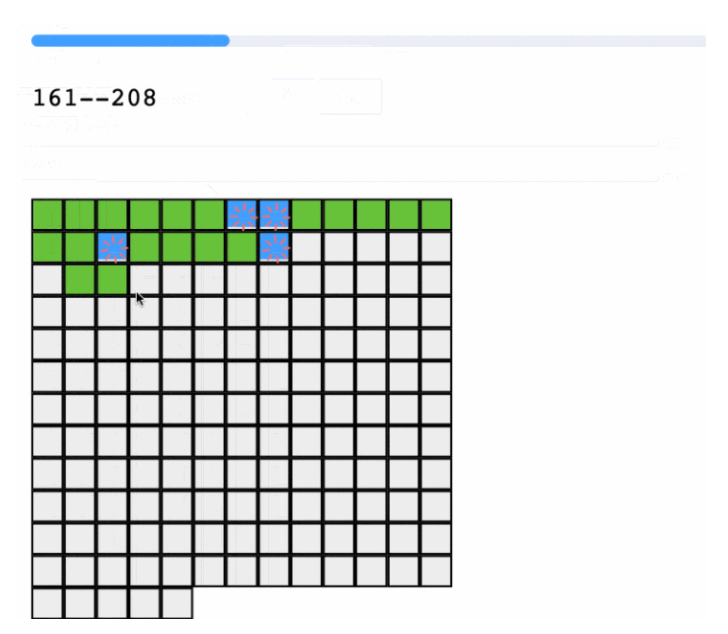
# **大文件上传和断点续传**

1. 计算 hash耗时的问题，不仅可以通过 web-workder，还可以参考 React的 FFiber架构，通过 requestIdleCallback来利用浏览器的空闲时间计算，也不会卡死主线程
2. 文件 hash的计算，是为了判断文件是否存在，进而实现秒传的功能，所以我们可以参考 布隆过滤器的理念, 牺牲一点点的识别率来换取时间，比如我们可以 抽样算hash
3. 文中通过 web-workder让 hash计算不卡顿主线程，但是大文件由于切片过多，过多的 HTTP链接过去，也会把浏览器打挂 (我试了4个G的，直接卡死了)， 我们可以通过控制异步请求的 并发数来解决，我记得这也是头条的一个面试题
4. 每个切片的上传进度不需要用表格来显示，我们换成方块进度条更直管一些(如图)

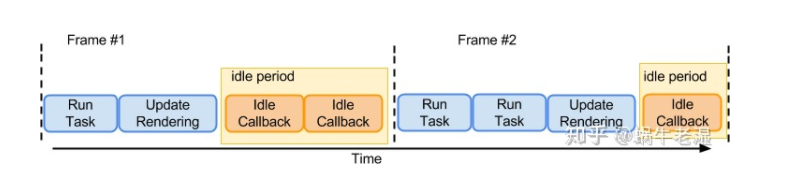


1. 并发上传中，报错如何重试，比如每个切片我们允许重试两次，三次再终止
2. 由于文件大小不一，我们每个切片的大小设置成固定的也有点略显笨拙，我们可以参考 TCP协议的 慢启动策略， 设置一个初始大小，根据上传任务完成的时候，来动态调整下一个切片的大小， 确保文件切片的大小和当前网速匹配
3. 小的体验优化，比如上传的时候
4. 文件碎片清理

## **时间切片计算文件hash**

其实就是 time-slice概念， React中 Fiber架构的核心理念，利用浏览器的空闲时间，计算大的diff过程，中途又任何的高优先级任务，比如动画和输入，都会中断diff任务， 虽然整个计算量没有减小，但是大大提高了用户的交互体验

window.requestIdleCallback()方法将在浏览器的空闲时段内调用的函数排队。这使开发者能够在主事件循环上执行后台和低优先级工作 requestIdelCallback执行的方法，会传递一个 deadline参数，能够知道当前帧的剩余时间。



该图中的两个帧，在每一帧内部， TASK和 redering只花费了一部分时间，并没有占据整个帧，那么这个时候，如图中 idle period的部分就是空闲时间，而每一帧中的空闲时间，根据该帧中处理事情的多少，复杂度等，消耗不等，所以空闲时间也不等。

而对于每一个 deadline.timeRemaining()的返回值，就是如图中， IdleCallback到所在帧结尾的时间（ms级）

## **抽样hash**

计算文件 md5值的作用，无非就是为了判定文件是否存在，我们可以考虑设计一个抽样的 hash，牺牲一些命中率的同时，提升效率，设计思路如下

1. 文件切成2M的切片
2. 第一个和最后一个切片全部内容，其他切片的取 首中尾三个地方各2个字节
3. 合并后的内容，计算 md5，称之为 影分身Hash
4. 这个 hash的结果，就是文件存在，有小概率误判，但是如果不存在，是100%准的的 ，和布隆过滤器的思路有些相似， 可以考虑两个 hash配合使用

js-spark-md5是做什么的？可以无需上传文件就快速获取本地文件md5.，每个文件的md5值都是唯一的，正因为每个文件的md5是一样的，那么，我们在做文件上传的时候，就只要在前端先获取要上传的文件md5，并把文件md5传到服务器，对比之前文件的md5，如果存在相同的md5，我们只要把文件的名字传到服务器关联之前的文件即可，并不需要再次去上传相同的文件，再去耗费存储资源、上传的时间、网络带宽。

1. 我在自己电脑上试了下1.5G的文件，全量大概要20秒，抽样大概1秒还是很不错的， 可以先用来判断文件是不是不存在

## **网络请求并发控制**

大文件 hash计算后，一次发几百个 http请求，计算哈希没卡，结果 TCP建立的过程就把浏览器弄死了，而且我记得本身异步请求并发数的控制，本身就是头条的一个面试题

思路其实也不难，就是我们把异步请求放在一个队列里，比如并发数是3，就先同时发起3个请求，然后有请求结束了，再发起下一个请求即可， 思路清楚，代码也就呼之欲出了

我们通过并发数max来管理并发数，发起一个请求 max--，结束一个请求 max++即可

## **慢启动策略实现**

[TCP拥塞控制的问题](https://www.zhihu.com/question/58517416/answer/158142955) 其实就是根据当前网络情况，动态调整切片的大小

1. chunk中带上 size值，不过进度条数量不确定了，修改 createFileChunk， 请求加上时间统计)
2. 比如我们理想是30秒传递一个
3. 初始大小定为1M，如果上传花了10秒，那下一个区块大小变成3M
4. 如果上传花了60秒，那下一个区块大小变成500KB 以此类推
5. 并发+慢启动的逻辑有些复杂，我自己还没绕明白，囧所以先一次只传一个切片，来演示这个逻辑，新建一个 handleUpload1函数

## **并发重试+报错**

1. 请求出错.catch 把任务重新放在队列中
2. 出错后progress设置为-1 进度条显示红色
3. 数组存储每个文件hash请求的重试次数，做累加 比如 [1,0,2],就是第0个文件切片报错1次，第2个报错2次
4. 超过3的直接 reject

## **文件碎片清理**

如果很多人传了一半就离开了，这些切片存在就没意义了，可以考虑定期清理，当然 ，我们可以使用[node-schedule](https://link.zhihu.com/?target=https://github.com/node-schedule/node-schedule" \t "https://zhuanlan.zhihu.com/p/_blank)来管理定时任务 比如我们每天扫一次 target，如果文件的修改时间是一个月以前了，就直接删除把