QCLab 快速上手指南 (uv 版本)

本指南旨在帮助开发者使用高效的包管理工具 uv 快速搭建围绕quarkstudio的的本地开发环境。

---

1. 先决条件

在开始之前，请确保您的系统已经安装了 uv。如果尚未安装，可以通过 pip 进行安装：

```
# On macOS and Linux.
curl -LsSf https://astral.sh/uv/install.sh | sh

# On Windows.
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

2. 环境安装与配置

请遵循以下步骤来创建虚拟环境、安装依赖并完成项目初始化。

第一步：创建虚拟环境

我们使用 uv 来创建一个指定 Python 版本的虚拟环境。这可以确保项目依赖与系统环境隔离。

打开终端，执行以下命令：

```
# 在工作目录(例如QCLab）下创建一个虚拟环境，并指定使用 Python 3.12.x
uv init --python 3.12 --vcs none QCLab
```

创建成功后，激活该虚拟环境：

```
# 切换到虚拟环境所在目录，这一步非常重要!
cd venv-qc

# macOS / Linux
source venv-qc/bin/activate

# Windows
.\venv-qc\Scripts\activate
```

第二步：安装依赖

为了加速下载，我们切换到清华大学的 PyPI 镜像源来安装所有依赖。quarkstudio[full] 会自动安装所有必需的库。

```
# 使用清华镜像源安装 quarkstudio 及其所有可选依赖
uv add quarkstudio[full] --index-url https://pypi.tuna.tsinghua.edu.cn/simple

# 如遇版本不对问题，则更新库版本（注意，这会更新所有库）
uv sync -U
```

> 注意:
> * [full] 选项非常重要，它确保了图形界面、数据分析等所有功能模块都被正确安装。
> * quarkstudio的画图模块'canvas'依赖pyside6.9.0，但自动安装目前会装最新版6.9.1，需要手动安装6.9.0: uv add pyside6==6.9.0
> * 运行测控程序还有个依赖需要手动加进去：uv add wath。这是一个计算波形的数学库。

第三步：初始化配置

quark 命令行工具需要一个全局配置文件 ~/quark.json 来定位项目的主目录。~ 代表您的用户主目录。

1. 在任意位置执行quark命令，会在您的用户主目录下 (e.g., /Users/your_username 或 /home/your_username 或 C:\Users\YourName) 创建一个名为 quark.json 的文件。
2. 修改server的主目录home，并确保 **home** 的路径指向您本地 **QCLab** 项目下的 **home** 目录。

```
{
   "server":{"home": "/Users/zhaoqin/Downloads/QCLab/home"}
}
```

> 重要提示: 请将 /Users/zhaoqin/Downloads/QCLab/home 替换为您自己电脑上的实际绝对路径。

第四步：拉取核心内容

完成配置后，运行 quark init 命令。该命令会读取 ~/quark.json 中的 home 路径，并从 Gitee 仓库自动拉取最新的驱动、门库等核心内容到该目录下。

我们使用 uv run 来确保命令在正确的虚拟环境中执行：

```
uv run quark init
```

第五步：启动与验证

现在，所有配置和依赖都已准备就绪。让我们启动后端服务来验证安装是否成功。

```
uv run quark server
```

如果您在终端看到服务器成功启动并开始输出日志信息，没有任何报错，那么恭喜您，您的开发环境已成功搭建！

---

3. 核心依赖库一览

quarkstudio[full] 会安装一系列强大的库来支撑整个测控系统。了解它们的功能有助于您更好地理解项目的工作原理和进行二次开发。

核心第三方库

- **numpy**: 用于所有数值计算的基础库，处理实验数据、扫描参数和波形数组。
- **scipy**: 提供高级科学计算功能，常用于实验结果的数据拟合与分析。
- **matplotlib**: 强大的2D绘图库，用于生成静态图表，如S21示例中的结果图。

- **pyvisa / pyserial**: 用于通过VISA标准（GPIB, TCP/IP等）或串口与物理仪器进行通信的核心库。
- **jupyter**: 支持交互式计算和数据可视化，是进行探索性实验和调试的理想环境（.ipynb 文件）。
- **fastapi** (或类似框架): 一个高性能的Web框架，用于构建 QuarkServer 的后端API服务。
- **typer / rich**: 用于构建美观、易用的命令行界面（如 quark server, quark init 等命令）。

主要内部库

除了上述通用库外，项目还包含几个自研的核心模块：

- **quark.app**: 项目的主应用程序接口（API），提供了与 QuarkServer 交互的核心工具，如 s 对象和 Recipe 类。
- **qlisp**: 定义了一套领域特定语言 (DSL)，用于以文本形式描述复杂的量子门序列，并将其解析为可执行的操作。
- **waveforms**: 负责生成、拼接和处理实验中使用的精确脉冲波形。

---

4. 配置文件说明

QuarkServer 的所有行为都由一个 JSON 配置文件驱动。这个文件定义了实验所需的所有硬件和参数。

核心步骤

1. 准备配置文件: 将您的 *.json 配置文件（例如，从一个模板复制而来）放置在 home/cfg/ 目录下。
2. 修改关键参数: 根据您的物理实验台，修改文件中的 dev (硬件设备) 部分，确保仪器的 addr (地址) 和 name (驱动) 正确无误。
3. 指定加载配置: 通过 s.signup('your_user', 'your_config_name') 命令，将您的用户名与您想要加载的配置文件（无需 .json 后缀）进行绑定。服务器启动时将加载此文件。

详细指南

关于配置文件中每个部分的详细解释，包括 dev, etc, station, gate 等，请参阅项目中的详细指南文档：

➡ **QCLab 配置文件指南 (CONFIG_GUIDE.md) (./CONFIG_GUIDE.md)**

该指南深入解释了每个参数的含义和作用，是进行高级配置和调试的必备参考。

---

5. 测试设备连接

在编写和运行完整的实验脚本之前，强烈建议先验证软件与物理仪器之间的通信是否正常。这个简单的步骤可以排除大量潜在问题。

我们将使用 s 对象在一个交互式环境（如 Jupyter Notebook 或普通 Python 脚本）中直接对仪器进行读写操作。

前提条件

- uv run quark server 进程正在运行。
- 您已经使用 s.login('your_user') 成功登录。
- 您 home/cfg/ 目录下的配置文件中，dev 部分已根据您的物理设备正确填写。

测试步骤

以下示例展示了如何与一个在配置文件中逻辑名为 "My_AWG" 的任意波形发生器进行通信。

1. 创建测试脚本：创建一个名为 test_connection.py 的文件。
2. 编写测试代码：将以下代码复制到文件中。

```python
from quark.app import s

# --- 登录到服务器 ---
# 请确保使用您在 `s.signup` 时注册的用户名
s.login('your_user')

# --- 定义要测试的设备 ---
# 将 'My_AWG' 替换为您在配置文件中为设备设置的逻辑名称
device_name = "My_AWG"
# 指定待操作的设备通道
device_channel = "CH1"
# 将 'Power' 替换为您想测试的参数，这必须是设备驱动支持的参数
parameter_to_test = "Power"

print(f"--- 开始测试设备 [{device_name}] ---")

# --- 1. 读取初始参数 ---
try:
    initial_value = s.read(f"{device_name}.{device_channel}.{parameter_to_test}")
    print(f"[成功] 读取到初始值: {initial_value}")
except Exception as e:
    print(f"[失败] 读取参数时出错: {e}")
    print("请检查：\n  1. QuarkServer 是否正在运行？\n  2. 配置文件中的设备逻辑名称是否正确？\n  3. 仪器的物理连接（如网线、电源）是否正常？")
    exit()

# --- 2. 写入新参数 ---
try:
    new_value = -20 # 假设我们要设置一个新的功率值
    print(f"\n--- 尝试写入新值: {new_value} ---")
    s.write(f"{device_name}.{device_channel}.{parameter_to_test}", new_value)
    print("[成功] 写入命令已发送。")
except Exception as e:
    print(f"[失败] 写入参数时出错: {e}")
    exit()

# --- 3. 再次读取以验证 ---
try:
    read_back_value = s.read(f"{device_name}.{device_channel}.{parameter_to_test}")
    print(f"\n--- 验证新值 ---")
    print(f"[成功] 再次读取到的值: {read_back_value}")

    # 注意：由于浮点数精度问题，直接比较可能不准确，这里仅为示例
    if read_back_value == new_value:
        print("\n[结论] 设备连接与控制测试成功！")
    else:
        print("\n[结论] 验证失败：回读的值与设置的值不符。请检查设备驱动或物理设备状态。")

except Exception as e:
    print(f"[失败] 验证读取时出错: {e}")
```

3. 运行测试：

```
uv run python test_connection.py
```

如果所有步骤都成功打印，没有任何失败信息，那么您的设备已经准备就绪，可以开始进行更复杂的实验了。如果遇到任何错误，请根据错误提示进行排查。

---

6. 编写第一个实验：S21 扫描

在确认硬件连接无误后，我们可以开始编写并运行第一个真正的量子实验。我们将以 **S21** 扫描为例，这是一个用于寻找读出谐振腔（Readout Resonator）共振频率的基础实验。

第一步：创建实验脚本

在您的项目根目录下创建一个新的 Python 文件，例如 run_s21.py。

第二步：编写 S21 实验代码

将以下代码复制到 run_s21.py 文件中。代码的每一部分都有详细的注释说明。

```
import numpy as np
import matplotlib.pyplot as plt
from quark.app import Recipe, s

# --- 1. 登录到服务器 ---
# 确保使用您在 `s.signup` 时注册的用户名
# 服务器会根据此用户名加载对应的配置文件
s.login('your_user')

# --- 2. 定义量子线路 ---
# 对于 S21 实验，线路非常简单：就是对指定的量子比特执行测量操作。
def s21_circuit(qubits: list[str], freq: float, ctx=None):
    """
    定义一个对每个 qubit 执行 'Measure' 操作的线路。
    `freq` 参数在这里虽然定义了，但我们会在 Recipe 中动态地覆盖它。
    """
    # [(('操作名', 索引), '目标比特'), ...]
    return [(('Measure', i), q) for i, q in enumerate(qubits)]

# --- 3. 构建实验配方 (Recipe) ---

# 3.1 初始化 Recipe
# - 第一个参数 's21' 是任务的名称。
# - `signal` 指定了我们关心的原始数据类型，'iq_avg' 表示平均后的 IQ 值。
rcp = Recipe('s21', signal='iq_avg')

# 3.2 指定线路和目标比特
rcp.circuit = s21_circuit
qubits_to_measure = ['Q0', 'Q1'] # <== 在这里修改您想测量的比特
rcp['qubits'] = tuple(qubits_to_measure)
```

```
# 3.3 定义扫描参数
# 我们将在每个比特的当前中心频率附近，扫描一个 +/- 10 MHz 的范围，共 101 个点。
scan_range_mhz = 10
num_points = 101
rcp['freq'] = np.linspace(-scan_range_mhz, scan_range_mhz, num_points) * 1e6

# 3.4 关联扫描参数
# 将上面定义的相对频率扫描范围 (`rcp['freq']`) 与每个比特的绝对测量频率关联起来。
for q in qubits_to_measure:
    # 从服务器查询该比特当前的中心频率
    center_freq = s.query(f'gate.Measure.{q}.params.frequency')
    # 设置扫描的绝对频率 = 中心频率 + 相对扫描范围
    rcp[f'gate.Measure.{q}.params.frequency'] = rcp['freq'] + center_freq

print("--- 实验配方 (Recipe) 构建完成 ---")

# --- 4. 提交任务并获取结果 ---

print("--- 提交 S21 任务到服务器 ---")
# `rcp.export()` 将配方转换为服务器可读的字典格式。
# `block=True` 表示程序会在此处暂停，直到实验完成。
s21_task = s.submit(rcp.export(), block=True)

print("--- 任务完成，获取结果 ---")
# `s21_task.result()` 从服务器下载实验结果。
results = s21_task.result()

# --- 5. 数据处理与可视化 ---

print("--- 正在绘制结果... ---")

# 从结果元数据中获取频率轴和信号名称
freq_axis = results['meta']['axis']['freq']['def']
signal_name = results['meta']['other']['signal'].split('|')[0]

# 获取测量数据
data = np.asarray(results['data'][signal_name])

# 为每个测量的比特绘制 S21 曲线
fig, axes = plt.subplots(1, len(qubits_to_measure), figsize=(6 * len(qubits_to_measure), 5))
if len(qubits_to_measure) == 1:
    axes = [axes] # 保证 axes 是一个可迭代对象

for i, q in enumerate(qubits_to_measure):
    # S21 信号的幅度通常与透射率相关
    amplitude = np.abs(data[:, i])

    # 找到谐振谷对应的频率
    resonant_index = np.argmin(amplitude)
    resonant_freq_ghz = freq_axis[resonant_index] / 1e9

    ax = axes[i]
```

```
    ax.plot(freq_axis / 1e6, amplitude, 'o-')
    ax.set_title(f"S21 Curve for {q}")
    ax.set_xlabel("Frequency (MHz)")
    ax.set_ylabel("Transmission Amplitude (a.u.)")
    ax.grid(True)

    # 在图上标记出谐振谷
    ax.axvline(x=resonant_freq_ghz * 1000, color='r', linestyle='--')
    ax.text(0.05, 0.95, f"Resonance: {resonant_freq_ghz:.4f} GHz",
            transform=ax.transAxes, verticalalignment='top', color='red')

fig.tight_layout()
plt.show()

print("--- 脚本执行完毕 ---")
```

第三步：运行实验

保存文件后，在您的终端中运行脚本：

```
uv run python run_s21.py
```

程序将会：

1. 连接到服务器。
2. 构建 S21 实验的配方。
3. 将任务提交给服务器执行，并等待其完成。
4. 下载结果数据。
5. 使用 matplotlib 绘制出每个比特的 S21 曲线，并标记出谐振频率点。

如果一切顺利，您将看到一个绘图窗口弹出，显示了清晰的谐振谷。这标志着您已经成功地完成了第一个量子实验！

一些有用的工具

quarkstudio提供了一些非常有用的工具，包括

1. studio，用于查看和编辑config表：uv run quark studio
2. viewer，用于实时显示测量数据：uv run quark viewer
3. canvas，用于显示所发波形数据：uv run quark canvas