Reusable Inline Caching for JavaScript Performance

> Jiho Choi, Thomas Shull, and Josep Torrellas. 2019. Reusable inline caching for JavaScript performance. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019). Association for Computing Machinery, New York, NY, USA, 889–901. https://doi.org/10.1145/3314221.3314587

本文提出了一种提高内联缓存启动性能的方案——通过将内联缓存的信息以上下文无关的形式保存，并在不同的执行之间重用，从而减少内联缓存miss，提升页面的初始化时间。

背景

隐藏类

> javascript - V8中的隐藏类（Hidden Classes）和内联缓存（Inline Caching） - 个人文章 - SegmentFault 思否
>
> Maps (Hidden Classes) in V8
>
> Objects in v8

JavaScript作为一个动态语言，它的对象可以动态地增加和删除属性，例如下面的例子中，可以给car对象动态地添加一个属性year。因此在规范中JS的对象被实现为一种字典。

```
function Car(make,model) {
    this.make = make;
    this.model = model;
}

const car = new Car(honda,accord);

car.year = 2005;
```

这种动态的类型阻碍了很多编译以及运行时优化，如果所有的属性访问都通过字典查表的形式进行那么JS的性能就太过低效。为了使能一些基于类型的优化，现代JS执行引擎使用了一种称为隐藏类的数据结构来描述一个对象的内存布局。

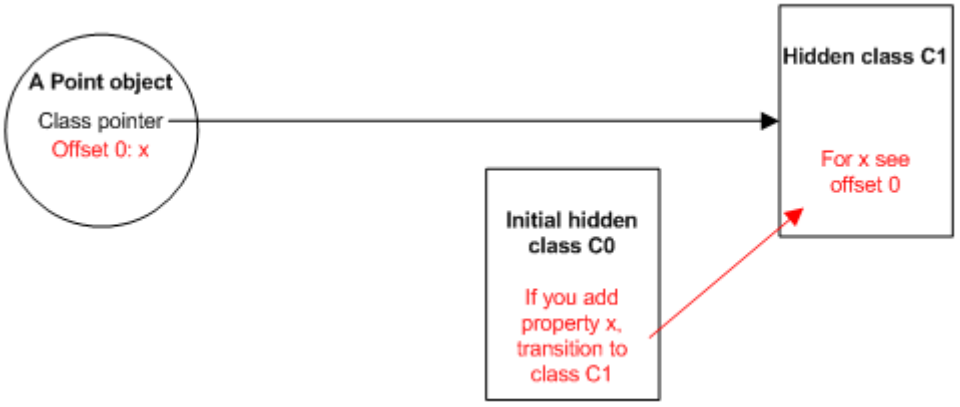
隐藏类有一个重要的性质：每个不同的结构布局由一个唯一的隐藏类标识，这样就可以通过比较隐藏类的地址来判断是否布局相同。我们可以把隐藏类看做一个在运行时维护的类型描述符。

以下面这个代码为例，我们来尝试说明隐藏类的创建和性质。

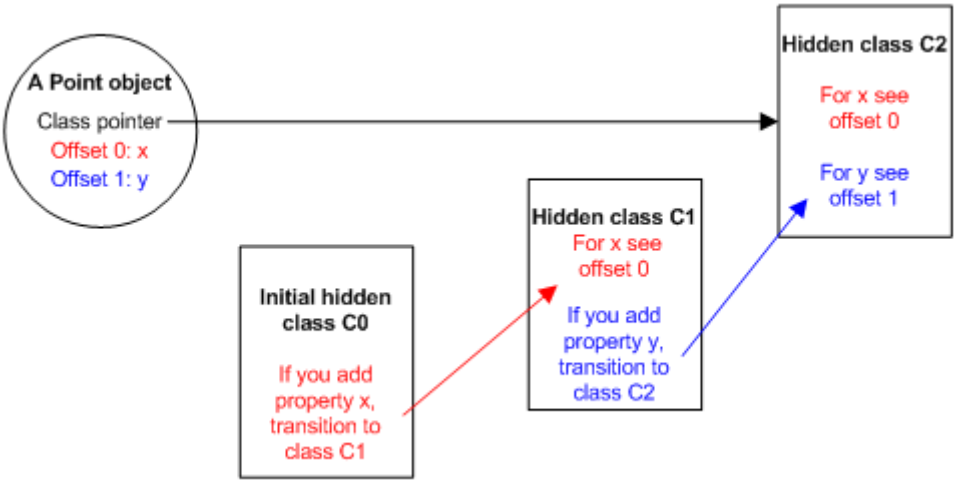
```
function Point(x,y) {
    this.x = x;
    this.y = y;
}
```

Point函数作为一个构造函数，在刚进入函数时会有一个空的隐藏类C0被创建，C0表示这个类型没有任何属性。

执行到`this.x = x`时，会基于C0创建一个新的隐藏类C1，C1描述了一个只有一个属性并且属性名为x的类型。同时，运行时会记录C0是从C1添加一个属性x而得来的，从而在下次重复这个操作的时候不需要重新创建一个新的隐藏类对象来描述这一类型。



然后是`this.y = y`

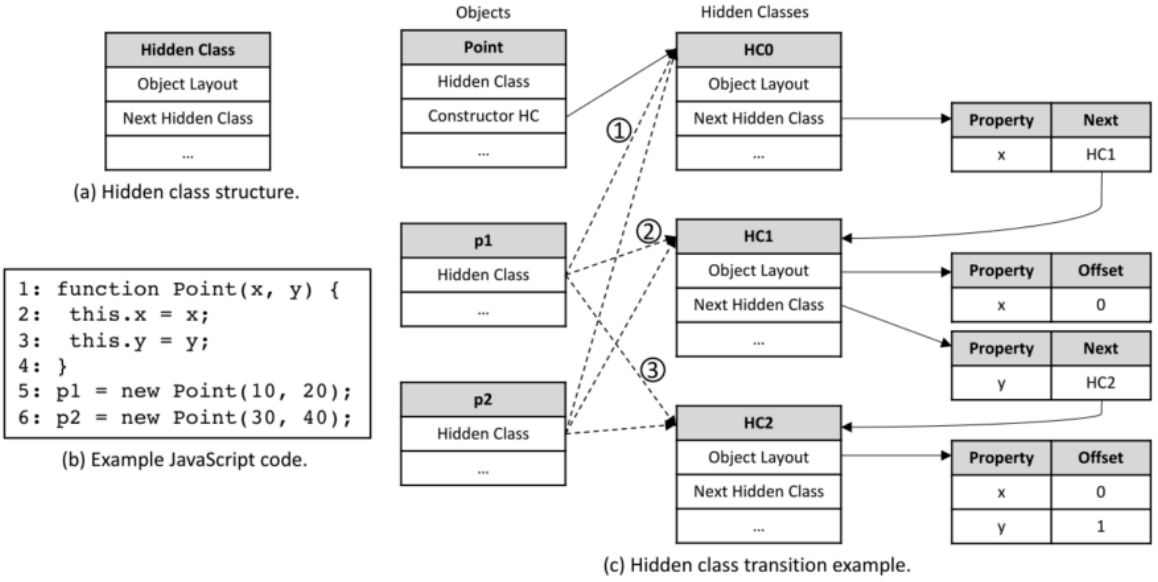


内联缓存

> [javascript - Google V8系列（三）V8提升函数执行效率的策略：Inline Cache（内联缓存） - 个人文章 - SegmentFault 思否]

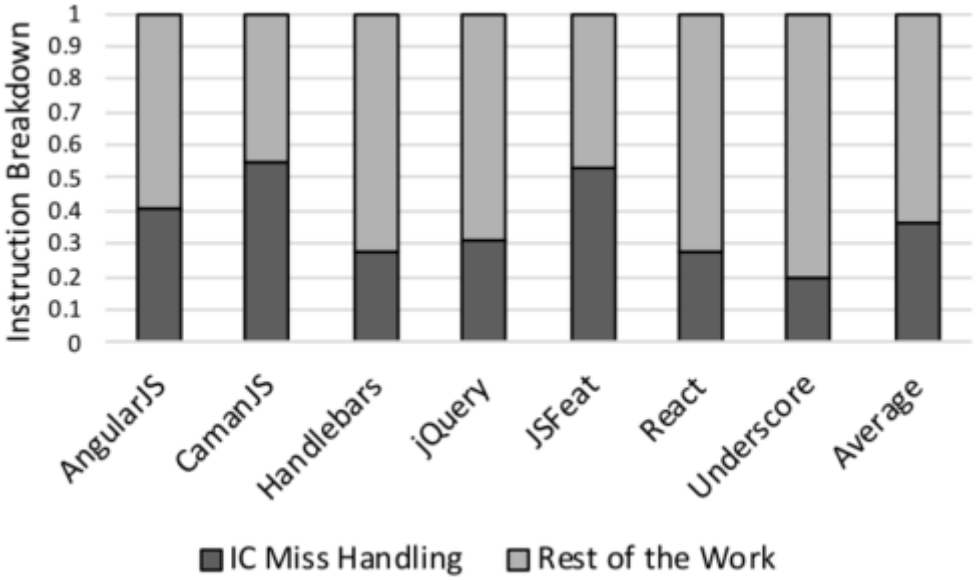
内联缓存（Inline Caching，IC）是一种优化技术，主要用于动态语言（如 JavaScript、Python 等）的方法调用和属性访问，以提高性能。

内联缓存通过存储最近使用的对象和其属性的查找方式来减少后续的查找开销。当一个方法被调用或属性被访问时，内联缓存会检查之前的调用是否与当前对象的类型相同(通过比较隐藏类的方式)。如果相同，则直接使用缓存的结果；如果不同，则进行正常的查找并更新缓存。



动机

1. IC Miss在应用的初始化过程中开销占比较高，平均达到了36%。本文通过插装的形式统计了在IC Miss期间执行的指令数量和程序执行的总指令数量，并使用他们的比例作为IC Miss的开销。



2. 有很多IC的handler是上下文无关的，这些上下文无关的handler是可以跨执行重用的。本文统计了几个js库中上下文无关的ic handler的占比，平均达到了59.6%。

Library	# of Diff. Hidden Classes	# of IC Misses	# of IC Misses per HC	% of Context Independent Handlers
AngularJS	138	799	5.8	62.5
CamanJS	99	383	3.9	61.8
Handlebars	88	541	6.2	63.2
jQuery	271	1547	5.7	57.3
JSFeat	116	323	2.8	51.7
React	360	2356	6.5	82.3
Underscore	123	295	2.4	38.1
Average	171	892	4.8	59.6

核心思路

初始执行与重用执行

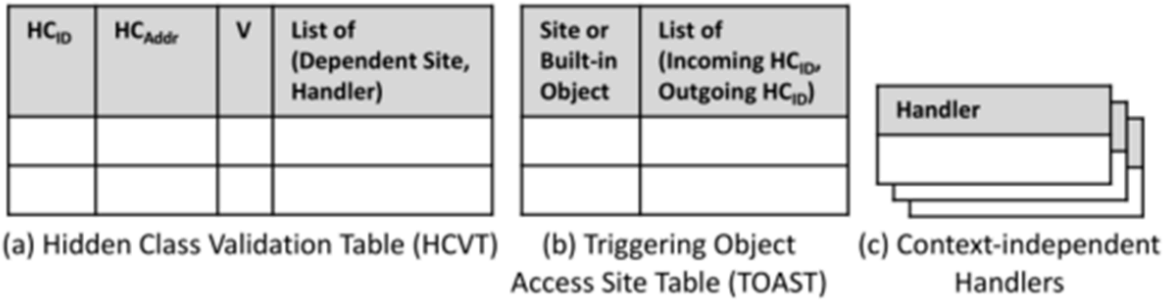
- Initial Run：程序初始运行，在这次运行中会记录IC信息以供后续执行重用
- Reuse Run：程序的后续运行，可以重用之前记录的IC信息

Idea：

- 在*触发点**(Triggering site)***的IC Miss会导致一些*关联点**(Dependent site)***的IC Miss
  - 触发点：发生IC Miss且有新隐藏类创建的对象属性访问点
  - 关联点：由于上述创建的新隐藏类而发生IC Miss的对象属性访问点
- 大部分的IC中的handler是上下文无关的
- 通过在运行时增量地验证在“Initial Run”中记录的IC信息能否在“Reuse Run”中重用

实现

ICRecord：用于重用IC的数据结构



- HCVT(Hidden Class Validation Table)
  - HC$_{ID}$: 隐藏类的序号
  - HC$_{Addr}$: 隐藏类的地址
  - V: 是否已经被验证
  - List of (Dependent Site, Handler): 记录该隐藏类的关联点和对应的handler
- TOAST(Triggering Object Access Site Table)

    - Site or Built-in Object: 触发点的序号或内建对象名
    - List of (Incoming HC_{ID}, Outgoing HC_{ID}): Incoming HC_{ID} 和Outgoing HC_{ID}分别为输入的隐藏类的序号和新创建的隐藏类的序号。对于内建对象，他的Incoming HC_{ID}为空，Outcoming HC_{ID}为内建对象的序号
- Handler：记录的上下文无关的Handler

Initial Run: 导出ICRecord

- 记录遇到的所有的隐藏类到HCVT，空置他们的HC_{Addr}域，V域置为0
- 识别所有的触发点和关联点，对于一个触发点，记录它的输入HC_{ID} 和输出HC_{ID}到TOAST，并且输出HC_{ID}以及和关联点和对应的Handler记录到HCVT

Reuse Run: 使用ICRecord

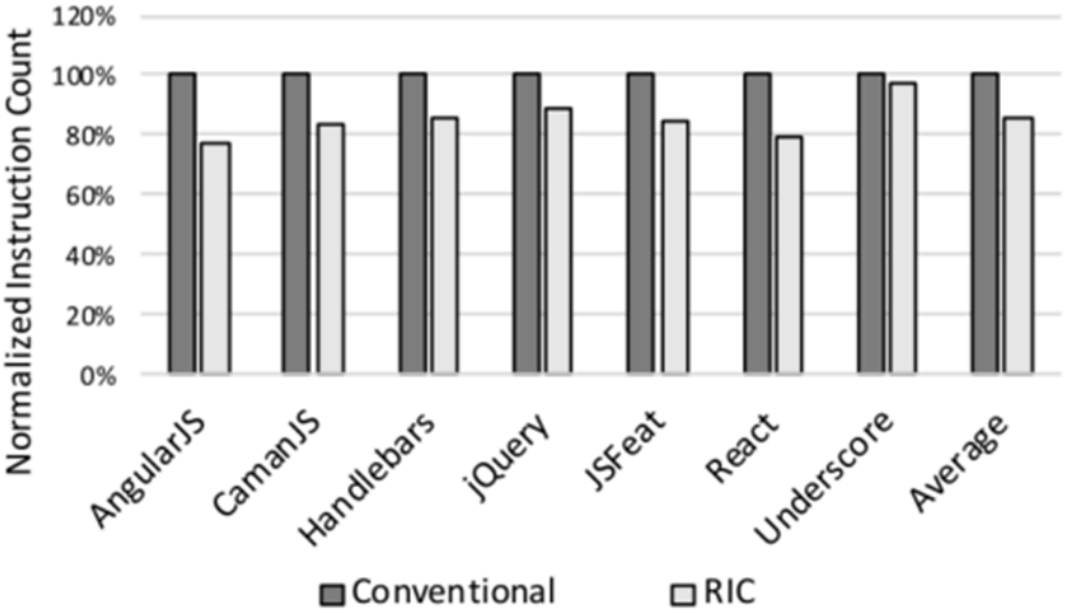
- 在JS运行时内创建内建对象时
    - 查找TOAST表中对应内建对象的条目，获取Outgoing HC_{ID}，根据Outgoing HC_{ID}查找HCVT中的条目，并且填写隐藏类的地址，将V置为1
- 对于触发点：
    - 查找TOAST表中的Incoming HC_{ID}，根据该ID找到HCVT中对应的条目，比较记录的HC_{Addr}和当前的隐藏类的地址。若地址匹配，则：
        - 根据当前操作生成的新的隐藏类更新Outgoing HC_{ID}对应的HCVT条目中的隐藏类地址，并将V置为1
        - 根据HCVT中Outgoing HC_{ID}对应条目的List of (Dependent Site, Handler)，更新实际的ICVector，从而能避免关联点的IC miss
    - 若地址不匹配则按照正常的流程处理

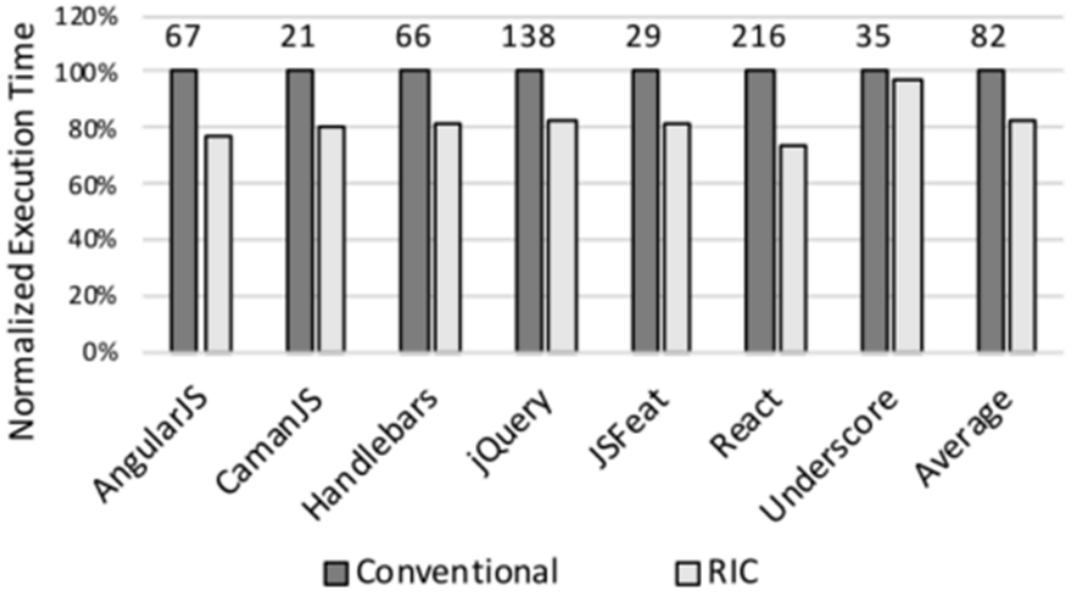
评估

1. RIC可以将IC Miss从49.19%降低到24.08%。其中Reuse Run中Cache Miss大多数是由于记录的RIC没有被命中导致的

Library	Initial Run	Reuse Run			
	IC Miss Rate (%)	IC Miss Rate (%)	Contribution To Miss Rate (%)		
			Handler	Global	Other
AngularJS	68.94	32.79	8.63	2.85	21.31
CamanJS	87.64	43.94	1.14	3.43	39.36
Handlebars	57.92	20.34	4.82	1.07	14.45
jQuery	48.50	29.28	6.49	1.13	21.66
JSFeat	18.96	8.16	0.18	1.82	6.16
React	18.67	3.83	1.90	0.31	1.62
Underscore	43.70	30.22	1.48	1.78	26.96
Average	49.19	24.08	3.52	1.77	18.79

2. 平均可以降低15%的执行的指令数量和17%的执行时间





评价与启发

该文章很好地发现了一个内联缓存Miss的规律：对象属性访问点之间的IC Miss的发生是有联系的。并且利用这个规律实现了对IC的重用。

本文从对背景和动机的描述到核心idea和实现方式的描述整个过程的逻辑很顺滑，并且使用了一个很巧妙的例子讲解了IC以及RIC的实现方式，简单易懂。

其他的启发：实际上在V8中，IC信息通过Feedback vector记录，而Feedback Vector不仅记录了IC信息，还记录了一些操作的类型信息，例如 `a + b` 这里的加法是字符串加法还是数值的加法。这些信息同样是上下文无关，可以考虑重用。