```
//ConstantPoolCache的内存布局
/*
    偏移(10) 偏移(16)  字段                  类型
    ---- ------    --------               --------------------
    0        0         _length               int
    4        4         _constant_pool        ConstantPool *

    ConstantPoolCacheEntry (0)
    --------------------------
    8        8         _indices              intx //占4个字节
    12       C         _f1                   Metadata*
    16       10        _f2                   intx
    20       14        _flags                intx
    --------------------------

    ConstantPoolCacheEntry (1)
    --------------------------
    24       18        _indices              intx //占4个字节
    28       1C        _f1                   Metadata*
    32       20        _f2                   intx
    36       24        _flags                intx
    --------------------------

    ......

    ConstantPoolCacheEntry (n)
    --------------------------
    ......
    --------------------------
*/


// The ConstantPoolCache is not a cache! It is the resolution table that the
// interpreter uses to avoid going into the runtime and a way to access resolved
// values.

// A ConstantPoolCacheEntry describes an individual entry of the constant
// pool cache. There's 2 principal kinds of entries: field entries for in-
// stance & static field access, and method entries for invokes. Some of
// the entry layout is shared and looks as follows:
//
// bit number |31                0|
// bit length |-8--|-8--|---16----|
// --------------------------------
// _indices   [ b2 | b1 |  index  ]  index = constant_pool_index
// _f1        [  entry specific   ]  metadata ptr (method or klass)
// _f2        [  entry specific   ]  vtable or res_ref index, or vfinal method ptr
// _flags     [tos|0|F=1|0|0|0|f|v|0 |0000|field_index] (for field entries)
// bit length [ 4 |1| 1 |1|1|1|1|1|1 |-4--|----16-----]
// _flags     [tos|0|F=0|M|A|I|f|0|vf|0000|00000|psize] (for method entries)
// bit length [ 4 |1| 1 |1|1|1|1|1|1 |-4--|--8--|--8--]

// --------------------------------
//
// with:
// index  = original constant pool index
// b1     = bytecode 1
// b2     = bytecode 2
// psize  = parameters size (method entries only)
// field_index = index into field information in holder InstanceKlass
//          The index max is 0xffff (max number of fields in constant pool)
//          and is multiplied by (InstanceKlass::next_offset) when accessing.
// tos    = TosState
// F      = the entry is for a field (or F=0 for a method)
// A      = call site has an appendix argument (loaded from resolved references)
// I      = interface call is forced virtual (must use a vtable index or vfinal)
// f      = field or method is final
// v      = field is volatile
// vf     = virtual but final (method entries only: is_vfinal())
//
// The flags after TosState have the following interpretation:
// bit 27: 0 for fields, 1 for methods
// f  flag true if field is marked final
// v  flag true if field is volatile (only for fields)
// f2 flag true if f2 contains an oop (e.g., virtual final method)
// fv flag true if invokeinterface used for method in class Object
//
// The flags 31, 30, 29, 28 together build a 4 bit number 0 to 8 with the
// following mapping to the TosState states:
//
// btos: 0
```

```
// ctos: 1
// stos: 2
// itos: 3
// ltos: 4
// ftos: 5
// dtos: 6
// atos: 7
// vtos: 8
//
// Entry specific: field entries:
// _indices = get (b1 section) and put (b2 section) bytecodes, original constant pool index
// _f1      = field holder (as a java.lang.Class, not a Klass*)
// _f2      = field offset in bytes
// _flags   = field type information, original FieldInfo index in field holder
//            (field_index section)
//
// Entry specific: method entries:
// _indices = invoke code for f1 (b1 section), invoke code for f2 (b2 section),
//            original constant pool index
// _f1      = Method* for non-virtual calls, unused by virtual calls.
//            for interface calls, which are essentially virtual but need a klass,
//            contains Klass* for the corresponding interface.
//            for invokedynamic, f1 contains a site-specific CallSite object (as an appendix)
//            for invokehandle, f1 contains a site-specific MethodType object (as an appendix)
//            (upcoming metadata changes will move the appendix to a separate array)
// _f2      = vtable/itable index (or final Method*) for virtual calls only,
//            unused by non-virtual.  The is_vfinal flag indicates this is a
//            method pointer for a final method, not an index.
// _flags   = method type info (t section),
//            virtual final bit (vfinal),
//            parameter size (psize section)
//
// Note: invokevirtual & invokespecial bytecodes can share the same constant
//       pool entry and thus the same constant pool cache entry. All invoke
//       bytecodes but invokevirtual use only _f1 and the corresponding b1
//       bytecode, while invokevirtual uses only _f2 and the corresponding
//       b2 bytecode.  The value of _flags is shared for both types of entries.
//
// The fields are volatile so that they are stored in the order written in the
// source code.  The _indices field with the bytecode must be written last.
```