对比

	odoo 原生	odoojs 独立部署	odoojs 组件式部署
请求地址	http://ip:8069/web	http://ip/xxx	http://ip:8069/xxx
web 服务	python.werkzeug = Web框架的底层库	nginx = web服务器	python.werkzeug
html 页面	jquery + 自己开的钩子 + xml文件 生成 页面。 html 页面 放在 python.werkzeug里 提供给 前端浏览器	提前 编译好 html 文件。 html文件 放在 nginx 的里。 由 nginx 负责 提供给 前端浏览器	提前 编译好 html 文件。 放在 python.werkzeug里 提供给 前端浏览器
请求	odoo的static里的 js 有请求相关代码 无跨域问题	axios 有跨域问题。通过 nginx 解决	两者 皆可 无跨域问题
html 架构	jquery 直接 操控 html 节点 odoo 自己做的的 钩子 用 xml文件 定义tree/form/kanban view	vue 架构 + antd 组件库 需要造钩子odoorpc+odooui 需要做各种组件 实现 tree/form/kanban	以下同 独立部署
钩子	odoo 官方开发的钩子 1. 处理 field (m2o, o2m), 2. 编辑页面用的 onchange	钩子 odoorpc 解决: 1. 处理 field (m2o, o2m), 2. 编辑页面用的 onchange	
页面	odoo官方开发的钩子 处理 xml 文件 转为 tree / form / kanban 页面 其中 用到 qweb 一一一 xml 中嵌入 python 语句 xml 中嵌入 css 一一 钩子中 有处理 python语句 的代码 一一 钩子中 有 处理 qweb 的代码 qweb 可以理解为 html 模版 + 服务端渲染的混合物	依然可以定义 类似 xml 功能的文件 这里先命名为 组件 一一一一 使用 vue 本身的 组件编程的方法 将组件组合为 tree / form / kanban 页面 一一一一 坚决不会嵌入 python 语法的代码 直接用 js 代码 实现即可 一一一 组件中 使用 css 这是理所当然的	
Tree View	用 xml 文件 定义 tree view 有钩子方法 将tree view 的xml 转为 html 一一 不同的模型 定义不同的 tree view tree view 可继承，以扩展 一一 xml 中嵌入 field 处理数据 field 绑定 数据源	用组件定义 tree view 定义基础的 TreeView 组件 处理公共的 TreeView 属性 一一 不同的模型 定义不同的组件 组件是可以继承的，以扩展 一一 组件中 用插槽slot 嵌入 field field 绑定数据源	
Form View	基本上雷同 tree view Form View的 xml中, 嵌入 各种 html 标签 被钩子直接 渲染	基本上雷同 tree view Form View的 组件中 嵌入 各种 html 标签 这是 很自然的事情，毫不违和	
Kanban / QWeb	这两个类似。一起说。 odoo创造一堆约定。 用 xml 文件 来描述 html 如何组织 包括 html 标签 和 数据 。混在一起。 qweb 有 if for 等语法 不同的 Kanban 定义。生成不同的页面	要做什么功能 就做什么样的组件即可 vue的组件里有 if for	
Menu	menu 是 纯 json 数据 直接读取后 前端渲染 menu 中的图片 用的是image。	menu 的组织。自己看着办。 当然也可以 读取自 服务器，再渲染。 以目前的经验看 menu 中的图片用 icon 更普遍	

	odoo 原生	odoojs 独立部署	odoojs 组件式部署
Action	本质是 定义了 model，domain，context，search 以及 动作结果： 如: 打开窗口的方式(弹窗还是主窗口)	Action 完全可以直接读取自服务端 根据返回的结果，做相应的页面处理	
再解释下VIEW	view 用xml 定义 view 定义中嵌入了过多的， 便于 jquery 和 bootstrap处理的内容 假定 将 view 定义拆分为: 纯业务定义部分 和 ui 相关部分 纯业务定义部分 与 前端实现逻辑无关 ui 相关部分，可以随前端开发方式而变	view中的纯业务部分，单独定义。 该部分与 odoo 原生定义是一样的。 view定义 中的 ui相关部分 让前端开发自己随心所欲吧	
VIEW的额外工作	现有的 view 直接保留不动	从服务端读取 view。 抽取其中的业务逻辑定义部分。 舍弃 其中的 ui 定义部分。 —— 每一个view 分别设计组件 —— 具体某个模型的view页面， 由 组件 + 业务逻辑定义 生成	
view中的特殊标签 group/notebook	form view 中的 group 是最常见的 odoo 原生页面将 group 渲染为 table odoo 原生页面将notebook 渲染为 tabs	处理 group/Notebook 。没必要了。 form view 中的 field 想怎么布局 属于 ui 范畴	
view的继承	模块安装的安装与否。view的内容不一样。最后渲染的结果不一样	view的业务逻辑部分直接服务端读取 获取的结果，是直接可用的。 ui部分的组件设计，可以实现继承机制 不同的模块，分别设计组件的细节部分内容。 渲染时，先读取模块是否安装。 再组织 组件最终的渲染结果。	
前端开发工作的 学习成本	需要掌握 odoo架构 需要 了解 model / field 精通 view 精通 qweb 的语法 精通 odoo做的 前端 各种 class	无需掌握 odoo架构 需要 了解 model / field 改变下思维。 发请求不再是直接用 axios 而是 用 封装有 axios 的 odoorpc 掌握 vue 或 react 前端架构 掌握 antd / elementui 等 ui 库 掌握 css	
前端工程师 市场储备	odoo 太小众了。知道的人不多	掌握 vue/react, antd/elementui, css 知识的 求职人员，大有人在。	
甲方的参与情况	让甲方的爷，做odoo 开发， 掌握view，可能性为 0	让甲方的爷。做点 前端页面开发。 他们觉得成就很高	
乙方的情况	无论怎么改，都是买的别人家的东西	可以独立开发自己的产品 后端开发的高效 前端开发的自由	
国家层面的要求	去IOE，国产化，信创的要求。 在 odoo原生上，无论怎么改。 都无法做到 满足 国产化的要求。	odoo 藏在了 后端 悄悄干活。 有实际价值的只有 model 开发了。 前端已经 完全国产化。 若要服务端 也国产化。重写 model 相关代码，是在可控范围内的。	