课程目标

组件注册

全局注册

```
1  Vue.component("custom-a", {
2    render() {
3      return <div>custom-a</div>;
4    }
5  });
```

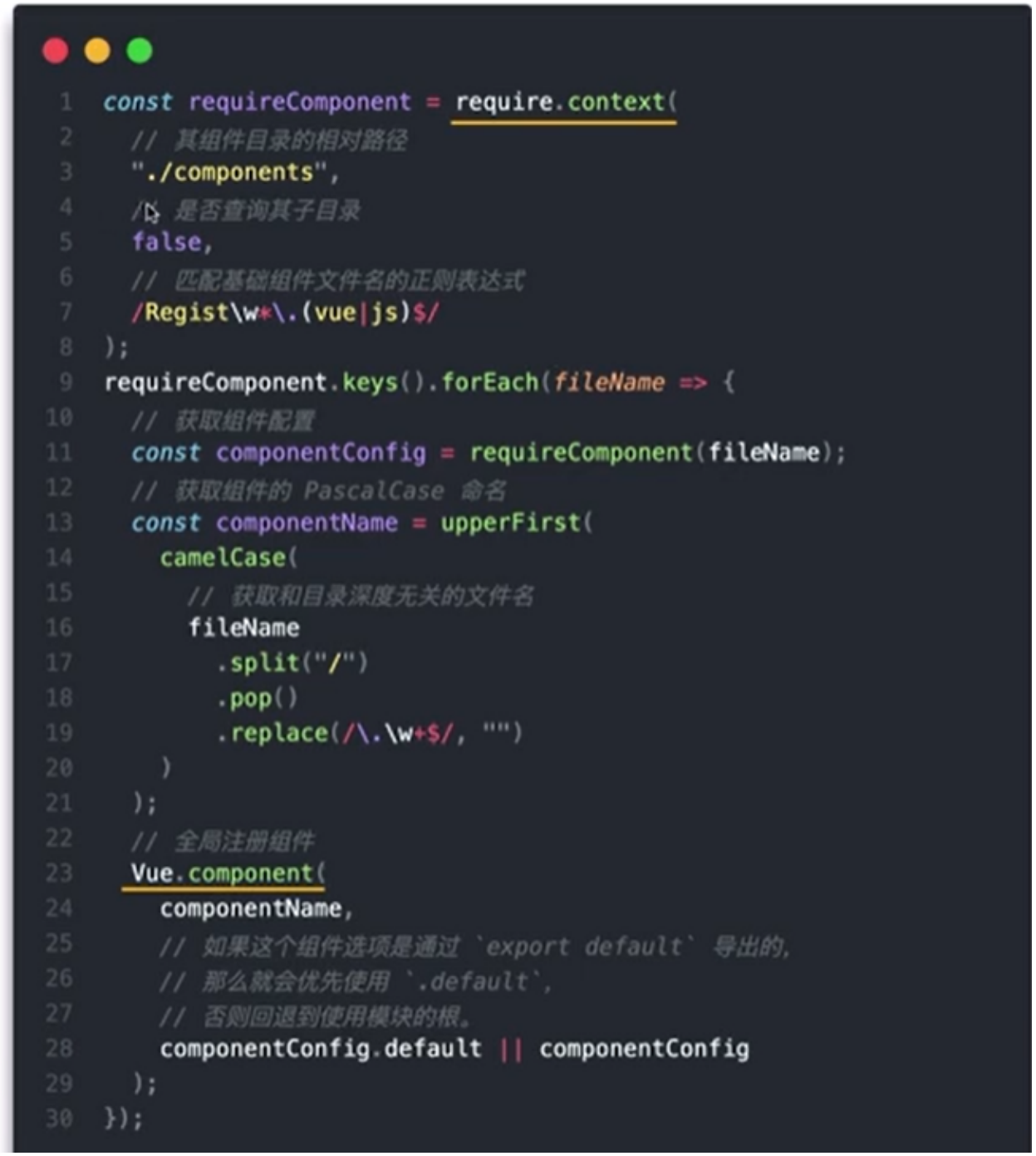
全局注册的组件可以在任何地方使用

不到必须，不要进行全局注册

局部注册

局部注册的组件只能在当前组件中使用

全局导入



```
const requireComponent = require.context(
  // 其组件目录的相对路径
  "./components",
  // 是否查询其子目录
  false,
  // 匹配基础组件文件名的正则表达式
  /Regist\w*\.(vue|js)$/
);
requireComponent.keys().forEach(fileName => {
  // 获取组件配置
  const componentConfig = requireComponent(fileName);
  // 获取组件的 PascalCase 命名
  const componentName = upperFirst(
    camelCase(
      // 获取和目录深度无关的文件名
      fileName
        .split("/")
        .pop()
        .replace(/\.\w+$/, "")
    )
  );
  // 全局注册组件
  Vue.component(
    componentName,
    // 如果这个组件选项是通过 `export default` 导出的,
    // 那么就会优先使用 `.default`,
    // 否则回退到使用模块的根。
    componentConfig.default || componentConfig
  );
});
```

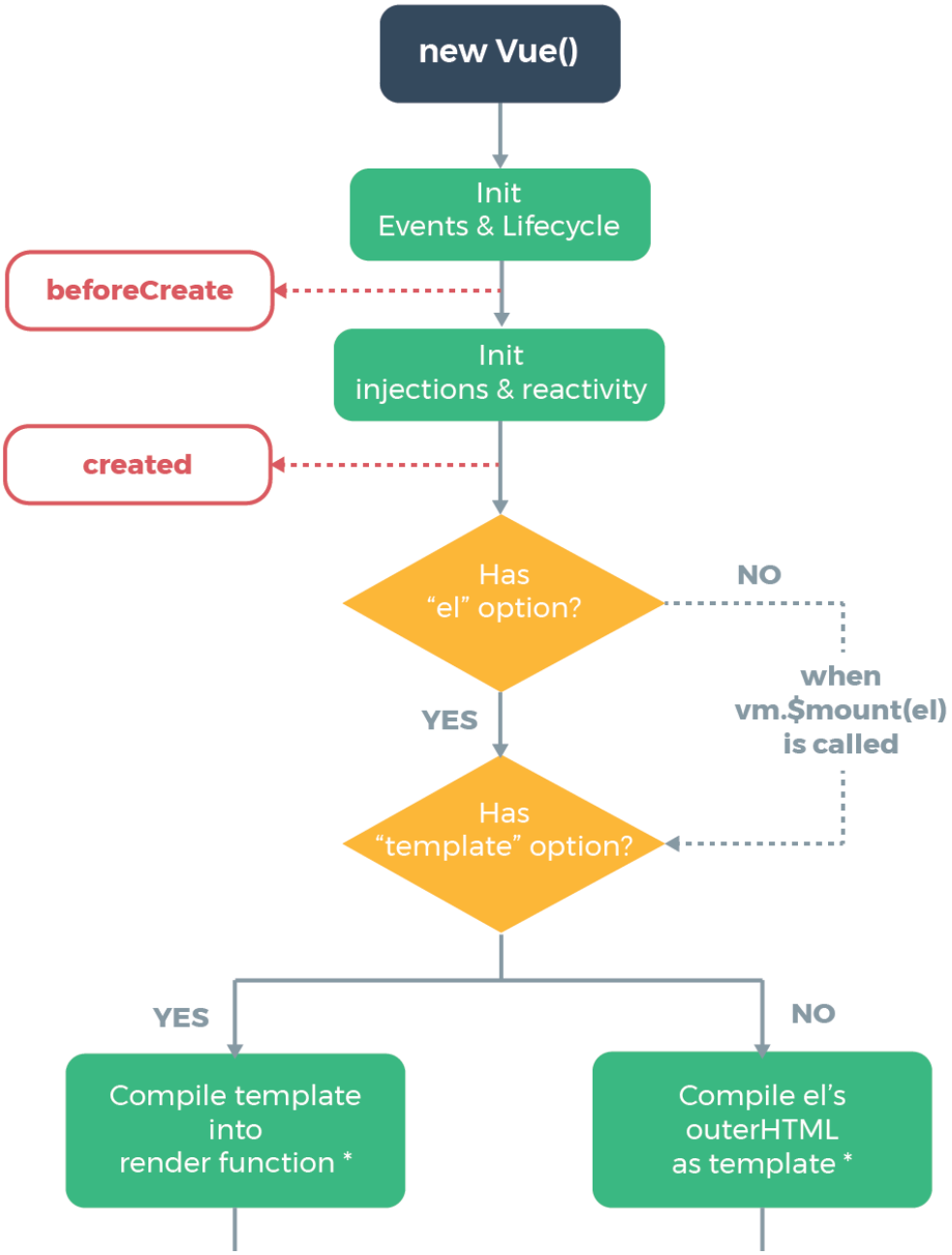
按需载入

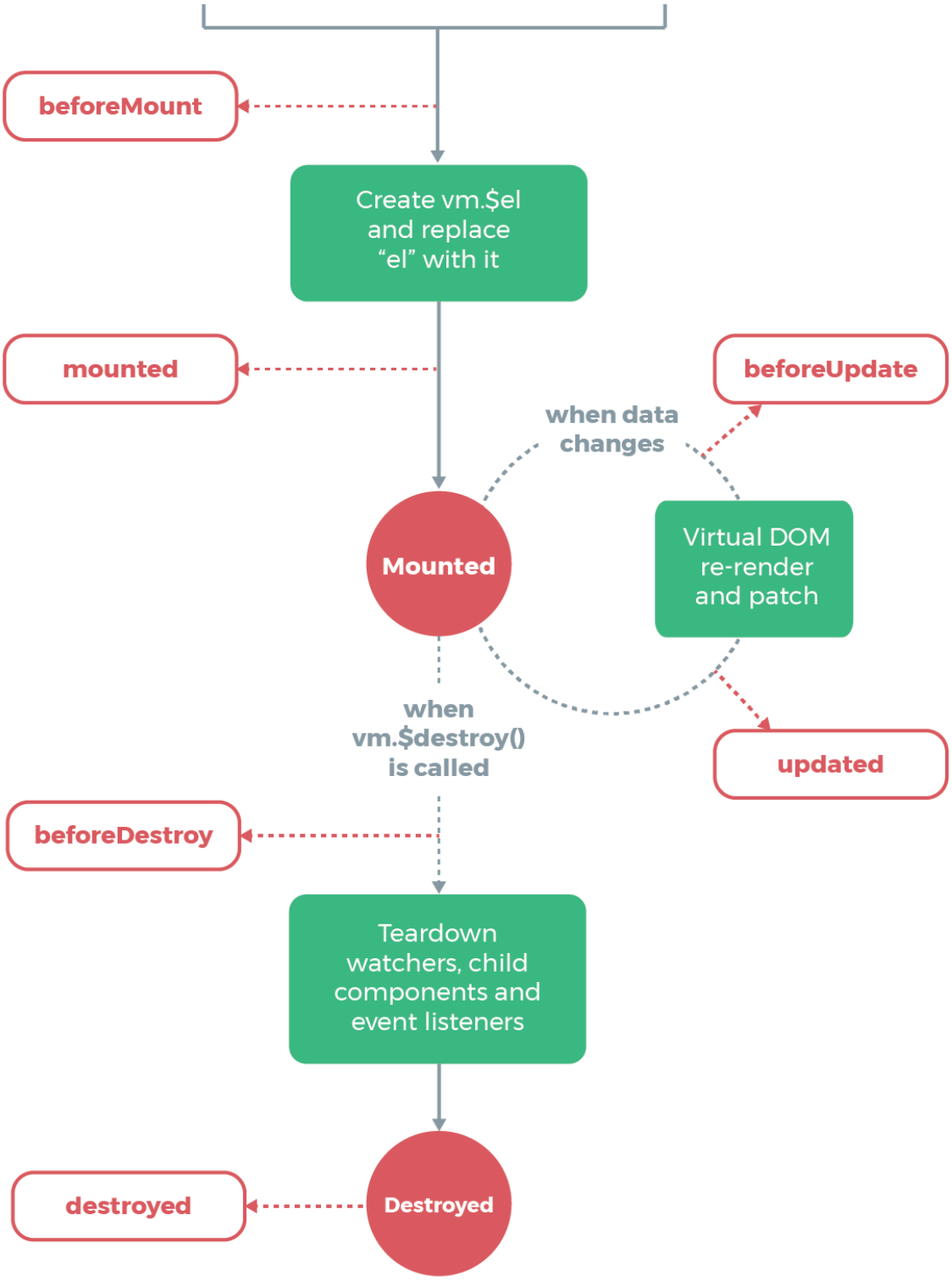
比如我们用了一些Element或者iview这些UI库，那我们在可能在最终发布的时候，我们不希望打包所有的组件，为了解决这个问题，我们可以babel-plugin-import babel-plugin-componnet(Element) 这两个插件去引入组件，这两个插件的区别是 babel-plugin-component是Element基于babel-plugin-import模改的一个插件。这个插件的主要作用是通过ES6 import的语法，通过这个插件可以帮助编译器把import编程require的语法 。如下图

```
import { Button } from 'components'


var button = require('components/lib/button')
require('components/lib/button/style.css')
```

生命周期





* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

生命周期流程：

1. 当我们创建一个Vue实例的时候，会进行初始化事件&生命周期，钩子函数。这时候就抛出beforeCreate的钩子函数(这时候还不能访问页面的data和dom节点)

2. 初始化 注入&校验完毕后 ， 就会抛出一个created函数(这时候可以访问data 以及和computed的一些响应式的数据)

3. 模板的解析和渲染的过程。模板的渲染完毕之后，及出现一个的beforeMount的钩子函数(模板已经编译好，但是没有根据响应式数据进行替换渲染)

4. 当替换渲染完毕之后就抛出mounted钩子函数，当mounted执行完毕之后，代表我们的组件已经初始化完毕，也就是挂在完毕了。

5. 当响应式数据data被修改时，这时候又进入一个新的循环，先抛出一个beforeUpdate钩子函数，这个钩子函数结束了之后，虚拟dom重新渲染，并应用更新(也就是diff patch的过程)。

6. diff patch之后，就抛出updated的钩子函数。

7. 当组件销毁的时候，也就是当调用vm.$destroy()函数时或者通过V-if来切换组件的时候，在组件卸载之前会有一个beforeDestroy的钩子函数

8. 接着解除绑定，销毁子组件以及事件监听器。销毁完毕后，还会抛出一个destoryed钩子函数。

生命周期图解

beforeCreate	最初调用触发，data 和 events 都不能用。可以在这里处理整个系统加载的Loading。
created	已经具有响应式的data，可以发送events。可以在这里去发送请求。
beforeMount	在模板编译后，渲染之前触发。SSR中不可用。 基本用不上这个Hook。
mounted	在渲染之后触发，并可访问组件中的DOM以及 $ref，SSR中不可用。 一般在用于需要在vue中嵌入非vue的组件时，不建议用于发送请求（放在created中）。
beforeUpdate	在数据改变后、模板改变前触发。切勿使用它监听数据变化（使用计算属性和watch监听）。
updated	在数据改变后、模板改变后触发。 常用于重渲染后的打点、性能检测或者触发vue组件中非vue组件的更新。
beforeDestroy	组件卸载前触发，可以在此时清理事件、计时器或者取消订阅操作。
destroyed	卸载完毕后触发，可以做最后的打点或事件触发操作。

动态组件

component

这样我们把逻辑尽量少写在模板里面，放在模板当中不利于我们单元测试。在切换路由或者切换组件的时候，vue有提供一个keep-alive的组件给我们使用。

keep-alive

```
<keep-alive>
  <component v-bind:is="currentComponent"></component>
</keep-alive>
```

这个keep-alive的原理是:缓存组件实例，通过vm.$el获取到先前DOM元素，切换之后，直接插入即可，这样就增加了渲染的效率。

keep-alive提供的props

include	字符串或正则表达式。只有名称匹配的组件会被缓存。
exclude	字符串或正则表达式。任何名称匹配的组件都不会被缓存。
max	数字。最多可以缓存多少组件实例。

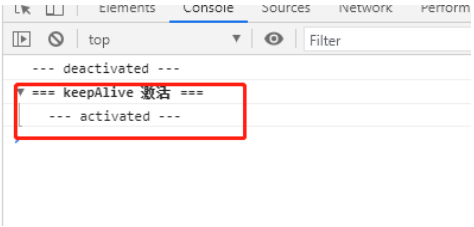
子组件life hook(这个是配合keep-alive使用的)

activated	Keep-alive 内组件加载成功后调用
deactivated	Keep-alive 内组件缓存成功后调用

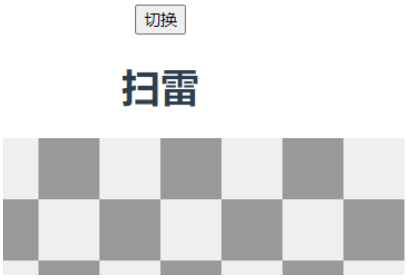
如下示例

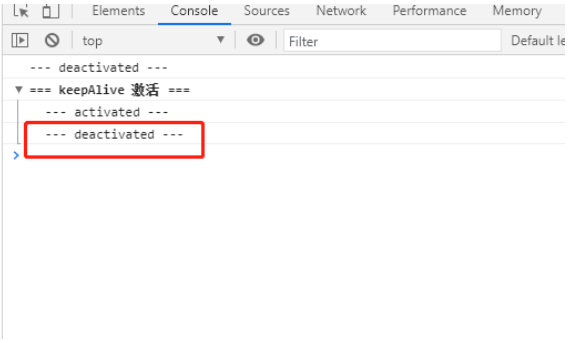
```
<keep-alive>
<component :is="currentComponent"></component>
</keep-alive>
```

activated keep-alive内加载成功后调用



deactivated keep-alive内组件缓存成功后调用


切换
扫雷


Elements
Console
Sources
Network
Performance
Memory
top
Filter
--- deactivated ---
=== keepAlive 激活 ===
--- activated ---
--- deactivated ---