrpcmethod(客户端方法)传过来的所有方法，在python wrapper中已经简单说明其原理。
- ▶ HTTPServer部分引用了 开源项目 jsonrpc4j和[Nanohttpd](#)
- ▶ [Nanohttpd](#): 一个轻量级的httpserver (把手机变成httpserver)
- ▶ 特性如下：处理各类http 1.1请求，支持POST/GET请求参数转换，支持HEAD/DELETE请求支持SSL，仅有一个java文件，不缓存任何数据，占用内存很少。
- ▶ [Jsonrpc4j](#): 处理json请求，Stream，作为httpserver/httpclient等
- ▶ 特性如下：可作为流处理服务器(InputStream/OutputStream),HTTP,Portlet,Socket服务器，也可作为客户端，支持自定义错误处理。Json转换和java对象转换部分引用了[Jackson 开源库](#) (高性能json处理)

结构

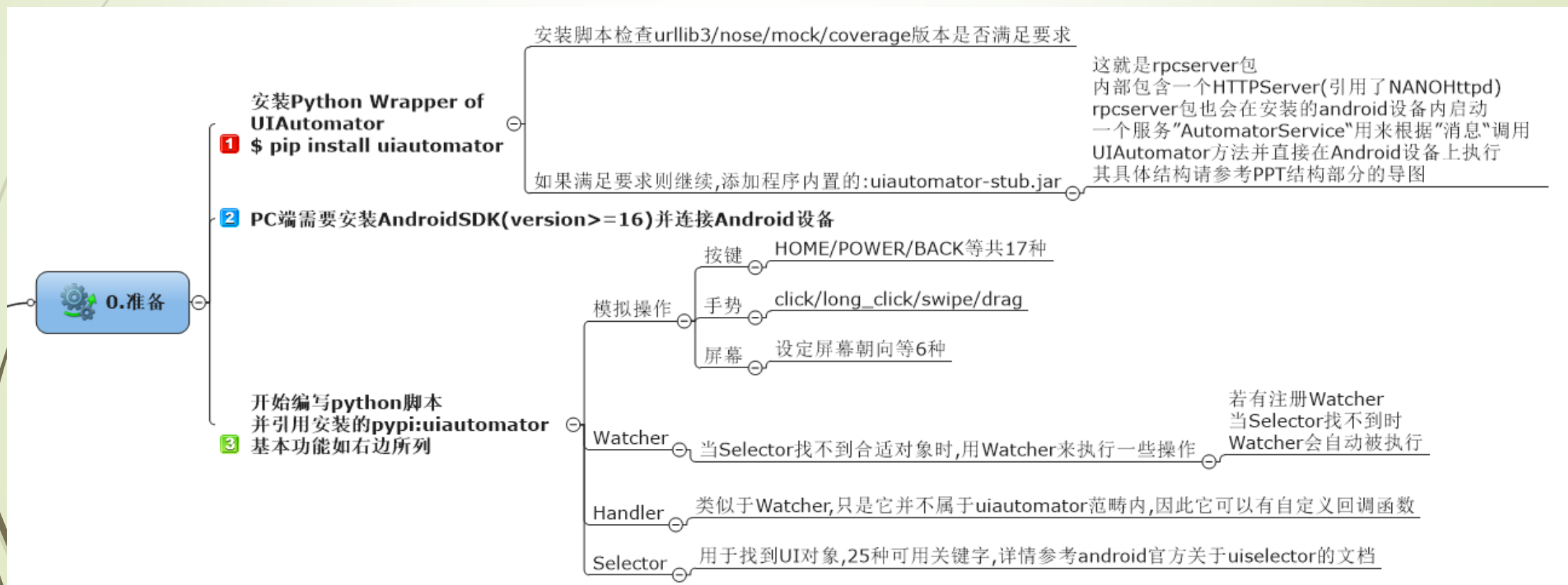
Structure

- **NanoHTTPD流程与原理分析**
- NanoHTTPD是一个抽象类，所以要继承NanoHTTPD，然后调用start()，在start()方法里面主要实现ServerSocket，然后等待客户端的连接，当有客户端连接的时候，myServerSocket.accept()获取socket实例，然后实例化InputStream和OutputStream，然后传递给HTTPSession类，在其execute()方法中解析HTTP的header和body
- **header解析**
- header和body直接使用两个回车换行符来间隔，由于Header的最大长度为8k，所以创建了一个8k的缓冲，来读取header，splitbyte = findHeaderEnd(buf, rlen);直到splitbyte 返回值大于0,说明已经知道分隔点，如果读取的长度大于splitbyte，则把多余的部分回写到流里面，然后创建一个BufferedReader来解析头部数据调用decodeHeader方法
- **body解析**
- 对于body的解析根据请求的方法分为PUT/POST，对于POST方法，使用serve方法来解析body，而serve这个方法也是我们实现NanoHTTPD类所要重写的方法，parseBody来真正解析body，首先创建一个临时文件把流里面的数据写到这个临时文件中，然后实例化ByteBuffer和BufferedReader，如果提交表单的类型是multipart/form-data，查询boundary，而boundary=就是内容数据的分割点，使用decodeMultipartData方法分离参数和内容数据，最后调用saveTmpFile方法来保存文件，为了保存到我们的本地文件中，所以我们还需要TempFileManagerFactory和TempFileManager，TempFileManager用来创建一个本地文件，用来存储真正的上传数据。

流程与逻辑

Process & Logics

- 逻辑流程在上一部分中已经说明，下面用导图详细解释一下完整流程
- 准备阶段



流程与逻辑

Process & Logics

- ▶ 运作阶段(同结构部分的流程描述, 这里使用一张图解释)

1. 运作



流程与逻辑

Process & Logics

➡ 举例

源python指令:`d.scroll(steps=50)` # 默认是垂直向下滚动

wrapper实现:如果是forward或者默认
调用`jsonrpc.scrollForward` 否则调用`jsonrpc.scrollBackward`

注:`jsonrpc`为wrapper中定义的一个方法
`jsonrpc.method`表示给http服务器(手机)
上的AutomatorService发送“scrollForward”的指令
让它执行自己定义的名为“scrollForward”的方法

2. 举例

1 Python Wrapper代码

```
AutomatorService中scrollForward的定义:  
public boolean scrollForward(Selector obj, boolean isVertical, int steps) throws UiObjectNotFoundException {  
    UiScrollable scrollable = new UiScrollable(obj.toUiSelector());  
    if (isVertical)  
        setAsVerticalList(scrollable);  
    else  
        setAsHorizontalList(scrollable);  
    return scrollable.scrollForward(steps);  
}
```

2 AutomatorService代码

注Automator引用了uiautomator,相当于直接执行uiautomator的方法



JsonRPC原理

Principles of JsonRPC

- ▶ Json-RPC: 以json为消息格式的远程调用服务, 这种远程过程调用可以使用http作为传输协议, 传输的内容是json消息体。(本项目使用的即为http协议)
- ▶ RPC采用客户端/服务器模式。请求程序就是一个客户端(PC), 而服务提供程序就是一个服务器(手机)。
- ▶ 下面介绍一下消息是如何在客户/服务器之间传输的

JsonRPC原理

Principles of JsonRPC

客户端发送有进程参数的调用信息到服务器

等待
应答

服务器保持睡眠直到有消息到达，当消息到达，
获取参数，计算结果处理并发送答复消息

服务器
继续等
待

客户端接收到答复消息，获得结果，继续执行

功能改进

Improvements

- ▶ 本项目已经开源并欢迎patch



Xiaocong He

发送至 我 ▾

是的，都是MIT license。你可以在readme中找到对应java工程的开源地址。欢迎有兴趣的同学fork并补充自己的patch。



- ▶ 尝试添加功能 “检查屏幕是否开启” 到本项目中(之前只能进行on/off操作)
- ▶ 现在可以使用 `d.screen.ison()`来判断屏幕的开关状态
- ▶ 代码基于screen方法进行补充，增加了“ison”调用逻辑。

xiaocong


136 commits / 6,828 ++ / 3,149 --

#1



fancylear

1 commit / 10



功能改进

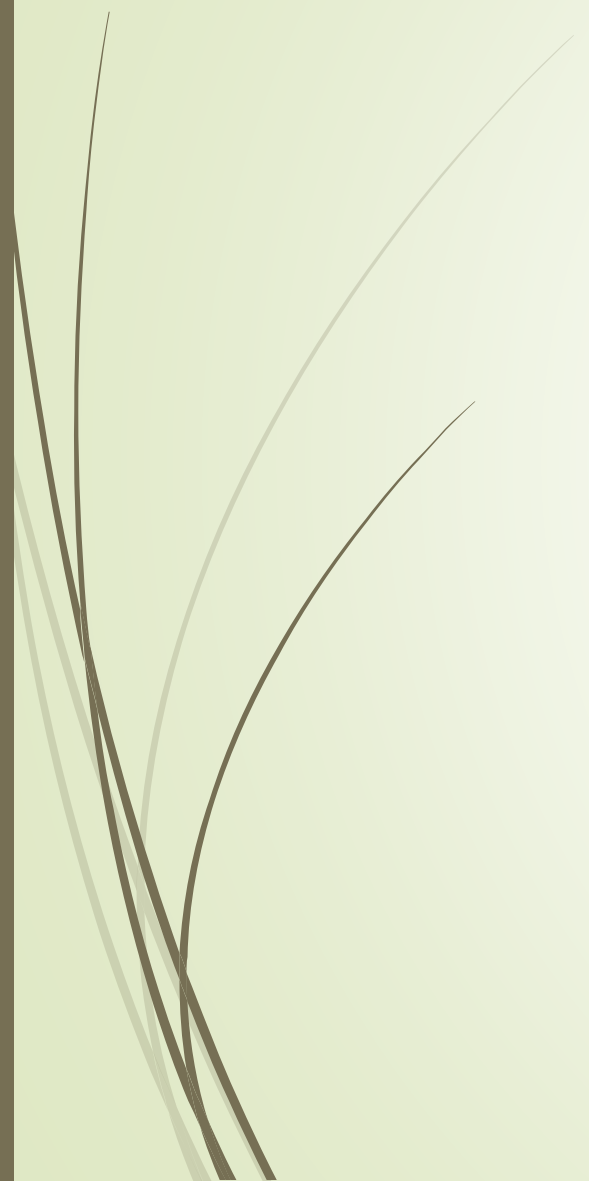
Improvements

- 进一步改进
- 1.使用其它语言？可以尝试制作Ruby Wrapper
- 2.GUI化脚本并制作GUI安装器？进一步降低上手和使用难度
- 3.改进并扩展UIAutomator本身？本项目没有对UIAutomator进行任何扩展，可以尝试对UIAutomator扩展，丰富其功能和适用范围。

资源/参考

Resources/References

- 参考资源
- 1. [UIAutomator](#) by xiaocong
- 2. [jsonrpcserver](#) for uiautomator by xiaocong
- 3. [NanoHTTPD](#) by NanoHTTPD.com
- 4. [jsonrpc4j](#) by briandilley
- 5. [Jackson libs](#)
- 我为本报告设立了一个博客，所有内容都放到了网上，包括HTML5功能介绍幻灯片
- <http://fancylear.github.io>
- 注：HTML5功能介绍Slide中大量引用了原作者的解释。



Fin
谢谢