

Linux 文件系统和 GFS2

LINUX VIRTUAL FILE SWITCH
GLOBAL FILE SYSTEM 2

龚溪东 编著
湖南麒麟

长沙 • CHANGSHA

序

本书献给所有热爱 Linux 内核的读者。我们学习它而获得的最大的乐趣和收获来源于满足了我们的好奇心。最早接触到 Linux 操作系统是在 2005 年暑假在大学校园,在一次中科红旗的推广活动中,接着开始使用 RedHat 9。印象最为深刻的是在 RedHat 9 上安装 Oracle 7,整整失败了 26 次,在第 27 次才安装成功。可见当时的折腾劲儿。之后所有的系统都安装成了 Linux,在 Linux 上做专业作业,上网,看视频,看课件,写应用。后来在研究生复试期间,面试老师问我会不会 Linux 内核,我才幡然醒悟,内核这一块是我完全不了解的,对我来说它还只是停留在 Linux 的传说中。虽然有所感触,但后来进入实验室之后,学习的分布式系统这一大领域,而 Linux 内核当时是不那么迫切需要去开发的。包括后来毕业之后参加的几分工作,都不需要面向 Linux 内核进行开发。直到现在的公司现在的岗位,现在的公司有一条 Linux 发行版的生产线,也有基于这一发行版的加密存储生态圈,定制和优化 Linux 内核态的存储软件栈必不可免,这才开始系统学习心仪已久的 Linux 内核。

由于之前接触过不少开源软件,也仔细研读过几个著名的软件代码,尤其是刚开始的 Lighttpd¹到 PostgreSQL²和后来的 Redis³,简直就是工程代码的典范,可读性上佳,阅读这样的代码对程序员来说可谓不可多得的享受。当时太年轻太天真了,以为这就是开源代码应该有的样子。直到看到 Linux 内核,对开源代码的刻板印象才被推翻,想直接通过 Linux 内核的源代码学习和掌握操作系统的原理,机制和策略,简直就是图样图森破。于是我改变策略,先老老实实从驱动开发开始,去实践 Linux 内核的 API,然后再切入 Linux IO Stack⁴。在 DEBUG 的过程中,通过上网、翻书、读代码、查自带文档的方式解决一个又一个问题,上班编写文档,下班看书读源码,从刚开始只是简单地实现功能,通过 3 年来的不断锻炼,到现在有了定制和优化 Linux 内核软件栈的信心。

市面上关于 Linux 文件系统的书籍汗牛充栋,其中不乏经典。《Linux 内核设计与实现》,《Linux 设备驱动程序》,《深入理解 Linux 内核》,《深入 Linux 内核架构》等都是豆瓣⁵上评分超 8.5 的经典。《Linux 内核设计与实现》我通读过两遍,《Linux 设备驱动程序》被我画满了记号,《深入理解 Linux 内核》和《深入 Linux 内核架构》虽没有完整地过一遍,但其中工作相关的章节拜读多次。不得不说,中文翻译还是存在问题的,鉴于原本实在经典,译本确实值得批评:阅读前三本,尤其是第一遍的时候,那酸爽简直呲牙,存在机器翻译

¹<https://www.lighttpd.net/>

²<https://www.postgresql.org/>

³<http://www.redis.cn/>, <http://www.redis.io/>

⁴https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram

⁵<https://book.douban.com>

的嫌疑;第四本的问题在于用语习惯有点水土不服,甚至把不应该翻译的 C 语言结构体翻译成了中文,存在不少印刷错误;如果外文功底非常好,还是建议阅读原版,。

当然,本书并不打算讲 Linux 内核的方方面面,事无巨细,否则难免会有狗尾续貂之嫌。本书仅仅围绕文件系统这一块。鉴于在中文世界里市面上专门讲解 Linux 下文件系统的书籍一直缺席,我就想是不是写一本以填补空白。写书本身是一件很严肃的事情,不比在工作中按照公司规范写的技术文档,也不同于业余撰写的博客。至少于我而言,有如下方面的好处:

1. 锻炼下自己的脸皮,所谓内向者的优势;
2. 把新近所学的 \LaTeX 赶出来练练兵;
3. 给自己前一段时间的工作做一个总结;
4. 温故而知新,解决 GFS2 产生的各种奇葩问题。

由于 Linux 文件系统在内核中至关重要,但它那集市般的开发模式,拿 C 语言强怼 OOP,导致这一子系统代码极度精炼又错综复杂,与企业级开发的工程代码相比,其可读性非常之差。而我精力和能力有限,难免有理解上的偏差。对于书中存在的不足与疏漏,欢迎读者将问题反馈到hillgreen012@hotmail.com。如果读者能在阅读本书的过程中,快速建立起“比较完整”和“大体正确”的认识,那将是我最大的欣慰。

龚溪东

湖南麒麟,长沙总部

前言

目 录

PART I

第一部分

Linux Virtual File Switch

核心知识

Angular 是一个用 HTML 和 TypeScript 构建客户端应用的平台与框架。Angular 本身使用 TypeScript 写成的。它将核心功能和可选功能作为一组 TypeScript 库进行实现,你可以把它们导入你的应用中。

1.1 架构概览

Angular 的基本构造块是 NgModule,它为组件提供了编译的上下文环境。NgModule 会把相关的代码收集到一些功能集中。Angular 应用就是由一组 NgModule 定义出的。应用至少会有一个用于引导应用的根模块,通常还会有很多特性模块。

- 组件定义**视图**。视图是一组可见的屏幕元素,Angular 可以根据你的程序逻辑和数据来选择和修改它们。每个应用都至少有一个根组件。
- 组件使用**服务**。服务会提供那些与视图不直接相关的功能。服务提供商可以作为依赖被注入到组件中,这能让你的代码更加模块化、可复用,而且高效。

强行在这里插入一个公式:

$$\lim_{x \rightarrow 0} \frac{e^x - 1}{2x} \stackrel{\left[\frac{0}{0}\right]}{=} \lim_{x \rightarrow 0} \frac{e^x}{2} = \frac{1}{2} \quad (1.1)$$

1.1.1 模块

Angular 定义了 NgModule,它和 JavaScript (ES2015) 的模块不同而且有一定的互补性。NgModule 为一个组件集声明了编译的上下文环境,它专注于某个应用领域、某个工作流或一组紧密相关的能力。NgModule 可以将其组件和一组相关代码(如服务)关联起来,形成功能单元。

每个 Angular 应用都有一个根模块,通常命名为 AppModule。根模块提供了用来启动应用的引导机制。一个应用通常会包含很多功能模块。

像 JavaScript 模块一样,NgModule 也可以从其它 NgModule 中导入功能,并允许导出它们自己的功能供其它 NgModule 使用。比如,要在你的应用中使用路由器 (Router) 服务,就要导入 Router 这个 NgModule。

把你的代码组织成一些清晰的功能模块,可以帮助管理复杂应用的开发工作并实现可复用性设计。另外,这项技术还能让你获得惰性加载 (也就是按需加载模块) 的优点,以尽可能减小启动时需要加载的代码体积。

然后,我在这里强行引用上述公式??。

1.1.2 模板、指令和数据绑定

模板会把 HTML 和 Angular 的标记 (markup) 组合起来,这些标记可以在 HTML 元素显示出来之前修改它们。模板中的指令会提供程序逻辑,而绑定标记会把你应用中的数据和 DOM 连接在一起。

- **事件绑定**让你的应用可以通过更新应用的数据来响应目标环境下的用户输入。
- **属性绑定**让你将从应用数据中计算出来的值插入到 HTML 中。

在视图显示出来之前,Angular 会先根据你的应用数据和逻辑来运行模板中的指令并解析绑定表达式,以修改 HTML 元素和 DOM。Angular 支持**双向数据绑定**,这意味着 DOM 中发生的变化 (比如用户的选择) 同样可以反映回你的程序数据中。

在视图显示出来之前,Angular 会先根据你的应用数据和逻辑来运行模板中的指令并解析绑定表达式,以修改 HTML 元素和 DOM。Angular 支持**双向数据绑定**,这意味着 DOM 中发生的变化 (比如用户的选择) 同样可以反映回你的程序数据中。

在视图显示出来之前,Angular 会先根据你的应用数据和逻辑来运行模板中的指令并解析绑定表达式,以修改 HTML 元素和 DOM。Angular 支持**双向数据绑定**,这意味着 DOM 中发生的变化 (比如用户的选择) 同样可以反映回你的程序数据中。

在视图显示出来之前,Angular 会先根据你的应用数据和逻辑来运行模板中的指令并解析绑定表达式,以修改 HTML 元素和 DOM。Angular 支持**双向数据绑定**,这意味着 DOM 中发生的变化 (比如用户的选择) 同样可以反映回你的程序数据中。

在视图显示出来之前,Angular 会先根据你的应用数据和逻辑来运行模板中的指令并解析绑定表达式,以修改 HTML 元素和 DOM。Angular 支持**双向数据绑定**,这意味着 DOM 中发生的变化 (比如用户的选择) 同样可以反映回你的程序数据中。

在视图显示出来之前,Angular 会先根据你的应用数据和逻辑来运行模板中的指令并解析绑定表达式,以修改 HTML 元素和 DOM。Angular 支持**双向数据绑定**,这意味着 DOM 中发生的变化 (比如用户的选择) 同样可以反映回你的程序数据中。

在视图显示出来之前,Angular 会先根据你的应用数据和逻辑来运行模板中的指令并解析绑定表达式,以修改 HTML 元素和 DOM。Angular 支持**双向数据绑定**,这意味着 DOM 中发生的变化 (比如用户的选择) 同样可以反映回你的程序数据中。

在视图显示出来之前,Angular 会先根据你的应用数据和逻辑来运行模板中的指令并解析绑定表达式,以修改 HTML 元素和 DOM。Angular 支持**双向数据绑定**,这意味着 DOM 中发生的变化 (比如用户的选择) 同样可以反映回你的程序数据中。

</> 代码 1.1: src/app/app.module.ts

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
3 import { NgModule } from '@angular/core';
4 import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
5 import { FormsModule, ReactiveFormsModule } from '@angular/forms';
6
7 import { NgZorroAntdModule, NZ_I18N, zh_CN } from 'ng-zorro-antd';
8 // 配置 angular i18n
9 import { registerLocaleData } from '@angular/common';
10 import zh from '@angular/common/locales/zh';
11 registerLocaleData(zh);
12
13 @NgModule({
14   imports:      [ BrowserModule ],
15   providers:    [ Logger ],
16   declarations: [ AppComponent ],
17   exports:      [ AppComponent ],
18   bootstrap:    [ AppComponent ]
19 })
20 export class AppModule { }
```

NgModule 系统与 JavaScript (ES2015) 用来管理 JavaScript 对象的模块系统不同,而且也没有直接关联。这两种模块系统不同但互补。你可以使用它们来共同编写你的应用。

JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 export 关键字,模块可以把它的某些对象声明为公共的。其它 JavaScript 模块可以使用 import 语句来访问这些公共对象。

1.2 NgModule 和 JavaScript 的模块

NgModule 系统与 JavaScript (ES2015) 用来管理 JavaScript 对象的模块系统不同,而且也没有直接关联。这两种模块系统不同但互补。你可以使用它们来共同编写你的应用。

JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 `export` 关键字,模块可以把它的一些对象声明为公共的。其它 JavaScript 模块可以使用 `import` 语句来访问这些公共对象。

JavaScript 模块

JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 `export` 关键字,模块可以把它的一些对象声明为公共的。其它 JavaScript 模块可以使用 `import` 语句来访问这些公共对象。

JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 `export` 关键字,模块可以把它的一些对象声明为公共的。其它 JavaScript 模块可以使用 `import` 语句来访问这些公共对象。



JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 `export` 关键字,模块可以把它的一些对象声明为公共的。其它 JavaScript 模块可以使用 `import` 语句来访问这些公共对象。

JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 `export` 关键字,模块可以把它的一些对象声明为公共的。其它 JavaScript 模块可以使用 `import` 语句来访问这些公共对象。



JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 `export` 关键字,模块可以把它的一些对象声明为公共的。其它 JavaScript 模块可以使用 `import` 语句来访问这些公共对象。



JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 `export` 关键字,模块可以把它的一些对象声明为公共的。其它 JavaScript 模块可以使用 `import` 语句来访问这些公共对象。



JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 `export` 关键字,模块可以把它的一些对象声明为公共的。其它 JavaScript 模块可以使用 `import` 语句来访问这些公共对象。

JavaScript 中,每个文件是一个模块,文件中定义的所有对象都从属于那个模块。通过 `export` 关键字,模块可以把它的一些对象声明为公共的。其它 JavaScript 模块可以使用 `import` 语句来访问这些公共对象。¹

¹JavaScript 模块可以使用 `import` 语句来访问这些公共对象。

服务与依赖注入

服务是一个广义的概念，它包括应用所需的任何值、函数或特性。狭义的服务是一个明确定义了用途的类。它应该做一些具体的事，并做好。

2.1 服务

服务是一个广义的概念，它包括应用所需的任何值、函数或特性。狭义的服务是一个明确定义了用途的类。它应该做一些具体的事，并做好。

</> 代码 2.1: src/app/app.module.ts

```
1 export class Logger {  
2   log(msg: any) { console.log(msg); }  
3   error(msg: any) { console.error(msg); }  
4   warn(msg: any) { console.warn(msg); }  
5 }
```

服务也可以依赖其它服务。比如，这里的 `HeroService` 就依赖于 `Logger` 服务，它还用 `BackendService` 来获取英雄数据。`BackendService` 还可能再转而依赖 `HttpClient` 服务来从服务器异步获取英雄列表。

2.2 依赖注入 (dependency injection)

组件是服务的消费者，也就是说，你可以把一个服务注入到组件中，让组件类得以访问该服务类。

在 Angular 中，要把一个类定义为服务，就要用 `@Injectable` 装饰器来提供元数据，以便让 Angular 可以把它作为依赖注入到组件中。

同样，也要使用 `@Injectable` 装饰器来表明一个组件或其它类（比如另一个服务、管道或 `NgModule`）拥有一个依赖。依赖并不必然是服务，它也可能是函数或值等等。



图 2.1: Dependency Injection

同样,也要使用 `@Injectable` 装饰器来表明一个组件或其它类(比如另一个服务、管道或 `NgModule`)拥有一个依赖。依赖并不必然是服务,它也可能是函数或值等等。

同样,也要使用 `@Injectable` 装饰器来表明一个组件或其它类(比如另一个服务、管道或 `NgModule`)拥有一个依赖。依赖并不必然是服务,它也可能是函数或值等等。

依赖注入(通常简称 DI)被引入到 `Angular` 框架中,并且到处使用它,来为新建的组件提供所需的服务或其它东西。如图??。

- 注入器是主要的机制。你不用自己创建 `Angular` 注入器。`Angular` 会在启动过程中为你创建全应用级注入器。
- 该注入器维护一个包含它已创建的依赖实例的容器,并尽可能复用它们。
- 提供商是一个创建依赖的菜谱。对于服务来说,它通常就是这个服务类本身。你在应用中要用到的任何类都必须使用该应用的注入器注册一个提供商,以便注入器可以使用它来创建新实例。

PART II

第二部分

Angular 应用安全防范

最佳实践

Web 应用程序的安全涉及到很多方面。针对常见的漏洞和攻击,比如跨站脚本攻击,Angular 提供了一些内置的保护措施。

3.1 最佳实践

- 及时把 Angular 包更新到最新版本。我们会频繁的更新 Angular 库,这些更新可能会修复之前版本中发现的安全漏洞。查看 Angular 的更新记录,了解与安全有关的更新。
- 不要修改你的 Angular 副本。私有的、定制版的 Angular 往往跟不上最新版本,这可能导致你忽略重要的安全修复与增强。反之,应该在社区共享你对 Angular 所做的改进并创建 Pull Request。
- 避免使用本文档中带“安全风险”标记的 Angular API。

3.2 防范跨站脚本 (XSS) 攻击

跨站脚本 (XSS) 允许攻击者将恶意代码注入到页面中。这些代码可以偷取用户数据 (特别是它们的登录数据),还可以冒充用户执行操作。它是 Web 上最常见的攻击方式之一。

为了防范 XSS 攻击,你必须阻止恶意代码进入 DOM。比如,如果某个攻击者能骗你把 `<script>` 标签插入到 DOM,就可以在你的网站上运行任何代码。除了 `<script>`,攻击者还可以使用很多 DOM 元素和属性来执行代码,比如 ``、``。如果攻击者所控制的数据混进了 DOM,就会导致安全漏洞。

3.2.1 Angular 的“跨站脚本安全模型”

为了系统性的防范 XSS 问题,Angular 默认把所有值都当做不可信任的。当值从模板中以属性 (Property)、DOM 元素属性 (Attribute)、CSS 类绑定或插值表达式等途径插入到 DOM 中的时候,Angular 将对这些值进行无害化处理 (Sanitize),对不可信的值进行编码。

Angular 的模板同样是可执行的：模板中的 HTML、Attribute 和绑定表达式（还没有绑定到值的时候）会被当做可信任的。这意味着应用必须防止把可能被攻击者控制的值直接编入模板的源码中。永远不要根据用户的输入和原始模板动态生成模板源码！使用离线模板编译器是防范这类“模板注入”漏洞的有效途径。

3.2.2 listings examples

C# code sample

</> 代码 3.1: C# code sample

```
1 // A Hello World! program in C#.
2 using System;
3 namespace HelloWorld
4 {
5     class Hello
6     {
7         static void Main()
8         {
9             Console.WriteLine("Hello World!");
10
11             // Keep the console window open in debug mode.
12             Console.WriteLine("Press any key to exit.");
13             Console.ReadKey();
14         }
15     }
16 }
```

PHP code sample

</> 代码 3.2: PHP code sample

```
1 <?php
2 namespace App;
3 use Illuminate\Database\Eloquent\Model;
4
5 class Comment extends Model
6 {
7     /**
8      * Get all of the owning commentable models.
9      */
10     public function commentable()
11     {
12         return $this->morphTo();
13     }
14 }
15
16 class Post extends Model
17 {
18     /**
19      * Get all of the post's comments.
20      */
21     public function comments()
22     {
23         return $this->morphMany('App\Comment', 'commentable');
24     }
25 }
26
27 class Video extends Model
28 {
29     /**
30      * Get all of the video's comments.
31      */
32     public function comments()
33     {
34         return $this->morphMany('App\Comment', 'commentable');
35     }
36 }
```

JavaScript/ES6 code sample

</> 代码 3.3: JavaScript code sample

```
1 // module "my-module.js"
2 function cube(x) {
3   return x * x * x;
4 }
5 const foo = Math.PI + Math.SQRT2;
6 var graph = {
7   options:{
8     color:'white',
9     thickness:'2px'
10  },
11  draw: function(){
12    console.log('From graph draw function');
13  }
14 }
15 export { cube, foo, graph };
```

TypeScript code sample

</> 代码 3.4: TypeScript code sample

```
1 abstract class Department {
2
3   constructor(public name: string) {
4   }
5
6   printName(): void {
7     console.log("Department name: " + this.name);
8   }
9
10  abstract printMeeting(): void; // must be implemented in derived classes
11 }
12
13 class AccountingDepartment extends Department {
14
15   constructor() {
16     super("Accounting and Auditing"); // constructors in derived classes must call super
17   }
18
19   printMeeting(): void {
20     console.log("The Accounting Department meets each Monday at 10am.");
21   }
22
23   generateReports(): void {
24     console.log("Generating accounting reports...");
25   }
26 }
27
28 let department: Department; // ok to create a reference to an abstract type
29 department = new Department(); // error: cannot create an instance of an abstract class
30 department = new AccountingDepartment(); // ok to create and assign a non-abstract
31      subclass
32 department.printName();
33 department.printMeeting();
34 department.generateReports(); // error: method doesn't exist on declared abstract type
```

golang code sample

</> 代码 3.5: golang code sample

```
1 // Go supports [anonymous functions] (http://en.wikipedia.org/wiki/Anonymous_function),
2 // which can form <a href="http://en.wikipedia.org/wiki/Closure_(computer_science)"><em>
3 //   closures</em></a>.
4 // Anonymous functions are useful when you want to define
5 // a function inline without having to name it.
6
7 package main
```

```

8 import "fmt"
9
10 // This function `intSeq` returns another function, which
11 // we define anonymously in the body of `intSeq`. The
12 // returned function `_closes` over_ the variable `i` to
13 // form a closure.
14 func intSeq() func() int {
15     i := 0
16     return func() int {
17         i++
18         return i
19     }
20 }
21
22 func main() {
23
24     // We call `intSeq`, assigning the result (a function)
25     // to `nextInt`. This function value captures its
26     // own `i` value, which will be updated each time
27     // we call `nextInt`.
28     nextInt := intSeq()
29
30     // See the effect of the closure by calling `nextInt`
31     // a few times.
32     fmt.Println(nextInt())
33     fmt.Println(nextInt())
34     fmt.Println(nextInt())
35
36     // To confirm that the state is unique to that
37     // particular function, create and test a new one.
38     newInts := intSeq()
39     fmt.Println(newInts())
40 }

```

为了系统性的防范 XSS 问题,Angular 默认把所有值都当做不可信任的。当值从模板中以属性 (Property)、DOM 元素属性 (Attribute)、CSS 类绑定或插值表达式等途径插入到 DOM 中的时候,Angular 将对这些值进行无害化处理 (Sanitize),对不可信的值进行编码。

Angular 的模板同样是可执行的:模板中的 HTML、Attribute 和绑定表达式 (还没有绑定到值的时候) 会被当做可信任的。这意味着应用必须防止把可能被攻击者控制的值直接编入模板的源码中。永远不要根据用户的输入和原始模板动态生成模板源码!使用离线模板编译器是防范这类“模板注入”漏洞的有效途径。

Angular CLI

```
ng new appname --style scss --skip-install
```

"Test, 3

|