1. 初始化
2. 删除之前Main类中的测试代码
3. DataTool类增加私有静态方法readConfigFromFile

//读取配置文件的数据

private static JObject readConfigFromFile()

{

if (!File.Exists(configPath))

{

//文件不存在，则返回null

return null;

}

try

{

//读取对应文件的数据

StreamReader file = new StreamReader(configPath);

String content = file.ReadToEnd();

file.Close();

//解析数据

JObject data = JObject.Parse(content);

//返回数据

return data;

}

catch (Exception e)

{

FiddlerApplication.Log.LogString("FiddlerExample出现错误(readConfigFromFile函数)：" + e.ToString());

return null;

}

}

1. DataTool增加公有静态方法initConfigData

//初始化配置数据

public static ArrayList initConfigData()

{

ArrayList result = new ArrayList();

//获取配置数据

JObject config = readConfigFromFile();

if (config != null)

{

JArray rules = config["host"] as JArray;

for (int i = 0, len = rules.Count; i < len; i++)

{

//获取规则

JObject rule = rules[i] as JObject;

//生成对应数据对象

HostModel item = new HostModel(i, (bool)rule["enable"], rule["ip"].ToString(), rule["port"].ToString(), rule["url"].ToString());

//添加到结果中

result.Add(item);

}

}

//返回数据

return result;

}

1. 修改Main类OnLoad函数初始化mainData逻辑

//初始化配置数据

mainData = DataTool.initConfigData();

1. Container类构造函数增加初始化逻辑

//初始化Rule面板

initRuleToUI();

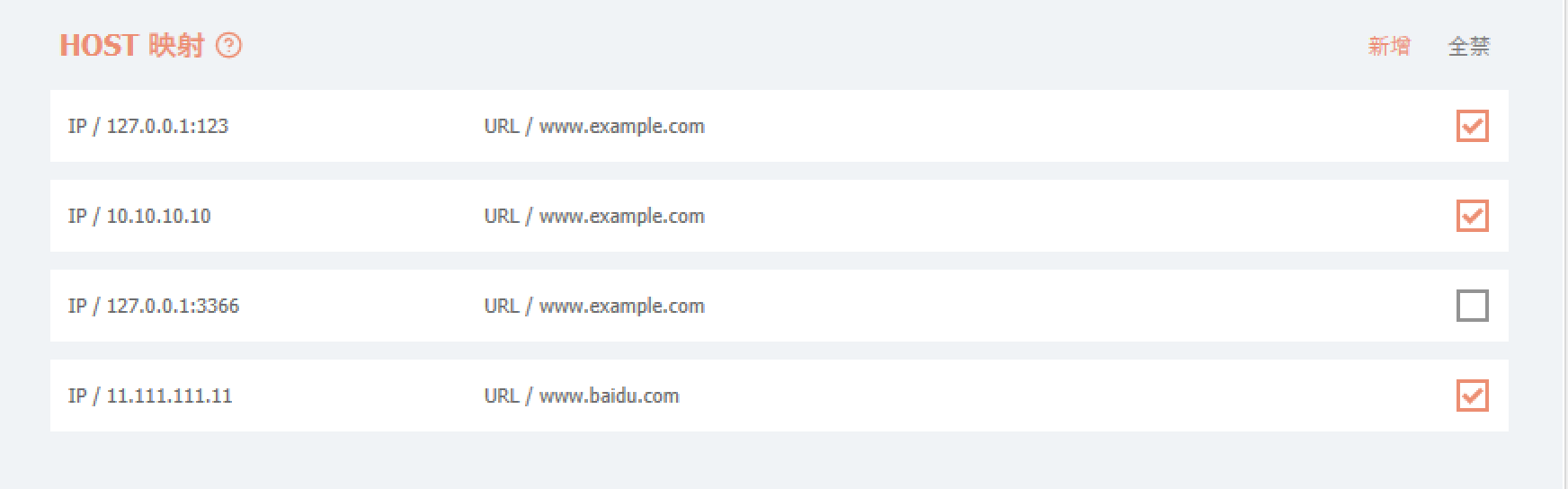
1. 修改新增规则的Bug

Main类的addHostRule方法增加写入文件逻辑

//重新写入文件

DataTool.writeConfigToFile();

1. 打包预览



1. Host映射
2. 在Tools文件夹中新增FiddlerTool类，同时新增私有静态方法getValidRulesByType、getPathFromSession

//根据类型获取有效的规则

private static ArrayList getValidRulesByType()

{

ArrayList rules = new ArrayList();

//遍历获取有效的数据

for (int i = 0, len = Main.mainData.Count; i < len; i++)

{

HostModel item = Main.mainData[i] as HostModel;

//为false则直接跳过

if (!item.Enable)

{

continue;

}

rules.Add(item);

}

return rules;

}

//从Session中获取path

private static string getPathFromSession(string fullUrl)

{

string path = fullUrl;

if (path.IndexOf("?") > 0)

{

path = path.Substring(0, path.IndexOf("?"));

}

return path;

}

1. FiddlerTool类中增加私有静态方法handleHostMapping

//HOST映射

private static void handleHostMapping(Session session)

{

ArrayList rules = getValidRulesByType();

//如果没有有效的host配置，直接返回

if (rules.Count == 0)

{

return;

}

//获取Url的Path

string path = getPathFromSession(session.fullUrl);

//遍历配置去修改映射值

for (int i = 0; i < rules.Count; i++)

{

//获取对应的各种参数

string url = (rules[i] as HostModel).Url.ToString();

string ip = (rules[i] as HostModel).IP.ToString();

string port = (rules[i] as HostModel).Port.ToString();

//新建正则表达式来检测

Regex urlRegex = new Regex(url);

//判断当前session的path是否在配置中

if (path.IndexOf(url) >= 0 || urlRegex.IsMatch(path))

{

//修改背景颜色、字体颜色

session["ui-color"] = "#FFFFFF";

session["ui-backcolor"] = "#9966CC";

if (port.Length > 0)

{

session["x-overrideHost"] = ip + ":" + port;

}

else

{

//映射到对应的ip和端口(这里必须写上端口号，不然https下会有问题)

session["x-overrideHost"] = ip + ":" + session.port;

}

session.bypassGateway = true;

break;

}

}

}

1. FiddlerTool类中增加公有静态方法handleRequest

//监听请求前的事件

public static void handleRequest(Session session)

{

//HOST映射

handleHostMapping(session);

}

1. 在Main类的OnLoad方法的末尾增加监听相关代码

//监听请求响应之前

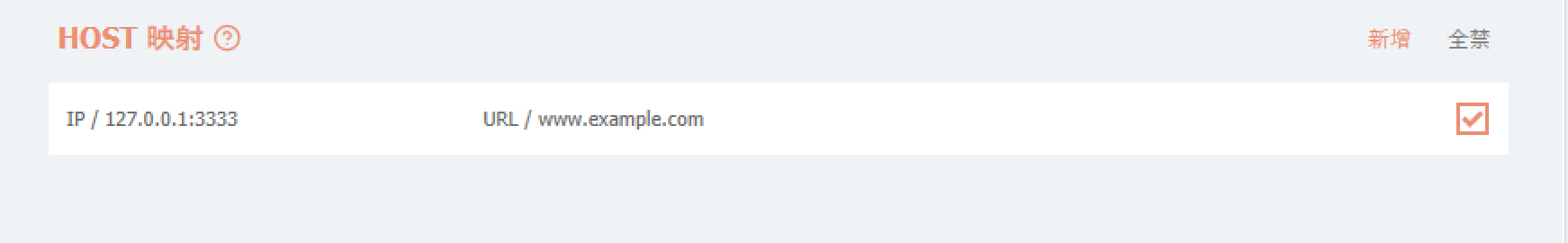
FiddlerApplication.BeforeRequest += delegate (Session session)

{

FiddlerTool.handleRequest(session);

};

1. 打包插件，然后起一个简单的Node服务，端口设置为3333，并重新设置插件规则



1. 浏览器访问www.example.com

