

- [指南](#)
- [论坛](#)
- [博客](#)
- [API文档](#)
- [构建](#)

🍴 [Fork Us!](#) 🍴 [Fork Ember!](#)

1. [入门指南](#)
 1. [介绍](#)
 2. [应用规划](#)
 3. [创建静态页面](#)
 4. [获取Ember.js和相应依赖](#)
 5. [添加第一个路由与模板](#)
 6. [建立模型](#)
 7. [使用静态数据](#)
 8. [显示模型数据](#)
 9. [显示模型的完成状态](#)
 10. [创建新的模型实例](#)
 11. [标记模型为完成或未完成](#)
 12. [显示未完成待办事项的数量](#)
 13. [切换显示和编辑状态](#)
 14. [接受修改](#)
 15. [删除模型](#)
 16. [添加子路由](#)
 17. [显示未完成待办事项](#)
 18. [显示已完成待办事项](#)
 19. [显示所有待办事项](#)
 20. [添加移除所有已完成待办事项的按钮](#)
 21. [提示所有待办事项已完成](#)
 22. [切换已完成和未完成待办事项](#)
 23. [更换FixtureAdapter](#)
2. [获取Ember](#)
 1. [获取Ember](#)
3. [概念](#)
 1. [核心概念](#)
 2. [命名惯例](#)
4. [对象模型](#)
 1. [类与实例](#)
 2. [计算属性](#)
 3. [计算属性和带@each的集合数据](#)
 4. [观察器](#)
 5. [绑定](#)
 6. [重新打开类和实例](#)
 7. [绑定，观察器，计算属性：如何选择？](#)
5. [应用](#)
 1. [介绍](#)
6. [模板](#)
 1. [应用模板](#)
 2. [Handlebars基础](#)
 3. [条件表达式](#)
 4. [显示项目列表](#)

5. [切换作用域](#)
6. [绑定元素属性](#)
7. [绑定元素类名称](#)
8. [链接](#)
9. [操作](#)
10. [输入助手](#)
11. [开发助手](#)
12. [用助手来渲染](#)
13. [编写助手方法](#)
7. [路由](#)
 1. [介绍](#)
 2. [定义路由](#)
 3. [生成的对象](#)
 4. [指定路由的模型](#)
 5. [设置控制器](#)
 6. [渲染模板](#)
 7. [重定向](#)
 8. [指定地址API](#)
 9. [查询参数](#)
 10. [异步路由](#)
 11. [加载中/错误子状态](#)
 12. [阻止和重试过渡](#)
8. [组件](#)
 1. [介绍](#)
 2. [定义组件](#)
 3. [传递属性](#)
 4. [包裹内容](#)
 5. [自定义组件元素](#)
 6. [使用Action处理用户交互](#)
 7. [从组件发送操作给应用](#)
9. [控制器](#)
 1. [介绍](#)
 2. [代表单一模型](#)
 3. [代表多模型](#)
 4. [管理控制器间的依赖](#)
10. [模型](#)
 1. [介绍](#)
 2. [定义模型](#)
 3. [创建和删除记录](#)
 4. [将记录推入仓库](#)
 5. [持久化记录](#)
 6. [查询记录](#)
 7. [使用记录](#)
 8. [使用Fixture](#)
 9. [连接HTTP服务器](#)
 10. [处理元数据](#)
 11. [自定义适配器](#)
 12. [常见问题](#)
11. [视图](#)
 1. [介绍](#)
 2. [定义视图](#)
 3. [处理事件](#)
 4. [在模板中插入视图](#)
 5. [为视图添加布局](#)

- 6. [自定义视图元素](#)
- 7. [内置视图](#)
- 8. [手动管理视图层级](#)
- 12. [枚举](#)
 - 1. [介绍](#)
- 13. [测试](#)
 - 1. [介绍](#)
 - 2. [集成测试](#)
 - 3. [测试助手](#)
 - 4. [测试用户交互](#)
 - 5. [单元测试](#)
 - 6. [单元测试基础](#)
 - 7. [测试组件](#)
 - 8. [测试控制器](#)
 - 9. [测试路由](#)
 - 10. [测试模型](#)
 - 11. [自动化测试](#)
- 14. [配置Ember.js](#)
 - 1. [禁用基本类型扩展](#)
 - 2. [嵌入式应用](#)
 - 3. [特性标识](#)
- 15. [Cookbook](#)
 - 1. [简介](#)
 - 2. [用户界面与交互](#)
 - 3. [事件处理和数据绑定](#)
 - 4. [助手与组件](#)
 - 5. [使用对象](#)
- 16. [理解Ember.js](#)
 - 1. [视图层](#)
 - 2. [管理异步](#)
 - 3. [模板自动更新](#)
 - 4. [调试](#)
 - 5. [运行循环](#)

[返回顶部](#)

核心概念 [编辑页面](#)

英文原文: <http://ember.js.com/guides/concepts/core-concepts/>

要开始学习Ember.js, 首先要了解一些核心概念。

Ember.js的设计目标是能帮助广大开发者构建能与本地应用相媲美的大型Web应用。要实现这个目标需要新的工具和新的概念。我们花了很大的功夫从Cocoa、Smalltalk等本地应用框架引入了其优秀的理念。

然而, 记住Web的特殊性非常重要。很多人认为一个应用是Web应用是因为其使用了像HTML、CSS和Javascript这些技术。实际上, 这只是实现的细节问题。

相反, Web应用是通过能收藏和分享链接来凸显它的作用的。URL是Web应用的一个最核心的特性, 正是URL使得Web应用有了卓越的可共享性和可协作性。现今, 很多Javascript框架都事后才考虑URL, 没有考虑这个让Web成功的主要因素。

Ember.js将本地GUI框架中的工具和概念与使得Web应用如此强大的URL嫁接在一起。

概念

模板

模板，用Handlebars模板语言来编写，它描述了一个应用程序的用户接口。每个模板背后都有一个模型，当模型发生改变时，模板将自动进行更新。

此外，相对于纯HTML，模板还提供了：

- 表达式，例如 `{{firstName}}`，它从模板对应的模型获取信息并将信息添加到HTML中。
- 出口（Outlets），它是其他模板的占位符。当用户使用应用时，不同的模板会通过路由插入到出口中。你可以使用 `{{outlet}}` 助手将出口放到模板中去。
- 组件，自定义的HTML元素，可以用来清理重复的模板或创建可重用的控件。

路由器

路由器将URL转换为一系列内嵌的有模型数据支撑的模板。当显示给用户的模板和模型发生改变时，Ember自动更新浏览器地址栏中的URL。

这意味着用户可以在任意点分享应用的URL。当某个用户点击了这个链接时，将看到与分享链接的用户看到的相同内容。

组件

组件是一个自定义的HTML标签，其行为用Javascript来实现，而显示使用Handlebars模板来描述。组件可以用来定义可重用的控件来简化应用的模板。

模型

模型是一个存储持久化状态的对象。它是应用将操作的数据，也是用来返回值给用户的数据。这些对象通常从服务器端加载，并当其在客户端发生改变后又保存到服务器端。

路由

路由是负责管理应用程序状态的对象。

控制器

控制器是存放应用状态的对象。模板除模型之外还可以有一个控制器与之对应，使其可以从这两者获取属性。

以上这些是在开发Ember.js应用时需要了解的核心概念。Ember.js设计为可以弹性的处理复杂的问题，因此需要为应用增加新功能、新特性时只需要改变很小的部分。

现在你已经理解了这些对象各自的角色，可以开始深入到Ember.js的世界中，进一步了解这些部分如何工作的细节。

[← 获取Ember: 获取Ember 命名惯例 →](#)

© 2014 Ember.js.CN

Design by [HeroPixel](#)

声明：本站的文章可以随意在网上转载，但必须注明原文出处！

