可通讯状态机(CSM)架构框架优势实例分析 Q&A

非常感谢大家参加我们关于《可通讯状态机(CSM)架构框架优势实例分析》的网络研讨会！直播期间共有 224 位伙伴进入直播间，其中 130 位全程参与了我们的分享，大家的热情让我们备受鼓舞。讨论环节也非常热烈，我们收到了超过 100 条提问和讨论留言！不过由于直播时间有限，我们当时没能详细解答所有问题，有些回答可能也不够深入。所以，我们专门整理了大家最关心的问题，在这里为大家进行集中回复。我们将问题按照类型分成了以下几组进行回复。

可通讯状态机(CSM)是什么？

大家关心的第一类问题，主要是关于可通讯状态机(CSM)本身： 很多伙伴可能是刚刚接触 CSM，对它究竟是什么以及目前的发展现状不太清楚。下面是一些典型的提问：

1. *CSM 系统架构是软件？还是硬件？*
2. *对于硬件配置有哪些要求？*
3. *CSM 本质能说一下吗？*
4. *CSM 是否需要 License？*
5. *能不能提供试用版让我们体验一下？*
6. *可通信状态机（CSM）对 LabVIEW 版本有要求吗？*
7. *CSM 是否兼容旧版本 LabVIEW？*
8. *CSM 未来会有新的 Feature 加进来么？*

可通信状态机（CSM） 是一个基于 JKI 状态机 (JKISM) 构建的开源 LabVIEW 应用框架（采用 MIT 协议）。它支持 LabVIEW 2017 及更高版本。

CSM 完全遵循 JKI SM 的核心模式，并在此基础上进行了扩展。它引入了新的关键词，专门用于描述模块之间的消息通信机制。这包括了同步消息、异步消息、状态订阅/取消订阅等概念，为实现模块化提供了核心的基础设施。在 LabVIEW 开发社区中，类似的框架有 DQMH、Workers、SMO、Actor Framework 等。

作为一个开源框架，CSM 的源代码托管在 GitHub 和 Gitee 上。您也可以通过 VIPM 轻

松安装。相关链接如下：

- GitHub: HTTPS://GitHub.com/NEVSTOP-LAB
- gitee: HTTPS://gitee.com/NEVSTOP-LAB
- vipm: Communicable State Machine Framework - Package List
- VIPM 打包好的 Release Package(VIPC)合集：(包含全部 Addon+Examples，可以离线安装)
  链接: HTTPS://pan.baidu.com/s/10fsnFmJpn-P_HLbpH9IFLg 提取码: CSMF

CSM 目前保持每月一次的更新节奏，新版本会通过 VIPM 定期发布给大家。每次更新的具体内容，欢迎关注我们的知乎专栏「可通讯状态机(CSM)框架」，那里会发布详细的月度更新说明。

需要说明的是，当前的更新重点主要放在这几个方面：持续完善周边工具链、补全测试用例和文档、以及修复使用者反馈的 BUG。核心框架本身已经非常稳定，因此对于核心功能的添加，我们会非常谨慎地进行评估。

为了保证每次代码提交的质量和稳定性，我们建立了自动化代码测试流程。这确保了每次更新都经过严格的验证。如果你对 CSM 是如何实现自动化测试的具体技术细节感兴趣，非常欢迎访问我们的 CSM GitHub 代码仓库，深入了解其背后的实现机制。

我们非常欢迎您下载试用 CSM！同时也诚挚邀请有经验的 LabVIEW 开发者加入我们，共同推进 CSM 的发展。

CSM 独特的优势

还有一类讨论聚焦在 LabVIEW 程序架构的价值本身，以及 CSM 与其他流行框架相比的独特优势。一些典型的问题有：

1. *CSM 和 DQMH 在标准流程自身优势是？*
2. *CSM 对于大型项目框架带来哪些便利？*
3. *CSM 技术优势在于哪些方面？*

我们认为，程序框架本质上是软件工程实践为解决复杂问题而提炼出的结晶。这里的“复杂问题”，不仅指功能需求的复杂性，更涵盖了代码的可维护性、多人协作效率等工程化要求，以及支撑业务发展和售后服务的长期需要。

回顾《可通讯状态机(CSM)架构框架优势实例分析》中的案例，项目挑战远不止实现数据采集功能本身。它还涉及到：如何高效地与新人同事协作、如何应对设备到位延迟的现实情况、如何确保项目的顺利交付和后续维护。一个设计精良的程序框架，正是为了有效应对这类综合性的工程挑战而存在的。

场景 – 数据采集、分析和记录的场景

项目任务与现状：

1. 需要创建一个连续的数据采集、存储和分析系统。
2. 分析算法暂时不明确。
3. 后期可能需要支持中控系统，需要被远程控制和数据备份。
4. 硬件仍在采购中，但是项目周期比较紧张，先要编写程序。
5. 新人同事小李与你一起完成这个项目。
6. 交付后，这个项目会部署偏远地区，可能需要长期支持。

EMERSON | NI

在 LabVIEW 开发中，我们通常建议选择成熟的编程框架。 除了 CSM，目前业内比较知名且成熟的选项还包括 DQMH、Workers、SMO 和 Actor Framework。我们做了一个简要的特性对比（请参考下图）。

	DQMH	SMO	Actor Framework	CSM
优点	• 继承于最广泛应用的框架QMH • 无OOP，对于大多数LabVIEW用户比较友好 • 社区活跃度高，资料/工具众多	• 非常符合面向对象的概念 • 模块独立性好，操作调用直观 • SMO 最适用的场景还是模块复用，不与调用方一起开发	• 操作者、消息均解耦，解耦程度高，便于多人协作开发 • 熟悉 OOP 的开发人员使用方便	• 继承于JKISM，代码集中度高 • 无OOP,对于大多数LabVIEW用户比较友好 • 通过文档定义接口分工，无需代码 • 容易实现操作序列化 (Serialization) • 内置多种高级模式 • AI(LLM)友好
缺点	• 代码接口冗余 • 两套模板实现 Singleton/Standalone模块	• 框架复杂度高，对用户要求高 • 目前社区用户少，更新频率不高 • 资料比较少	• 框架复杂度高，对用户要求高 • 项目需要架构师角色	• 所有消息必须使用 STRING 格式传递，有转换的开销 • 参数不通过 LabVIEW 数据类型定义，不严格要求

CSM 的诞生源于一个实际项目的需求。 当时，在评估了市面上所有成熟的 LabVIEW 框架后，我们发现没有哪个能完美支撑该项目的要求，这才促使我们创建了 CSM。它的独特之处主要体现在：

- 用户可见代码逻辑简单统一： 对新人非常友好，极大简化了将代码移交给客户维护的流程。
- 基于虚拟总线实现插件机制： 使模块解耦和团队分工协作变得异常简便。
- 优雅完备的消息通信机制： 保障了模块的高度独立性，进一步提升了分工协作的效率。
- 核心交互采用纯文本： 既方便架构师设计整体框架，也极大简化了相关设计文档的编写和整理工作。

作为 LabVIEW 框架领域的后来者，CSM 充分借鉴了现有优秀框架的长处，同时也融入了自身鲜明的特色。它是由中国本地团队创建并维护的开源框架，这意味着我们能够为中国用户提供更及时、更贴近需求的本地化支持和服务。

当然，不同的框架各有其优势。 在实际项目中，我们建议开发者结合自身技术熟悉度以及项目的具体需求来做出最合适的选择。

CSM 使用技巧

不少提问聚焦在 CSM 的具体开发实践上，比如调试技巧和典型状态机场景的实现方法。下面是一些代表性的问题：

1. *CSM 中有哪些 debug 的方式？CSM Console 支持哪些操作？*
2. *如何在 CSM 实现周期性执行指定操作？*
3. *太多接口不知道选哪个？*

上面列出的问题，主要与我们提供的 调试工具链 和 功能插件(Addon) 相关。目前 CSM 内置了以下内容来提升开发效率：

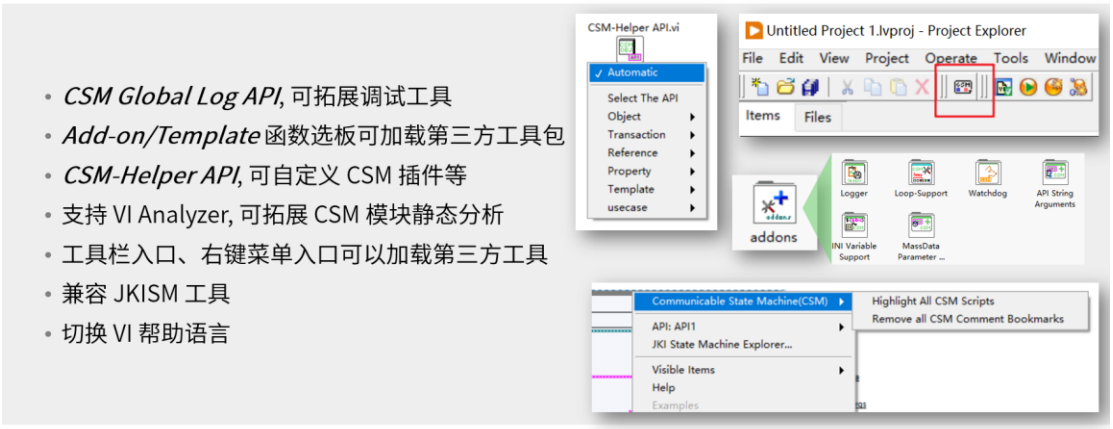
核心调试工具：

- CSM Debug Console： 无需编写额外代码，即可直接向模块发送测试消息，甚至模拟其他模块发布状态变化，是交互式调试的利器。
- CSM Log Viewer： 实时监控程序运行时的关键信息，包括模块状态、错误日志和活跃的订阅关系，一目了然。
- Interface Browser： 自动扫描项目中的 CSM 模块及其接口，方便开发者快速查看、理解和选择合适的接口进行交互。
- Mermaid Viewer： 利用流行的 Mermaid 图表工具，直观地可视化展示模块间的接口定义和状态订阅关系，便于架构理解。

实用功能插件 (Addons)：

- CSM File Logger： 将程序运行状态和日志持久化保存到文件，便于离线分析和问题回溯。
- CSM Watchdog： 确保程序优雅退出。它会在主程序关闭时，自动向所有后台模块发送 Macro: Exit 消息，保证模块安全终止。
- CSM Loop Support： 提供 CSM 框架推荐的、清晰可靠的方式来实现需要周期性执行操作的状态。

拓展式的设计



如果现有的工具或插件还不能完全满足你的特定需求，CSM 提供了丰富的 API 接口，你可以基于这些接口灵活定制开发自己所需的调试工具或功能插件。 我们也非常欢迎大家将自行开发的有用工具或插件共享给社区！ 这种互利互惠的协作模式，能让 CSM 生态和所有开发者共同受益。

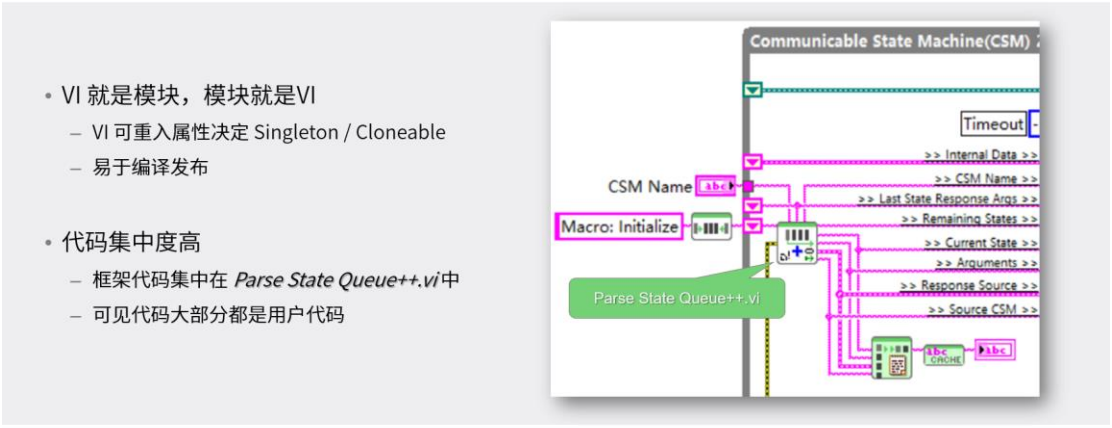
对于深入的技术内容，推荐大家关注：知乎专栏：可通讯状态机(CSM)框架 和 CSM Wiki (项目文档)。

CSM 性能表现

关于 CSM 的性能表现，大家集中提出了以下问题：

1. *多通道并发时，CSM 的内存峰值比传统 QMH 少几成？*
2. *CSM 与 DQMH 相比，消息吞吐延迟能降低多少 ms？*
3. *CSM 支持的最大事件队列深度是多少？溢出时自动丢包还是阻塞？*
4. *最大可并行状态机数量上限是多少，CPU 占用率如何？*
5. *用 CSM 拆分模块后，代码量平均减少多少%？*

VI 即模块



EMERSON | NI

CSM 的核心设计目标之一，就是最大化减少用户需要编写和维护的“框架代码”量。因此，相较于 DQMH 或 Actor Framework 等框架，CSM 的一个显著特点是能够有效减少项目中 VI（虚拟仪器）的总数量。

- 关于代码简化效率：我们目前尚未进行全面的代码复杂度简化比较测试。但根据 NI 合作伙伴的实际反馈，在一个需要灵活管理 200+ 测试项的复杂项目中，从 DQMH 迁移到 CSM 后，开发工作量节省了约 50%。同时，该团队反馈，具备两年左右经验的工程师能够快速融入开发，并独立完成现场调试任务，这印证了 CSM 在降低门槛和提升团队协作效率方面的价值。

CSM 的另一个核心设计目标，是追求尽量少的占用系统资源，将系统资源留给核心业务逻辑。 框架内部采用了多种缓存机制和 LabVIEW 编程优化技巧来实现这一目标。关于性能边界与资源占用：

- 框架本身无硬性限制： CSM 本身不预设消息队列深度上限，也不限制同时运行的模块数量。
- 性能表现高度依赖实现与硬件： CSM 的实际性能（CPU、内存占用）与您的具体代码实现质量以及底层硬件配置密切相关。
- 实践验证： 虽然我们没有专项性能对比报告，但 CSM 已在包含 30+ 模块、且模块间 LOG 峰值达到约 5000 条/秒 的复杂项目中成功部署和应用，运行稳定，未出现性能瓶颈。
- 性能自测： 您可以在 CSM 范例库中找到性能测试范例：<CSM Example>\4. Advance Examples\6. Global Log Handling Capability。这个范例专门用于测试多模块、高吞吐场景下的性能极限，欢迎您在自己的硬件平台上运行测试。

由于篇幅所限，关于性能的介绍先到这里。如果您有更深入的问题，欢迎在文章下方留言，或在 GitHub / Gitee / 知乎专栏 中向我们反馈，我们会尽力为您解答！

CSM 的应用场景

还有一大类问题聚焦于 CSM 的实际应用场景。 问题整理如下：

1. *对于使用 cDAQ 进行数采+控制+人机界面的工业产品应用，CSM 是不是更好的选择？*
2. *当网络中断或节点失败时，CSM 框架如何确保状态机的可靠性和系统整体的稳定性？*
3. *如果用在整车的 LINCAN 通讯网络中，需要不断地收发指令，接收硬件反馈信息，这种实时通讯的情况，可以使用 CSM 吗？*
4. *在 NIVeriStand 环境中，如何将自定义的 CSM 框架部署到实时目标？需要哪些特定的配置或工具支持？*
5. *CSM 框架是否支持如 TCP/IP、UDP、CAN 或 OPCUA 等协议？如何配置这些协议以实现状态机之间的通信？*
6. *NI 的可通讯状态机框架如何通过状态机实现模块化设计，以及它在分布式测量与控制系统中的通信机制是如何工作的？*
7. *CSM 在 FPGA 半实物仿真验证中是否有相关应用案例？*
8. *如果用在仪器通讯上，上位机一直发命令，硬件一直反馈信息到上位机，这样能用这个框架吗？*

首先需要明确的是，CSM 作为一个程序框架，本身并无特定的行业限制。 它的核心目标是辅助设计更清晰、更易维护的软件架构，促进团队分工协作，提升代码复用率，从而避免大型项目陷入难以维护的“意大利面条式代码”困境。因此，理论上 CSM 可以赋能任何需要良好架构的 LabVIEW 应用场景。

- 对于问题中提到的数据采集+控制、整车 LIN/CAN 网络测试、仪器通讯等场景，CSM 完全适用。
- 对于 TCP/IP、UDP、CAN 等通讯协议的支持，关键在于在 CSM 框架下实现相应的功能模块。事实上，在 NI 中国工程团队的多个项目中，这些协议都已被成功集成到基于 CSM 的解决方案中。

为了帮助大家理解 CSM 如何应用于大规模的程序，除了普通的范例，我们提供了两个主要的开源范例（可通过 VIPM 获取）：

- CSM 示例: 连续测量和记录应用程序 - 展示数据采集与存储的典型实现。
- CSM 示例: TCP 通讯应用 - 演示如何在 CSM 中集成网络通信。

目前，NI 中国工程团队已将 CSM 成功应用于以下多样化场景，积累了丰富的实践经验：

- 标准化测试执行： 与 TestStand 紧密集成，构建标准化测试流程。
- 并行测试系统： 高效管理多种采集设备（如 PXI 硬件、CAN/LIN/485 等总线设备）。
- 灵活流程控制： 实现脚本化的、可动态配置的控制逻辑。
- 汽车 HIL 测试： 构建硬件在环系统，用于汽车电子故障注入验证。
- 实时硬件在环 (HIL)： 结合 MIT 工具包和 NI 实时系统，实现高性能实时仿真。
- 卫星通信验证 & 实时信号处理： 配合 FPGA 技术，处理高速信号和卫星通信协议。

如果您的业务涉及以上领域或类似挑战，我们非常欢迎您咨询洽谈！ NI 中国工程团队期待

与您合作，共同探索 CSM 如何助力您的项目成功。

其他讨论

在整理留言时，我们也注意到一些关于 如何在具体项目中应用 CSM 进行架构设计的深度问题，例如：

如何保证 CSM 模块间的数据一致性？
如何有效避免资源竞争 (Race Condition)？
如何优化消息传递机制以减少延迟？

这些问题通常需要结合具体的应用场景和实现细节来深入探讨。受限于本文篇幅，我们无法在此一一展开详述。非常欢迎大家将这些具体问题发布在 CSM 的 GitHub / Gitee 仓库讨论区或知乎专栏留言中！ 我们团队会密切关注，并针对性地进行回复和交流。未来，我们也很乐意就这些普遍关注的主题策划专门的线上或线下分享会。

不少伙伴也问到了当前火热的 AI 技术如何与 CSM 框架结合，以及 NI 在 AI 领域的探索。这是一个宏大且充满潜力的方向！CSM 得益于其核心交互基于纯文本的设计，天然具备与大语言模型 (LLM) 等 AI 技术进行深度集成的潜力。NI 中国工程团队目前正在积极进行相关的探索性研究和尝试，探索 AI 如何与 CSM 结合，一旦我们在这些结合点上取得突破性的进展或实用的成果，我们一定会通过专题研讨会、技术文章或新版本发布等形式，第一时间与大家分享！

总结与诚挚邀请

再次衷心感谢大家在网络研讨会期间的积极参与和宝贵提问！大家的热情是我们持续优化 CSM 的强大动力。

为了构建更活跃的 CSM 生态，我们诚挚邀请您：

- 持续反馈： 请继续通过 NI 官方公众号、CSM GitHub / Gitee 仓库 或 CSM 知乎专栏提出您的建议、疑问或使用心得。您的反馈将直接指引 CSM 的改进方向！
- 分享经验： 如果您已经在项目中成功应用了 CSM，非常欢迎您以文章形式向 CSM 知乎专栏投稿！分享您的实践案例、技巧和挑战，您的经验将惠及整个 LabVIEW 开发者社区。
- 支持项目： 如果觉得 CSM 有价值，请到 GitHub 上 Star 我们的 CSM 仓库！这是对我们开源工作的巨大鼓励。
- 探讨合作： 如果您的项目涉及测试自动化、数据采集、控制系统、汽车电子、实时仿真等领域，并希望探讨如何提升开发效率与软件质量，欢迎随时联系 NI 中国工程团队咨询合作！我们拥有丰富的落地经验，期待与您携手共赢。

让我们共同努力，推动 LabVIEW 开发生态更加繁荣高效！