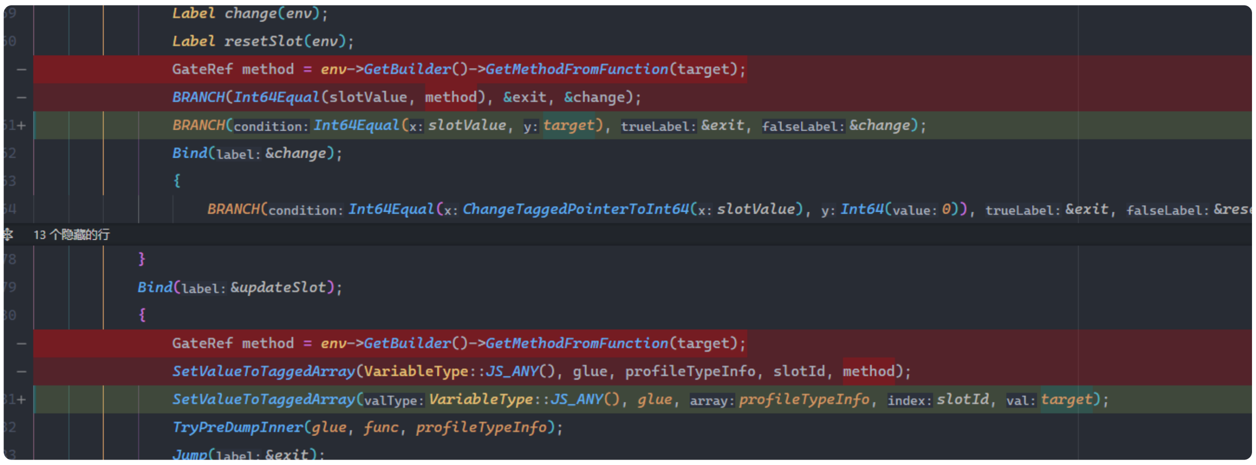
inline 实现方案

inline时需要callee的一些信息，依赖callee对应的JSFunction.

1. 通过ProfileTypeInfo传递JSFunction

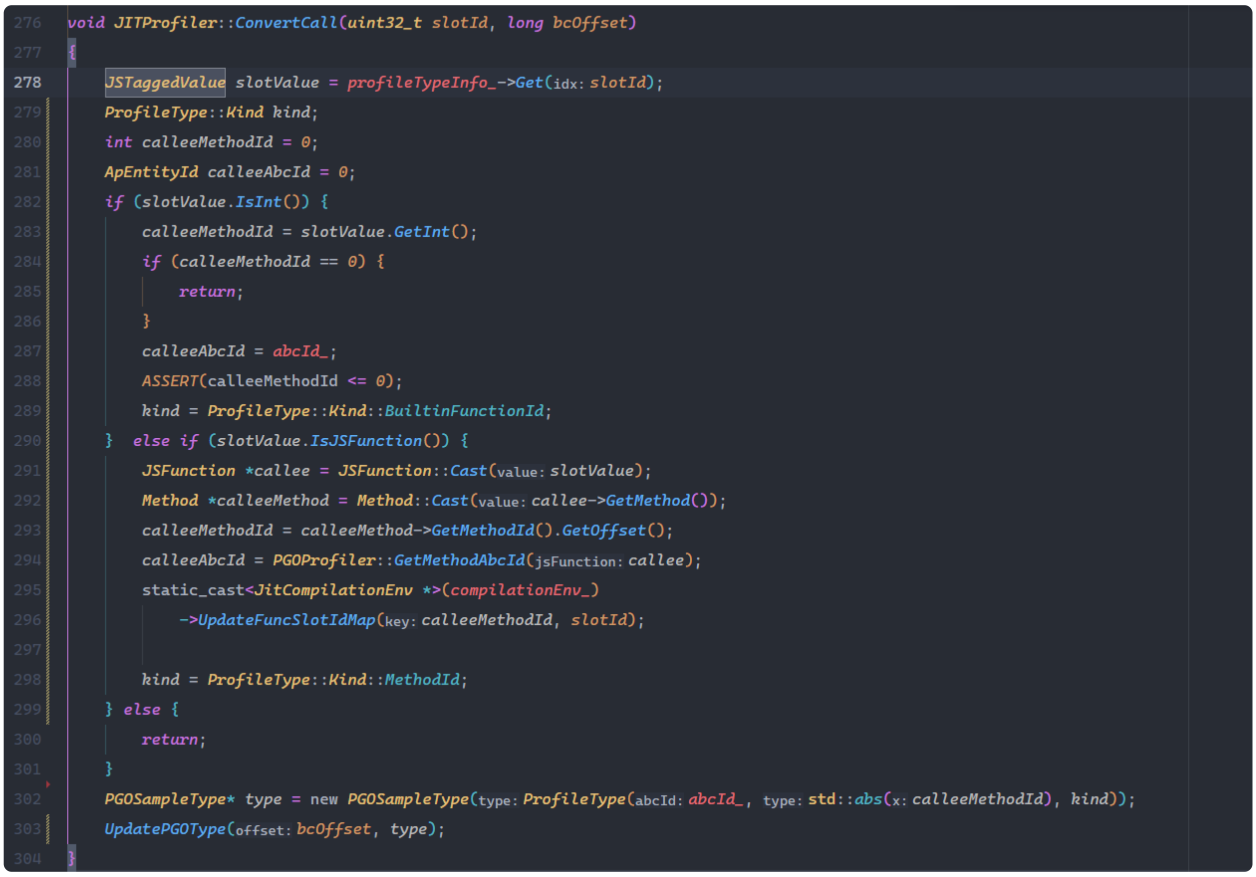
`ProfilerStubBuilder::ProfileCall`原本在对应的slot放置一个`Method*`，修改为放置一个`JSFunction*`



```
        Label change(env);
        Label resetSlot(env);
-       GateRef method = env->GetBuilder()->GetMethodFromFunction(target);
-       BRANCH(Int64Equal(slotValue, method), &exit, &change);
+       BRANCH(condition: Int64Equal(x: slotValue, y: target), trueLabel: &exit, falseLabel: &change);
        Bind(label: &change);
        {
            BRANCH(condition: Int64Equal(x: ChangeTaggedPointerToInt64(x: slotValue), y: Int64(value: 0)), trueLabel: &exit, falseLabel: &rese
13 个隐藏的行
    }
    Bind(label: &updateSlot);
    {
-       GateRef method = env->GetBuilder()->GetMethodFromFunction(target);
-       SetValueToTaggedArray(VariableType::JS_ANY(), glue, profileTypeInfo, slotId, method);
+       SetValueToTaggedArray(valType: VariableType::JS_ANY(), glue, array: profileTypeInfo, index: slotId, val: target);
        TryPreDumpInner(glue, func, profileTypeInfo);
        Jump(label: &exit);
```

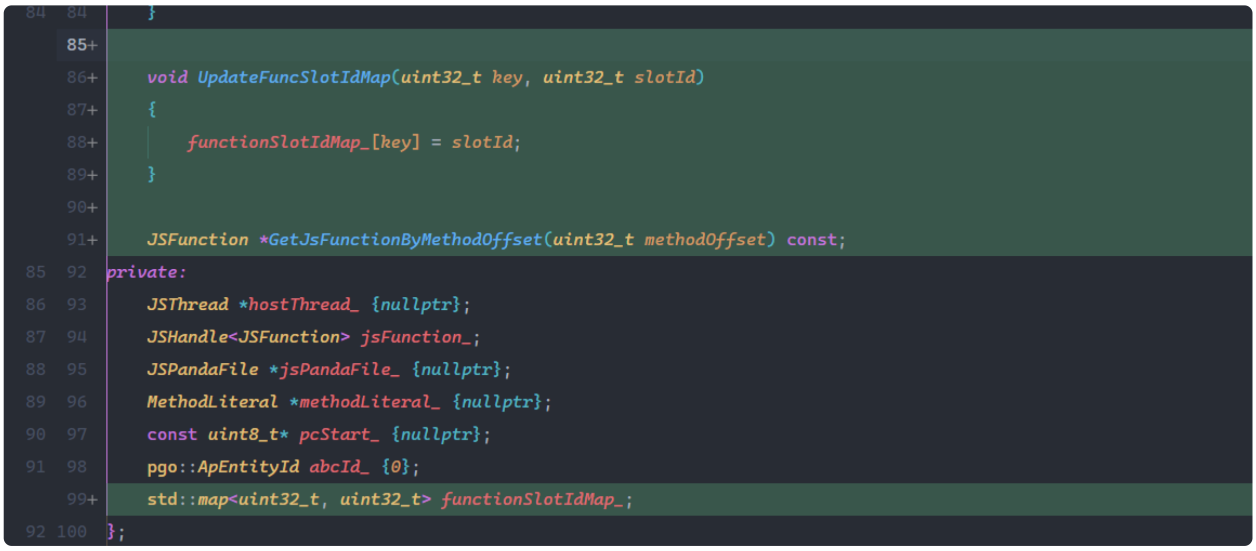
2. JITProfiler::ConvertCall完善

这一步主要是处理pgo采集的数据，转换为PGOType


```cpp
void JITProfiler::ConvertCall(uint32_t slotId, long bcOffset)
{
    JSTaggedValue slotValue = profileTypeInfo_->Get(idx: slotId);
    ProfileType::Kind kind;
    int calleeMethodId = 0;
    ApEntityId calleeAbcId = 0;
    if (slotValue.IsInt()) {
        calleeMethodId = slotValue.GetInt();
        if (calleeMethodId == 0) {
            return;
        }
        calleeAbcId = abcId_;
        ASSERT(calleeMethodId <= 0);
        kind = ProfileType::Kind::BuiltinFunctionId;
    }  else if (slotValue.IsJSFunction()) {
        JSFunction *callee = JSFunction::Cast(value: slotValue);
        Method *calleeMethod = Method::Cast(value: callee->GetMethod());
        calleeMethodId = calleeMethod->GetMethodId().GetOffset();
        calleeAbcId = PGOProfiler::GetMethodAbcId(jsFunction: callee);
        static_cast<JitCompilationEnv *>(compilationEnv_)
            ->UpdateFuncSlotIdMap(key: calleeMethodId, slotId);

        kind = ProfileType::Kind::MethodId;
    } else {
        return;
    }
    PGOSampleType* type = new PGOSampleType(type: ProfileType(abcId: abcId_, type: std::abs(x: calleeMethodId), kind));
    UpdatePGOType(offset: bcOffset, type);
}
```

`PGOProfiler::DumpCall`也需要做一些修改，以对应ProfileTypeInfo中的变化

3. 保存callee的JSFunction


```cpp
    }

    void UpdateFuncSlotIdMap(uint32_t key, uint32_t slotId)
    {
        functionSlotIdMap_[key] = slotId;
    }

    JSFunction *GetJsFunctionByMethodOffset(uint32_t methodOffset) const;
private:
    JSThread *hostThread_ {nullptr};
    JSHandle<JSFunction> jsFunction_;
    JSPandaFile *jsPandaFile_ {nullptr};
    MethodLiteral *methodLiteral_ {nullptr};
    const uint8_t* pcStart_ {nullptr};
    pgo::ApEntityId abcId_ {0};
    std::map<uint32_t, uint32_t> functionSlotIdMap_;
};
```

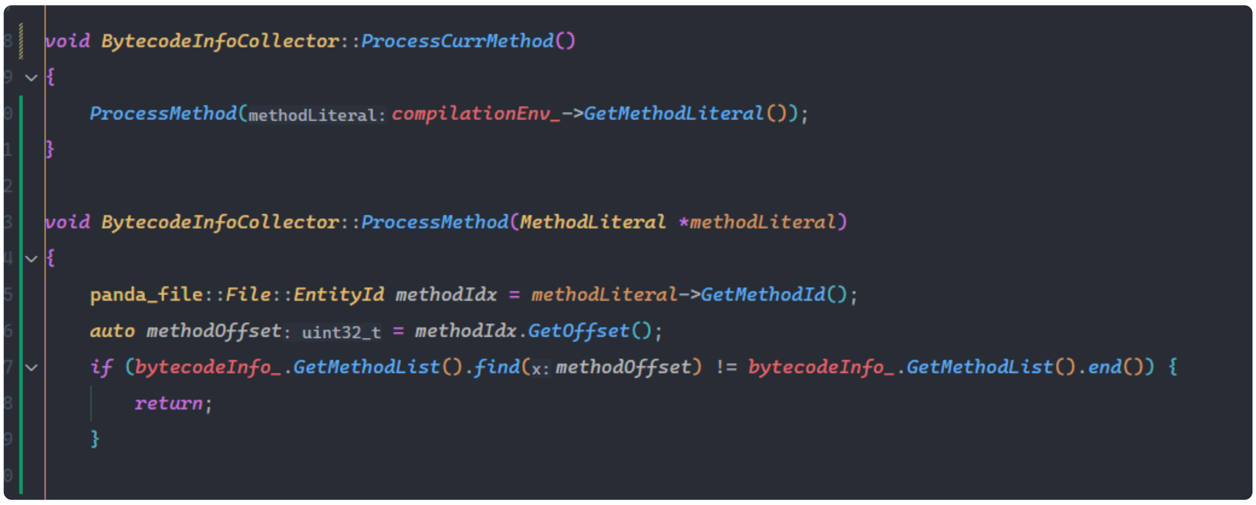
将`JSFunction*`对应的MethodOffset->slotId的映射保存在`JitCompilationEnv`中，这样后续优化可以从中获取callee对应的

JSFunction在caller的ProfileTypeInfo中的slotId，进而获取JSFunction*

4. inline优化的补全

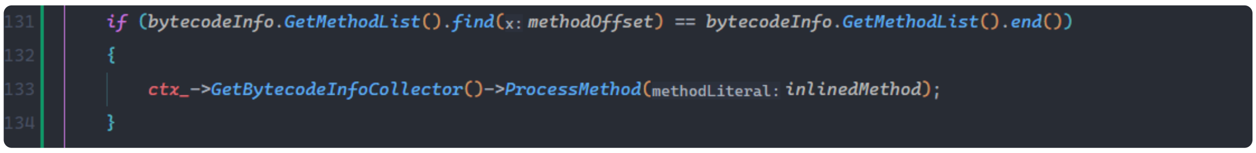
1. BCInfo

inline需要callee对应的BCInfo，而Ctx中只保存了当前caller的Caller的BCInfo在`BytecodeInfoCollector::ProcessMethod`中获取，可以稍作修改以获取callee对应的BCInfo



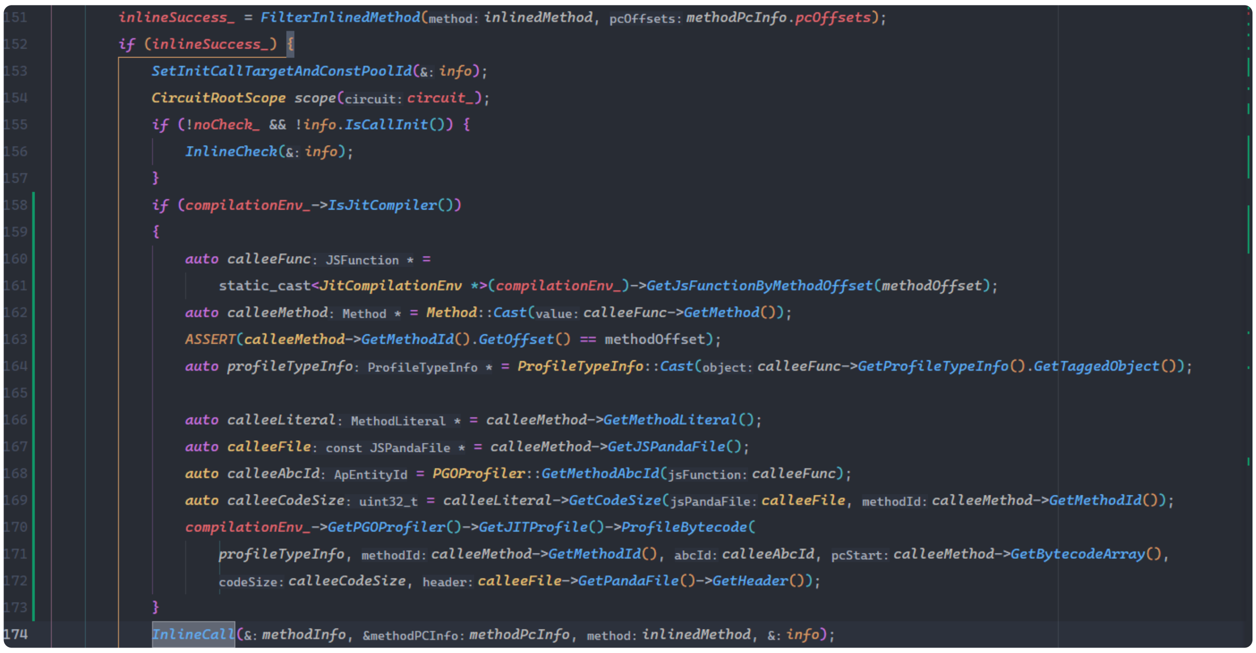
```
void BytecodeInfoCollector::ProcessCurrMethod()
{
    ProcessMethod(methodLiteral: compilationEnv_->GetMethodLiteral());
}

void BytecodeInfoCollector::ProcessMethod(MethodLiteral *methodLiteral)
{
    panda_file::File::EntityId methodIdx = methodLiteral->GetMethodId();
    auto methodOffset: uint32_t = methodIdx.GetOffset();
    if (bytecodeInfo_.GetMethodList().find(x: methodOffset) != bytecodeInfo_.GetMethodList().end()) {
        return;
    }
```

inline时调用



```
    if (bytecodeInfo.GetMethodList().find(x: methodOffset) == bytecodeInfo.GetMethodList().end())
    {
        ctx_->GetBytecodeInfoCollector()->ProcessMethod(methodLiteral: inlinedMethod);
    }
```

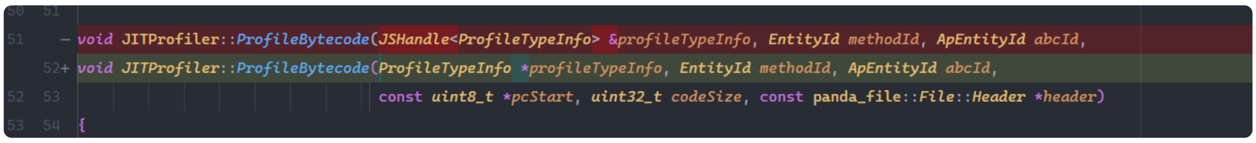
2. 满足inline条件时获取callee的ProfileTypeInfo转换为PGOType



```cpp
inlineSuccess_ = FilterInlinedMethod(method: inlinedMethod, pcOffsets: methodPcInfo.pcOffsets);
if (inlineSuccess_) {
    SetInitCallTargetAndConstPoolId(&: info);
    CircuitRootScope scope(circuit: circuit_);
    if (!noCheck_ && !info.IsCallInit()) {
        InlineCheck(&: info);
    }
    if (compilationEnv_->IsJitCompiler())
    {
        auto calleeFunc: JSFunction * =
            static_cast<JitCompilationEnv *>(compilationEnv_)->GetJsFunctionByMethodOffset(methodOffset);
        auto calleeMethod: Method * = Method::Cast(value: calleeFunc->GetMethod());
        ASSERT(calleeMethod->GetMethodId().GetOffset() == methodOffset);
        auto profileTypeInfo: ProfileTypeInfo * = ProfileTypeInfo::Cast(object: calleeFunc->GetProfileTypeInfo().GetTaggedObject());

        auto calleeLiteral: MethodLiteral * = calleeMethod->GetMethodLiteral();
        auto calleeFile: const JSPandaFile * = calleeMethod->GetJSPandaFile();
        auto calleeAbcId: ApEntityId = PGOProfiler::GetMethodAbcId(jsFunction: calleeFunc);
        auto calleeCodeSize: uint32_t = calleeLiteral->GetCodeSize(jsPandaFile: calleeFile, methodId: calleeMethod->GetMethodId());
        compilationEnv_->GetPGOProfiler()->GetJITProfile()->ProfileBytecode(
            profileTypeInfo, methodId: calleeMethod->GetMethodId(), abcId: calleeAbcId, pcStart: calleeMethod->GetBytecodeArray(),
            codeSize: calleeCodeSize, header: calleeFile->GetPandaFile()->GetHeader());
    }
    InlineCall(&: methodInfo, &methodPCInfo: methodPcInfo, method: inlinedMethod, &: info);
```

因为这里无法构造带JSHandle的profileTypeInfo，将

JITProfiler::ProfileBytecode的第一个参数修改为裸指针



```diff
- void JITProfiler::ProfileBytecode(JSHandle<ProfileTypeInfo> &profileTypeInfo, EntityId methodId, ApEntityId abcId,
+ void JITProfiler::ProfileBytecode(ProfileTypeInfo *profileTypeInfo, EntityId methodId, ApEntityId abcId,
                                    const uint8_t *pcStart, uint32_t codeSize, const panda_file::File::Header *header)
  {
```