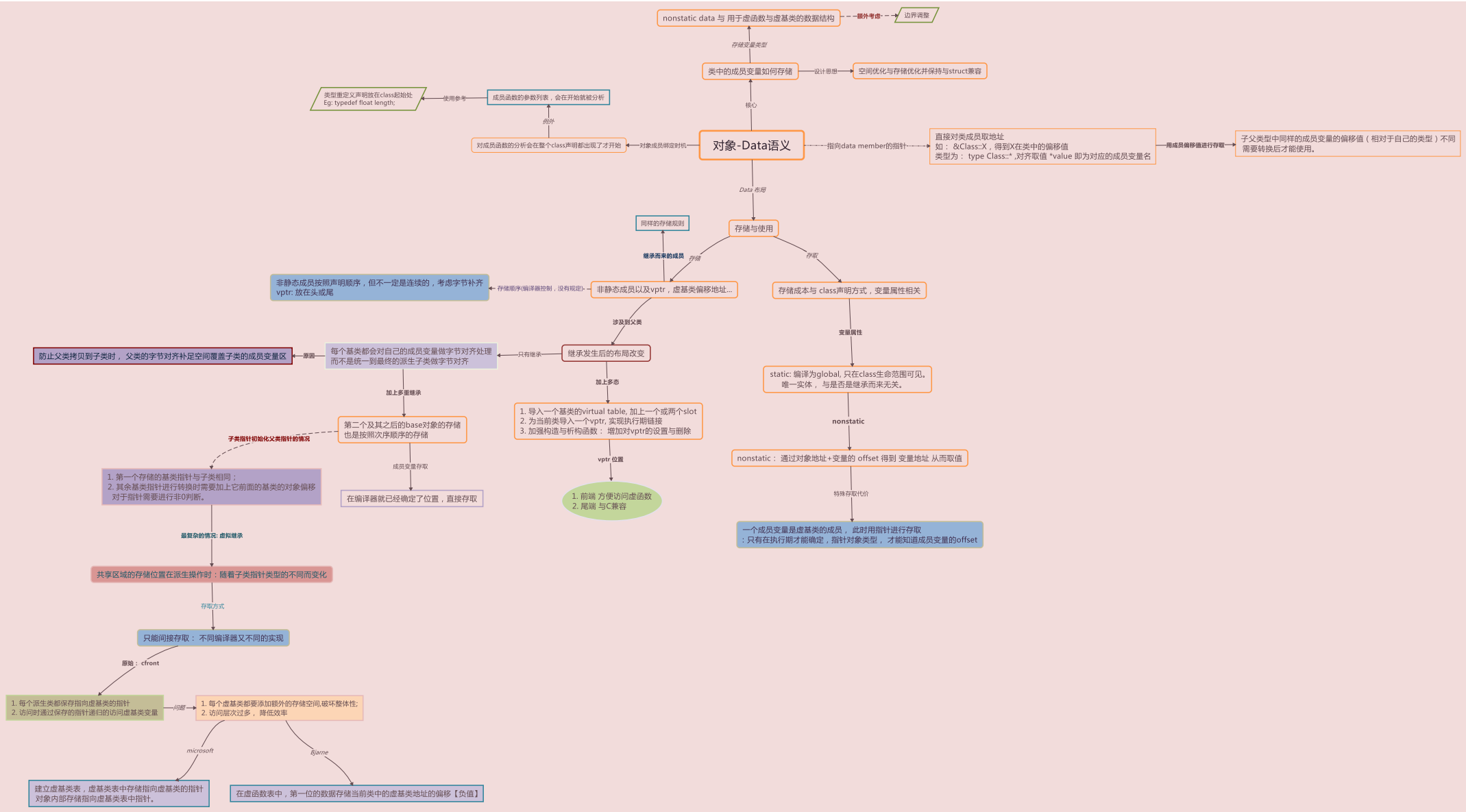
nonstatic data 与 用于虚函数与虚基类的数据结构
额外考虑
边界调整
存储变量类型
类中的成员变量如何存储
设计思想
空间优化与存储优化并保持与struct兼容
核心
类型重定义声明放在class起始处
Eg: typedef float length;
使用参考
成员函数的参数列表，会在开始就被分析
例外
对成员函数的分析会在整个class声明都出现了才开始
对象成员绑定时机
对象-Data语义
指向data member的指针
直接对类成员取地址
如： &Class::X，得到X在类中的偏移值
类型为： type Class::* ,对齐取值 *value 即为对应的成员变量名
用成员偏移值进行存取
子父类型中同样的成员变量的偏移值（相对于自己的类型）不同
需要转换后才能使用。
Data 布局
存储与使用
同样的存储规则
继承而来的成员
存储
存取
非静态成员按照声明顺序，但不一定是连续的，考虑字节补齐
vptr: 放在头或尾
存储顺序(编译器控制，没有规定)
非静态成员以及vptr，虚基类偏移地址...
存储成本与 class声明方式，变量属性相关
涉及到父类
变量属性
防止父类拷贝到子类时， 父类的字节对齐补足空间覆盖子类的成员变量区
原因
每个基类都会对自己的成员变量做字节对齐处理
而不是统一到最终的派生子类做字节对齐
只有继承
继承发生后的布局改变
static: 编译为global, 只在class生命范围可见。
唯一实体， 与是否是继承而来无关。
加上多重继承
加上多态
1. 导入一个基类的virtual table, 加上一个或两个slot
2. 为当前类导入一个vptr, 实现执行期链接
3. 加强构造与析构函数： 增加对vptr的设置与删除
nonstatic
第二个及其之后的base对象的存储
也是按照次序顺序的存储
子类指针初始化父类指针的情况
nonstatic： 通过对象地址+变量的 offset 得到 变量地址 从而取值
vptr 位置
成员变量存取
1. 第一个存储的基类指针与子类相同；
2. 其余基类指针进行转换时需要加上它前面的基类的对象偏移
对于指针需要进行非0判断。
在编译器就已经确定了位置，直接存取
1. 前端 方便访问虚函数
2. 尾端 与C兼容
特殊存取代价
一个成员变量是虚基类的成员， 此时用指针进行存取
；只有在执行期才能确定，指针对象类型， 才能知道成员变量的offset
最复杂的情况: 虚拟继承
共享区域的存储位置在派生操作时；随着子类指针类型的不同而变化
存取方式
只能间接存取： 不同编译器又不同的实现
原始： cfront
1. 每个派生类都保存指向虚基类的指针
2. 访问时通过保存的指针递归的访问虚基类变量
问题
1. 每个虚基类都要添加额外的存储空间,破坏整体性;
2. 访问层次过多， 降低效率
microsoft
Bjarne
建立虚基类表，虚基类表中存储指向虚基类的指针
对象内部存储指向虚基类表中指针。
在虚函数表中，第一位的数据存储当前类中的虚基类地址的偏移【负值】