1，无需配置就能实现打包的方案

目前已经有很多的打包工具了，包括webpack和browserify。许多的打包工具都是围绕着配置和插件构建的，而且为了让应用正常的工作，超过三四百行的配置并不罕见。这些配置不仅繁琐而且耗时。通常情况下，这可能导致次优化的应用发送到生产环境。该方案设计零配置：只需要将它指向应用程序的入口点，它就能正常工作。

目前现存的打包工具都非常慢。拥有大量文件和依赖的大型应用可能需要花费几分钟的时间来构建，这在开发过程中随着时间的变化而变得尤为痛苦。监听文件变更能够帮助重新构建，但初始的启动仍然非常慢。该方案利用工作线程编译你的代码，利用现代的多核处理器能力。这导致了初始构建的速度大大提升。它还具有一个文件系统缓存，可以保存每一个文件的编译结果，以便后续能够更快的启动。

最后，现有的打包工具都是围绕字符串加载/转换构建的，其中转换需要一个字符串，解析它，进行一些转换，然后再次生成代码。通常这样会导致许多的解析和代码生成在单个文件上运行，这是非常低效的。相反，该方案转换工作在AST上，因此每个文件只有一个解析，多个转换以及一个代码生成。

1.2，运作方式：

将 资源 树转换成 bundle 树。许多其它的打包工具基本上是基于 JavaScript 资源，还有附加在其上的其它格式的资源。例如，在 JS 文件中内联成字符串。 该方案对文件类型无感知的，它能按你所期待的方式那样与任意类型的资源工作，且毋须配置。打包流程共有三个步骤。

1）构建资源数

接受单个入口资源作为输入，可以是任意类型：一个 JS 文件、HTML、CSS等等。资源会被解析，资源的依赖会被提取，资源会被转换成最终编译好的形态。此过程创建了一个资源树。



2） 构建bundle树

一旦资源树被构建好，资源会被放置在 bundle 树中。首先一个入口资源会被创建成一个 bundle，并为动态通过import()导入的资源 创建子 bundle树，这引发了代码的拆分。当不同类型的文件资源被引入，兄弟 bundle 就会被创建。例如你在 JavaScript 中引入了 CSS 文件，那它会被放置在一个与 JavaScript 文件对应的兄弟 bundle 中。如果资源被多于一个 bundle 引用，它会被提升到 bundle 树中最近的公共祖先中，这样该资源就不会被多次打包。

3) 打包

在 bundle 树被构建之后，每一个包都有特定的文件类型的包装器写入文件。打包器知道如何将每个资源的代码合并到由浏览器加载的最终文件中。例如，分析和生成 CSS 代码的 CSS 资源类型和将 CSS 资源组合成最终打包的 CSS 包。 JS，HTML 等存在类似的类型。

以下是打包的原理流程设计,打包过程不需要人为操作配置和设置加载插件或者依赖。



2，实现对批量操作DOM的优化方案

为了节约使用DOM。每次JavaScript对DOM的操作都会改变页面的变现，并重新刷新整个页面，从而消耗了大量的时间。为解决这个问题，可以创建一个文档碎片，把所有的新节点附加其上，然后把文档碎片的内容一次性添加到document中。

javascript操作dom是一个很耗性能的过程，在某些情况下，不得不进行dom循环操作，我们每次对dom的操作都会触发"重排"，这严重影响到能耗，一般通常采取的做法是尽可能的减少dom操作来减少"重排"。  
　　面对循环操作dom的过程，我们选择使用文档碎片（creatDocumentFragment），将需要添加到dom中的内容一次性添加到文档碎片中，然后将文档碎片添加到dom树，这样就可以有效的减少操作dom的次数。



