react 核心

课程目标：

P6：

- 知道react大致实现思路，能对比react和js控制原生dom的差异，能口喷一个简化版的react。
- 知道diff算法大致实现思路。
- 对state和props有自己的使用心得，结合受控组件、hoc等特性描述，需要说明各种方案的适用场景。

P6+ ~P7:

- 能说明白为什么要实现 fiber。
- 能说明白为什么要实现 hook。
- 知道react不常用的特性，比如context，portal，errorBoundry。

视频：
0–20 背景
20–100 框架
100–120 源码调试

视频2：
0–15： diff 概念
15–90 diff 实现– mini react
90– schedule 实现

课程大纲与背景：

如何最快速的掌握 `react` ，并且让自己成为一个 `react` 的高手。
几个问题：

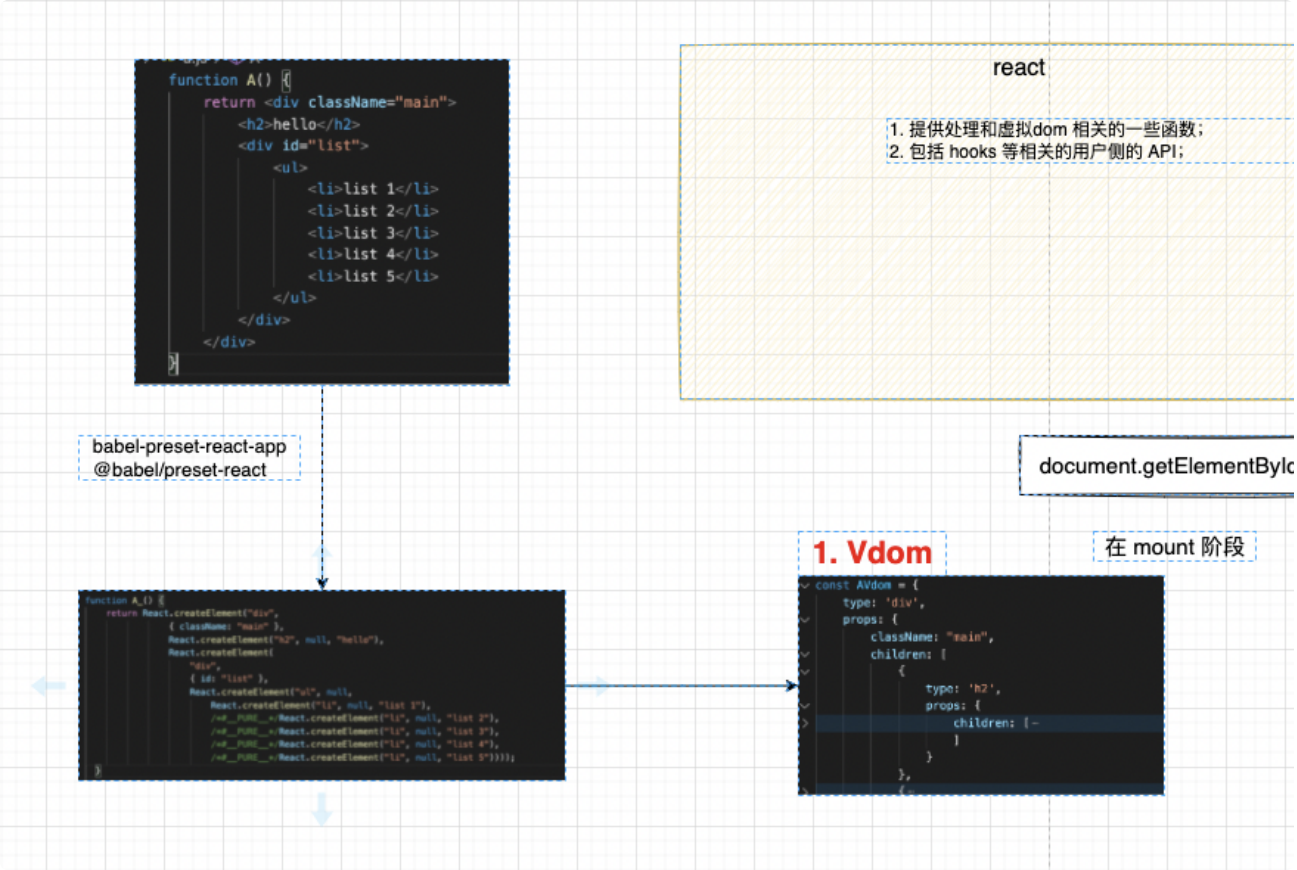
- 为什么要读源码？怎么读源码？需不需要读源码？
  - 跟着一起去调试一下；
  - 读了源码，加分项；
  - 面试官想要招什么样的人？
- 怎么学 `react` ？
  - 第一个官方文档；
  - 第二个了解 `react` / `preact` 这样的库的一个主要是实现思路；（vdom , diff ）（2节，调度）
  - 熟悉源码框架、实现流程、机制；（1节）

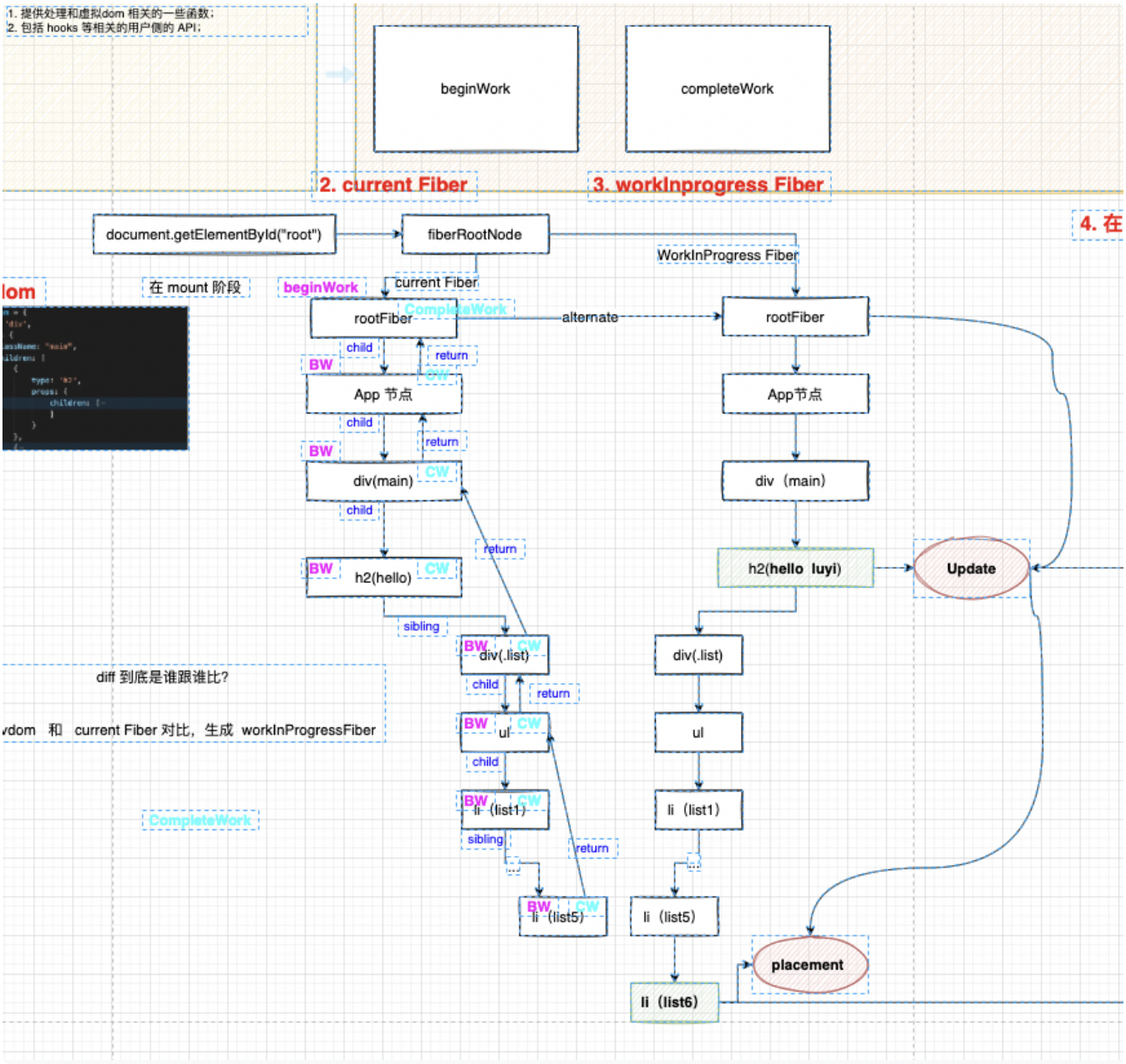
课程内容

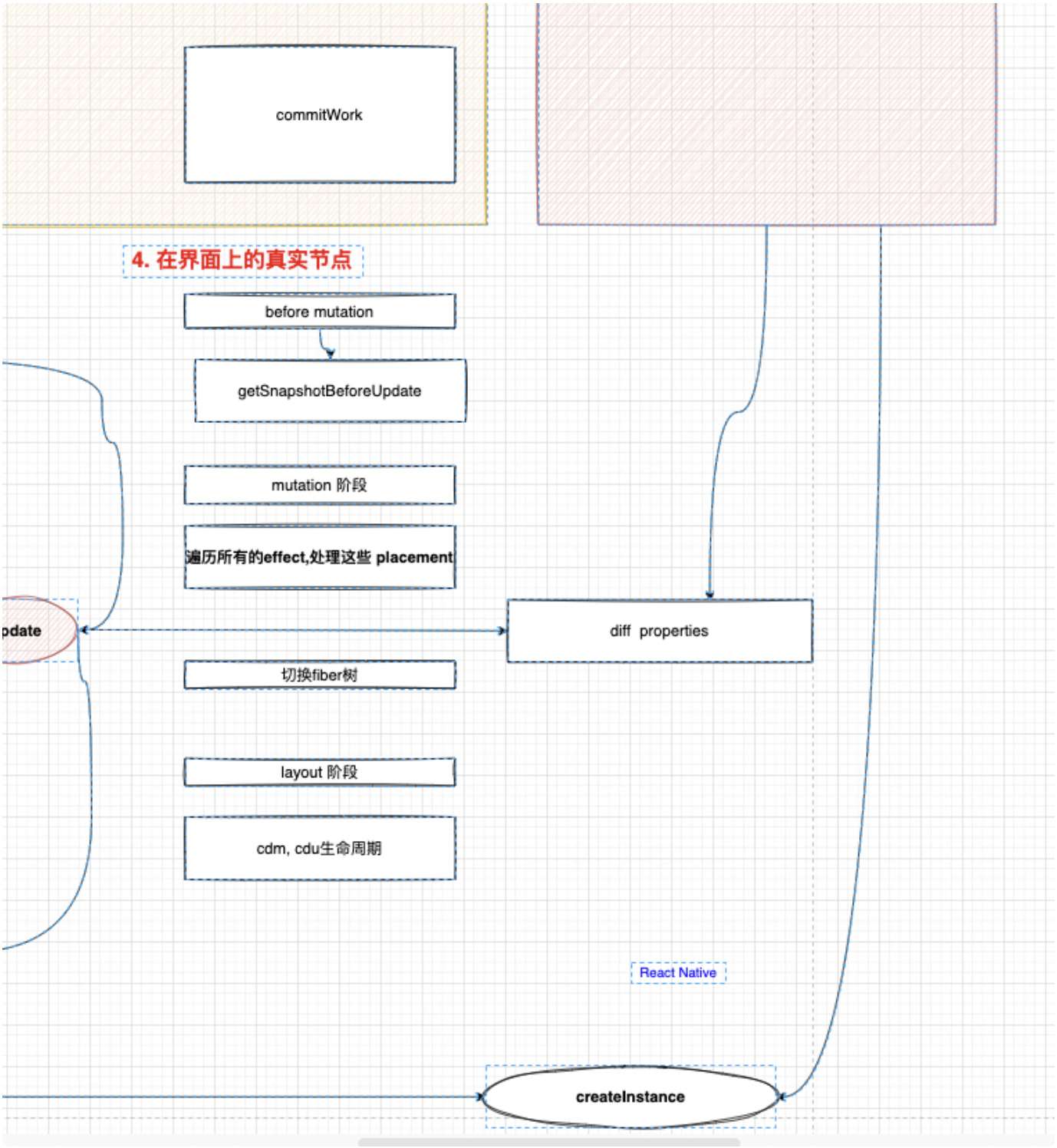
虚拟dom

不管是 react 还是 vue ， 都有 虚拟 dom？虚拟 dom 一定快吗？
为什么要用虚拟 dom ？

- 某种程度上，保证，性能的一个下限；
- 中间层，vdom –> fiber对象 –> 真实dom




1. 提供处理和虚拟dom 相关的一些函数；
2. 包括 hooks 等相关的用户侧的 API；
beginWork
completeWork
2. current Fiber
3. workInprogress Fiber
4. 在
document.getElementById("root")
fiberRootNode
WorkInProgress Fiber
在 mount 阶段
beginWork
current Fiber
rootFiber
CompleteWork
alternate
rootFiber
child
return
BW
CW
App 节点
App节点
div(main)
div（main）
h2(hello)
h2(hello luyi)
Update
sibling
div(.list)
div(.list)
ul
ul
li（list1）
li（list5）
li（list6）
placement
diff 到底是谁跟谁比?
dom 和 current Fiber 对比，生成 workInProgressFiber
CompleteWork



源码跟踪

- `render`
- `legacyRenderSubtreeIntoContainer`
  - 创建 `fiberroot` 的根节点
- `updateContainer`

- scheduleUpdateOnFiber
- `ensureRootIsScheduled`
- `performSyncWorkOnRoot`
- `renderRootSync`
- `workLoopSync`
- `performUnitOfWork`
- `beginWork`
- `completeWork`

diff 算法

- react 和 vue 的 diff 算法有什么异同？
- react 为什么不去优化 diff 算法?
- 传统 diff O(n^3)，React Diff O(n)？怎么来的？还可以优化吗？

- 最好时间复杂度、最坏时间复杂度、平均时间复杂度、均摊时间复杂度；
- 《数据结构和算法之美》极客时间

diff 算法并不是近年才有的，早在多年以前就已经有人在研究 diff 算法了，最早复杂度基本是 O(m^3n^3)然后优化了 30 多年，终于在 2011 年把复杂度降低到 O(n^3) 这里的 n 指的是节点总数
所以 1000 个节点，要进行 10亿次操作；
而今天，站在巨人的肩膀上，我们将探究的 diff 算法主要指 react 横空出世之后提出的近代同层比较的 diff 算法，因为是同层嘛，复杂度就到了 O(n)；

为什么需要diff，为什么现在对 diff 好像提及的不是那么多?

本质上就是为了性能，性能，性能

为什么传统的是 O(n^3)

在计算机中，比较两棵树的区别；-- 借鉴字符串的编辑距离，莱温斯坦最短距离算法
比如： 字符串"hello" -> "hallo"；

react 是如何将 diff 算法的复杂度降下来的?

其实就是在算法复杂度、虚拟 dom 渲染机制、性能中找了一个平衡，react 采用了启发式的算法，做了如下最优假设：
a. 如果节点类型相同，那么以该节点为根节点的 tree 结构，大概率是相同的，所以如果类型不同，可以直接「删除」原节点，「插入」新节点
b. 跨层级移动子 tree 结构的情况比较少见，或者可以培养用户使用习惯来规避这种情况，遇到这种情况同样是采用先「删除」再「插入」的方式，这样就避免了跨层级移动

c. 同一层级的子元素，可以通过 key 来缓存实例，然后根据算法采取「插入」「删除」「移动」的操作，尽量复用，减少性能开销

d. 完全相同的节点，其虚拟 dom 也是完全一致的；

调度

`concurrent` 模式 / 18里面 -- 调度。

`scheduler` -- 有一个任务，耗时很长，filter。

--> 把任务，放进一个队列，然后开始以一种节奏进行执行。

---- ---- ----

`requestIdleCallback` ->

- 兼容性；
- 50ms 渲染问题；

chrome 60hz 每16.666ms 执行一次事件循环。

|--- task queue ---|--- micro task ---|--- raf ---|--- render ---|--- requestIdleCallback ---|

为什么没有用 `generator`

为什么没有用 `setTimeout` -- 4-5ms

// 1. useTransition API

// startTransition 👉 https://github.com/reactwg/react-18/discussions/41

// https://github.com/reactwg/react-18/discussions/65

// 2. Scheduler / why not generator ? why web-worker ?

// https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Generator

// [why not use generator] https://github.com/facebook/react/issues/7942#issuecomment-254987818

// why not setTimeout? why not raf?

// https://developer.mozilla.org/zh-CN/docs/Web/API/MessageChannel

// [event-loop] https://www.youtube.com/watch?v=u1kqx6AenYw&t=853s
// https://www.imaginea.com/the-javascript-event-loop-micro-tasks-and-macro-tasks/

// -----task queue -----| ----- micro-tasks ------| -------render --------| --------macro-tasks--------|
// https://image-static.segmentfault.com/165/372/1653721873-5adb68e2247cf

// inputing
// https://github.com/WICG/is-input-pending
// [idle] https://developer.mozilla.org/zh-CN/docs/Web/API/Window/requestIdleCallback

思考题：

react 和 Vue 的 diff 算法有什么区别?

相同点：
Vue和react的diff算法，都是不进行跨层级比较，只做同级比较。

不同点：
1.Vue进行diff时，调用patch打补丁函数，一边比较一边给真实的DOM打补丁
2.Vue对比节点，当节点元素类型相同，但是className不同时，认为是不同类型的元素，删除重新创建，而react则认为是同类型节点，进行修改操作
3.① Vue的列表比对，采用从两端到中间的方式，旧集合和新集合两端各存在两个指针，两两进行比较，如果匹配上了就按照新集合去调整旧集合，每次对比结束后，指针向队列中间移动；
②而react则是从左往右依次对比，利用元素的index和标识lastIndex进行比较，如果满足index < lastIndex就移动元素，删除和添加则各自按照规则调整；
③当一个集合把最后一个节点移动到最前面，react会把前面的节点依次向后移动，而Vue只会把最后一个节点放在最前面，这样的操作来看，Vue的diff性能是高于react的

目前的 diff 是什么？真的是 O(n) 吗?

准确的来说，React的 diff 的最好时间复杂度是 O(n)， 最差的话，是 O(mn)；

diff 比较的是什么?

比较的是 current fiber 和 vdom，比较之后生成 workInprogress Fiber

为什么要有 key ?

在比较时，会以 key 和 type 是否相同进行比较，如果相同，则直接复制。

react 为什么要用 fiber ?

stack reconciler –> fiber reconciler
在V16版本之前 协调机制 是 Stack reconciler， V16版本发布Fiber 架构后是 Fiber reconciler。
在setState后，react会立即开始reconciliation过程，从父节点（Virtual DOM）开始递归遍历，以找出不同。将所有的Virtual DOM遍历完成后，reconciler才能给出当前需要修改真实DOM的信息，并传递给renderer，进行渲染，然后屏幕上才会显示此次更新内容。

对于特别庞大的DOM树来说，reconciliation过程会很长(x00ms)，在这期间，主线程是被js占用的，因此任何交互、布局、渲染都会停止，给用户的感觉就是页面被卡住了。

在这里我们想解决这个问题的话,来引入一个概念,就是任务可中断,以及任务优先级,也就是说我们的reconciliation的过程中会生成一些任务和子任务,用户的操作的任务优先级是要高于reconciliation产生的任务的,也就是说用户操作的任务是可以打断reconciliation中产生得任务的,它会优先执行.

react 为什么要用 hook？

https://react.docschina.org/docs/hooks-intro.html#motivation

hook 是怎么玩的？

我的diff，是不是 current fiber 和 vdom 比较?

JavaScript | 复制代码

```
function App() {
  const [state, setState] = useState(0);
  const divRef = useRef(null);
  return <div>XXX</div>
}
// App 是不是也是一个 Fiber
// 在beginWork的时候 App(); -> vdom

AppFiber.memoizedState -> hook.next -> hook.next -> hook
                          hook.memoizedState - 保存了 hook 对应的属性

class Main extends Component {
  render(){
    return <div>xxx</div>
  }
}
```

最后再说一下，关于项目上的亮点的事情。

什么是亮点?

- **前提1：一定是有思考的；熵减。**
  - 使用 Vue 、Vuex、Vue-router、elementUI 和设计师对接、和后端接口对接，实现XXX功能；
  - emotion，css module
- **前提2：一定是有创造性的；**
  - **a —> b ，提升了什么?**
- **前提3：一定是有复杂度的；**

XXX项目：
A:负责公司的前端工作，使用 webpack 打包代码并发布上线，使用 webpack 对整体性能优化，用 happyPack 优化打包速度；

B: 完善公司内容构建、部署流程一体化，根据业务XX特点，通过热切换部署实现。

B: 建设内部云构建体系，产出通用命令行指令工具，将发布、环境切换、快速回滚能力平台化，保证线上高稳定性，同时管控研发流程，定期产出研发报告。