Prim 语言设计与实现项目计划书

项目名称：Prim — 一种基于"一切皆 Prim"哲学的动态类型语言设计与虚拟机实现

一、项目背景与动机

1.1 研究背景

现代编程语言（如 C++、Python、JavaScript）在语义层面存在概念割裂：语句、函数、对象、作用域被建模为不同的语言构造，这种设计导致以下问题：

1. **语义不统一**：不同抽象层次的元素无法在统一的模型下表达，增加了语言实现的复杂性；
2. **优化困难**：JIT 编译器难以在中间表示层面进行统一优化，必须针对不同概念实现多套优化策略；
3. **内存语义混乱**：引用与拷贝语义缺乏显式区分，难以精确控制内存分配与释放；
4. **抽象冗余**：函数、结构体、闭包等概念在本质上有相似性，但仍需分别建模。

1.2 研究目标

本项目旨在设计并实现 Prim 语言，通过"一切皆 Prim"的哲学理念，将上述问题统一化处理。具体目标包括：

1. **统一抽象模型**：将可执行逻辑统一为 Prim，静态状态统一为 Closure，实现语义的统一表达；
2. **显式内存管理**：通过 Slot/Symbol 分离机制，明确区分值语义与引用语义；
3. **虚拟机架构**：实现基于字节码的虚拟机，支持解释执行与热点即时编译（Hot Spot JIT）；
4. **可扩展设计**：架构清晰、接口明确，便于后期优化与功能扩展。

二、设计理念与总体架构

2.1 设计哲学

Prim 语言的核心哲学是"一切皆 Prim"（Everything is Prim），通过两种基本抽象构建完整的语言系统：

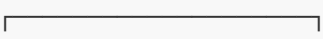
- **Prim（Primitive）**：表示所有可执行逻辑，包括语句、表达式、函数调用等。Prim 的本质是状态变换器，接受输入并产生副作用或返回值。
- **Closure**：表示静态的数据结构，是符号与槽位的绑定集合。Closure 可以是普通变量环境、数据结构实例或函数闭包。

在这种模型下，所有程序都可以表达为：**程序 = Prim（可执行逻辑）+ Closure（静态状态）**。

可变状态由 Closure 保存，执行行为由 Prim 定义，两者构成了完整的计算模型。

2.2 系统架构

Prim 语言的实现采用多阶段编译+虚拟机执行的架构，包括以下五个核心阶段：



三、核心设计原理

3.1 内存管理模型

Prim 语言采用显式的 Slot/Symbol 分离机制来管理内存：

- **Slot（槽位）**：实际存储数据的内存单元，具有引用计数属性
- **Symbol（符号）**：指向 Slot 的引用，多个 Symbol 可以指向同一个 Slot

这种设计实现了以下语义：

- **值绑定**：创建新的 Slot，复制数据内容
- **引用绑定**：多个 Symbol 共享同一个 Slot
- **自动回收**：当 Slot 的引用计数降为 0 时自动释放内存

3.2 统一抽象模型

在 Prim 语言中，所有语言构造都被统一为两种基本抽象：

1. **Prim 抽象**：所有可执行逻辑都表示为 Prim
    - 语句（if、loop、let）是立即执行的 Prim
    - 函数是延迟执行的 Prim（通过 $ 操作符捕获）
    - 表达式是返回值的 Prim
2. **Closure 抽象**：所有数据结构都表示为 Closure
    - 变量环境是 Closure
    - 对象实例是 Closure
    - 函数闭包也是 Closure

3.3 虚拟机设计

虚拟机采用基于栈的执行模型，支持以下核心功能：

- **字节码解释执行**：将 AST 编译为字节码指令序列
- **垃圾回收**：基于引用计数的轻量级内存管理
- **热点检测**：统计指令执行频率，识别热点代码
- **JIT 编译**：对热点代码进行即时编译优化

四、实现方案

4.1 技术栈选择

模块	技术选择	理由
词法分析	re2c	高性能 C++ 词法分析器生成器，支持 Unicode
语法分析	Bison	成熟的 LALR(1) 语法分析器生成器
实现语言	C++20	现代 C++ 特性支持，性能优异
构建系统	CMake + Ninja	跨平台构建，增量编译快速
测试框架	GoogleTest	完善的单元测试支持

4.2 实现阶段

阶段一：词法与语法分析（10月下旬-11月上旬）

- 使用 re2c 实现词法分析器，支持标识符、数字、字符串、操作符等基本 Token
- 使用 Bison 实现语法分析器，构建完整的 AST 结构
- 实现基本的语法错误检测与报告机制

阶段二：字节码编译器（11月中旬-11月下旬）

- 设计字节码指令集，包括算术运算、控制流、函数调用等指令
- 实现 AST 到字节码的编译过程
- 支持基本的变量绑定与作用域管理

阶段三：虚拟机实现（12月上旬-12月中旬）

- 实现基于栈的虚拟机执行引擎
- 实现 Slot/Symbol 内存管理系统
- 实现基本的垃圾回收机制

阶段四：热点 JIT 优化（12月下旬）

- 实现指令执行频率统计
- 设计热点代码检测算法
- 预留 LLVM IR 生成接口，为 JIT 编译做准备

五、预期成果与创新点

5.1 预期成果

功能性目标

- **完整的语言实现**：支持基本语法结构（变量、控制流、函数、闭包）
- **虚拟机系统**：基于字节码的解释执行引擎
- **内存管理**：基于引用计数的自动垃圾回收
- **性能优化**：热点代码检测与 JIT 编译接口
- **开发工具**：REPL 交互式环境与调试支持

技术指标

- **代码规模**：预计实现约 3000 行 C++20 代码
- **性能目标**：解释执行性能达到 Python 的 50% 以上
- **内存效率**：支持引用计数的轻量级内存管理
- **可扩展性**：模块化设计，便于功能扩展

5.2 创新点

1. **统一抽象模型**：通过 Prim/Closure 二元模型统一表达所有语言构造，消除了传统语言中语句、函数、对象等概念的割裂。
2. **显式内存语义**：Slot/Symbol 分离机制明确区分值语义与引用语义，为精确的内存管理提供基础。
3. **虚拟机架构**：采用多阶段编译+虚拟机执行的架构，为后续 JIT 优化提供良好的基础。
4. **热点优化**：集成热点检测与 JIT 编译接口，为性能关键代码提供优化路径。
5. **教育价值**：清晰的架构设计便于理解语言实现原理，具有重要的教学价值。

六、进度安排与风险评估

6.1 进度安排

主要工作内容	里程碑
语言规范设计与文档编写	完成项目计划书
词法分析器实现（re2c）	输出标准 Token 流
语法分析器实现（Bison）	构建完整 AST
字节码编译器实现	生成可执行字节码
虚拟机实现	支持基本程序执行
内存管理与垃圾回收	稳定的运行时系统
热点 JIT 接口设计	完成项目答辩

七、结论

Prim 语言项目通过"一切皆 Prim"的设计哲学，旨在解决现代编程语言中存在的语义割裂问题。通过统一的 Prim/Closure 抽象模型、显式的内存管理机制以及多阶段编译+虚拟机执行的架构，本项目将为编程语言设计提供新的思路。

项目的成功实施将验证统一抽象模型的可行性，为后续的语言优化与扩展奠定基础。同时，清晰的架构设计也将为编程语言教学提供有价值的参考案例。