Oilpan

Garbage Collection for Blink

Mads Ager, ager@chromium.org

Outline

- Introduction: Reference Counting vs Tracing
- Oilpan: Mostly Precise GC for Blink
- Programming Blink with Oilpan
- Status Update
- Q&A

Reference Counting

- Reference counting
  - Automatic memory management technique
  - Associate counter with all objects
  - Count using smart pointers (RefPtr)
  - When count reaches zero, deallocate object

Reference Counting

- Reference counting issues
  - Cycles cannot be (easily) reclaimed
    - Leads to raw pointers to break cycles,
    - or hacks to ensure that objects die together.
  - Hard to understand and bad for security

Reference Counting

- Reference counting issues

```
class HTMLMediaElement {
public:
    ~HTMLMediaElement() { if (m_textTracks) m_textTracks->clearOwnerNode(); }
private:
    RefPtr<TextTrackList> m_textTracks;
};


class TextTrackList {
public:
    void clearOwnerNode() { m_owner = 0; }
    ExecutionContext* context() { return m_owner->context(); }
private:
    HTMLMediaElement* m_owner;
};
```

Reference Counting

- Reference counting issues

```
class IDBRequest { RefPtr<IDBCursor> m_result; };
class IDBCursor { RefPtr<IDBRequest> m_request; };

void IDBRequest::checkForReferenceCycle()
{
    // If this request and its cursor have the only references
    // to each other, then explicitly break the cycle.
    IDBCursor* cursor = getResultCursor();
    if (!cursor || cursor->request() != this)
        return;

    if (!hasOneRef() || !cursor->hasOneRef())
        return;

    m_result.clear();
}
```

Reference Counting

- Reference counting issues
  - It is just too slow to inc/dec the ref count all the time
  - Cheat! Someone else has a ref; I can just use a raw pointer!
  - But then what if that ref disappears?

```
void ChildNodeInsertionNotifier::notifyNodeInsertedIntoDocument(Node* node)
{
    ASSERT(m_insertionPoint.inDocument());
    RefPtr<Node> protect(node);
    // Do stuff that might remove a ref to node.
    ...
    // Then use node again.
    if (node->isContainerNode())
        ...;
}
```

Tracing

- Tracing
  - Automatic memory management technique
  - Compute transitive reachability in object graph starting from roots
  - Anything that is not reachable can be reused

Tracing

- Tracing, the good part
  - Cycles are not a problem!
    - Unreachable cycles are reclaimed
    - No more back pointer clearing
    - No more hacks to make object structures live and die together
  - No cost when updating pointer structure
    - No more cheating and protect pointers

Tracing

- Tracing issues
  - All pointers to objects in the heap need to be known to the garbage collector:
    - from the stack,
    - from objects outside the heap, and
    - between objects in the heap
  - Garbage collection pause time?

Outline

- Introduction: Reference Counting vs Tracing
- Oilpan: Mostly Precise GC for Blink
- Programming Blink with Oilpan
- Status Update
- Q&A

Oilpan: Mostly Precise GC for Blink

- Oilpan: a tracing GC for Blink
- Precise tracing of pointers into the heap for
  - off-heap objects,
  - between objects in the heap
- Conservative tracing of pointers on the stack
- Mostly precise because we postpone GCs to the event loop where there is no stack!

Oilpan

- Oilpan implements a simple stop the world mark-sweep GC
  - When we need to GC, get all threads to a safepoint
  - Compute transitive reachability starting from roots and mark all objects that are reachable
  - Each thread then sweeps its part of the heap and reclaims (adds to free-lists) anything that was not marked

Oilpan

- Mark-sweep requirements
  - Given a pointer to an object, we need to know the size of that object for sweeping
  - Needs room for a mark bit
  - Needs to be able to find all pointers from a given heap object to other heap objects during tracing

Oilpan

- Blink objects are just C++ objects
- We don’t control object layout!

Oilpan

- Allocate objects with a two-word header containing size, mark bit and type information

- Let the objects themselves tell the garbage collector about pointers using a visitor pattern

Outline

- Introduction: Reference Counting vs Tracing
- Oilpan: Mostly Precise GC for Blink
- Programming Blink with Oilpan
- Status Update
- Q&A

Programming Blink with Oilpan

- Getting objects allocated in the Oilpan heap
- Implementing tracing
- Dealing with finalization-order issues

Programming Blink with Oilpan

- Getting objects allocated in the Oilpan heap
  - GarbageCollectedFinalized super class
    - Overrides new/delete
    - Call destructor when object is GCd
  - GarbageCollected super class
    - Same as GarbageCollectedFinalized except that the destructor of the object is not called

Programming Blink with Oilpan

- Getting objects allocated in the Oilpan heap

```
class ClientRect FINAL
    : public RefCounted<ClientRect>
    , public ScriptWrappable {
};
```

Programming Blink with Oilpan

- Getting objects allocated in the Oilpan heap

```
class ClientRect FINAL
    : public GarbageCollectedFinalized<ClientRect>
    , public ScriptWrappable {
};
```

Programming Blink with Oilpan

- Getting objects allocated in the Oilpan heap

```
class ClientRect FINAL
    : public GarbageCollectedFinalized<ClientRect>
    , public ScriptWrappable {
};

class ScriptWrappable {
public:
    ~ScriptWrappable()
    {
        doSecurityRelatedPointerManipulation();
    }
};
```

Programming Blink with Oilpan

- GarbageCollected sounds dangerous!

```
class ClientRect FINAL
    : public GarbageCollected<ClientRect>
    , public ScriptWrappable {
};

class ScriptWrappable {
public:
    // This destructor never called for ClientRects.
    ~ScriptWrappable()
    {
        doSecurityRelatedPointerManipulation();
    }
};
```

Programming Blink with Oilpan

- Clang plugin to the rescue:

```
Source/core/dom/ClientRect.h:40:1: error: [blink-gc] Class 'ClientRect' requires finalization.
class ClientRect FINAL : public GarbageCollected<ClientRect>, public ScriptWrappable {
^
Source/core/dom/ClientRect.h:40:79: note: [blink-gc] Base class 'ScriptWrappable' requiring
finalization declared here:
class ClientRect FINAL : public GarbageCollected<ClientRect>, public ScriptWrappable {
```

Programming Blink with Oilpan

- Implementing tracing
  - Pointers between heap objects: Member<T> and virtual void trace(Visitor*)
  - Pointers from non-heap objects: Persistent<T>
  - Pointers on the stack are conservatively scanned if not at event loop

Programming Blink with Oilpan

- Implementing tracing: inter-object pointers

```
class Node : public GarbageCollectedFinalized<Node> {
public:
    virtual void trace(Visitor*) { ... }
};
class ContainerNode : public Node {
public:
    virtual void trace(Visitor*) OVERRIDE { ... }
};
class Document : public ContainerNode {
public:
    virtual void trace(Visitor* visitor) OVERRIDE
    {
        visitor->trace(m_elemSheet);
        visitor->trace(m_cssCanvasElements);
        ContainerNode::trace(visitor);
    }
private:
    Member<CSSStyleSheet> m_elemSheet;
    HeapHashMap<String, Member<HTMLCanvasElement> > m_cssCanvasElements;
};
```

Programming Blink with Oilpan

- That looks dangerous! Forgetting to trace?!?

```
class Document : public ContainerNode {
public:
    virtual void trace(Visitor* visitor) OVERRIDE
    {
        // Not tracing anything! Dangling pointers?
    }
private:
    Member<CSSStyleSheet> m_elemSheet;
    HeapHashMap<String, Member<HTMLCanvasElement> > m_cssCanvasElements;
};
```

Programming Blink with Oilpan

- Clang plugin to the rescue!

```
Source/core/dom/Document.cpp:5679:1: error: [blink-gc] Base class 'ContainerNode'
of derived class 'Document' requires tracing.
void Document::trace(Visitor* visitor)
^
Source/core/dom/Document.cpp:5679:1: error: [blink-gc] Class 'Document' has
untraced fields that require tracing.
../../third_party/WebKit/Source/core/dom/Document.h:1215:5: note: [blink-gc]
Untraced field 'm_elemSheet' declared here:
    Member<CSSStyleSheet> m_elemSheet;
    ^
Source/core/dom/Document.h:1307:5: note: [blink-gc] Untraced field
'm_cssCanvasElements' declared here:
    HeapHashMap<String, Member<HTMLCanvasElement> > m_cssCanvasElements;
    ^
```

Programming Blink with Oilpan

- Implementing tracing: pointer from non-heap objects

```
class DOMWindow : public GarbageCollectedFinalized<DOMWindow> {
public:
    void trace(Visitor) { … }
};
class Frame : public RefCounted<Frame> {
public:
    DOMWindow* domWindow() const { return m_domWindow; }
private:
    Persistent<DOMWindow> m_domWindow;
}
```

Programming Blink with Oilpan

- Implementing tracing: pointer on stack
- Just use raw pointers for arguments and local variables
- Will be found by conservative stack scanning if needed

```
void ChildNodeInsertionNotifier::notifyNodeInsertedIntoDocument(Node* node)
{
    ASSERT(m_insertionPoint.inDocument());
    // Do stuff that might remove a pointer to node.
    ...
    // Then use node again. Safe, no protection needed.
    if (node->isContainerNode())
        ...;
}
```

Programming Blink with Oilpan

- Finalization-order issues
  - Objects that have traced Member pointers to each other die together
  - There are no destruction order guarantees
  - Destructors cannot touch other GCd objects because they could have been already destructed!

Programming Blink with Oilpan

- Finalization-order issues
  - Most of the time you don’t have to touch other objects in destructors!
  - Destructors used for clearing out pointers
    - Let the objects involved live and die together
    - Many destructors become empty
  - Destructors used for clearing out actual weak pointers
    - Oilpan has support for real weak pointers!

Programming Blink with Oilpan

- Finalization-order issues: WeakMember
  - Sometimes you do want to observe and not keep alive
  - Use WeakMember pointers for such cases
  - WeakMember pointers need to be traced in the trace method
  - WeakMember pointers do not keep the target alive
  - WeakMember pointers are set to zero on GC if the target is not reachable from any strong pointers

Programming Blink with Oilpan

- Finalization-order issues: WeakMember
  - With WeakMember:

```
class Observee : public GarbageCollected<Observee> { };

class Observer : public GarbageCollected<Observer> {
public:
    void Observer(Observee* observee) : m_observee(observee) { }
    void doStuff() { if (m_observee) m_observee->doStuff(); }
    void trace(Visitor* visitor) { visitor->trace(m_observee); }
private:
    WeakMember<Observee> m_observee;
};
```

Programming Blink with Oilpan

- Finalization-order issues: WeakMember
  - Before Oilpan:

```
class Observee {
public:
    ~Observee()
    {
        HashSet<Observer*>:const_iterator it = m_observers.begin();
        for (; it != m_observers.end(); ++it)
            it->clearObservee();
    }
    void registerObserver(Observer* o) { m_observers.add(o); }
    void unregisterObserver(Observer* o) { m_observers.remove(o); }
private:
    HashSet<Observer*> m_observers;
};
```

Programming Blink with Oilpan

- Finalization-order issues: WeakMember
  - Before Oilpan:

```
class Observer {
public:
    Observer(Observee* observee) : m_observee(observee)
    {
        observee->registerObserver(this);
    }
    ~Observer()
    {
        if (m_observee)
            m_observee->unregisterObserver(this);
    }
    void clearObservee() { m_observee = nullptr; }
    void doStuff() { if (m_observee) m_observee->doStuff(); }
private:
    Observee* m_observee;
};
```

Programming Blink with Oilpan

- Finalization-order issues
  - Cannot touch potentially dead objects in destructors
  - Usually easy to avoid
    - Prefer strong pointer relationships where objects die together
    - If you don’t want to keep something alive, use weak processing locally in the object that holds the weak pointer

Outline

- Introduction: Reference Counting vs Tracing
- Oilpan: Mostly Precise GC for Blink
- Programming Blink with Oilpan
- Status Update
- Q&A

Status

- Simple modules shipping with Oilpan by default in Chrome 36
- Trade-off
  - Getting input as soon as possible, vs
  - getting full performance picture before shipping

Status

- CSS and Node hierarchies under way with transition types (urgh!)

```
class Node : public TreeSharedWillBeRefCountedGarbageCollected<Node> {
public:
    virtual void trace(Visitor*) { }
};

class ContainerNode : public Node {
public:
    virtual void trace(Visitor* visitor) OVERRIDE
    {
        visitor->trace(m_firstChild);
        visitor->trace(m_lastChild);
        Node::trace(visitor);
    }
private:
    RawPtrWillBeMember<Node> m_firstChild;
    RawPtrWillBeMember<Node> m_lastChild;
};
```

Status

- CSS hierarchy is mostly moved
- Nodes are in the Oilpan heap and traced
  - Still need to remove all RefPtrs though
- Expecting to finish Node hierarchy at end of quarter
- Then measure performance for real and optimize before shipping!

Status

- Performance (from experimental branch)
- Benchmark performance on par
  - Some better, some worse
- Pause time
  - GMail pause time on the experimental branch is typically around 10 ms
  - Lots of things we can do to improve
  - (reference counting has pauses too)
- Conservative stack scanning
  - Almost always followed by precise GC at event loop
  - On GMail the worst case kept ~1 MB alive

Q&A

Oilpan Team: oilpan-reviews@chromium.org

Google, Aarhus: Mads Ager, Erik Corry, Vyacheslav Egorov, Ian Zerny, Gustav Wibling

Google, Tokyo: Kentaro Hara, Kent Tamura, Kouhei Ueno, Keishi Hattori

Opera, Oslo: Sigbjørn Finne