第三章 Microbot 框架需求分析与概要设计

3.1 Microbot 项目整体概述

Microbot 框架是一个基于 Node.js 的支持微服务架构的物联网开发框架，主要用户是期望使用 JavaScript 进行物联网应用开发的开发者。Microbot 的主要功能包括以下两大部分：

一是将底层的硬件抽象，提供简单统一的接口，使开发者不再需要关注硬件实现细节，快速开发物联网应用。目前，Microbot 支持 Arduino、Raspberry Pi 等微处理器和基于 GPIO 和 I2C 总线的设备器件；

二是支持微服务架构，Microbot 实现了两种服务类型，基于 MQTT 的发布/订阅消息服务和基于 HTTP 的 REST API。开发者可以根据需要配置参数，调用相应的接口即可实现服务的发布或订阅，而不需要关注协议的细节实现。

3.2 Microbot 框架的需求分析

3.2.1 用例图

Microbot 框架的只有一类，即物联网应用开发者。他们通过框架来开发物联网应用，主要需求分两部分，一是对硬件的管理和操作，以实现具体应用的业务逻辑，包括配置连接件、配置设备和访问并操作设备；二是与微服务架构相关，包括服务配置、服务发布、服务订阅和取消服务；另外框架还提供了查看当前运行的应用与服务功能。用例图如下所示：

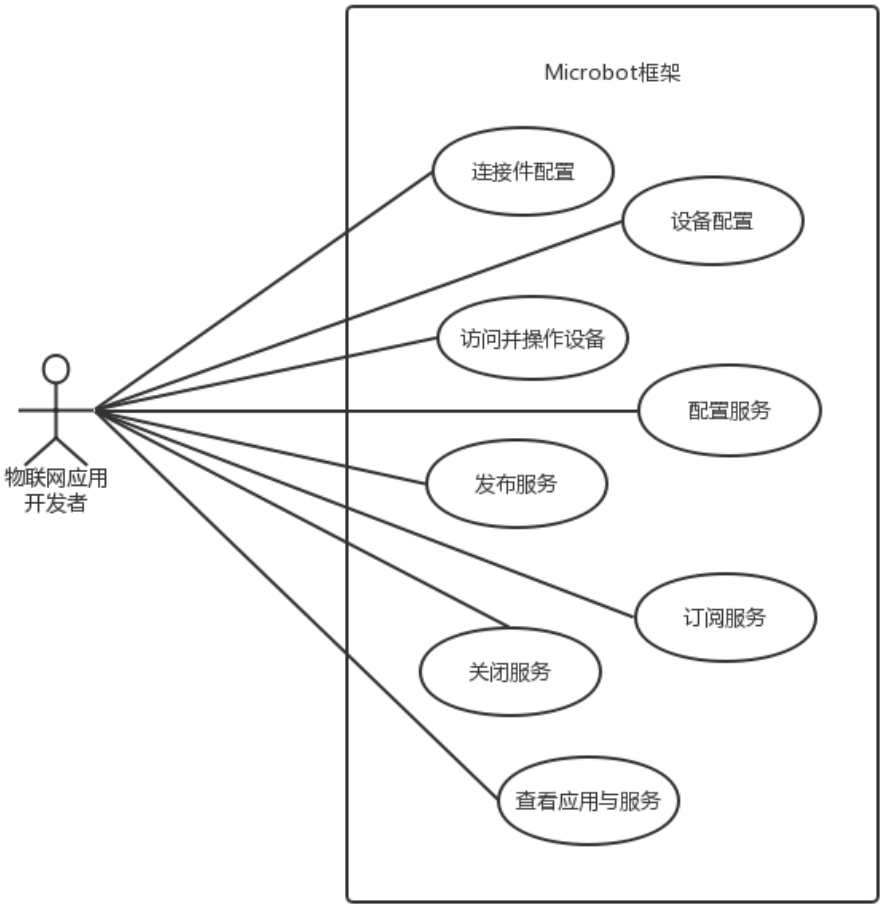

图 3.1 Microbot 框架用例图

3.2.2 功能性需求

通过分析用例图和关键用例，提取出 Microbot 的功能需求，如下表所示：

表 3.8 Microbot 框架功能需求列表

编号	需求名称	需求描述	优先级
R1	配置连接件信息	开发者能够根据开发需要配置所用到的硬件连接件信息，并在程序启动时完成硬件连接	高
R2	配置设备信息	开发者能够根据开发需要配置所用到的硬件设备信息，并在程序启动时完成设备与连接件的衔接与通信	高
R3	访问并操作设备	开发者能够在主程序中访问并操作设备来实现具体应用的业务逻辑	高
R4	配置服务信息	开发者能够通过配置相关信息将应用封装成微服务	高
R5	发布服务	开发者能够发布 MQTT 服务到 Broker 或开	高

		放 RESTful API 接口接受客户端请求	
R6	订阅服务	开发者能在应用的业务逻辑中订阅 MQTT 服务或者发送调用其他服务提供的 RESTAPI	高
R7	关闭服务	开发者能根据应用业务逻辑的需要关闭服务	中
R8	查看运行中的应用和服务	开发者能够查看当前正在运行的子应用和服务信息	中

3.2.3 非功能性需求

Microbot 的非功能性需求主要来自两个方面：

第一，Microbot 作为一个框架类项目，用户将使用框架进行二次开发，因此对框架自身的易用性要求较高；

第二，Microbot 是针对物联网应用开发的框架，物联网硬件平台和设备种类繁多，并且目前还没有非常统一的标准，框架除了要支持对目前主流硬件的支持外，还必须考虑自身的可扩展性，尽量降低新增支持某个硬件平台的成本。

表 3.9 Microbot 框架非功能需求列表

易用性	框架的学习成本低，接口和参数简单明了、命名规范，注释说明完整，易于理解；有合理的错误提示，便于开发者区分。
可扩展性	可灵活删减对硬件平台和设备的支持，不需要修改大量原有代码，不会对其他模块造成影响

3.3 Microbot 框架的概要设计

3.3.1 系统框架结构

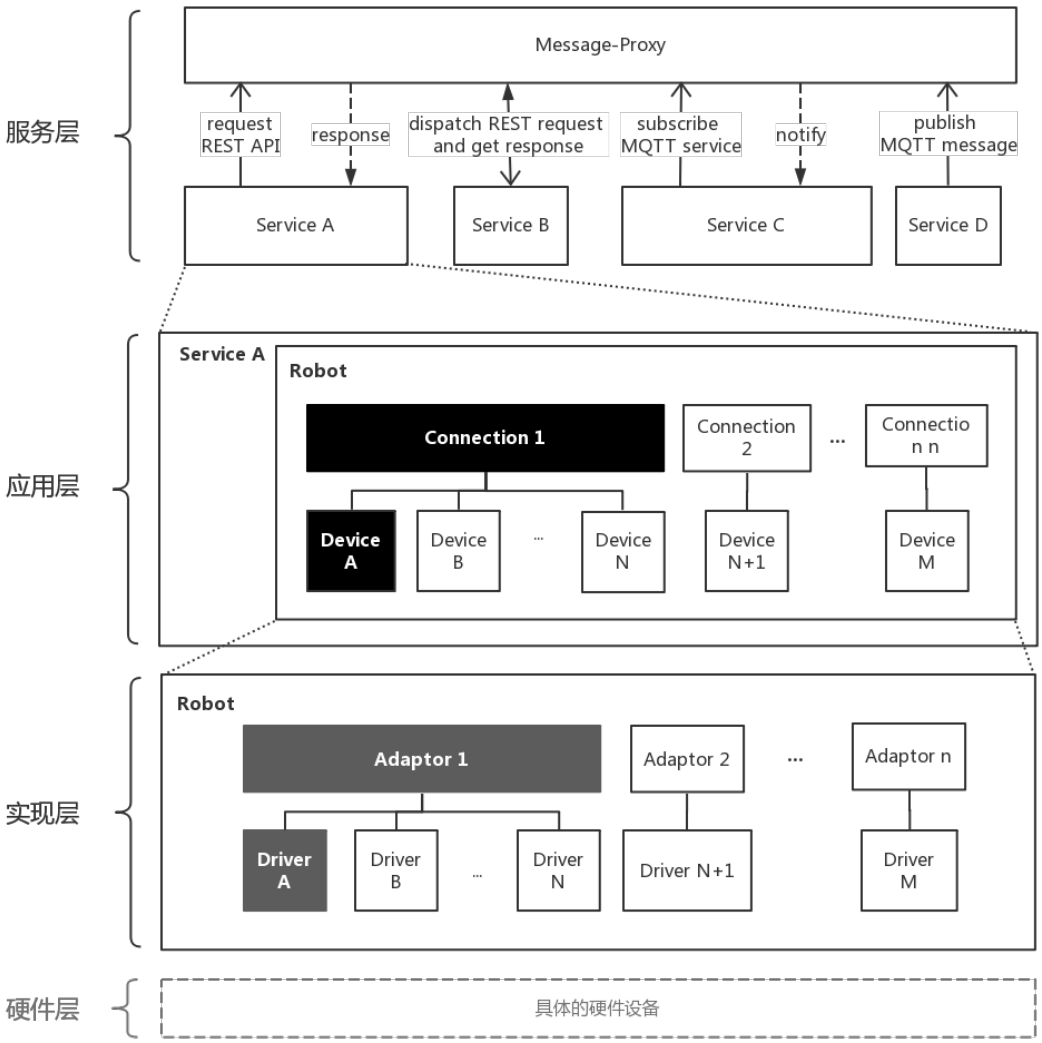

图 3.2 Microbot 框架架构分层视图

注：上图展示了 Microbot 框架的整体架构，这里按照从上到下的顺序依次加以说明：

服务层，提供对微服务架构的支持。Message-Proxy 是一个服务代理中心，所有服务间的通信均通过消息代理完成。服务实例通过配置使用的协议，按照接口说明传入参数并调用即可实现服务的请求或发布。图中 Sevice A 和 Service B 展示了对 REST API 请求的处理，服务请求方发送调用服务请求接口，Microbot 内部将通过服务代理中心接收请求并转发请求至最终服务地址，获得响应后再将响应转发给服务请求方。这一过程虽然增加了一次服务转发，但是通过服务代理中

心，可以完成诸如统一接口、格式、权限验证等增值服务，为框架使用者节省了开发成本，也能使最终代码更简洁明了。图中 Service C 和 Service D 则展示了 MQTT 服务的实现，服务发布方调用服务发布接口发布的主题和消息将由服务代理中心接管，并将其分发给服务订阅方。服务代理中心维护了一份每个主题的客户列表，当服务订阅方调用服务订阅接口订阅服务时，服务代理中心将更新客户列表信息。

应用层，可以说是现实连接的硬件在代码层次的整体高层抽象。Microbot 中把一个完整的物联网应用定义为一个 Robot，每个 Robot 可以由一个或多个连接件(Connection)组成，每个连接件对应于一个硬件平台，例如 Arduino[1]板、Raspberry Pi[2]等等。与现实中一个硬件开发板上可以连接多个硬件器件一样，每个 Connection 又可以连接一个或多个设备(Device)。连接件和设备通过在应用初始化时配置每个连接件对应的适配器（Adaptor）和设备对应的驱动（Driver）等相关信息，由框架初始化模块完成实例化过程。

实现层，是比应用层更底层一些的硬件抽象，涉及到与平台相关的具体实现。适配器(Adaptor)是硬件平台的抽象，每个硬件平台（如 Arduino）的具体实现就是一个适配器，它将硬件的功能与最终的代码开放的接口一一对应，提供给上层使用。驱动（Driver）类似 Adaptor，不过驱动是硬件设备的抽象，它对 Adaptor 有依赖关系。

硬件层，对应现实中的硬件设备，实际上不属于本框架的实现范围，放入图中主要是为了体现出整个应用的完整性。

3.3.2 模块间接口

上一小节从分层视图上介绍了 Microbot 的体系架构，本小节则从模块组成上进一步分析 Microbot 的架构和模块间接口。

由于物联网设备的特殊性，他们的功能通常都有较大的差异，接口也大相径庭。很难抽象出一个有很多公共接口父类，因此 Microbot 参考 Cylon.js 的实现，在抽象类 Adaptor 中，只提供了 connect 和 disconnect 两个公共方法给 Robot 使用，用于程序启动时完成与连接件的连接，保证硬件平台准备就绪，其他功能接口由具体实现与平台相关的子类提供；在 Driver 类中类似，只提供了 start 和 halt 两个公共方法，用于开启或关闭设备，其他功能接口由具体实现某个设备的子类提供。

---

[1] 关于 Arduino 的详细信息，参见：https://www.arduino.cc/

[2] 关于 Raspberry Pi 的详细信息，参见：https://www.raspberrypi.org/

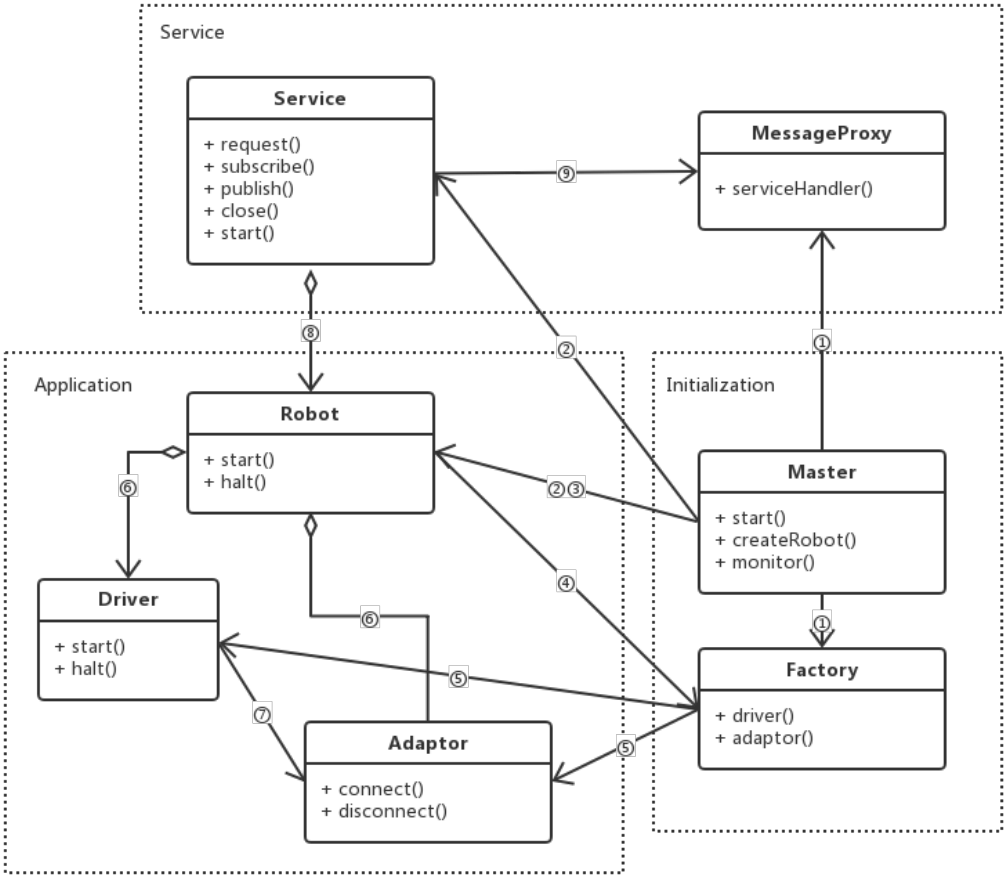

图 3.3 Microbot 框架模块视图及关键类图

注：上图展示了 Microbot 框架内部主要模块间接口的关系，按功能主要划分成 Service（服务）、Application（应用）和 Initialization（初始化）三大模块，另外还有部分辅助模块没有在上图展示，主要是工具类。具体说明如下：

① Master 的 start 方法是程序运行时的主入口，负责初始化 Factory 和 MessageProxy 的单例。

② Master 维护了当前正在运行的 Service 和 robot 的实例，通过访问本地浏览地址可以查看当前运行的服务和应用的信息。

③ Master 的 createRobot 的方法是创建 robot 的接口，开发者通过该方法创建 robot 实例。

④ Robot 在启动时调用 Factory 的工厂方法完成连接件和设备的初始化。

⑤ Factory 是一个应用了工厂模式的单例，根据传入参数生成对应的 driver 或 adaptor 的子类。

⑥ Robot 中聚合了 driver 和 adaptor，使其可以在 robot 中访问并被操作，实现 robot 的具体功能。

⑦ Driver 子类中包含 Adaptor 的依赖，二者的关系在 robot 的配置中指定，

并在初始化过程中绑定，driver 通过调用其 connect 和 disconnect 方法实现与连接件的连接。

⑧ Service 中聚合了 Robot，使得在 Service 的 API 中可以访问到 robot 实例获取数据，实现业务逻辑，对外提供服务；同时在 robot 实例中也包含 service 的依赖，使得在 robot 的主程序运行时可以发布服务或请求外部服务实现内部的复杂功能。

⑨ Service 实例调用 MessageProxy 的接口实现服务的发布、订阅或请求。

表 3.10 Microbot 框架各模块职责表

模块	**职责**
Initialization（启动模块）	负责初始化主程序，服务代理中心实例和适配器及驱动工厂单例，启动应用
Service（服务模块）	负责微服务实例化和通信
Application（应用模块）	负责物联网应用各组成部分的组建和初始化，提供业务逻辑实现接口