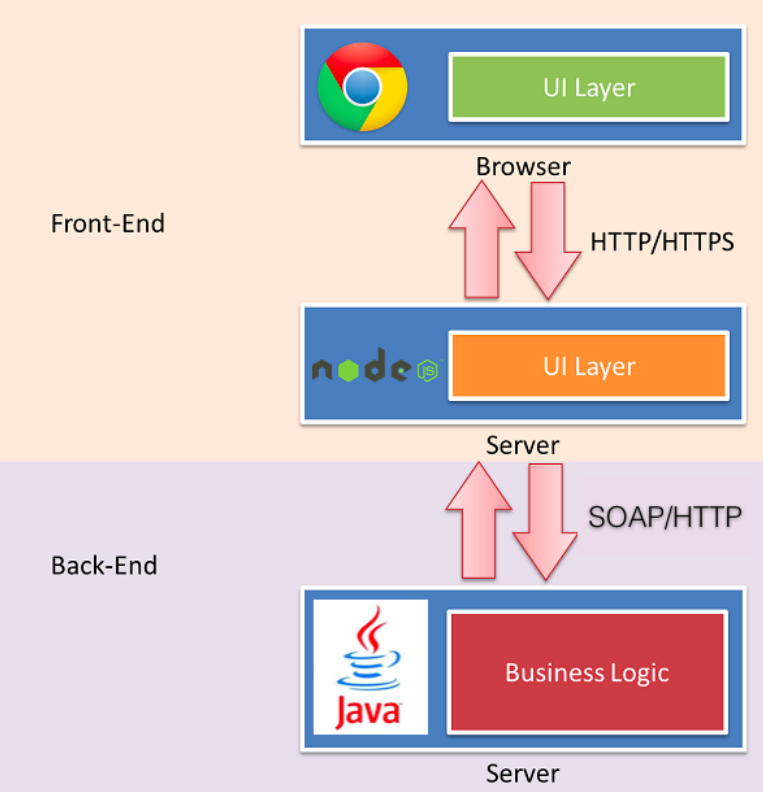
React On JVM

李传忠

Index

- 畅想
- Javascript on JVM
- React Case

畅想



Full Stack

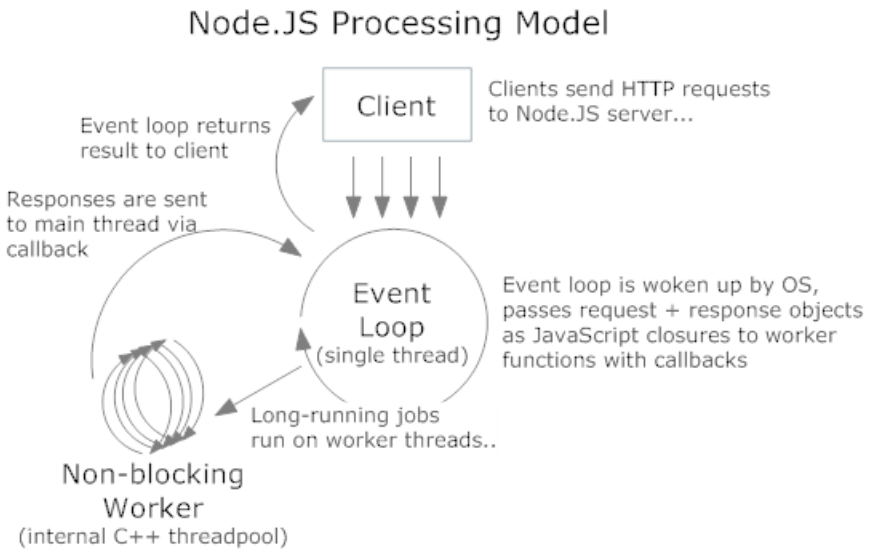
1.Front-end UI layer 处理浏览器层的展现逻辑。通过 CSS 渲染样式，通过 JavaScript 添加交互功能，HTML 的生成也可以放在这层，具体看应用场景。
2、Back-end UI layer 处理路由、模板、数据获取、cookie 等。

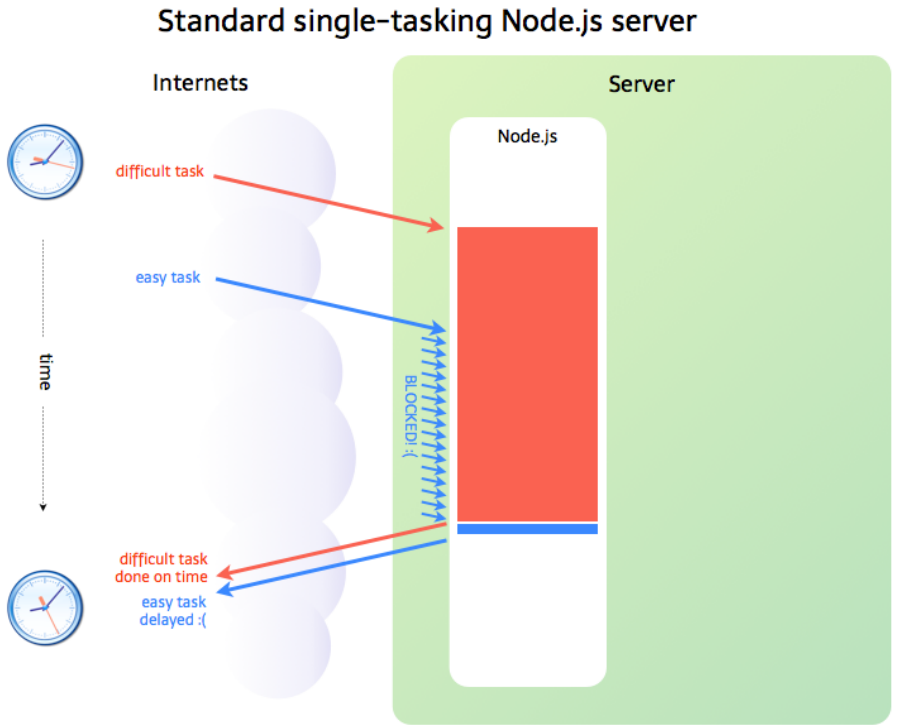
SEO友好	代码复用
前后端分离	减少异步请求

https://github.com/lifesinger/lifesinger.github.io/issues/184

现实的问题

- Node.js软肋- CPU密集型任务
- Node 层与 Java 层的高效通信
- 不同的部署、运维方案
- 大量的历史遗留



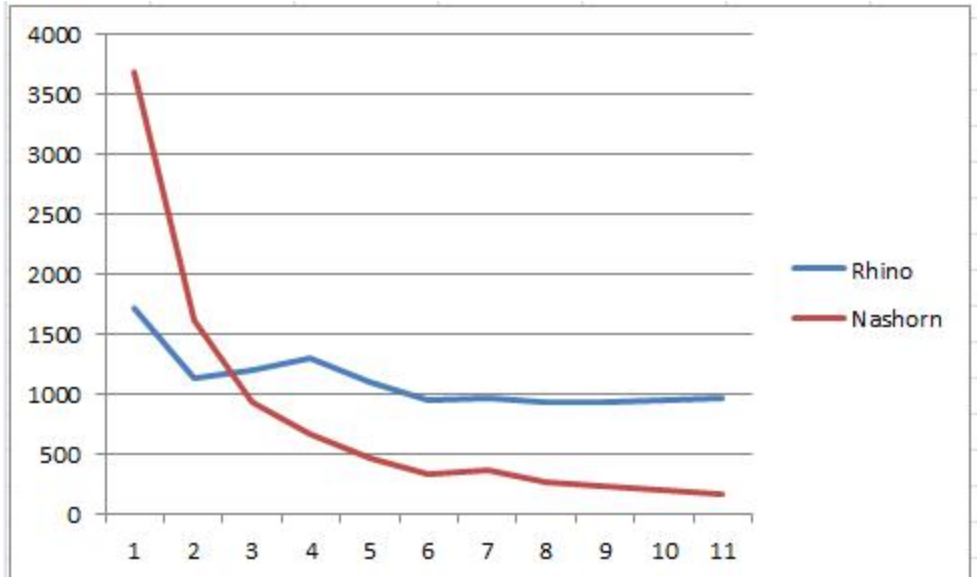


Javascript on JVM

- Rhino & Nashorn
- 典型应用
- 访问Java组件

Rhino & Nashorn

- 成熟的GC
- 动态编译
- 多线程支持
- 丰富的标准库和第三方库
- ES5.1



Esprima解析Jquery

访问JAVA组件

- 直接访问
- 多线程
- 无额外的IO开销

```
// 访问Java类Thread
var Thread = Java.type("java.lang.Thread");

// 带有run方法的子类
var MyThread = Java.extend(Thread, {
    run: function() {
        print("Run in separate thread");
    }
});
var th = new MyThread();
th.start();
th.join();
```

典型应用

- R.js for ant
- Avatar.js

React Case

- JSX编译
- React组件
  - 前端引用
  - 服务器端调用
- 性能

JSX 编译

- 编译动态生成的JSX
- for Ant

```
ScriptEngineManager sem = new ScriptEngineManager();
ScriptEngine engine = sem.getEngineByName("nashorn");
engine.eval("var process = {env:{}}");
engine.eval("var global = this;");
engine.eval(ScriptResourceUtil.getScriptCode("jvm-npm.js"));
engine.eval(ScriptResourceUtil.getScriptCode("react.js"));
engine.eval(ScriptResourceUtil.getScriptCode("JSXTransformer.js"));
engine.eval("var getJsx = function(a){ return JSXTransformer.transform(a)['code'];} ");
```

React组件

- 前端引用
- 服务器端调用

```
    //on browser
    React.render(React.createElement(TodoList, {items: params}), mountNode);

    //on server
    String str = (String)engine.eval("React.renderToStaticMarkup(React.createElement(TodoList,
{items: params}))");
```

性能

预热(Narshorn:692ms,Rhino:200ms;FM:78ms)后,*Nashorn 中的性能已经达到FreeMarker的70%以上了,*

Thanks

iUAP前端技术小组