

# CommonJS时代的 模块机制和编译工具

Dexter.Yy @ 土豆网

标题看上去好内涵喔！.....



标题看上去好内涵喔！.....

CommonJS是神马？



标题看上去好内涵喔！.....

CommonJS是神马？

时代！？



标题看上去好内涵喔！.....

CommonJS是神马？

时代！？

为神马要讨论  
模块和工具？





土豆前端team，和谐的一塌糊涂啊.....  
其实是在上班时间打游戏.....



先看看  
前端日常工作中  
存在的问题

# 现状

- 前端开发越来越成熟，大多数团队和项目里都使用了流行的库或框架。
- 自身代码库逐步积累，越来越重视代码复用和抽象封装——比如常用模式、工具、UI组件、交互行为。

## Inside the TUI JavaScript Library

又名《TUI打包权威指南》



# Inside the TUI JavaScript Library

又名《TUI打包权威指南》



# 现状

- 前端开发越来越成熟，大多数团队和项目里都使用了流行的库或框架。
- 自身代码库逐步积累，越来越重视代码复用和抽象封装——比如常用模式、工具、UI组件、交互行为。

# 问题

- `jQuery`、`YUI`、`MooTools`、自己开发的框架——不同的代码组织结构、不同的接口风格。
- 积累下的代码有很强的依赖性，被库/框架束缚，各种山寨风格，只能在特定环境里复用。
- 难于迁移，难于管理，难于分享，难于传播。
- 各自闭门造车轮。

# 现状

- 工作流程更专业更严谨——版本控制、调试、静态校验、单元测试、自动远程测试、开发环境/测试环境/模拟线上环境、静态文件管理、发布系统.....

```
href="http://www.shjubao.cn">上海市举报中心</a> <a target="_blank"
title="网络违法犯罪举报网站" href="http://www.cyberpolice.cn">网络违法犯罪举报网站</a></li>
<li><a title="“扫黄打非”办公室举报中心: 12390" href="javascript:;">“扫黄打非”办公室举报中心: 12390<
TUI.module.join("http://js.tudouui.com/js/fn/xnconnect_26.js");
var focusKits = { swf: "http://js.tudouui.com/bin/contentplayer/focus_player_16.swf?id=66316509" };
var adKits = { seed: "http://js.tudouui.com/bin/seedplayer/SeedPlayerLayer_9.swf", board: "http://js.tud
$(function(){window.temptest_dom=new Date()});
/* @modified: $Author: jyan $ * @version: $Rev: 12614 $ */
document.domain="tudou.com";TUI.domain.join({minichannel:"www.tudou.com/my/channel"});ClickS
function cacheAdData(json){window.adExDataCache = json}
cacheAdData({"mulSel":[],"commonAdvReturnEntityList":[{"throwId":10346,"textColor":"","ownerName":""}
</script></body></html>
```

# 现状

- 工作流程更专业更严谨——版本控制、调试、静态校验、单元测试、自动远程测试、开发环境/测试环境/模拟线上环境、静态文件管理、发布系统.....

# 问题

- 环节越来越多，做事越来越不“直接”，开发和测试变得麻烦。
- 过程的改进不应该增加开发的成本。
- 好的过程应该能隐藏复杂性，减少出错的机会。
- 对代码管理的要求更高，代码如何方便的分离？如何方便的调用？代码单元和文件的粒度如何掌控？

# 现状

- 前端性能越来越受重视。优化准则得到广泛应用。  
——减少请求数，利用缓存，避免阻塞。
- 指标：首屏时间，TTI（用户可交互前等待时间）  
——需要延后、按需加载等手段。

# 现状

- 前端性能越来越受重视。优化准则得到广泛应用。  
——减少请求数，利用缓存，避免阻塞。
- 指标：首屏时间，TTI（用户可交互前等待时间）  
——需要延后、按需加载等手段。

# 问题

- 优化手段需要封装，需要工具的支持，否则容易成为滑向hack技巧和魔幻代码的深渊.....
- 跟上一个问题里需要加强的代码管理出现矛盾！能方便开发、调试、代码复用和维护的措施 vs 能提高性能的措施——经常截然相反。
- 要权衡.....要河蟹.....

# 以上三个方面都涉及同一个问题

- 越来越重视代码复用和抽象封装。
- 工作流程越来越专业越来越严谨。
- 前端性能和优化手段越来越受重视。



代码的组织结构、管理、复用、部署

解决问题的途径! .....之一.....

模块机制  
(module)

+

编译工具  
(autobuild / preprocessor / compiler)

# 不过标题其实是分成三段的

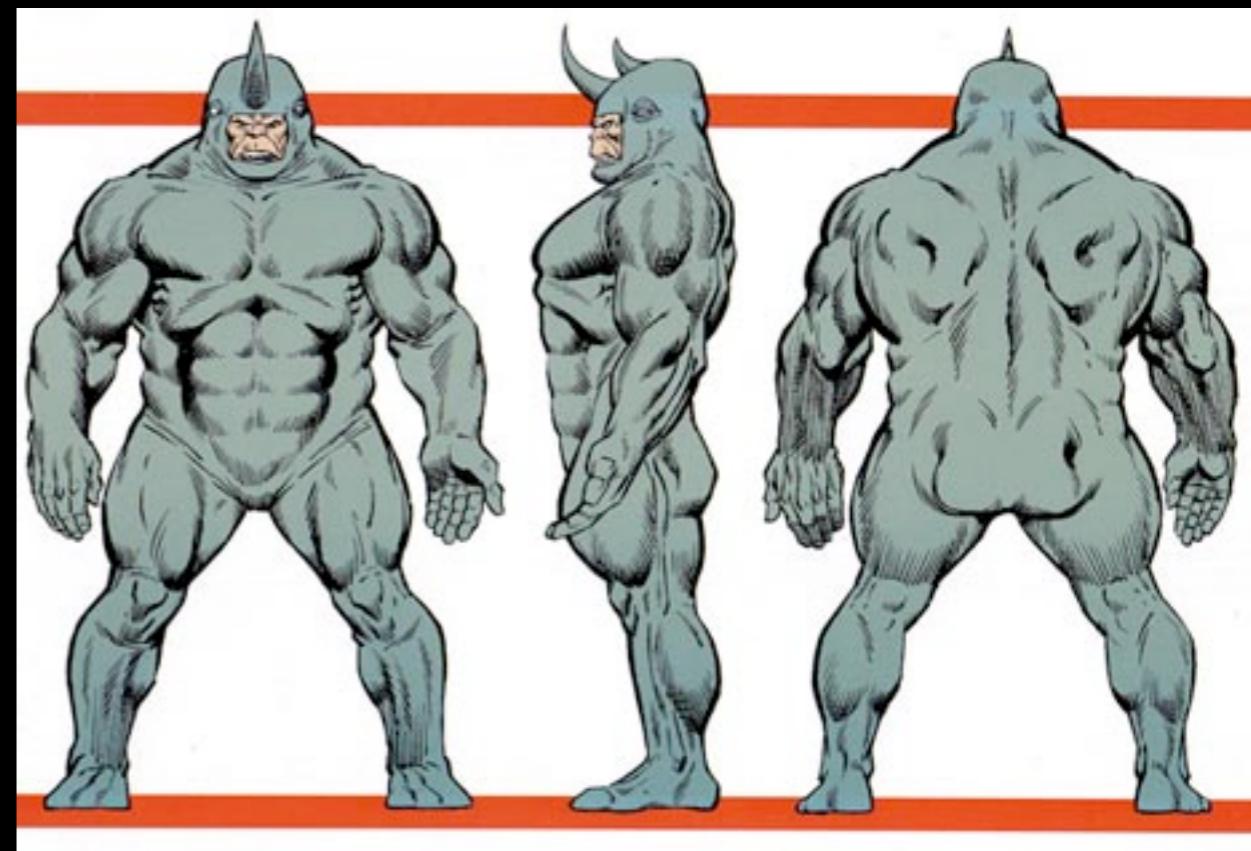
CommonJS时代的模块机制和编译工具

后两个是手段，第一个是背景和环境

先讨论背景罢

# 什么是“通用JS” (CommonJS)

这位只是蜘蛛侠里的犀牛人.....



# 狭义的 CommonJS

- 一套API
- 一种规格、协议、或约定
- 一个事实标准 (*de facto*) <> *de jure*
- 面向服务器、桌面、命令行和浏览器里的JS开发
- 目标：建设JS的生态系统 (ecosystem)

# JS的生态系统有什么问题？

- JS缺少服务器端开发需要的API（不是重点）
- 很多流行的、开源的现代编程语言都拥有由一个标准库、一个优秀的第三方代码库和依赖管理机制组成的技术生态圈。
- Python: PyPI (egg) + setuptools
- Ruby: Gem
- Perl: CPAN

# 在Python里 使用第三方的 module

```
9
10 import sys, io, os, re
11 import yaml
12 from mako.template import Template
13 import subprocess as sub
14 from optparse import OptionParser
15 import pickle
16
17
18 class Pacbot():
19     """auto proxy configuration toolkit
20     """
21
```

```
$ sudo easy_install mako
Searching for mako
Best match: Mako 0.3.4
Processing Mako-0.3.4-py2.6.egg
Mako 0.3.4 is already the active version in easy-install.pth
Installing mako-render script to /usr/local/bin

Using /Library/Python/2.6/site-packages/Mako-0.3.4-py2.6.egg
Processing dependencies for mako
Finished processing dependencies for mako
```

# 创始人的黑历史

- Kevin Dangoor在09年初建立CommonJS讨论组
- 《What Server Side JavaScript needs》
- 09年加入Mozilla的web开发者工具团队。
- 08年10月Mozilla雇佣了Ajaxian的创始人Dion Almaer和Ben Galbraith，建立开发者工具实验室，开发过Skywriter/Bespin，Kevin Dangoor现在是lead
- 曾在SitePen工作，TurboGears和Paver项目的创始人。

# CommonJS提供什么

- 提供约定用JS开发桌面/服务器程序需要的API，比如操作系统接口、文件系统、二进制数据、网络通信、等等。
- 更重要、更基础的约定：module机制
- 跟JS引擎无关，跟操作系统无关，使程序不作修改就可以在任何CommonJS兼容的平台上运行——包括第三方module。

# 广义的“通用JS”

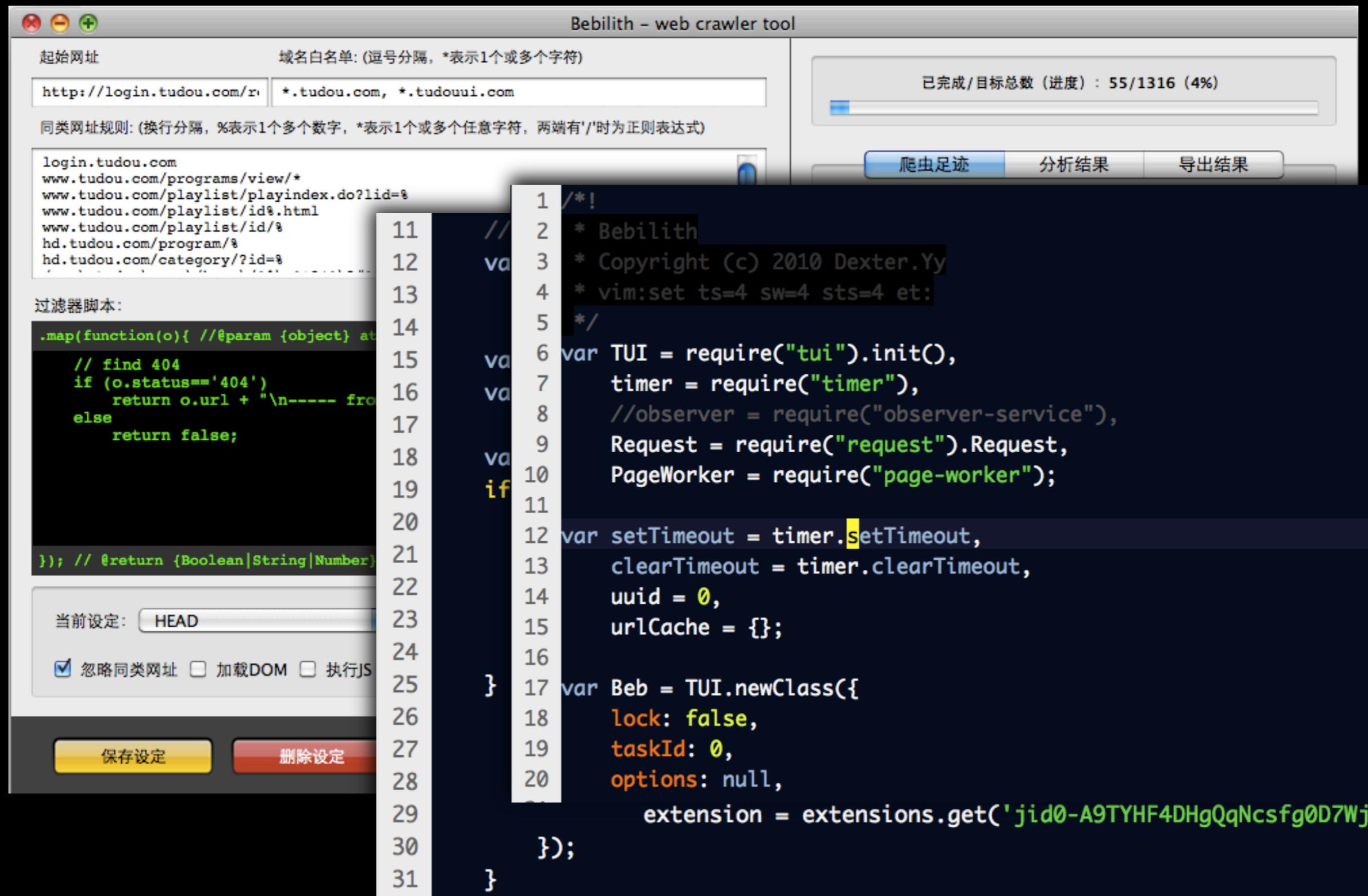
- 身为“浏览器之王”的JavaScript，在更多领域、更多环境里流行起来，其中很多仍然围绕前端UI。
- 现有JavaScript代码库里，有很多功能可以写的更“一般化”，跟特殊环境解耦，成为更通用的module。

# JS“通用”的例子

# 浏览器扩展里的JS

- Firefox的新Add-on SDK: Jetpack
- 已经发布1.0beta1
- 以前的Firefox扩展开发需要熟悉XUL、XPCOM等技术，门槛较高，包含大量JS代码，不便于维护和测试。
- 现在提供CommonJS兼容的module机制，更易用的JS API、更方便的开发测试工具。

# DEMO：爬虫机器人Bebilith



# DEMO: Safari扩展

- 跟Chrome扩展非常类似。
- 缺乏API和module机制，开发复杂应用很吃力。
- Chrome上能诞生Firebug这样给力的重量级扩展么？
- 这个“迷你土豆”用了hack手段才能实现，曾经担心Apple审核不通过.....



# 桌面/移动GUI里的JS

- Nokia的QT Quick
- 在MeeGo和Symbian里开发移动App
- 虽然是QT，但是不需要会讨厌的C++
- QML：由JavaScript改造的声明式语言。可嵌入JavaScript代码



# 桌面/移动GUI里的JS

- WebOS 2.0
- 新版内建了nodejs运行环境，可以开发后台服务，支持多任务。



hp  
webOS 2.0

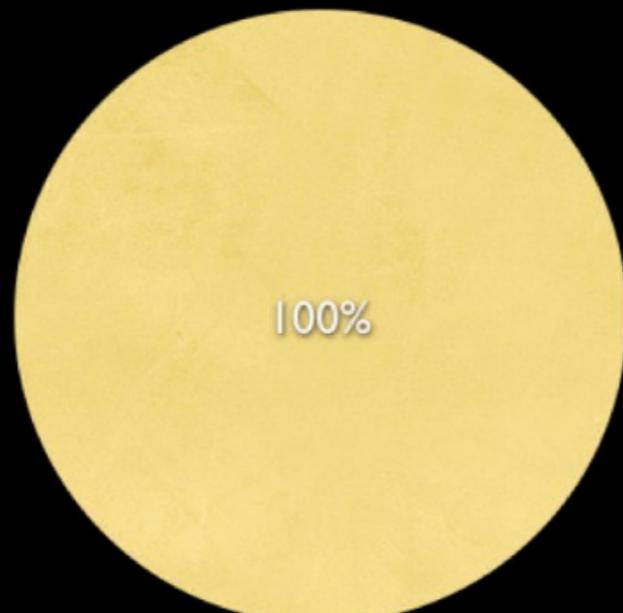
# 桌面/移动GUI里的JS

- 豆瓣的[OneRing](#)
- “跨平台的桌面应用开发库，使用HTML5+CSS3制作用户界面，用Javascript编写交互逻辑，用写web后端的技术编写后台逻辑”。
- Issue 25：建议兼容CommonJS API.....

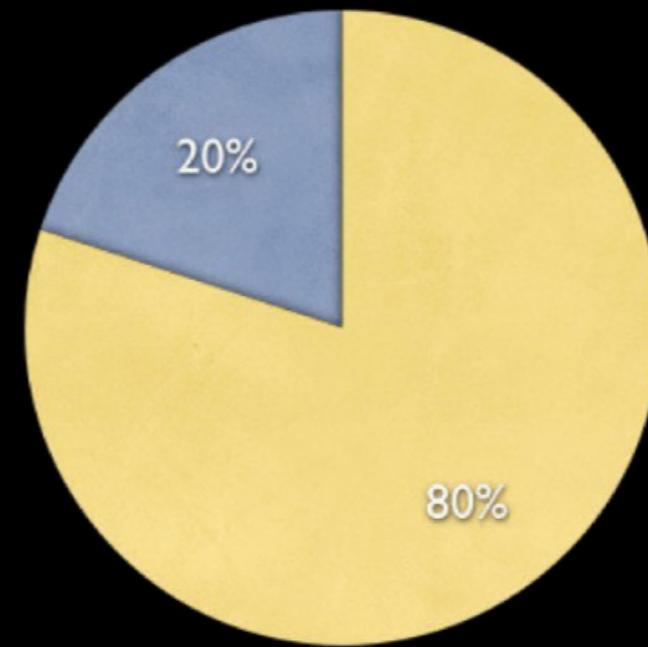
# JSConf.eu上的饼图

● Client  
● Embedded  
● Server

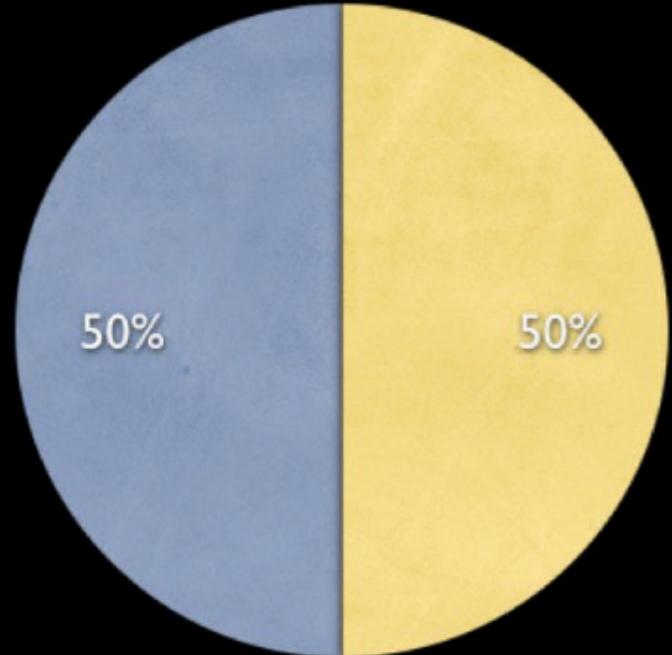
JsConf 2009 DC



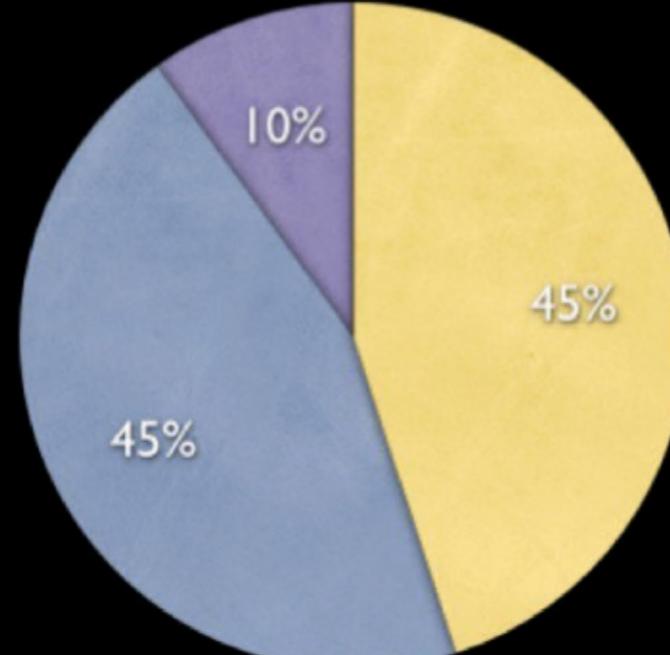
JsConf 2009 Berlin



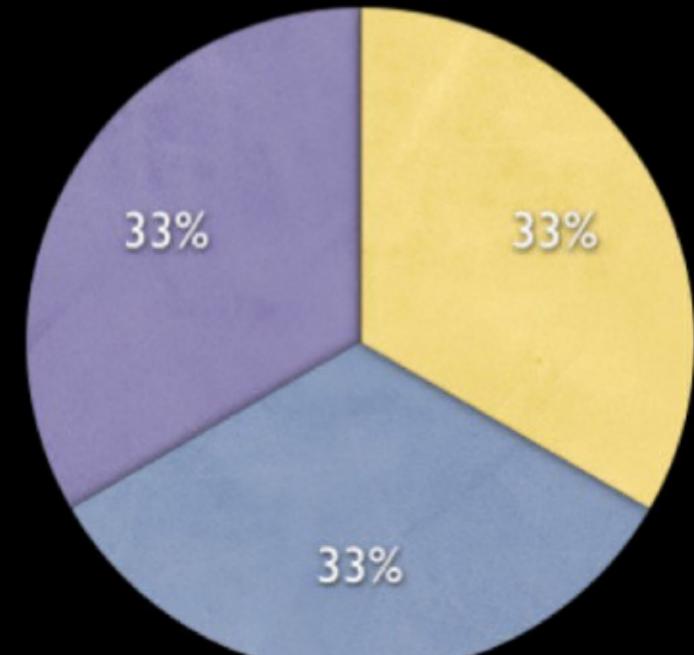
JsConf 2010 DC



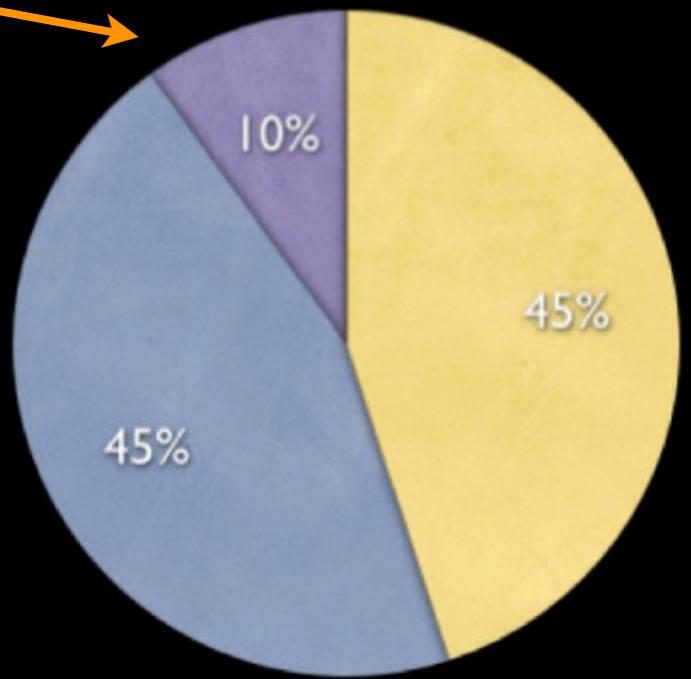
JsConf 2010 Berlin



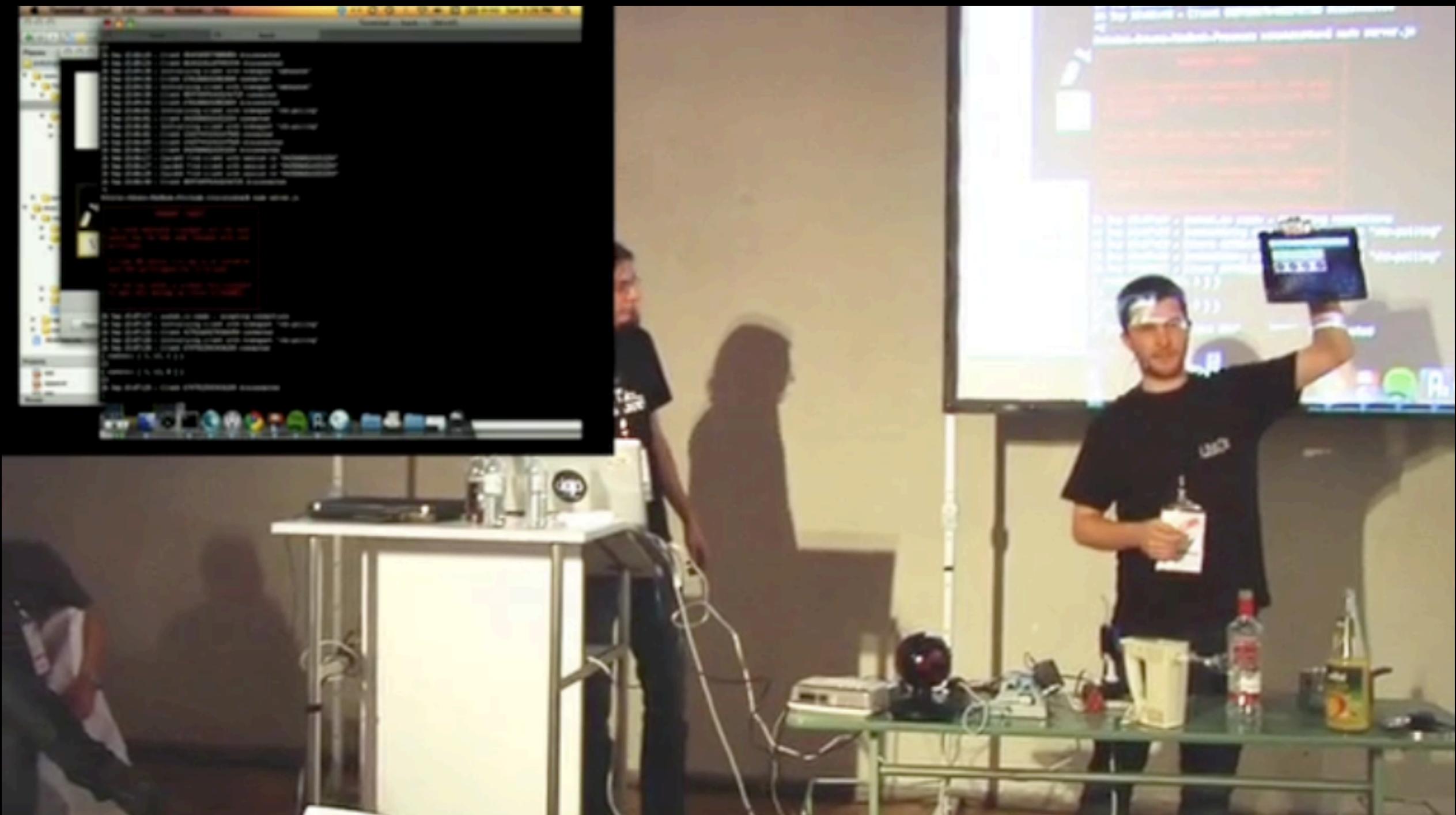
JsConf 2011 DC



10%的embedded →  
是指演讲者自己



- 用js做嵌入式开发，控制硬件设备
- Arduino是一个非常流行的开源硬件平台
- 淘宝上有卖
- arduinojs: Arduino will run JavaScript one day
- 视频演示非常酷非常Geek



# 终于轮到服务器端的JS

- Ryan Dahl在09年11月的JSConf柏林上介绍了NodeJS
- 跟Aptana Jaxer等服务器端JS相比，NodeJS的出发点不同，它强调的是基于事件的异步编程，卖点是并发和实时。
- 不提供server，而是提供基础设施。
- 利用v8引擎，被拿来跟erlang比性能。
- CommonJS兼容实现

# NodeJS的module数量

- 在github上收录了550+
- web框架，各类服务器，数据库封装，模板引擎，网络中间件，测试框架，图形，文本解析，构建工具.....

# 回头来看浏览器上的JS

- 通用的module数量太少
- 社区分散、独立
- 缺乏有利于协作和分享的基础设施

# 基础设施之一： Module机制 和依赖管理



# 其他流行语言的module

main.py

```
import moduleB  
  
print moduleB.say(3) # 5
```

moduleA.py

```
a = 1  
  
class cat():  
    def say(self, n):  
        return n
```

moduleB.py

```
from moduleA import cat  
  
b = 2  
a = 0  
  
def say(n):  
    print a + b + n  
    return cat().say(a + b + n)
```



# 其他流行语言的module

- 一个文件就是一个module
- 用声明语句导入想要的东西
- 不会意外获得不想要的东西
- 导入前模块不会解析执行
- 不同的文件有各自独立的命名空间
- 模块文件内顶层的命名空间转变成模块对象
- 模块文件自带依赖信息

浏览器里的JS不符合任意一条

# 其他流行语言的module

浏览器里的JS不符合任意一条

- 很难把JS的文件当作**module**单元来使用
- 没有相关的声明语句，只能用异步加载文件的方式模拟导入的效果
- 可能隐式的导入很多全局变量，覆盖或修改当前的内容
- 所有文件一开始都会解析执行
- JS存在全局作用域，不同文件会共享同一个全局命名空间
  - 导入之后不管你想不想要，那个文件里的东西会全部注入到当前的命名空间里
- JS文件自身也不包含依赖信息，如果有打合并之类的操作，我们可能会把依赖信息写在另外的配置文件里

# JS代码的传统组织方式

## ① 很多很多script tag在HTML里排队.....

```
46 <script type="text/javascript" src="http://s.xnimq.cn/a16398/n/core/js/minifeed-all.js"></script><s  
47 src="http://s.xnimq.cn/a9860/xnapp/vip/jspro/profile/flashWithAS3.js"></script>  
48 <script type="text/javascript" src="http://a.xnimq.cn/a2550/jspro/swfupload.v2.2.0.1/swfupload.js">  
49 <script type="text/javascript" src="http://s.xnimq.cn/a6866/js/swfobject.js"></script>  
50 <script type="text/javascript" src="http://s.xnimq.cn/a16393/jspro/xn.app.status.js"></script>  
51 <script type="text/javascript" src="http://s.xnimq.cn/a14957/jspro/xn.app.gossip.js"></script>  
52 <script type="text/javascript" src="http://s.xnimq.cn/a16387/jspro/xn.page.profile.js"></script>  
53 <script type="text/javascript" src="http://s.xnimq.cn/a11680/jspro/xn.ui.minieditor.js"></script>  
54 <script type="text/javascript" src="http://s.xnimq.cn/a16052/jspro/xn.app.ilike.js"></script>  
55 <script type="text/javascript" src="http://s.xnimq.cn/a16328/jspro/page/jquery.js"></script>  
56 <script type="text/javascript" src="http://s.xnimq.cn/a12291/jspro/popPage.js"></script>  
57 <script type="text/javascript">  
58   jQuery(function(){  
59     new popPage({mark: '.pop-module-mark', reqUrl:'http://page.renren.com/community/ajaxQuery'});  
});
```

## ② 很多很多script在combo后的文件里排队.....

```
<script>YUI_config = { app: {tabview: '#demo' } };</script>  
<script src="http://company.cdn.com/combo?  
3.2.0/build/yui/yui-min.js&  
3.2.0/build/loader/loader-min.js&  
3.2.0/build/tabview/tabview-min.js& /* plus other dependencies */  
1.0/build/app.js&  
1.0/build/init.js"></script>
```

# 浏览器端的特殊文化

- 代码的依赖关系表现为加载或放置的顺序
- 代码文件不是抽象的连接在一起，在实际应用中需要直接合并或分割代码文件
- 需要自己实现异步加载和延后加载

# Script loader

特殊文化下形成的解决方案

不是Module loader

# 各种script loader库



# 各种script loader库

```
// the most simple case. load and execute single script without blocking  
head.js("/path/to/file.js");
```

HeadJS

```
// load files in parallel but execute them in sequence  
head.js("file1.js", "file2.js", ... "file10.js");
```



并行下载，却按顺序执行?  
解决之前提到的缺陷?

```
// execute function after all scripts have been loaded  
head.js("file1.js", "file2.js", function() {  
...});
```

串行加载

```
// files are loaded in parallel and executed in order they arrive  
head.js("file1.js").js("file1.js").js("file3.js");
```

其实内部实现只是先并行的“预加载”，  
然后重新串行的加载（利用缓存）

```
// call a function immediately after jQuery Tools is loaded  
head.ready("tools", function() {  
...});
```

本质上还是没啥区别.....

```
// load scripts by assigning a label for them  
head.js(  
  {tools: "http://cdn.jquerytools.org/1.2.5/tiny/jquery.tools.min"},  
  {heavy: "http://a.heavy.library/we/dont/want/to/wait/for.js"},  
  // label is optional  
  "http://can.be.mixed/with/unlabeled/files.js"  
);
```

标签命名

# 各种script loader库

这个旧版API虽然叫TUI.module.use，实际上仍然只能算script loader

其实土豆也有.....

注册脚本文件，解析域名和版本号

只在第一次使用时发请求。有依赖的代码自动延迟执行

```
<li><a href="http://www.soso.com/a?" target="_blank" kname="w">网页</a></li>
TUI.module.join("http://js.tudouui.com/js/fn/saleloader_54.js");
TUI.module.join("http://js.tudouui.com/js/fn/iosplayer_6.js");
TUI.module.join("http://js.tudouui.com/js/delate_4.js");
TUI.module.join("http://js.tudouui.com/js/fn/xnconnect_26.js");
</script><script src="http://js.tudouui.com/js/page/play_program_158.js"></script>
```

```
TUI.module.use("/fn/saleloader", function(){
    // 限定关键字的广告
    adExtension.addTempBusiness(1, function(opt, it, box){
        return it.adTempOpt.split("/").indexOf(window.kw_res) != -1;
    });
    // @TODO 临时广告 7-28至9-30
    adExtension.addTempBusiness(2, function(opt, it, box){
        if ((new Date()).getMonth() > 8)
            return false;
    });
});
```

# 特殊文化下形成的 另一种解决方案

## Module Pattern

```
var MODULE = (function () {
    var my = {},
        privateVariable = 1;

    function privateMethod() {
        // ...
    }

    my.moduleProperty = 1;
    my.moduleMethod = function () {
        // ...
    };

    return my;
}());
```

# Dave Herman: JavaScript needs modules!

Q: 说的是好！但是.....妈的，他到底是谁？

A: Dave Herman是Mozilla实验室的研究员，ECMA TC39成员，**ECMAScript5 Simple Modules**提案的发起人。

# ES5 Simple Modules

- 提案：[http://wiki.ecmascript.org/doku.php?id=strawman:simple\\_modules](http://wiki.ecmascript.org/doku.php?id=strawman:simple_modules)
- 幻灯演示：<http://www.ccs.neu.edu/home/dherman/talks/javascript-needs-modules.pdf>
- 实现：Narcissus

# ES5 Simple Modules

```
export var a = 2;
export function cat(){}
cat.prototype.say = function(n){
    return n;
};
```

moduleA.js

大致上是这个样子：

```
<script type="text/es-harmony">

    module moduleB {
        module A = load "moduleA.js";
        var b = 2;
        export function say(){
            return A.cat().say(A.a + b + n);
        }
    }

    import moduleB

    moduleB.say(3) // 6
```

# ES5 Simple Modules

- 跟CommonJS的关系不大，出发点不同：[Simple Modules and Current Modules](#)

## Simple Modules and Current Modules

Brendan Eich [brendan at mozilla.com](mailto:brendan@mozilla.com)

Thu Nov 4 10:42:15 PDT 2010

- Previous message: [Simple Modules and Current Modules](#)
- Next message: [Simple Modules and Current Modules](#)
- Messages sorted by: [\[ date \]](#) [\[ thread \]](#) [\[ subject \]](#) [\[ author \]](#)

- 此贴的参与者：Kris Zyp（Sitepen工程师，Dojo贡献者），Alex Russell（Dojo创始人），Brendan Eich（JS创始人）
- Simple Modules目前还只是“稻草人提案”（strawman proposal）

# ES5 Simple Modules

我们可以先借鉴他们的目标：

- 消除全局对象
- 兼容已存在的特性和旧代码
- 不损害web上的用户体验和性能
- 平滑的重构旧代码
- 标准的代码分享机制

现在可以仔细看看  
CommonJS的module规范了

# 爬虫机器人Bebilith的代码

# 两个工具 require 和 exports

```
1 /*!
2 * Bebilith
3 * Copyright (c) 2010 Dexter.Yy
4 * vim: set ts=4 sw=4 sts=4 et:
5 */
6 var TUI = require("tui").init(),
7     timer = require("timer"),
8     //observer = require("observer-service"),
9     Request = require("request").Request,
10    PageWorker = require("page-worker");
11
12 var setTimeout = timer.setTimeout,
13     clearTimeout = timer.clearTimeout,
14     uuid = 0,                      452      self.notify("capture", "complete", url);
15     urlCache = {};
16
17 var Beb = TUI.newClass({          453      },
18   lock: false,                   454      output: function(result){
19   taskId: 0,                     455         // hook
20   options: null,                456         }
21                                         457     });
22                                         458   });
23                                         459 });
24                                         460
25                                         461 exports.Bebilith = Beb;
26                                         462
```

# 两个工具： require 和 exports

# CommonJS Module

```
var say = require("moduleB").say;  
  
say(3) // 6
```

main.js

moduleB.js

```
var A = require("moduleA");  
  
var a = 1;  
var b = 2;  
  
function say(){  
    return A.cat().say(a + b + n);  
}  
  
exports.say = say;
```

moduleA.js

```
var a = 2;  
function cat(){  
}  
cat.prototype.say = function(n){  
    return n;  
};  
  
exports.a = a;  
exports.cat = cat;
```

能在浏览器端实现  
CommonJS的module吗？



# 能在浏览器端实现 CommonJS的module吗？

怨念～

怨念～

怨念～

怨念～

怨念～

怨念～

怨念～

如果是几个月前，  
我接下来就会开始介绍  
自己实现的 **oz.js**  
兼容CommonJS，  
复用彼此的代码，  
API风格也简洁一致。  
.....可惜



# 能在浏览器端实现 CommonJS的module吗？

最近我发现  
.....杯具性的撞车了！  
有人实现了一模一样的API，  
还提交到CommonJS  
成了新提案。  
于是.....  
今天只好先介绍他的实现了



# RequireJS

- 作者James Burke， dojo开发者，在Mozilla工作。——跟CommonJS社区联系紧密
- 我曾经觉得它只是普通的script loader， 提供按需加载功能。
- 那个时候它还只有**require**方法
- 新增加的**define**方法让它产生质变， 成为  
CommonJS的“AMD”规范 (**Asynchronous Modules Definition**)

# Asynchronous Modules

```
define("moduleB", ["moduleA"], function(A){  
    var a = 1;  
    var b = 2;  
    function say(){  
        return A.cat().say(a + b + n);  
    }  
  
    return {  
        say: say  
    };  
});
```

```
require(["require", "moduleB"], function(require, B){  
    var say = require("moduleB").say;  
    say(3) // 6  
    B.say(4) // 7  
});
```

```
define(["exports"], function(exports){  
    var a = 2;  
    function cat(){  
    }  
    cat.prototype.say = function(n){  
        return n;  
    };  
  
    exports.a = a;  
    exports.cat = cat;  
});
```

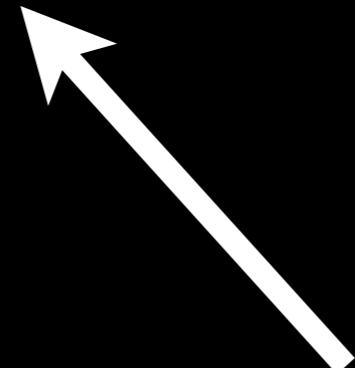
moduleA.js

main.js

# Asynchronous Modules

```
require(["http://search.twitter.com/trends/current.json?callback=define"], function (json) {  
    console.log(json);  
});
```

可以用来请求  
JSONP接口



```
define({  
    "trends":{  
        "2010-12-16 04:20:00":[{  
            "name": "#femalesneedto",  
            "events" :null,  
            "query": "#femalesneedto",  
            "promoted_content": null  
        }]  
    },  
    "as_of":1292472358  
})
```

# Asynchronous Modules

- 不污染，也不使用全局命名空间
- 自身包含依赖关系信息，显示的声明
- 依赖的模块自动注入到作用域里，彼此独立
- 兼容传统的Module Pattern
- exports和require本身也是module，导入之后，代码风格跟服务器端module一致
- 如果需要的module未定义，自动加载相应的文件。

# OzJS也是异步模块的兼容实现， 除此之外的优点

- require.js: 5.3K
- oz.js: 1.5K
- 主要的功能和接口设计完全一致，模块名称（MRL）的用法有不同的考虑
- OzJS要求用 .def 方法显示的声明模块文件的URI，多个模块可以声明为同一个文件

# OzJS的def

模块名称对应的  
URI必须声明

只声明URI，会自动绑  
定文件内的匿名模块

声明的优点之一  
是下载一个文件可  
以获得多个模块

```
oz.def('moduleA.js');
oz.def('B', ['lang'], 'moduleB.js');
oz.def('jQuery/1.4.3', 'jQuery-1.4.3.js');
oz.def('jQuery/1.3.2', 'jQuery-1.4.3.js');

oz.def('lang', ['require', 'B', 'jQuery/1.3.2'], function(require, B, $){
    return {
        type: function(){}
    };
});

oz.require(['jQuery', 'B', 'moduleA.js'], function($, B, A){
});
```

支持多版本

自动关联最新版本

# OzJS的内部

- `def`只注册模块，不执行任何代码，在需要立刻使用的时候才执行模块的初始化，生成模块对象
- 模块的初始化代码只执行一次。
- 依赖关系的拓扑结构经常是树状甚至网状的，需要先作深度优先的遍历，记录节点状态，最后展开成一个排序过的队列
- 遍历中如果遇到没有注册代码块（或称工厂函数）的模块，先异步加载对应的文件，更新计数器。等到计数器归零后重新开始遍历。当计数器为零且遍历结束后，才开始按顺序执行模块队列
- 模块文件并行加载，只有模块对象和初始化代码块之间才存在依赖关系，模块文件本身不应该存在依赖。
- 多个版本并存时，原始模块名自动指向最新版本的模块
- 容忍依赖配置出错：忽略未注册的模块，循环依赖时忽略其中一方。

Make it easier to share code,  
and people will share more.

-- Dave Herman, “JavaScript needs modules”

# 支持模块化开发的工具



# 模块化和性能的矛盾

```
oz.def('jQuery', 'lib/jquery_src.js');
oz.def('url', 'tui/src/utilities/urlkit.js');
oz.def('tui/src/utilities/domain.js');
oz.def('swf', 'tui/src/utilities/flash.js');
oz.def('tui/src/utilities/key.js');
oz.def('perf', 'tui/src/utilities/performance.js');
oz.def('tui/src/plugins/loginBox.js');
oz.def('tui/src/plugins/quickPlaylist.js');
oz.def('tui/src/plugins/searchHint.js');
```

```
oz.def('main', ['jQuery', 'url', 'perf'], function($, urlkit, perfkit){
```

The screenshot shows the Page Speed Insights tool interface. At the top, it displays 'ALL (23)' items, 'FILTER BY: CONTENT (6) | COOKIE' options, and tabs for 'Performance', 'Resources', 'Export', and 'Help'. Below this, the 'Performance' tab is active, showing a 'Page Speed Score: 85/100' with a yellow triangle icon indicating improvement. A 'Refresh Analysis' button is also present. The main area lists several optimization suggestions:

- A Make fewer HTTP requests
- F Use a Content Delivery Network (CDN)
- A Avoid empty src or href
- F Add Expires headers
- A Compress components with gzip
- ! Combine external JavaScript
- ! Leverage browser caching
- ! Serve static content from a cookieless domain
- ▲ Combine external CSS
- ▲ Minimize DNS lookups
- ▲ Minimize request size

# OzJS能做到， 而RequireJS做不到的

```
function(DEFER_URL){  
    oz.def('jQuery', DEFER_URL);  
    oz.def('url', DEFER_URL);  
    oz.def('domain', DEFER_URL);  
    oz.def('swf', DEFER_URL);  
    oz.def('key', DEFER_URL);  
    oz.def('perf', DEFER_URL);  
    oz.def('loginBox', DEFER_URL);  
    oz.def('quickPlaylist', DEFER_URL);  
    oz.def('searchHint', DEFER_URL);  
}  
)('lib/defer_combo.js')
```

```
oz.def('main', ['jQuery', 'url', 'perf'], function($, urlkit, perfkit){
```

```
});
```

```
1  /**  
2   * defer  
3   * @modified: $Author$  
4   * @version: $Rev$  
5   * @!note 不包含 plugin 和业务逻辑  
6   * @charset gb18030  
7   * @import lib/jquery.js  
8   * @import tui/src/utilities/urlkit.js  
9   * @import tui/src/utilities/domain.js  
10  * @import tui/src/utilities/flash.js  
11  * @import tui/src/utilities/key.js  
12  * @import tui/src/utilities/performance.js  
13  * @import tui/src/plugins/loginBox.js  
14  * @import tui/src/plugins/quickPlaylist.js  
15  * @import tui/src/plugins/searchHint.js  
16  */
```

# OzJS能做到， 而RequireJS做不到的

```
$ tuicompile js/lib/defer_src.js -z  
[CMD] python /Users/dexteryy/code/cmd/tuicompiler/tuipacker.py j  
s/lib/defer_src.js
```

```
build:  
total files: 10  
[read]import: lib/jquery.js  
[read]import: tui/src/utilities/urlkit.js  
[read]import: tui/src/utilities/domain.js  
[read]import: tui/src/utilities/flash.js  
[read]import: tui/src/utilities/key.js  
[read]import: tui/src/utilities/performance.js  
[read]import: tui/src/plugins/loginBox.js  
[read]import: tui/src/plugins/quickPlaylist.js  
[read]import: tui/src/plugins/searchHint.js  
[read]import: js/lib/defer_src.js  
charset: gb18030  
[write]packed file: js/lib/defer_combo.js  
BUILD SUCCESS!
```

```
[CMD] /Users/dexteryy/code/cmd/tuicompiler/bin/dos2unix js/lib/d  
efer_combo.js  
dos2unix: converting file js/lib/defer_combo.js to UNIX format ...
```

```
Compress:  
[write]compressed file: js/lib/defer.js  
[CMD] java -jar /Users/dexteryy/code/cmd/tuicompiler/bin/yuicom  
pressor.jar --charset gb18030 js/lib/defer_combo.js -o js/lib/defer.j  
s  
COMPRESS SUCCESS!
```

TUICompiler是土豆代码库  
使用的构建工具

```
jquery.js  
/utilities/urlkit.js  
/utilities/domain.js  
/utilities/flash.js  
/utilities/key.js  
/utilities/performance.js  
/plugins/loginBox.js  
/plugins/quickPlaylist.js  
/plugins/searchHint.js
```

```
erfkit){
```

# OzJS + TUICompiler

OzJS的依赖查找算法来自历史更长的TUICompiler，  
它们导入模块的行为完全一致

同一种模式在不同层面上的体现

动态 Module Loader (运行时)

+

静态 Moduler Loader (编译时)

# “编译工具”其实是标题党

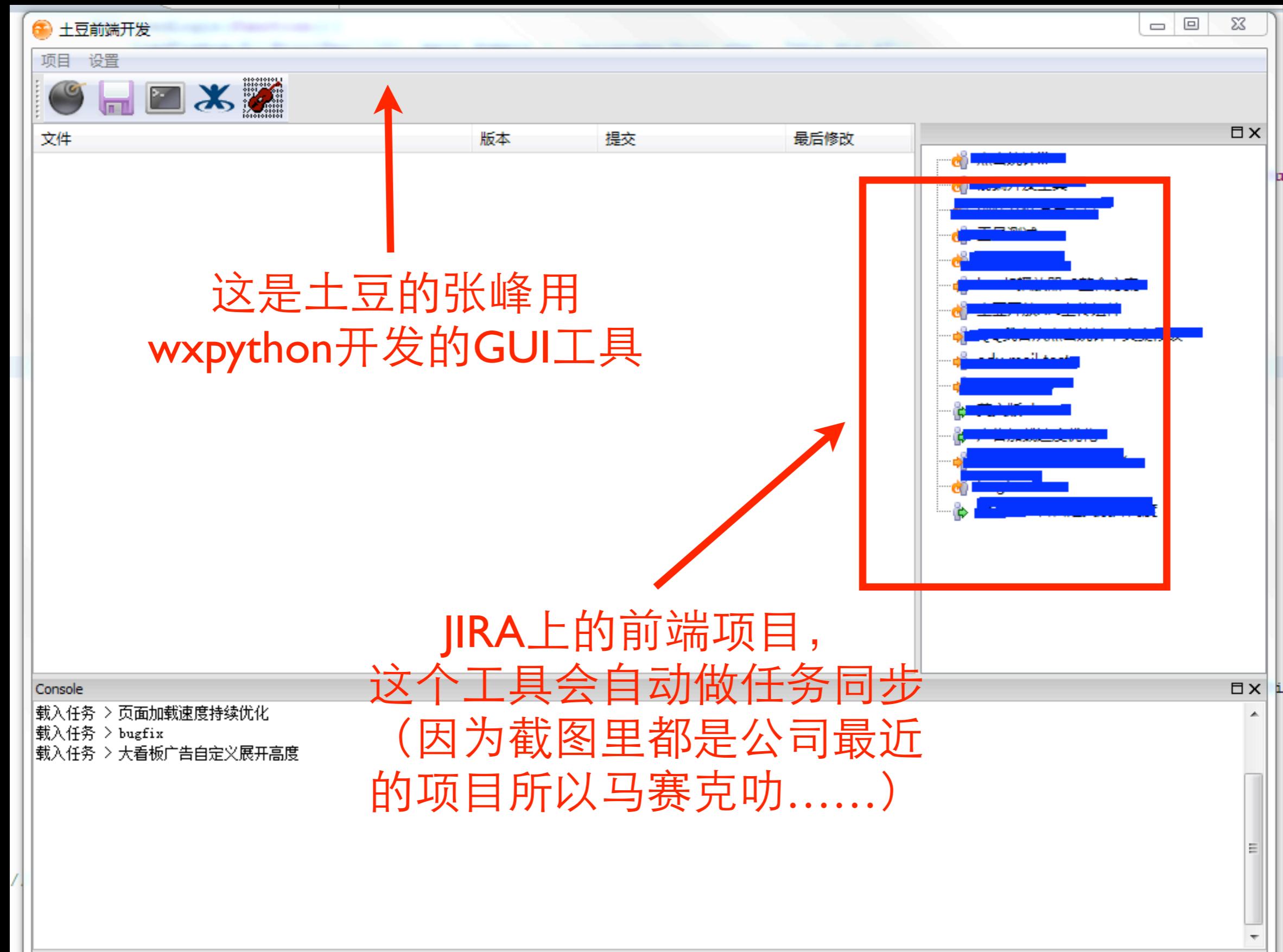
- 用Python开发，有扩展机制
- <https://github.com/dexteryy/TUICompiler>
- 过滤器插件类似WSGI的洋葱皮模式
- 目前还不算编译器（compiler），只是预处理器（preprocessor）
- 下一步：引入静态语法分析

# 作为Vim的:make来使用 效果更佳

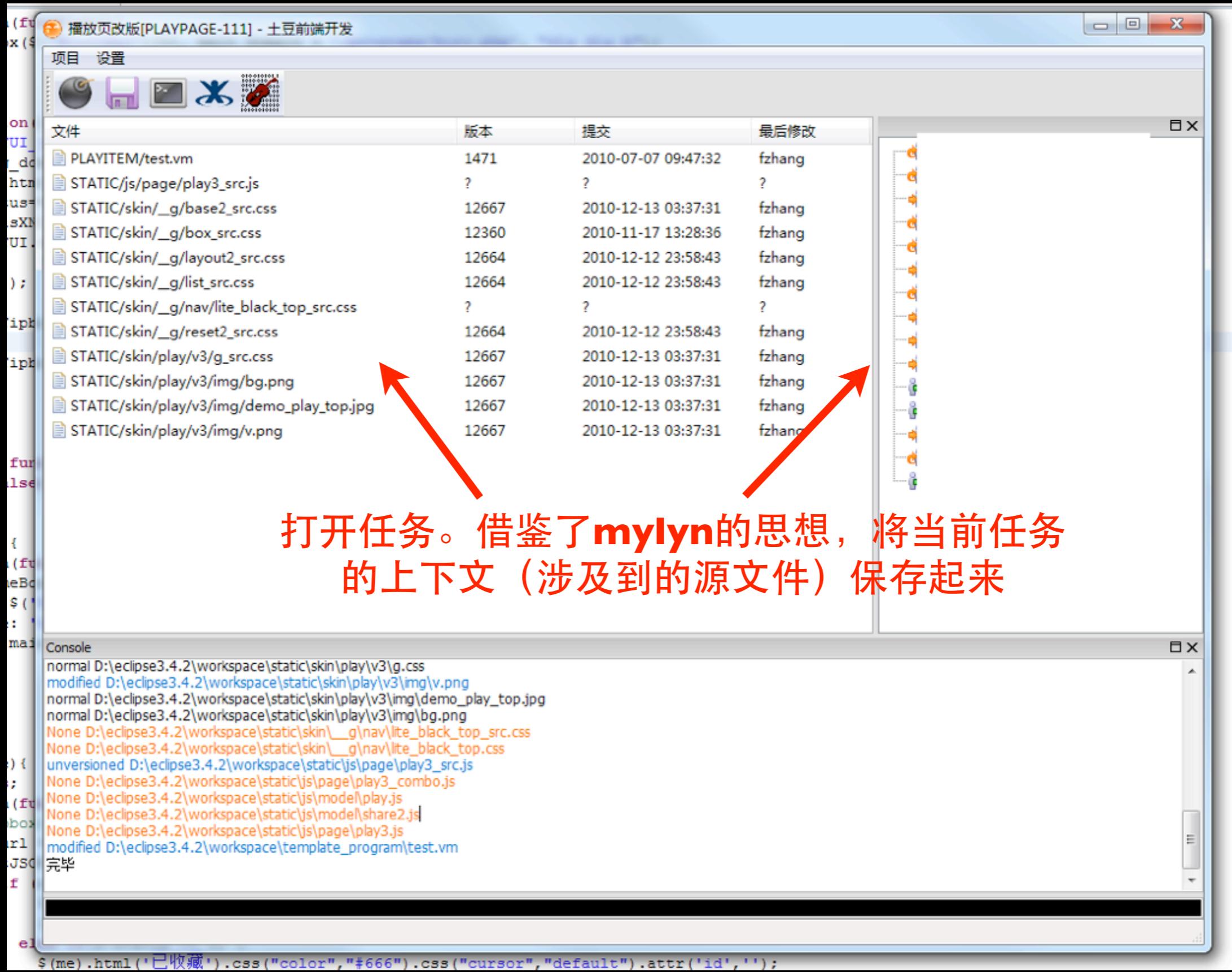
Q: 如果老子不用vim，而且用windows，咋办！？

A: 凉拌.....呃我是说.....请看接下来的好东西.....

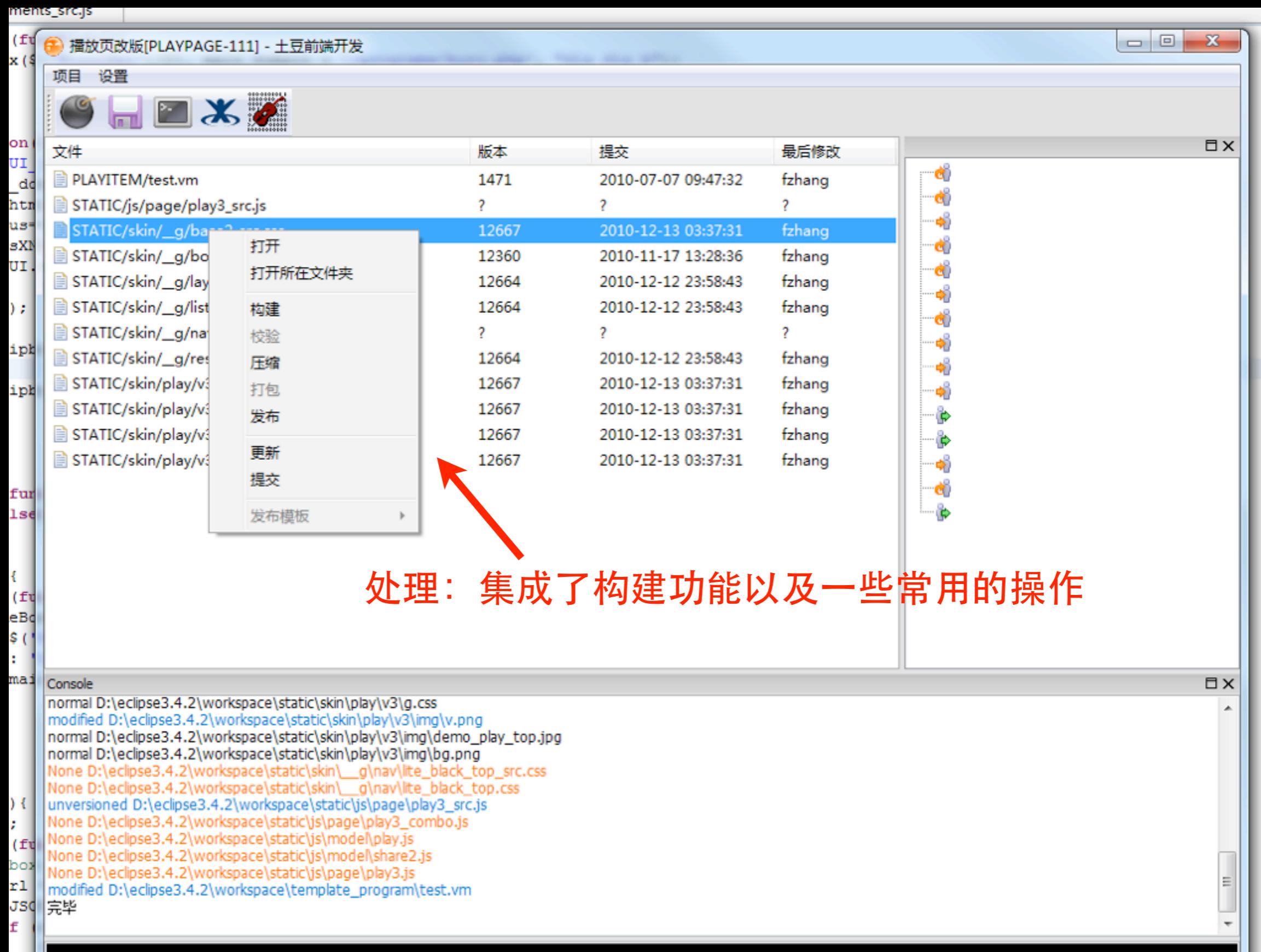
# 集成TUICompiler的前端项目工具



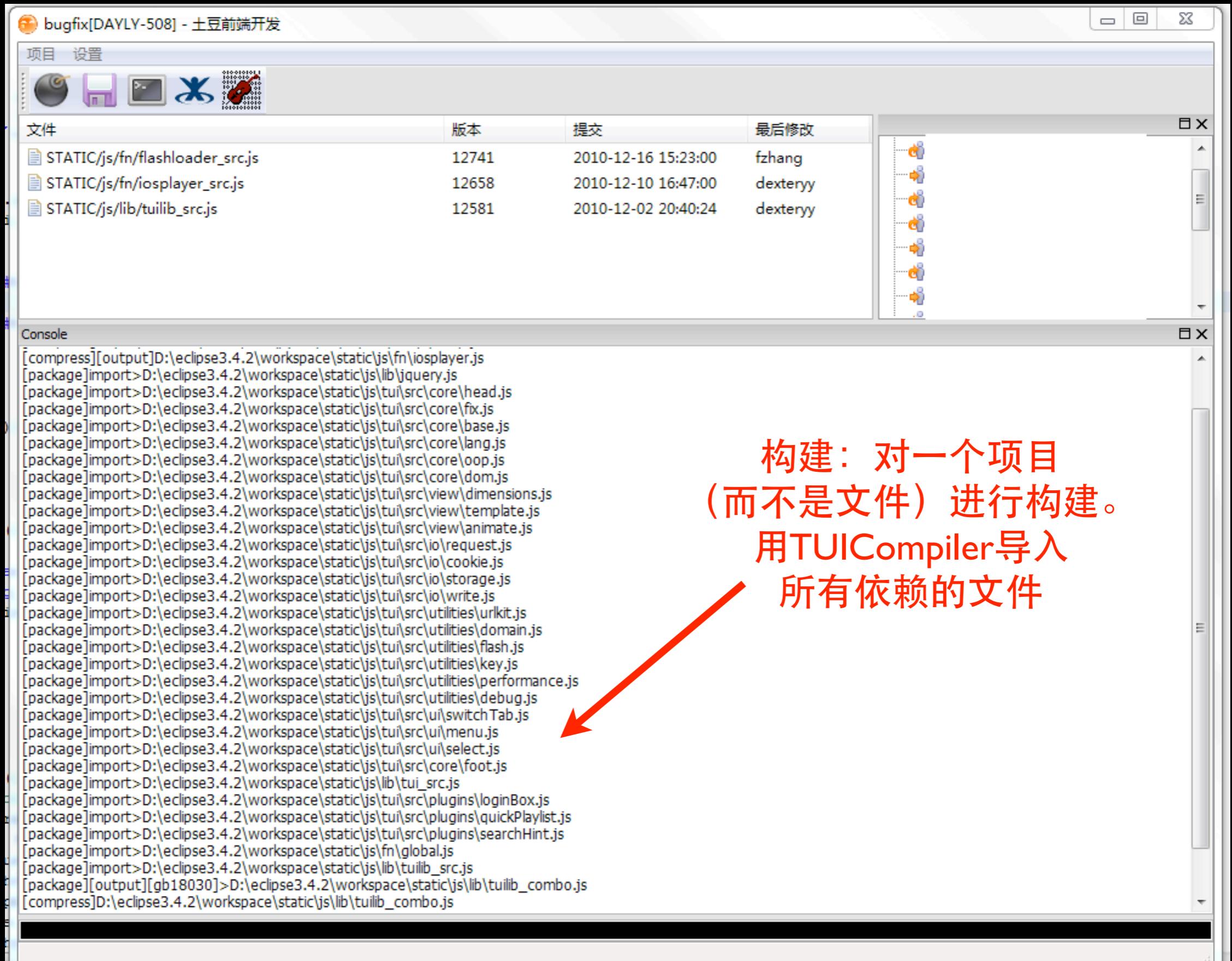
# 集成TUICompiler的前端项目工具



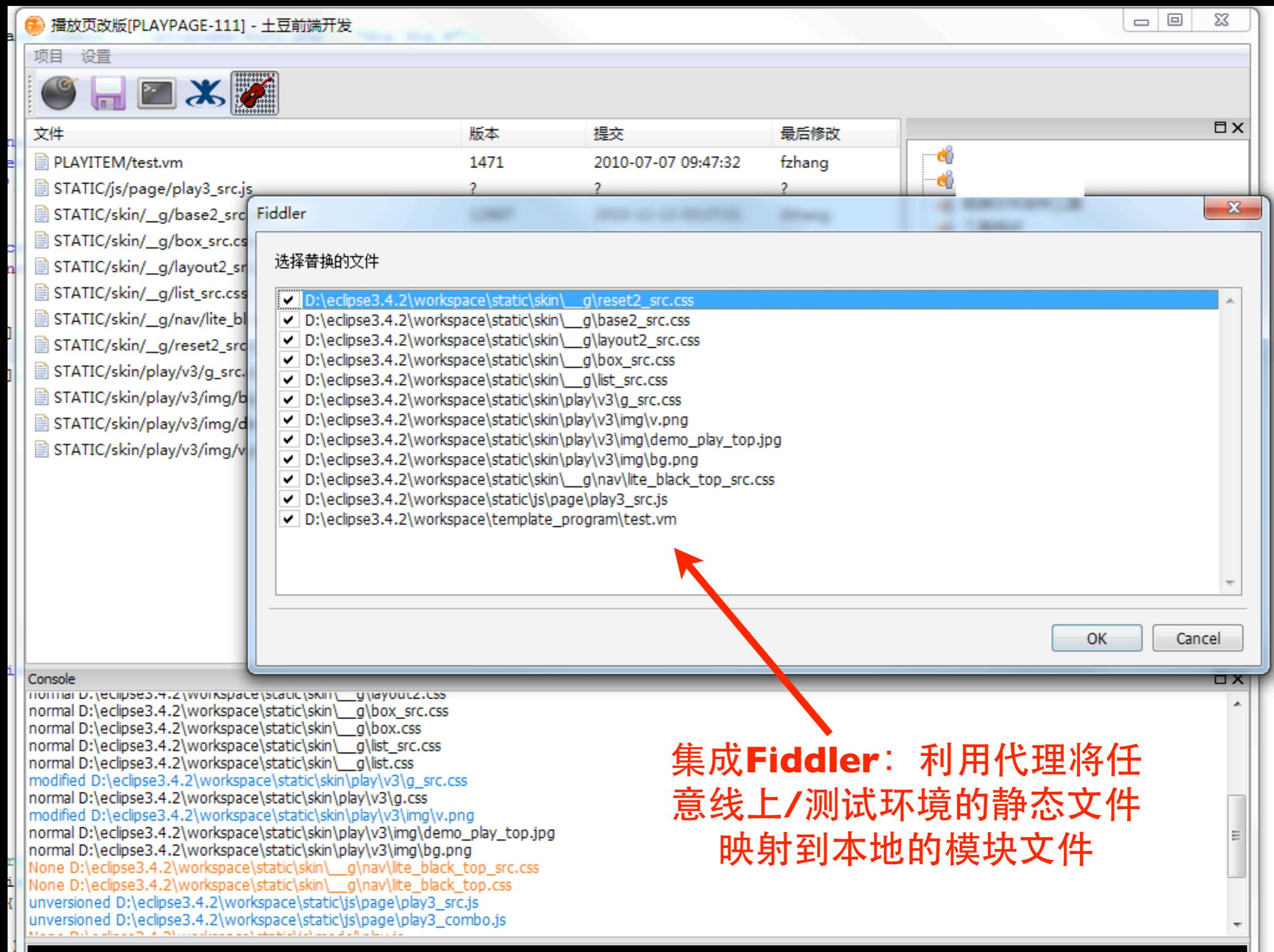
# 集成TUICompiler的前端项目工具



# 集成TUICompiler的前端项目工具

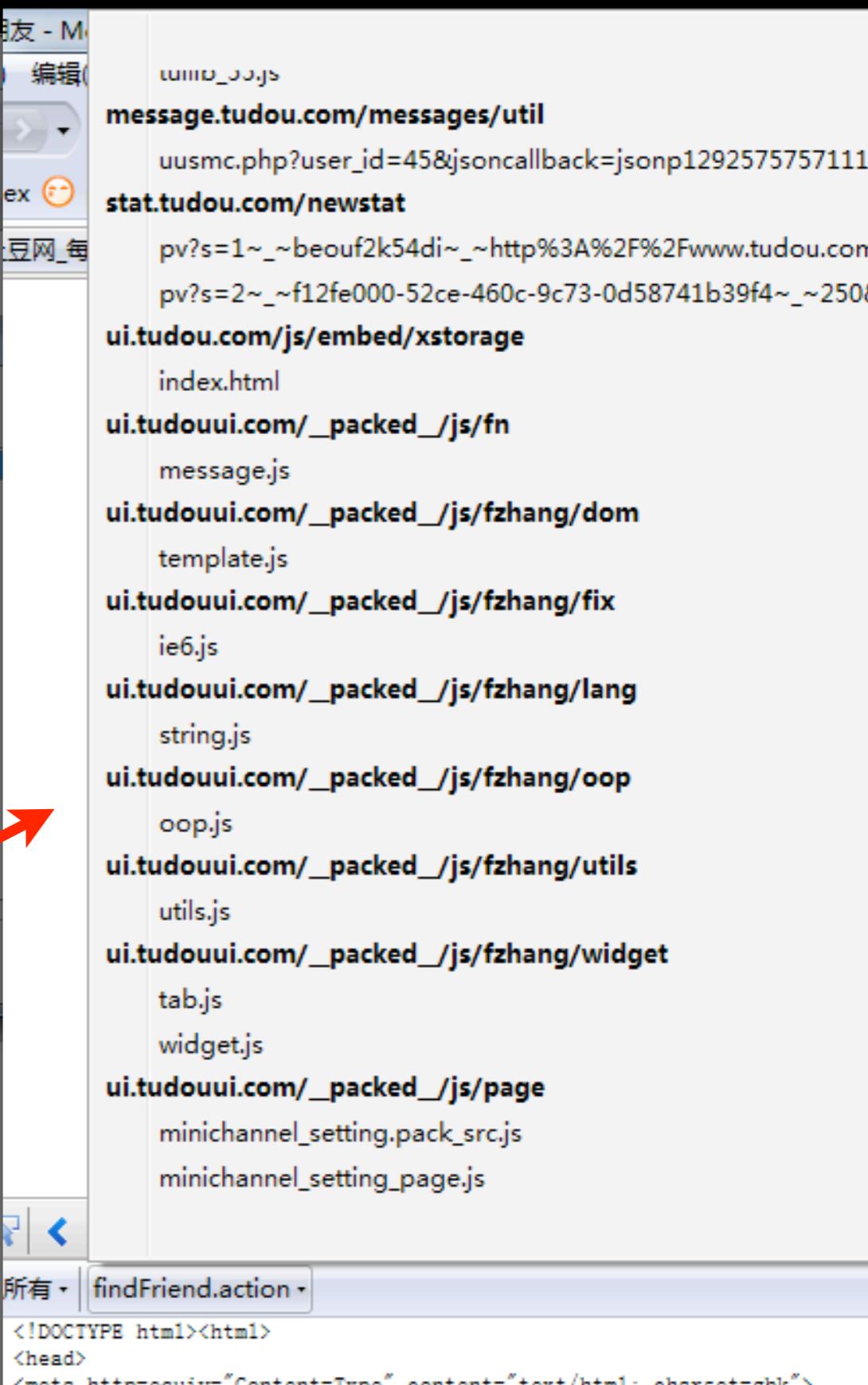


# 集成TUICompiler的前端项目工具



# 集成**Fiddler**: 利用代理将任意线上/测试环境的静态文件映射到本地的模块文件

# 集成TUICompiler的前端项目工具



The screenshot shows a browser developer tools window with the Network tab selected. It lists several requests made to `message.tudou.com/messages/util` and `ui.tudouui.com/_packed_/js/fzhang/`. The requests include files like `uusmc.php`, `stat.tudou.com/newstat`, `index.html`, `message.js`, `template.js`, `ie6.js`, `string.js`, `oop.js`, `utils.js`, `tab.js`, and `widget.js`.

A red arrow points from the text below to the Fiddler interface, indicating the integration of Fiddler for proxying static files.

集成**Fiddler**: 利用代理将任意线上/测试环境的静态文件映射到本地的模块文件

Fiddler interface showing a list of files to replace:

- D:\eclipse3.4.2\workspace\static\skin\\_\_g\reset2\_src.css
- D:\eclipse3.4.2\workspace\static\skin\\_\_g\base2\_src.css
- D:\eclipse3.4.2\workspace\static\skin\\_\_g\layout2\_src.css
- D:\eclipse3.4.2\workspace\static\skin\\_\_g\box\_src.css
- D:\eclipse3.4.2\workspace\static\skin\\_\_g\list\_src.css
- D:\eclipse3.4.2\workspace\static\skin\play\v3\g\_src.css
- D:\eclipse3.4.2\workspace\static\skin\play\v3\img\bv.png
- D:\eclipse3.4.2\workspace\static\skin\play\v3\img\demo\_play\_top.jpg
- D:\eclipse3.4.2\workspace\static\skin\play\v3\img\bg.png
- D:\eclipse3.4.2\workspace\static\skin\\_\_g\nav\lite\_black\_top\_src.css
- D:\eclipse3.4.2\workspace\static\js\page\play3\_src.js
- D:\eclipse3.4.2\workspace\template\_program\test.vm

沒了.....



- Twitter: <http://twitter.com/dexteryy>
- Google buzz: <http://www.google.com/profiles/dexter.yy>
- Blog: <http://www.limboy.com>
- OzJS: <https://github.com/dexteryy/OzJS>
- TUICompiler: <https://github.com/dexteryy/TUICompiler>