SpaceX 软件工程师问答摘录

今天偶然发现了 SpaceX 软件工程师在论坛上与网友问答互动的两个帖子：

https://old.reddit.com/r/spacex/comments/gxb7j1/we_are_the_spacex_software_team_ask_us_anything/
https://old.reddit.com/r/spacex/comments/ncj4vz/we_are_the_spacex_software_team_ask_us_anything/

现作摘抄汇总如下：

1. 关于计算架构

整体上，我们的飞行器上有许多计算机，每个计算机都是根据其分配的任务进行构建和配置的。它们都以时间同步的方式运行，飞行计算机负责监督所有的动作。几乎所有的操作都可以被表达为实时控制循环：你读取一些传感器数据，做出决策（结合传感器数据和过去的状态），然后将决策的输出发送回硬件。这个过程每秒钟发生多次。 - Dietrick

我们使用 Linux 操作系统，并应用了 PREEMPT_RT 补丁以获得更好的实时性能。我们不使用任何第三方发行版，而是维护自己的内核和相关工具的副本。多年来，我们对内核进行了一些小的修改，尽管大部分仍未修改。唯一的例外是添加了几个自定义驱动程序，用于与我们的硬件进行接口。我们使用各种 CPU 架构。我不能提供太多详细信息，只能说它是一个由许多个体计算机组成的分布式系统。 – Josh

就 Starlink 而言，每次发射 60 颗卫星都包含超过 4,000 台 Linux 计算机。目前，该星座在太空中拥有超过 30,000 个 Linux 节点（以及超过 6,000 个微控制器）。由于我们与 Falcon 和 Dragon 共享许多 Linux 平台基础设施，它们受益于我们

超过 180 个 vehicle-years 的在轨测试时间。 - Matt

Linux 仅用于运行我们的应用程序和与硬件进行接口。所有的故障管理和计算冗余都在我们开发的自定义软件处理。时间同步使用了硬件和软件功能的组合，其中一些是行业标准，一些是我们自己开发的。- Josh

2. 关于软件安全

自主飞行安全系统（Autonomous Flight Safety System）软件运行在与飞行计算机独立的一组微控制器上。它直接接收传感器输入（例如 IMU 测量值），以及一些由飞行计算机计算得出的输入。任务数据装载特定的 AFSS 参数，以确定哪些条件可能需要终止飞行，例如火箭偏离航线太远、失去所有加速度等。- Jeff

在 Dragon 上，我们在硬件方面具有很多冗余（多个计算机、传感器、执行器等），但也使用软件来处理故障响应。NASA 的要求是，我们的飞行器必须具备 2 个 fault tolerant （即，能够安全地从空间站撤离和/或让有人驾驶的飞行器安全地返回），因此我们进行分析和测试以确保满足这种 fault tolerant 。- Wendy

在 Starlink 上，我们设计了系统，以便在发生故障时，卫星会因大气阻力而迅速被动降轨（尽管我们会尽力主动将其降轨）。在飞行器内部，我们仍然具有一些冗余，但我们主要依靠系统级的 fault tolerant ：多个卫星可以为用户提供服务。发射更多的卫星是我们的核心竞争力，这使我们能够在大多数时间内提供更好的服务，并且在个别卫星发生故障的时候提供足够冗余。- Matt

我们拥有许多传统的网络安全措施，用于保护我们的企业网络，监测内外部的威胁，进行钓鱼攻击防范等。我们还需要分析针对我们的飞行器的潜在攻击，特别是在指令路径和最终部署到飞行器上的代码的可信度方面。我们有一个专门的团队来识别我们的飞行器和卫星可能受到的黑客攻击方式，以便在构建飞

行器时消除或禁止这些威胁。我们还充分利用静态和动态分析来对我们的代码进行检测。- Jeff

在安全关键的软件中，重要的是如果单个软件组件出现问题，不会影响整个软件系统。软件通常还依赖于与之交互的硬件，因此必须考虑硬件故障。崩溃和重新启动并不是一个选项。我们通过构建模块化组件、编写防御性逻辑和检查每个操作的状态来解决这个问题。如果我们期望完成的操作失败了，我们定义了错误处理路径和恢复策略。有时，这个策略只是跳过该操作。有时，策略可能更复杂，涉及到切换到备份系统等响应措施。-Jarrett

3. 关于软件部署与维护

在进行更改时，我们希望我们的工程师们能够进行批判性思考（并相互质疑），考虑功能测试（如何知道我的更改有效？）和回归测试（如何知道是否破坏了其他功能，或者是否会在将来出现问题？）。构建可以在地面上运行的测试用例是回答这些问题的一个很好的方法，我们确实做了很多这样的工作，但这并不是唯一的方法。

对于 Starlink 来说，我们需要将卫星视为数据中心中的服务器，而不是特殊的独一无二的飞行器。有一些事情我们需要绝对确定（命令、软件更新、电源和硬件安全），因此需要有特定的测试用例。但对于其他许多事情，我们可以更加灵活一些 - 对于这些事情，我们可以采用更类似于开发网络服务的方式。我们可以将测试版本部署到我们的一小部分卫星上，然后与其余的卫星进行比较，看它的表现如何。如果它不能达到我们的要求，我们可以进行调整并再次尝试，然后再合并。如果在推出时出现问题，我们可以暂停、回滚，然后再次尝试。这对于我们思考太空飞行器的方式来说是一个非常强大的变革，对于能够快速迭代我们的系统来说绝对至关重要。– Matt

我们尝试每周向我们的整个设备群（卫星、地面站、用户终端和 WiFi 路由器）推出新的构建版本。每个设备定期与我们的服务器进行通信，以查看是否需要

获取新的构建版本，如果有可用的版本，设备将在最佳时间下载并应用更新，以最小化对用户的影响。这意味着我们可以在一个小型设备池中轻松测试构建版本，并通过在数据库中更改一些配置来实现指数级的部署。

我们的系统设计使得每个设备（可能包含数十台独立计算机）通过首先将新软件包获取到一个中央节点，然后让所有其他计算机从该中央节点获取更新。每个设备还保留了上一个良好软件版本的备份副本，因此如果在更新过程中出现任何问题（如辐射引起的电源故障），设备将自动通过引导到备份副本来恢复。

我们几乎所有的部署和测试工具都是自主开发的，主要是因为我们的架构非常独特，而且我们需要处理各种约束条件，这要求对现有工具进行大量定制。- Natalie

在思考 SpaceX 的软件工程时，很容易只考虑到运行在我们的飞行器上的软件，但那只是冰山一角。我们的一些工程师是 C++开发人员，为 Starship 编写代码，但我们还有工程师正在构建核心软件测试基础设施，为工厂、星座和客户开发内部 Web 应用程序，自动化生产测试，或编写代码来协调 Starlink 网络。我们的软件工程师使用 C++、Python、C#.NET、Java、Javascript、Angular 等多种编程语言。- Kristine and Jeanette

4. 关于工具栈

龙飞船宇航员使用的 UI 界面
我们在飞行软件中使用 C 和 C++，在显示方面使用 HTML、JavaScript 和 CSS，并且大量使用 Web 组件。而在测试方面使用 Python。- Sofian

没错，我们在显示界面上使用了 Chromium 作为渲染引擎。这个项目最初是一个模拟器原型，用来向 NASA 展示设计愿景。然后我们尝试在飞行硬件上运行它，并进行了一些修改，效果非常好。随着我们开发原型的过程，我们对这个技术栈更加有信心，因此我们在设计飞行软件时就以此为基础。我们喜欢浏览

器带来的现代化功能，也喜欢能够接触到已经接受过这种技术培训的人才。或许在 SpaceX，我们并不害怕以稍微不同的方式做事情。我们喜欢采用从第一原理出发的方法来解决问题，而不仅仅依赖行业标准。- Sofian

我们不使用现成的 Linux 发行版 - 我们有自己的发行版。- Dietrick

Dragon 和 Falcon 不使用任何机器学习技术，但这并不意味着类似的技术不会出现在 SpaceX 的未来！- Dietrick

我们使用了一个我们自己开发的响应式编程库。不同的团队成员依照个人喜好，使用不同的编辑器，我使用 VSCode。我们对所有代码都有代码检查工具。- Sofian

我们有一个专门的团队负责构建 CI/CD 工具以及我们的飞行器使用的测试和仿真的核心基础设施。为了测试软件而运行高保真度的物理仿真，这些工具面临着一些有趣的挑战，而大多数现成的 CI/CD 工具并不能很好地处理这些挑战。这些仿真不仅需要大量计算资源，而且可能持续时间很长（比如从发射到对接的整个 Dragon 飞行过程），我们需要能够运行“硬件脱机”仿真以及“硬件在环”仿真，其中我们将软件加载到带有实际计算机和电子设备副本的测试平台上。

我们已经做了很多工作，以确保开发人员在日常工作中可以轻松运行这些测试 – 我们可以在我们的工作站上运行与 CI 集群相同类型的测试，并且我们可以在合并更改之前在硬件在环测试平台上运行测试用例。这使我们对我们编写的代码非常有信心。

我们也利用了一些现成的工具；例如，我们广泛使用 Bazel 来满足我们的构建和单元测试需求。- Natalie

确保稳定的性能是最重要的。在新程序的早期开发阶段，我们不害怕尝试新的

系统、策略、标准或语言。然而，任务的成功是至关重要的，我们需要关注未来代码的可维护性，所以相比于普通的初创公司，我们会保持一定的距离，不过于追求最前沿的技术。 - Asher

我们对 Rust 感到非常兴奋！它注重安全性、性能和现代工具，这些都是它的亮点。我们也很高兴能够在嵌入式系统、模拟器、工具和 Web 应用程序中使用同一种语言。我们正在开始使用 Rust 原型设计一些新项目，但我们当然只是处在这个旅程的开始阶段。 - Asher

在 SpaceX，我们大量使用 Python！我们的许多地面工具都有很大的 Python 组成部分 - 例如我们的数据分析服务、测试基础设施和 CI/CD 系统。虽然它不直接用于飞行器，但我们经常使用 Python 来构建许多其他系统。

Python 的一个独特之处在于它是非软件工程师（机械、推进等）学习和工作的绝佳语言。我们在使用 Python 编写软件和硬件的测试用例、自动化数据分析流程以及其他需要各种背景的工程师能够贡献的领域取得了很大成功。 - Kristine

5. 关于软件模拟和测试

我们尽可能采用各种方式进行测试！单元测试、容器化的集成测试（您可以在自己的计算机上运行这些测试，并进行完整的物理模拟），以及在真实飞行硬件上进行完整的“HITL”（硬件在环）测试 - 同样，配合完整的模拟。将飞行软件与模拟器配对是我们拥有的最强大的工具，尤其是在实际硬件上运行时。我们可以在实验室的桌子上只是将飞行器硬件放置在那里，就能模拟整个任务，甚至许多详细的故障场景。 - Dietrick

对于每辆飞行器，我们都有一个 hardware in the loop 模拟器（包括所有飞行关键硬件以及模拟的物理和感知），在将其部署到生产飞行器或进行飞行之前，我们会对其进行大量的测试。每当我们进行新的软件更改（对于开发飞行器来说，这经常发生！），我们都会确保运行代码的单元测试，功能测试以确保软件

按预期工作，以及系统级测试来测试正常和非正常情况下的任务阶段。- Wendy

在 Dragon 上，我们会模拟任何影响飞行器安全关键软件的故障。我们使用单元测试和组件级测试的组合，以确保单一和双重故障会导致飞行器按照我们设计的方式做出反应。我们还会进行模拟任务测试，包括正常情况下的完整任务配置文件以及在这些情况下引入故障，以确保各个系统之间的依赖关系得到充分理解。我们还通过持续集成系统进行这些测试，并运行自动化数据检查，以确保没有出现意外行为。 - Wendy

在 SpaceX，我们不将质量保证与开发分开 - 每个编写软件的工程师都被期望参与测试工作。我们通常会在高保真度的硬件测试平台上尽可能多地进行合并前的测试。我们的测试代码和测试结果会与飞行代码一起进行同行评审，以确保我们测试的是正确的内容。我们还有独立的工程师开发端到端测试，对整个系统进行压力测试。

对于大型卫星星座的测试，一个独特之处在于我们实际上可以使用“金丝雀”卫星来测试新功能。我们对软件进行回归测试，以确保它不会破坏关键功能，然后我们可以选择一个卫星，部署新功能，并监控其行为，对星座的风险最小化。 - Natalie

6. 关于实时性

保持容错计算系统的关键在于确保所有飞行计算机之间的正确时序。对于响应较慢的子系统（如生命支持或热控制），响应时间会有一些弹性（取决于我们采取的故障类型），大约在几秒钟的数量级上。 - Wendy

内核错误绝对是最有趣和令人难忘的。我们的大部分控制软件是单线程的，以避免同步问题引入的非确定性，但操作系统在任何给定时间都会有很多事情发生。我们花了很多心思将 Linux 打造成一个可靠的实时控制平台，其确定性要比您在桌面操作系统中看到的要高得多。正如在其他地方提到的，我们使用了

CONFIG_PREEMPT_RT 补丁，这对我们非常有帮助。但即使如此，在早期的开发中，有时我们会发现系统的实时性不如我们所希望的那样，解决这些问题总是一次冒险。- Dietrick

我们所有的机载计算机要么运行 Linux（带有 PREEMPT_RT 补丁），要么是运行裸机代码的微控制器。对于在 Linux 上运行的应用程序，我们会小心地设置进程和内核线程的优先级，以避免优先级反转。我们通常以最大化确定性的方式编写代码，例如避免在运行时进行内存分配或无限循环。最后，我们有遥测数据，可以显示所有进程的性能，以确保它们在飞行的所有阶段都能按时完成任务，即使在出现意外或过多的输入时也是如此。- Josh

7. 关于硬件

我们的硬件与特斯拉的不同。 - Sofian

我们使用了一款专用的四核处理器，算力类似于五年前（2015）的手机。- - Sofian

我们在某些应用中使用卡尔曼滤波器。对于许多传感器数据，我们采用更简单的方法，例如基本的合理性检查或低通滤波。总体而言，我们处理传感器错误的方法是使用多个冗余传感器，并以容错的方式将它们的输入组合起来，以确保坏传感器不会导致危险的飞行器行为。- Josh

8. 关于数据量

对于 Dragon 飞船来说，典型任务的数据量在数百 GB 左右，每次飞行后我们会进行相当数量的数据审查，以确保我们了解系统是否按照我们的意图运行。- Wendy

对于 Starlink 来说，我们目前每天产生超过 5TB 的数据！我们正在积极减少每

个设备发送的数据量，但同时我们也在快速扩展卫星（和用户）的数量。就分析而言，进行在线问题检测是减少我们需要发送和存储的遥测数据量的最佳方法（只在有趣的时候发送）– Matt