在线程与进程中我们讲述了为啥需要需要进程，为啥需要线程。那么在浏览器里都有哪些进程与线程，他主要的工作是做什么？本节将从大概的讲述浏览器的线程与进程以及他们各自的作用。

在浏览器中，进程主要有以下几个：

1. GPU进程。浏览器全局只有这么一个进程。主要是与图形渲染有关。
2. 其他插件的进程，比如你给你浏览器装了一个插件，那么这一个插件就是一个进程。
3. Browser进程：浏览器的主进程（负责协调、主控），只有一个。主要功能有以下：

   > - 负责浏览器界面显示，与用户交互。如前进，后退等
   > - 负责各个页面的管理，创建和销毁其他进程

4. 浏览器渲染进程，也被称为浏览器内核。分类有：

   > **Google Chrom:** Chrome 28开发版本的版本说明中还在使用**WebKit**，而最新的Chrome 28.0.1469.0中已经替换为**Blink**。
   >
   > **Internet Explorer：** **Trident**内核，也是俗称的IE内核
   >
   > **Mozilla Firefox：** Gecko内核，俗称**Firefox**内核。
   >
   > **Safari：** **WebKit**
   >
   > **Opera：** 最初是自己的**Presto**内核，后来是**Webkit**，现在是**Blink**内核

   浏览器渲染进程的作用：负责页面的渲染，脚本执行，时间处理，网络请求等功能。

浏览器渲染进程

在一个进程中，至少有一个线程。线程被称为CPU任务调度的最小执行单位。那么在浏览器的渲染进程中，有以下几个线程：

1. **GUI渲染线程**： 解析html文档，生成DOM树与CSS树（**需要注意的是css树不会阻塞dom树的生成**）。当生成DOM树与CSS树之后，就根据这两 个数生成一个render树（**在生成render树的时候，如果有一方没有解析完毕就会等待解析完成。此时此刻是双方会互相阻塞**），然后将这个render树渲染到界面上。当页面触发回流或者重绘的时候，会再次执行此次操作。
2. **JS线程**：用来执行JS代码。
3. **定时器线程**：用来处理定时器线程，当定时器到期的时候，将回调放到任务队列里面，等待JS线程的执行。那么有了JS线程，我们为啥还需要定时器线程呢？看下面代码解释：

```
function test() {
  setTimeout(() => {
  console.log('我是计时器1');
}, 1000);
  setTimeout(() => {
    console.log('我是计时器2');
  }, 2000);

};
test();
```

假如没有定时器线程，又因为JS是单线程的，我只能一个一个的压入栈中执行，那么首先是计时器1入栈，接着是计时器2入栈。但是因为计时器1的时间小于计时器2的时间，那么应该计时器1首先出栈。但是因为栈是一个先进后出的数据结构。那么这就发火说呢过了冲突。因为计时器2还没有到时间，所以计时器1也就不能出栈。

4. **事件触发线程**。用来管理事件的触发，例如：点击事件，鼠标移动事件。当这些个事件被触发的时候，就会将这些事件的回调添加到任务队列里，等待JS执行。
5. **异步HTTP请求线程**。在XMLHttpRequest在连接后新启动的一个线程，线程如果检测到请求的状态变更，如果设置有回调函数，该线程会把回调函数添加到事件队列，同理，等待JS引擎空闲了执行。

JS线程与GUI线程互斥的原因

主要是因为当一个文档在加载的时候，如果此时JS线程也在加载执行，例如要获取一个id为demo1的节点，此时此刻我们渲染过程中，还并没有生成render树，也就不会进行布局和渲染。那么此时这个节点是没有的，那么肯定就会找不到。因此为了避免这种情况的发生，在浏览器中，JS线层和GUI线程是互斥的，当一个执行的时候，另外一个就会被强制挂起。这样就会导致一个问题，当JS执行一个时间复杂度非常高的算法的时候，因为迟迟不能执行完毕，导致GUI渲染线程被挂起太久，就会导致页面看起来卡顿，事件响应变慢。解决办法，可以通过Web Worker解决。