- Runtime
  - smalltalk
    - 万物皆对象
    - 所有的方法调用即发送消息
  - 对象（id）定义objc_object
    - isa指针： isa_t isa
      - isa 指针不总是指向实例对象所属的类，不能依靠它来确定类型，而是应该用 class 方法来确定实例对象的类
      - 实例对象->类->元类->根元类（NSObject的元类），根元类的isa指向自己；
      - http://ios.jobbole.com/89485/
  - 类(Class)的定义objc_class：继承自objc_object
    - objc_class super_class;//父类
      - superclass：类->父类->...->根类NSObject，元类->父元类->...->根元类->根类，NSObject的superclass指向nil
    - cache_t cache
      - 作用：为方法调用的性能进行优化，通俗地讲，每当实例对象接收到一个消息时，它不会直接在isa指向的类的方法列表中遍历查找能够响应消息的方法，因为这样效率太低了，而是优先在 cache 中查找。Runtime 系统会把被调用的方法存到 cache 中（理论上讲一个方法如果被调用，那么它有可能今后还会被调用），下次查找的时候效率更高。
    - class_data_bits_t bits
      - class_rw_t
        - 运行时对类拓展的能力，运行时加载category时，把category中的方法、属性、协议添加到这里。
        - method_array_t methods;
property_array_t properties;
protocol_array_t protocols;
        - 没有实例变量列表，即表示运行时不允许对类增加实例变量
          - 引入概念Non Fragile ivars，健壮实例变量
            - 背景：编译阶段，类的实例变量内存布局已经确定，当增删父类实例变量时，必须重新编译才能保证子类实例变量没有问题。
            - 场景：iOS系统版本升级，如果是Fragile ivars特性，会导致旧的包不能正常运行，需要重新编译上传新包。
            - 特点：在健壮的实例变量下编译器生成的实例变量布局跟以前一样，但是当 runtime 系统检测到与超类有部分重叠时它会调整你新添加的实例变量的位移，那样你在子类中新添加的成员就被保护起来了。
        - class_ro_t
          - 存储类在编译期间确定的信息
          - method_list_t
protocol_list_t
property_list_t
            - 全部是entsize_list_tt的子类
            - 实现了Non Fragile ivars特性，编译阶段会给instanceStart 和 instanceSize赋值，确定好编译时每个类的所占内存区域起始偏移量和大小，这样只需将子类与基类的这两个变量作对比即可知道子类是否与基类有重叠
          - ivar_list_t，实例变量列表
            - 根据实例查找其在类中的名字，也就是“反射”：
    - 问题：为什么不能利用Non Fragile ivars的特性实现在runtime阶段增加实例变量?
    - http://yulingtianxia.com/blog/2014/11/05/objective-c-runtime/
  - 消息发送
    - objc_msgSend
objc_msgSend_stret,
objc_msgSendSuper
objc_msgSendSuper_stret
      - 流程
        - 1、检测selector是否需要忽略(ARC中的retain，release)
        - 2、检测target是不是nil
        - 3、在cache中查找该类的IMP
        - 4、在方法分发表中查找IMP
        - 5、在superClass方法分发表中查找IMP
        - 6、动态方法解析
      - 参数
        - 对象：id（objc_object）
        - SEL：映射到方法的C字符串。
      - 隐藏参数
        - 在源代码方法的定义中并没有声明这两个参数。它们是在代码被编译时被插入实现中的。
        - 接收消息的对象self
        - 方法选择器 _cmd
    - 消息转发
      - 动态方法解析：resolveInstanceMethod:和resolveClassMethod:
        - 动态添加方法实现
      - 重定向：forwardingTargetForSelector
        - 替换消息的接受者为其他对象
      - 消息转发：forwardInvocation
        - 方法就像一个不能识别的消息的分发中心，将这些消息转发给不同接收对象。或者它也可以象一个运输站将所有的消息都发送给同一个接收对象。它可以将一个消息翻译成另外一个消息，或者简单的”吃掉“某些消息，因此没有响应也没有错误。
forwardInvocation:方法也可以对不同的消息提供同样的响应，这一切都取决于方法的具体实现。该方法所提供是将不同的对象链接到消息链的能力。
  - Method Swizzling
  - 经典问题
    - 为什么super.class输出与self.class相同，怎样获取父类
      - class_getSuperclass